On the All-Pairs Shortest Path Algorithm of Moffat and Takaoka*

Kurt Mehlhorn and Volker Priebe**

Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany

Abstract. We review how to solve the all-pairs shortest path problem in a non-negatively weighted digraph with n vertices in expected time $O(n^2 \log n)$. This bound is shown to hold with high probability for a wide class of probability distributions on non-negatively weighted digraphs. We also prove that for a large class of probability distributions $\Omega(n \log n)$ time is necessary with high probability to compute shortest path distances with respect to a single source.

1 Introduction

Given a complete digraph in which all the edges have non-negative length, we want to compute the shortest path distance between each pair of vertices. This is one of the most basic questions in graph algorithms, since a variety of combinatorial optimization problems can be expressed in these terms. As far as worst-case complexity is concerned, we can solve an n-vertex problem in time $O(n^3)$ by either Floyd's algorithm [3] or by n calls of Dijkstra's algorithm [2]. Fredman's algorithm [4] uses efficient distance matrix multiplication techniques and results in a running time of $O(n^3((\log\log n)/\log n)^{1/3})$ (slightly improved to $O(n^3((\log\log n)/\log n)^{1/2})$ by Takaoka [14]). Recently, Karger, Koller, and Phillips [8] presented an algorithm that runs in time $O(nm^* + n^2 \log n)$, where m^* denotes the number of edges that are a shortest path from their source to their target.

However, worst-case analysis sometimes fails to cover the advantages of algorithms that perform well in practice; average-case analysis has turned out to be more appropriate for these purposes. We are not only interested in algorithms with good expected running time but in algorithms that finish their computations within a certain time bound *with high probability* (and might therefore be called reliable).

Two kinds of probability distributions on non-negatively weighted complete digraphs have been considered in the literature. In the so-called *uniform model*, the edge lengths are independent, identically distributed random variables. In

* This work is supported by the ESPRIT II Basic Research Actions Program of the EC under contract no. 7141 (project ALCOM II) and the BMFT-project "Softwareökonomie und Softwaresicherheit" ITS 9103

** Research supported by a Graduiertenkolleg graduate fellowship of the Deutsche Forschungsgemeinschaft. E-mail: priebe@mpi-sb.mpg.de

the so-called *endpoint-independent model*, a sequence c_{vj}, $1 \leq j \leq n$, of n non-negative weights is fixed for each vertex v of K_n arbitrarily. These weights are assigned randomly to the n edges with source v, i.e., a random injective mapping π_v from $[1..n]$ to V is chosen and c_{vj} is made the weight of edge $(v, \pi_v(j))$ for all $j, 1 \leq j \leq n$.

Frieze and Grimmett [6] gave an algorithm with $O(n^2 \log n)$ expected running time in the uniform model when the common distribution function F of the edge weights satisfies $F(0) = 0$, $F'(0)$ exists, and $F'(0) > 0$. Under these assumptions, $m^* = O(n \log n)$ with high probability and so the algorithm of Karger et al. also achieves running time $O(n^2 \log n)$ with high probability.

The endpoint-independent model is much more general and therefore harder to analyze. Spira [12] proved an expected time bound of $O(n^2(\log n)^2)$, which was later improved by Bloniarz [1] to $O(n^2 \log n \log^* n)$. (We use log to denote logarithms to base 2 and ln to denote natural logarithms; $\log^* x := 1$ for $x \leq 2$ and $\log^* x := 1 + \log^* \log x$ for $x > 2$.) In [10] and [11], Moffat and Takaoka describe two algorithms with an expected time bound of $O(n^2 \log n)$. The algorithm in [11] is a simplified version of [10]. In this paper,

- we present an even simpler version of the algorithm in [11] and also correct a small oversight in the analysis given by Moffat and Takaoka.
- Moreover, we prove that the running time of the modified version is $O(n^2 \log n)$ with high probability.
- We show that under modest assumptions $\Omega(n \log n)$ edges need to be inspected to compute the shortest path distances with respect to a single source.

2 Preliminaries

We mention some results from discrete probability theory. Suppose that in a sequence of independent trials, the probability of success is $\geq p$ for each of the trials. Then the expected number of trials until the first successful one is $\leq 1/p$. In the so-called *coupon collector's problem*, we are given a set of n distinct coupons. In each trial, a coupon is drawn (with replacement) uniformly and independently at random. Let X denote the number of trials required to have seen at least one copy of each coupon. By the above argument, $E[X] = \sum_{0 \leq i < n} \frac{n}{n-i} \sim n \ln n$. Actually, it is rather unlikely that we deviate from the expected number of trials by more than a constant multiplicative factor, since the probability that a particular coupon has not been collected after r trials equals $(1 - \frac{1}{n})^r$. Hence, for any $\beta > 1$,

$$\Pr(X > \beta n \ln n) \leq n \left(1 - \frac{1}{n}\right)^{\beta n \ln n} \leq n e^{-\beta \ln n} = n^{-(\beta - 1)} . \qquad (1)$$

For a problem of size n, we will say that an event occurs *with high probability*, if it occurs with probability $\geq 1 - O(n^{-C})$ for an arbitrary but fixed constant C and large enough n. For example, (1) tells us that the number of trials in the coupon collector's problem is $O(n \ln n)$ with high probability.

