# Finding the Hidden Path: Time Bounds for All-Pairs Shortest Paths

David R. Karger[*] Daphne Koller Steven J. Phillips[†]

Department of Computer Science
Stanford University
Stanford, California 94305

**Abstract**

*We investigate the all-pairs shortest paths problem in weighted graphs. We present an algorithm—the Hidden Paths Algorithm—that finds these paths in time $O(m^*n + n^2 \log n)$, where $m^*$ is the number of edges participating in shortest paths. Our algorithm is a practical substitute for Dijkstra's algorithm. We argue that $m^*$ is likely to be small in practice, since $m^* = O(n \log n)$ with high probability for many probability distributions on edge weights. Finally, we prove an $\Omega(mn)$ lower bound on the running time of any* path-comparison-based *algorithm for the all-pairs shortest paths problem. Path-comparison-based algorithms form a natural class containing the Hidden Paths Algorithm, as well as the algorithms of Dijkstra and Floyd.*

## 1 Introduction

Oh what a peaceful life is theirs
Who worldly strife and noise have flown
And follow the hidden path from cares
Which none but the wise of the world have known
Fray Luis de Leon, c. 1527 - 1591

Let $G$ be a directed graph with $n$ vertices and $m$ edges, with non-negative edge weights. The all-pairs shortest paths problem is to find a shortest path between each pair of vertices in $G$.

Our contributions to this problem lie in two areas. In Section 2 we present a new algorithm for all-pairs shortest paths, called the Hidden Paths Algorithm. Let an edge be called *optimal* if it is a shortest path, and let $m^*$ be the number of optimal edges in the graph. The Hidden Paths Algorithm runs in time $O(m^*n+n^2 \log n)$ if we use a Fibonacci heap [8] to implement a priority queue; the running time increases to $O(m^*n \log n)$ if a standard heap is used instead. The algorithm operates by running Dijkstra's single-source shortest paths algorithm [3] in parallel from all nodes in the graph, using information gained at each node to reduce the work done at other nodes. Our algorithm is likely to be fast in practice, because it is known [9, 15] that $m^* = O(n \log n)$ with high probability when the input graph is the complete graph with edge weights chosen independently from any of a large class of probability distributions, including the uniform distribution on the interval $[0, 1]$.

Our second contribution is a new lower bound, given in Section 3. Most algorithms for the all-pairs shortest path problem use the edge weight function only in comparing the weights of two paths in the graph. We call such algorithms *path-comparison-based*, and prove that any path-comparison-based algorithm requires $\Omega(mn)$ time on graphs with $m$ edges and $n$ vertices. The idea of the lower bound is to construct a weighted directed graph with $\Theta(mn)$ directed paths. If an algorithm $\mathcal{A}$ fails to examine any path, we show how to modify the weight function so that the path is optimal, without $\mathcal{A}$ detecting the change.

In proving the lower bound, we show that any path-comparison-based algorithm requires $\Omega(mn)$ time even to verify the output of an all-pairs shortest paths algorithm. Thus in the path-comparison model, verification is as hard as finding the paths. We show a lower bound of $\Omega(n^3)$ for the problem of checking that the edge weights of a graph satisfy the triangle inequality. These lower bounds also hold for randomized path-comparison-based algorithms.

One can generalize the notion of path weight so that the weight of a path may no longer be the sum of the weights of its edges. For example, one might define the weight of a path to be the maximum of the weights of

[*]Supported by a National Science Foundation Graduate Fellowship.

[†]Supported by NSF PYI grant CCR-8858030-03.

its edges. In Section 4 we show that many all-pairs shortest paths algorithms work even for generalized path weight functions. We show a lower bound of $\Omega(mn)$ on the running time of any algorithm solving the all-pairs shortest path problem for all generalized path weight functions.

### Previous Work

The most widely known algorithms for the all-pairs shortest paths problem are those of Dijkstra [3] and Floyd [5]. Dijkstra's algorithm for the single-source shortest path problem can be run from each vertex (as noted by Johnson [11]), resulting in a running time of $\Theta(mn + n^2 \log n)$ if Fibonacci heaps are used to implement priority queues. Floyd's algorithm, which can handle negative edge weights, works by dynamic programming and runs in time $\Theta(n^3)$. If the edge weights are independently and identically distributed random variables, a variant of Dijkstra's algorithm developed by Spira [18] has an expected running time of $O(n^2 \log^2 n)$. Bloniarz [1] provided an algorithm with an expected running time of $O(n^2 \log n \log^* n)$. Another algorithm, developed by Frieze and Grimmet [9], achieves an expected running time of $O(n^2 \log n)$, but is suitable only for random graphs. All of these algorithms are path-comparison-based, so by the lower bound of Section 3 they have a worst case running time of $\Omega(mn)$. Fast algorithms exist for special cases of the all-pairs shortest paths problem, for instance when the graph is unweighted [4] or planar [6].