2.1 A Probabilistic Experiment

We will refer to the following probabilistic experiment: An urn contains n balls that are either red or blue; let m be the number of red balls. The balls are repeatedly drawn from the urn (without replacement) uniformly and independently at random. For $1 \leq k \leq m$, let the random variable W_k denote the waiting time for the k-th red ball. In addition, we define the random variables Y_i, $1 \leq i \leq m$, by $Y_1 := W_1$ and $Y_i := W_i - W_{i-1}$ for $2 \leq i \leq m$. Note that both the W_k's and the Y_i's are not independent, e.g., $W_m = \sum_{1 \leq i \leq m} Y_i \leq n$ whereas each Y_i can take values in $\{1, \ldots, n-m+1\}$. It is shown in the appendix that the Y_i's are exchangeable random variables; in particular, for any i with $1 \leq i \leq m$,

$$E[Y_i] = \frac{n+1}{m+1} \ . \tag{2}$$

It will prove convenient to normalize the Y_i's. For $1 \leq i \leq m$, define random variables $Z_i := (Y_i - 1)/(n-m)$. The Z_i's take values in $[0,1]$; for any j, $1 \leq j \leq n-m+1$, $\Pr\left(Z_i = \frac{j-1}{n-m}\right) = \Pr(Y_i = j)$. By linearity of expectation,

$$E[Z_i] = \frac{E[Y_i] - 1}{n-m} = \frac{1}{m+1} \quad \text{for} \quad 1 \leq i \leq m \ . \tag{3}$$

The Z_i's are dependent as well, therefore, we do not expect the relation $E[\prod Z_i] = \prod E[Z_i]$ to hold. However, considering the underlying experiment, we may conjecture that if Z_1 is 'large', then it is less likely to occur that Z_2 is 'large' as well. The following lemma proves that the Z_i's are indeed negatively correlated. (A proof is given in the appendix.)

Lemma 1. *For any $I \subseteq \{1, \ldots, m\}$ with $|I| = k \geq 1$,*

$$E\left[\prod_{i \in I} Z_i\right] = \frac{[n-m]_k}{(n-m)^k} \cdot \frac{1}{[m+k]_k} \leq \frac{1}{(m+1)^k} = \prod_{i \in I} E[Z_i] \ ,$$

where $[x]_k := x \cdot (x-1) \cdots (x-k+1)$.

Lemma 1 suffices to establish large deviation estimates of the Chernoff-Hoeffding kind for $Z := \sum_i Z_i$. Let $X_1, \ldots, X_n$ be random variables that take values in $[0,1]$ and let $X := \sum_{1 \leq i \leq n} X_i$. Following [13], we call the X_i's 1-correlated if for all non-empty $I \subseteq \{1, \ldots, n\}$

$$E\left[\prod_{i \in I} X_i\right] \leq \prod_{i \in I} E[X_i] \ .$$

Note that if the random variables $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$ are both 1-correlated and if the X_i's are independent of the Y_j's, then the whole set of random variables $X_1, \ldots, X_n, Y_1, \ldots, Y_m$ is 1-correlated as well.

Lemma 2 ([13]). *Let X be the sum of 1-correlated random variables $X_1, \ldots, X_n$ with values in $[0,1]$. Then for any $\varepsilon > 0$,*

$$\Pr(X > (1+\varepsilon)E[X]) \leq \left(\frac{e^{\varepsilon}}{(1+\varepsilon)^{(1+\varepsilon)}}\right)^{E[X]} \ . \tag{4}$$

Note that for $\varepsilon \geq 2e - 1$, (4) implies that

$$\Pr(X > (1+\varepsilon)E[X]) \leq 2^{-(1+\varepsilon)E[X]} .$$

We will also use *Azuma's inequality*. The following formulation appears in [9].

Lemma 3. *Let $X_1, \ldots, X_n$ be independent random variables, with X_k taking values in a set A_k for each k. Suppose that the function $f : \prod A_k \to \mathbb{R}$ satisfies $|f(x) - f(y)| \leq c$ whenever the vectors x and y differ only in a single coordinate. Let Y be the random variable $f(X_1, \ldots, X_n)$. Then for any $t > 0$,*

$$\Pr(|Y - E[Y]| \geq t) \leq 2e^{-2t^2/(nc^2)} .$$

We use the following terminology for weighted digraphs. For an edge $e = (u, v)$, we call u the *source* and v the *target* or *endpoint* of e. The weight of an edge is denoted by $c(e)$. We will interpret an entry in the adjacency list of a vertex u either as the endpoint v of an edge with source u or as the edge (u, v) itself, as is convenient.

3 The Algorithm of Moffat and Takaoka

We will now review the algorithm of Moffat and Takaoka in [11] and their analysis. We are given a complete digraph on n vertices with non-negative edge weights. The algorithm first sorts all adjacency lists in order of increasing weight (total time $O(n^2 \log n)$) and then solves n single source shortest path problems, one for each vertex of G. The single source shortest path problem, say, with source $s \in V$, is solved in two phases. Both phases are variants of Dijkstra's algorithm [2] and only differ in the way the priority queue is handled.

Dijkstra's algorithm labels the vertices in order of increasing distance from the source. We use S to denote the set of labeled vertices and $U = V - S$ to denote the set of unlabeled vertices. Initially, only the source vertex is labeled, i.e., $S = \{s\}$. For each labeled vertex v, its exact distance $d(v)$ from the source is known. For the source node s, we have $d(s) = 0$. For each labeled vertex v, one of its outgoing edges is called its *current edge* and is denoted $ce(v)$. We maintain the invariant that all edges preceding the current edge $ce(v)$ in v's (sorted) adjacency list have their endpoint already labeled. Both phases use a priority queue. A priority queue stores a set of pairs (x, k) where k is a real number and is called the key of the pair. We assume that priority queues are implemented as Fibonacci heaps [5]. Fibonacci heaps support the insertion of a new pair (x, k) in constant time and the deletion of a pair with minimum key (*delete min* operation) in amortized time $O(\log p)$ where p is the number of pairs in the priority queue. They also support an operation *decrease key* in constant amortized time. A *decrease key* operation takes a pointer to a node in a Fibonacci heap containing, say, the pair (x, k), and allows the replacement of k by a smaller key k'.

Phase I. In Phase I, the additional invariant is maintained that the targets of all current edges are unlabeled. For each vertex $u \in U$, we also maintain a list $L(u)$ of all vertices $v \in S$ whose current edge ends in u. The priority queue contains all vertices in U. The key of a vertex $u \in U$ is $\min_{v \in L(u)} d(v) + c(v,u)$. In each iteration of Phase I, the vertex $u \in U$ with minimal key value $d(u)$ is selected and is deleted from the priority queue by a *delete min* operation. The vertex u is added to S and for each vertex $v \in \{u\} \cup L(u)$, the current edge $ce(v)$ is advanced to the first edge in v's (sorted) adjacency list whose target is in $V - S$. For any $v \in \{u\} \cup L(u)$, let $ce(v) = (v,w)$ be the new current edge of v and denote w's current key by d_w. We add v to $L(w)$, and if $d(v) + c(v,w) < d_w$, we decrease d_w appropriately. This implies a *decrease key* operation on the priority queue. By our assumption on the implementation of the priority queue, the cost of an iteration of Phase I is $O(\log n)$ plus the number of edges scanned. Phase I ends when $|U|$ becomes $n/\log n$.

Remark: Moffat and Takaoka use a binary heap instead of a Fibonacci heap to realize the priority queue; Fibonacci heaps did not exist at that time. Since a *decrease key* operation in a binary heap takes logarithmic time, they use a slightly different strategy for Phase I. They keep the vertices in S in the priority queue. The key of vertex $v \in S$ is $d(v) + c(ce(v))$. In each iteration, the vertex of minimum key, say, vertex $v \in S$, is selected from the heap. Let w be the target of the current edge of v. The current edge $ce(z)$ is advanced for all $z \in \{w\} \cup L(w)$. Then w is inserted into the priority queue with key $d(w) + c(ce(w))$, and for each $z \in L(w)$, the key of z is increased (since the weight of the new current edge is greater than the weight of the old current edge of z). Moffat and Takaoka show that the expected cost of an *increase key* operation is constant in a binary heap. We believe that the implementation using Fibonacci heaps is slightly simpler since it does not use a non-standard operation on priority queues.

What is the total number of edges scanned in Phase I? In [11], Moffat and Takaoka argue as follows: Let U_0 be the set of unlabeled vertices at the end of Phase I. Then $|U_0| = n/\log n$. Since for every vertex v the endpoints of the edges out of v form a random permutation of V, we should expect to scan about $\log n$ edges in each adjacency list during Phase I and hence about $n \log n$ edges altogether. This argument is incorrect as U_0 is determined by the orderings of the adjacency lists and cannot be fixed independently. The following example makes this fact obvious. Assume that all edges out of the source have length one and all other edges have length two. Then Phase I scans $n - n/\log n$ edges out of the source vertex and U_0 is determined by the last $n/\log n$ edges in the adjacency list of the source. Thus the conclusion that one scans about $\log n$ edges in each adjacency list is wrong. However, the derived conclusion that the expected total number of scanned edges is $O(n \log n)$ is true, as the following argument shows.

Consider Phase I′, the following modification of Phase I (similar to Spira's algorithm [12]). In this modification, the target of a current edge may be labeled and the vertices $v \in S$ are kept in a priority queue with keys $d(v) + c(ce(v))$. When a vertex v with minimum key is selected, let w be the target of $ce(v)$. If w does not belong to S, then w is added to S and to the priority queue. In any

case, $ce(v)$ is advanced to the *next* edge and v's key is increased appropriately. The modified algorithm finishes Phase I' when $|V - S| = n/\log n$. Let S_0 be the set of vertices that have been labeled in Phase I'. For every vertex $v \in S_0$, denote by $A(v)$ the set of edges out of v that have been scanned by the modified algorithm and denote by $S(v)$ the targets of the edges in $A(v)$.

The analysis of the coupon collector's problem implies that $E[\sum_v |A(v)|] = O(n \log n)$. Indeed, let X_i be the number of edges scanned when $|S| = i$. Since the targets of the edges are random, we have $E[X_i] \leq n/(n-i)$ and hence

$$E\left[\sum_{i \leq n - n/\log n} X_i\right] \leq n \sum_{i \leq n} 1/i \leq n(\ln n + 1) \ .$$

This proves that $E[\sum_v |A(v)|] = O(n \log n)$.

It turns out that Phase I of the algorithm by Moffat and Takaoka shows basically the same behavior. In fact, at the end of Phase I, their algorithm has labeled exactly the vertices in S_0, and all the edges in $\bigcup_v A(v)$ have been scanned by the Moffat and Takaoka algorithm as well. However, for the purpose of maintaining the invariant, the current edge pointer of each vertex $v \in S_0$ has been advanced to the first vertex in U_0 in v's adjacency list. For every vertex $v \in S_0$, let e_v and e'_v be the current edge of v at the end of Phase I in the algorithm by Moffat and Takaoka and at the end of Phase I' in the algorithm by Spira, respectively. e'_v precedes e_v in v's adjacency list and the edge e_v is the first edge after e'_v with target in U_0. We can imagine scanning the edges after e'_v only when Phase I' has terminated. Due to the endpoint-independent distribution of edge weights, the targets of the edges after e'_v in the adjacency list form a random permutation of $V - S(v) \supseteq V - S_0 = U_0$. This is the setting of the probabilistic experiment in Sect. 2.1, where the number of edges between e'_v and e_v corresponds to Y_1. Since $|V - S(v)| \leq n$ and $m := |U_0| = n/\log n$, we deduce from (2) in Sect. 2.1 that the expected number of edges between e'_v and e_v is $O(\log n)$. This completes the analysis of Phase I.

Phase II. In Phase II, the weaker additional invariant is maintained that the endpoint of every current edge belongs to U_0. We now keep the vertices $v \in S$ in the queue with key $d(v) + c(ce(v))$.

In each iteration of Phase II, a vertex with minimum key is selected from the queue. Say that vertex v is selected and that w is the endpoint of $ce(v)$. The vertex w is a random vertex in U_0 but it is not necessarily unlabeled. If w is unlabeled, it will be labeled, $d(w)$ is set to $d(v) + c(ce(v))$, and $ce(v)$ and $ce(w)$ are advanced to the next edge whose endpoint is in U_0. If w is already labeled, only $ce(v)$ is advanced. In either case the heap is updated appropriately. All of this takes time $O(\log n + \#\text{edges scanned})$.

We have already stated that w is a random element of U_0 and hence, when $|U| = i < n/\log n$, an expected number of $(n/\log n)/i$ iterations is required to decrease the cardinality of U by one. The expected number of iterations in Phase II is therefore bounded by

$$\frac{n}{\log n} \sum_{1 \leq i \leq n/\log n} \frac{1}{i} = O(n) \ .$$

Moreover, whenever the current edge of a vertex is advanced, it is advanced to the next edge having its endpoint in U_0. As argued above, for any vertex $v \in S$, the vertices in U_0 are distributed randomly in $V - S(v)$, and (2) in Sect. 2.1 shows that whenever $ce(v)$ is advanced in Phase II, it is advanced by an expected number of $O(\log n)$ edges. We conclude that the expected cost of one iteration in Phase II is $O(\log n)$ and, given that we do k iterations in Phase II, the expected cost of Phase II is $O(k \log n)$. Hence, the total expected cost of Phase II is $O(n \log n)$.

The above discussion is summarized in the following theorem.

Theorem 1. *For endpoint-independent distributions the algorithm of Moffat and Takaoka runs in expected time $O(n^2 \log n)$.*

We will next prove that the algorithm by Moffat and Takaoka is reliable, i.e., that, with high probability, its running time does not exceed its expectation by more than a constant multiplicative factor.

Theorem 2. *The running time of the all-pairs shortest path algorithm by Moffat and Takaoka is $O(n^2 \log n)$ with high probability.*

Proof. It is sufficient to prove that solving a single source shortest path problem takes time $O(n \log n)$ with high probability. As in the proof of Theorem 1, we analyze Phase I', the remaining part of Phase I, and Phase II separately, and we prove that each of them takes time $O(n \log n)$ with high probability. We use the notation that has been introduced for the proof of Theorem 1.

Recall that the running time of Phase I' is $O(n \log n)$ plus $\sum_{v \in S_0} |A(v)|$, the number of edges scanned. The tail estimate for the coupon collector's problem, (1) in Sect. 2, implies that $\sum_{v \in S_0} |A(v)|$ is $O(n \log n)$ with high probability.

For the analysis of the remaining part of Phase I, for any $v \in S_0$, define the random variable Y_v as being the number of edges between e'_v and e_v. With $m := n/\log n$ and $Z_v := (Y_v - 1)/(n - m)$, (2) and (3) in Sect. 2.1 imply that for $Y_I := \sum_{v \in S_0} Y_v$ and $Z_I := \sum_{v \in S_0} Z_v$, the expected values are $E[Y_I] = |S_0| \frac{n+1}{m+1} = \Theta(n \log n)$ and $E[Z_I] = |S_0|/(m+1) = \Theta(\log n)$. Since the Z_v's are independent random variables, we get from the usual Chernoff-Hoeffding bound (which is subsumed in Lemma 2) that

$$\Pr(Y_I > (1+\varepsilon)E[Y_I]) \leq \Pr(Z_I > (1+\varepsilon)E[Z_I]) \leq 2^{-(1+\varepsilon)E[Z_I]}$$

for large enough ε. This proves that $Y_I = O(n \log n)$ with high probability.

We now turn to the analysis of Phase II. Let the random variable Y_{II} denote the total number of edges scanned in Phase II; we know that $E[Y_{II}] = O(n \log n)$ from the proof of Theorem 1. Suppose that we perform k iterations in Phase II; then we can express Y_{II} as the sum of random variables Y_i, $1 \leq i \leq k$, where Y_i denotes the number of advances of the current edge pointer in the i-th iteration. With $m := n/\log n$, we introduce the normalized random variables $Z_i := (Y_i - 1)/(n - m)$, $1 \leq i \leq k$. Since some of the Z_i's might refer to the adjacency list of the same vertex, the Z_i's are not necessarily independent random variables.

However, Lemma 1 tells us that $Z_1, \ldots, Z_k$ are 1-correlated random variables. Lemma 2 provides a tail estimate for $Z^{(k)} := \sum_{i=1}^{k} Z_i$;

$$\Pr\left(Z^{(k)} > (1+\varepsilon)E[Z^{(k)}]\right) \leq 2^{-(1+\varepsilon)E[Z^{(k)}]}$$

for large enough ε. For $k = O(n)$, we set $(1+\varepsilon) = \Theta(n/k)$ to obtain that $Z^{(k)} = O(\log n)$ with high probability. If we abbreviate by I_k the event that we perform k iterations in Phase II, then

$$\begin{aligned}\Pr(Y_{II} > (1+\varepsilon)E[Y_{II}]) &= \textstyle\sum_k \Pr(Y_{II} > (1+\varepsilon)E[Y_{II}] \mid I_k) \cdot \Pr(I_k) \\ &\leq \textstyle\sum_k \Pr\left(Z^{(k)} > (1+\varepsilon)E[Z^{(k)}]\right) \cdot \Pr(I_k) \ .\end{aligned}$$

By the tail estimate for the coupon collector's problem, the number of iterations in Phase II is $O(n)$ with high probability. Hence, $Y_{II} = O(n \log n)$ with high probability. Again, because the number of iterations is $O(n)$ with high probability, the total time needed for updating the heap in Phase II is $O(n \log n)$ with high probability.

Thus we have proved that the running time of the algorithm is $O(n^2 \log n)$ with high probability. □

4 A Lower Bound for the Single Source Problem

Can we achieve running time $o(n^2 \log n)$ for solving the all-pairs shortest path problem? In certain situations we certainly can, e.g., if all edge weights are equal to one. However, in the general case of endpoint-independent distributions, it takes expected time $\Omega(n \log n)$ to compute the shortest path distances with respect to a single source, as we now argue.

Our underlying graph is $\tilde{K}_n = (V, E)$, the complete digraph on n vertices with loops. We restrict ourselves to the case of *simple* weight functions on the edges, i.e., for every vertex v and each integer k, $1 \leq k \leq n$, there is exactly one edge with weight k and source v. A single source shortest path algorithm gets as its input the problem size n, a source vertex s, and a simple weight function c. We assume that c is provided by means of an oracle that answers questions of the following kind:

(1) What is the weight $c(e)$ of a given edge e?
(2) Given a vertex $v \in V$ and an integer $k \in \{1, \ldots, n\}$, what is the target of the edge with weight k and source v?

The algorithm is supposed to compute the function d of shortest distances from s. It is allowed to ask the oracle questions of type (1) and (2), thereby gaining partial information on c. The complexity of the algorithm on a fixed simple weight function c is defined to be the number of questions the algorithm asked in order to compute the distance function d with respect to c.

For simple weight functions, the distance function d maps the set of vertices into $\mathbb{N}_0$. Define $D := \max\{d(v) \ ; \ v \in V\}$ and for all i, $0 \leq i \leq D$, let $V_i :=$

$\{v \ ; \ d(v) = i\}$. We call V_i the i-th layer with respect to d. For all i, $0 \leq i \leq D$, let $\ell(i) := |\{j \ ; \ j > i \text{ and } V_j \neq \emptyset\}|$ be the number of non-empty layers above layer i. Clearly, D, the sets V_i, and the function ℓ depend on c; for ease of notation, we do not make this dependence visible in the notation.

We first argue intuitively how to provide a lower bound on the complexity of a single source shortest path algorithm in terms of ℓ. Consider any vertex v of distance $d(v)$ from the source and suppose that the algorithm has not inquired about one of v's outgoing edges, say e, of length $c(e) < D - d(v)$. By omitting the check of e, the algorithm cannot exclude that $d(v) + c(e)$ is smaller than the distance label of the target of e, in which case the distance function computed by the algorithm would be incorrect.

Lemma 4. *Let c be a simple weight function and let d be the distance function with respect to c. Then any shortest path algorithm has complexity at least*

$$\sum_{u \in V} (\ell(d(u)) - 1) \ .$$

Proof. Let E' be the set of edges queried by the algorithm by a question of either type (1) or type (2). For an arbitrary but fixed vertex $u \in V$, let $E(u)$ be the set of edges with source u and let $E'(u) := E' \cap E(u)$. We prove that $|E'(u)| \geq \ell(d(u)) - 1$. This is clear if E' contains edge $e \in E(u)$ of weight $c(e) = j$ for all j, $1 \leq j < \ell(d(u))$. If there is an edge $e_i \in E(u) - E'$ with weight $c(e_i) = i < \ell(d(u))$, then every non-empty layer V_j above layer $d(u) + i$ must contain the target of an edge in $E'(u)$. Assume otherwise, then there is an edge $e_j = (u, v) \notin E'$ with $v \in V_j$ for a $j > d(u) + i$. Define the simple weight function c' by

$$c'(e) := \begin{cases} c(e), & \text{if } e \notin \{e_i, e_j\} \ ; \\ c(e_j), & \text{if } e = e_i \ ; \\ c(e_i), & \text{if } e = e_j \ . \end{cases}$$

Then $c'(e) = c(e)$ for all $e \in E'$, and therefore the algorithm will output d, the distance function with respect to c, on input c' as well. However, $d(v) = j > d(u) + i = d(u) + c'(e_j)$ for $e_j = (u, v)$, which shows that d is incorrect with respect to c'.

We choose $i = \min\{c(e) \ ; \ e \in E(u) - E'\}$. Note that all edges in $E(u)$ with targets in layer V_j, $j > d(u) + i$, must have weight at least $j - d(u) > i$ by the correctness of the algorithm. Hence, $|E'(u)| \geq i - 1 + \ell(d(u) + i) \geq \ell(d(u)) - 1$. □

Table 1 shows the distribution of vertices over distances for a (typical) simple weight function on a graph of $n = 10000$ vertices. Most vertices have distance about 14 ($\approx \log n$) from the source but there are vertices that have distance as much as 24 ($\approx 2 \log n$). By the argument of Lemma 4, we can guess that any (correct) algorithm must inquire about $\Omega(n \log n)$ edges.

In the remainder of this section, we make this argument more precise. We derive a lower bound of $\Omega(n \log n)$ on the expected value of $\sum_{u \in V} \ell(d(u))$ for random simple weight functions c. More generally, we show that any algorithm has to ask $\Omega(n \log n)$ questions with high probability.

Table 1. A typical distribution of vertices over distances for $n = 10000$

distance d	0	1	2	3	4	5	6	7	8	9	10	11	12
# vertices	1	1	2	4	8	16	32	64	120	237	449	796	1306

distance d	13	14	15	16	17	18	19	20	21	22	23	24
# vertices	1845	1952	1562	910	415	181	58	20	16	2	1	2

Our proof strategy is as follows. The lower bound given by Lemma 4 depends only on the distance function d. For random simple weight functions, we re-interpret the calculation of d and the construction of the layers V_i as the outcome of a random labeling process. Note that a random simple weight function is given by n independent permutations of V, one for each vertex. The i-th vertex on the permutation for vertex v is the target of the edge with weight i and source v. The labeling process proceeds in stages. In the 0-th stage, V_0 is set to $\{s\}$ and $d(s)$ is set to 0. In the i-th stage, $i \geq 1$, each vertex $v \in S^{(i)} = \bigcup_{0 \leq j < i} V_j$ picks the $(i - d(v))$-th vertex in its adjacency list. Note that each vertex that v has not yet seen is equally likely to occur. The newly reached vertices are put into V_i and their d-value is set to i. Instead of fixing the n permutations beforehand, we may also view them as being fixed on-line (this is sometimes called the principle of deferred decisions). This leads to the following re-interpretation of the random labeling process: In the i-th stage, each vertex in $S^{(i)} = \bigcup_{0 \leq j < i} V_j$ chooses a vertex uniformly and independently at random from the set of vertices it has not yet seen. The labeling process stops if $S^{(k)} = V$ for some k.

A related process was considered by Frieze and Grimmett in [6]. They assumed that each vertex in $S^{(i)}$ chooses a vertex uniformly and independently at random from the set of *all* vertices. If D_A denotes the number of stages taken by this version of the process, then it is clear that D_A stochastically dominates D, i.e., for all m, $\Pr(D > m) \leq \Pr(D_A > m)$. Frieze and Grimmett prove in [6] that D_A (and hence D) is $O(\log n)$ with high probability. However, we need a lower bound on D and hence their result is of no use to us. (Nevertheless, our proof strategy was inspired by theirs.)

The random labeling process is said to be in state j, if $|S^{(i)}| = j$. We call stage i of the labeling process *central*, if $n/e \leq |S^{(i)}| \leq n - \sqrt{n}$. Layers constructed in central stages are called central.

Our proof will proceed in two steps. First, we show in Lemma 5 that there are $\Omega(\log n)$ central stages with high probability. Second, we prove in Lemma 6 that each central stage gives rise to a non-empty layer with high probability.

Lemma 5. *With high probability, the labeling process has $\Omega(\log n)$ central stages.*

Proof. For a random simple weight function c, let i_0 be the first central stage with respect to c. Then $n/e \leq |S^{(i_0)}| \leq 2n/e$, since $|S^{(i+1)}| \leq 2|S^{(i)}|$ for any $i \geq 0$. We will show that $|S^{(i_0+k)}| \leq n - \sqrt{n}$ with high probability for $k = (\ln n)/17$. Let $U = V - S^{(i_0)}$ be the set of vertices that are still unlabeled after stage i_0. Note that $|U| \geq (e-2)n/e \geq n/4$.

Let us condition on $m = |U|$. Construct an $n \times m$ matrix A with 0-1 entries as follows. The rows correspond to the vertices in V and the columns correspond to the vertices in U; entry a_{vu} is 1 if and only if the edge (v, u) is among the k shortest edges in v's adjacency list whose head is an element of U. Let $f(A)$ be the number of all-zero columns in A. Then $|S^{(i_0+k)}| \leq n - f(A)$ because no vertex in U corresponding to an all-zero column will be labeled in the k stages following stage i_0. Since A models a process in which all vertices (and not only those that are currently labeled) are allowed to label new vertices, and in which each vertex is prevented from choosing vertices that have been labeled by other vertices before stage i_0, $f(A)$ may seem to be a rather crude lower bound on $|V - S^{(i_0+k)}|$. However, we will now prove that even $f(A) \geq \sqrt{n}$ with high probability.

A row of A is a random 0-1 vector of length m with exactly k ones. Moreover, the row entries $A_{i\cdot}$, $1 \leq i \leq n$, are independent random variables, and if A, A' differ only in a single row, then $|f(A) - f(A')| \leq k$. Hence, by Azuma's inequality (Lemma 3), we get the following tail estimate for $f(A) = f(A_{1\cdot}, \ldots, A_{n\cdot})$,

$$\Pr(f(A) \leq E[f(A)]/2) \leq 2\exp(-E[f(A)]^2/(2nk^2)) \ .$$

The probability that a fixed column is all-zero is $(1 - k/m)^n$; therefore,

$$E[f(A)] = m\left(1 - \frac{k}{m}\right)^n \ . \tag{5}$$

Remember that $m = |U| \geq n/4$ and $k = (\ln n)/17$; since $(1 - 1/x)^x \geq e^{-2}$ for large enough x, we get from (5) that

$$E[f(A)] \geq me^{-2kn/m} \geq \tfrac{1}{4}n^{1-8/17} > 2\sqrt{n} \tag{6}$$

for large enough n, where $E[f(A)]$ is conditioned on m. However, the lower bound in (6) is independent of m. Hence,

$$\Pr(f(A) < \sqrt{n}) \leq 2\exp(-\Theta(n^{1/17})/(\ln n)^2) = O(n^{-C})$$

for any fixed $C > 0$ and large enough n. Since $|S^{(i_0+k)}|$ is increasing in k, we have thus proved that, with high probability, it will take $\Omega(\ln n) = \Omega(\log n)$ stages to label all but $\sqrt{n}$ vertices. □

Lemma 6. *With high probability, each central layer contains at least one vertex.*

Proof. Suppose the process is in state j at the beginning of stage i. For any vertex in $S^{(i)}$, the probability of selecting a vertex in $S^{(i)}$ during this stage is $\leq j/n$. Therefore, the next layer will remain empty with probability $\leq (j/n)^j$. Note that $x \mapsto (x/n)^x$ is an increasing function for $x > n/e$.

Let B denote the event that at least one central layer remains empty. By the estimates provided in the preceding paragraph,

$$\Pr(B) \leq \sum_{j=n/e}^{n-\sqrt{n}} \left(\frac{j}{n}\right)^j \leq n\left(\frac{n-\sqrt{n}}{n}\right)^{n-\sqrt{n}} \leq ne^{-\sqrt{n}+1} = O(n^{-C})$$

for sufficiently large n. □

Theorem 3. *Any algorithm for the single source shortest path problem has complexity $\Omega(n \log n)$ with high probability on random simple weight functions.*

Proof. Suppose that i is the first central stage of the labeling process; as before, let $S^{(i)}$ denote the set of vertices that have already been labeled up to this stage. By Lemma 5, with high probability, the process has $\Omega(\log n)$ central layers. Lemma 6 tells us that all these layers will be non-empty with high probability. With the notation introduced in the discussion of the labeling process, this reads

$$\sum_{u \in S^{(i)}} (\ell(d(u)) - 1) = \Omega(n \log n) \quad \text{with high probability.}$$

By Lemma 4, the left-hand side term is a lower bound on the complexity of any shortest path algorithm. □

Acknowledgements

We learned from discussions with Paul Spirakis that analyzing the Moffat and Takaoka algorithm [11] is not as easy as it might appear at first glance. The remarks of an anonymous referee for ICALP'94 allowed considerable simplification of our proofs of Theorems 1 and 2. Rudolf Fleischer suggested the use of Fibonacci heaps in the implementation of the algorithm. Finally, numerous discussions with Hannah Bast were particularly insightful, as conversations with Devdatt Dubhashi and Torben Hagerup helped to clarify our ideas.

References

1. P.A. Bloniarz, A shortest-path algorithm with expected time $O(n^2 \log n \log^* n)$, *SIAM J. Comput.* **12** (1983) 588–600
2. E.W. Dijkstra, A note on two problems in connexion with graphs, *Numer. Math.* 1 (1959) 269–271
3. R.W. Floyd, Algorithm 97: Shortest path, *Comm. ACM* **5** (1962) 345
4. M.L. Fredman, New bounds on the complexity of the shortest path problem, *SIAM J. Comput.* **5** (1976) 83–89
5. M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *J. ACM* **34** (1987) 596–615
6. A.M. Frieze and G.R. Grimmett, The shortest-path problem for graphs with random arc-lengths, *Discrete Appl. Math.* **10** (1985) 57–77
7. R.L. Graham, D.E. Knuth, and O. Patashnik, *Concrete Mathematics* (2nd ed.), Addison-Wesley, Reading, MA, 1994
8. D.R. Karger, D. Koller, and S.J. Phillips, Finding the hidden path: Time bounds for all-pairs shortest paths, *SIAM J. Comput.* **22** (1993) 1199–1217
9. C. McDiarmid, On the method of bounded differences, in: J. Siemons (Ed.), *Surveys in Combinatorics, 1989* (London Mathematical Society Lecture Notes Series; 141), Cambridge University Press, Cambridge, 1989
10. A. Moffat and T. Takaoka, An all pairs shortest path algorithm with expected time $O(n^2 \log n)$, *Proc. of the 26th Annual Symposium on Foundations of Computer Science*, Portland, OR, 1985, 101–105

11. A. Moffat and T. Takaoka, An all pairs shortest path algorithm with expected time $O(n^2 \log n)$, *SIAM J. Comput.* **16** (1987) 1023–1031
12. P.M. Spira, A new algorithm for finding all shortest path in a graph of positive arcs in average time $O(n^2 \log^2 n)$, *SIAM J. Comput.* **2** (1973) 28–32
13. A. Srinivasan, Techniques for probabilistic analysis and randomness-efficient computation, Ph.D. Thesis, Cornell University, Ithaca, NY, Technical Report 93-1378, August 1993
14. T. Takaoka, A new upper bound on the complexity of the all pairs shortest path problem, *Inform. Process. Lett.* **43** (1992) 195–199

Appendix

Recall the probabilistic experiment from Sect. 2.1: An urn contains n balls that are either red or blue; let m be the number of red balls. The balls are repeatedly drawn from the urn (without replacement) uniformly and independently at random. For $1 \leq k \leq m$, let the random variable W_k denote the waiting time for the k-th red ball. In addition, we define the random variables Y_i, $1 \leq i \leq m$, by $Y_1 := W_1$ and $Y_i := W_i - W_{i-1}$ for $2 \leq i \leq m$. The W_k's are distributed according to the negative hypergeometric distribution, i.e., for k, r with $1 \leq k \leq m$ and $k \leq r \leq n - m + k$,

$$\Pr(W_k = r) = \binom{r-1}{k-1}\binom{n-r}{m-k} \Big/ \binom{n}{m}.$$

The waiting time for the k-th red ball equals r if and only if there is a k-tuple $(j_1, \ldots, j_k)$ of positive integers with $j_1 + \cdots + j_k = r$ and $Y_i = j_i$ for all i, $1 \leq i \leq k$. Hence, for $j_1, \ldots, j_k \geq 1$ with $j_1 + \cdots + j_k = r$,

$$\Pr\left(\bigwedge\nolimits_{1 \leq i \leq k} Y_i = j_i\right) = \binom{r-1}{k-1}^{-1} \Pr(W_k = r) = \binom{n - (j_1 + \cdots + j_k)}{m-k} \Big/ \binom{n}{m}.$$

By using the well-known convolution identity

$$\sum_{0 \leq k \leq l} \binom{l-k}{m}\binom{q+k}{n} = \binom{l+q+1}{m+n+1} \tag{A.1}$$

for integers $l, m, n, q \geq 0$, $n \geq q$ (see [7] for a proof), it is easy to see that

$$\begin{aligned}
\Pr\left(\bigwedge\nolimits_{2 \leq i \leq k} Y_i = j_i\right) &= \sum_{1 \leq j_1 \leq n-m+1} \Pr\left(\bigwedge\nolimits_{1 \leq i \leq k} Y_i = j_i\right) \\
&= \sum_{1 \leq j_1 \leq n-m+1} \binom{n - (j_2 + \cdots + j_k) - j_1}{m-k} \Big/ \binom{n}{m} \\
&= \binom{n - (j_2 + \cdots + j_k)}{m-k+1} \Big/ \binom{n}{m}
\end{aligned}$$

and, more generally, for any non-empty $I \subseteq \{1, \ldots, m\}$ and positive integers j_i, $i \in I$,

$$\Pr\left(\bigwedge_{i\in I} Y_i = j_i\right) = \binom{n - \sum_{i\in I} j_i}{m - |I|} \Big/ \binom{n}{m}, \tag{A.2}$$

i.e., the Y_i's are exchangeable random variables. Making use of (A.1), we conclude that for any i, $1 \le i \le m$,

$$E[Y_i] = \sum_{j=1}^{n-m+1} j \binom{n-j}{m-1} \Big/ \binom{n}{m} = \binom{n+1}{m+1} \Big/ \binom{n}{m} = \frac{n+1}{m+1}.$$

This proves (2) in Sect. 2.1.

For $1 \le i \le m$, we introduced normalized random variables $Z_i := (Y_i - 1)/(n-m)$. The Z_i's take values in $[0,1]$; for any j, $1 \le j \le n-m+1$, $\Pr\left(Z_i = \frac{j-1}{n-m}\right) = \Pr(Y_i = j)$. By linearity of expectation, $E[Z_i] = 1/(m+1)$ for any i, $1 \le i \le m$. Lemma 1 proves that the Z_i's are negatively correlated.

Lemma 1. *For any $I \subseteq \{1, \ldots, m\}$ with $|I| = k \ge 1$,*

$$E\left[\prod_{i\in I} Z_i\right] = \frac{[n-m]_k}{(n-m)^k} \cdot \frac{1}{[m+k]_k} \le \frac{1}{(m+1)^k} = \prod_{i\in I} E[Z_i],$$

where $[x]_k := x \cdot (x-1) \cdots (x-k+1)$.

Proof. Only the first equation has to be proved and because of (A.2), we can restrict ourselves to the case $I = \{1, \ldots, k\}$. Using (A.1), one can prove by induction on k that

$$\sum_{\substack{j_1,\ldots,j_k \ge 1 \\ j_1+\cdots+j_k = r}} (j_1 - 1) \cdots (j_k - 1) = \binom{r-1}{2k-1}.$$

Therefore, by (A.2) and (A.1),

$$\begin{aligned}
&(n-m)^k E\left[\prod_{1\le i\le k} Z_i\right] \\
&= \sum_{k\le r\le n-m+k} \sum_{\substack{j_1,\ldots,j_k \ge 1 \\ j_1+\cdots+j_k = r}} (j_1 - 1) \cdots (j_k - 1) \cdot \Pr\left(\bigwedge_{1\le i\le k} Y_i = j_i\right) \\
&= \binom{n}{m}^{-1} \sum_{k\le r\le n-m+k} \binom{n-r}{m-k} \binom{r-1}{2k-1} = \binom{n}{m}^{-1} \binom{n}{m+k} = \frac{[n-m]_k}{[m+k]_k}.
\end{aligned}$$

□