Fredman [7] shows that $O(n^{5/2})$ comparisons between sums of edge weights suffice to solve the all-pairs shortest paths problem. He uses this fact to do preprocessing, producing an algorithm that runs in time $O(n^3(\log\log n \,/\, \log n)^{1/3})$. This result should be contrasted with the lower bound in Section 3. Fredman's algorithm is not path-comparison-based, since it performs comparisons involving sums of weights of edges that do not form a path (this capability distinguishes the algebraic decision tree model from the path-comparison-based model). It is surprising that we must allow such comparisons in order to improve on the $\Omega(n^3)$ bound.

An algorithm similar to the Hidden Paths Algorithm, with the same time bound, has been developed independently by McGeoch [16]. A variant of our algorithm has been developed independently by Jakobsson [10] as a transitive closure algorithm. Both these algorithms require data structures which are more complex than those used by the Hidden Paths Algorithm.

Lower bounds on the computational complexity of the all-pairs shortest paths problem have been proved in some other models. If the permissible operations are addition and minimum in a straight line computation, Kerr [12] shows that any algorithm requires $\Omega(n^3)$ running time. Regarding algebraic decision tree complexity, Spira and Pan [19] show that $\Omega(n^2)$ comparisons between sums of edge weights are necessary to solve the single-source shortest paths problem.

## 2 Finding Shortest Paths

In this section, we describe an algorithm for solving the all-pairs shortest paths problem in a directed graph $G = (V, E)$ with nonnegative edge weights. In order to present our algorithm, we need to make the following definitions.

### 2.1 Preliminary Definitions

We use $(u_1, u_2, \ldots, u_k)$ to denote a path from $u_1$ to $u_k$ going through the vertices $u_2, \ldots, u_{k-1}$. An unspecified path from $u$ to $v$ (which can also be an edge) will be denoted by $(u \rightsquigarrow v)$. The length of a path $(u_1, \ldots, u_k)$ is

$$|(u_1, \ldots, u_k)| = k - 1 \ .$$

Let $\|(u,v)\|$ denote the weight of the edge $(u,v)$. We extend the weight function by setting $\|(u,v)\| = \infty$, for any $(u,v) \notin E$, so that $\|(u,v)\|$ is defined for all pairs $u, v$. The weight of a path $(u_1, \ldots, u_k)$ is

$$\|(u_1, \ldots, u_k)\| = \sum_{i=1}^{k-1} \|(u_i, u_{i+1})\|$$

**Definition 2.1** *An edge $e$ is said to be* optimal *if it participates in some shortest path. Let $m^*(G)$ denote the number of optimal edges in $G$.*

**Fact 2.1** *The edge $(u,v)$ is optimal iff $(u,v)$ is a shortest path from $u$ to $v$.*

Note that in an unweighted graph each edge is optimal, so $m^* = m$ and our algorithm provides no improvement.

### 2.2 The Hidden Paths Algorithm

The Hidden Paths Algorithm presented in this section finds the shortest path between every pair of vertices in the graph. Essentially it runs Dijkstra's single-source shortest paths algorithm in parallel for all points in the graph. The different single-source

- For each vertex $v$, a list $E_v^*$ of the optimal edges directed into $v$ that have been found so far.
- A path array $P$, which for every pair $u, v$ contains the current best path from $u$ to $v$.
  Each entry $P[u,v]$ consists of
  *first*: The first vertex on this path (other than $u$ itself).
  *weight*: The weight of the path.
  *opt*: A flag indicating whether this path is known to be optimal.
  *heaptr*: A pointer to the heap entry for this path, if one exists.
- A heap $H$ of pointers to candidate paths which appear in the path array, ordered by weight.

Figure 1: Data structures maintained by the Hidden Paths Algorithm.

```
Initialize
    For all $u, v$
        Set $P[u,v].weight := \|(u,v)\|$.
        Insert the pair $u, v$ into the heap with priority $P[u,v].weight$.
        Initialize the rest of the record $P[u,v]$.
    For all $v$, set $E_v^*$ to be empty.

While (heap not empty and weight of top item $\neq \infty$) do
    Step 1  Remove the top element $(u \leadsto v)$ from the heap.
            Mark $P[u,v].opt := True$.
    Step 2  If $(u \leadsto v)$ is an edge then
                Add $(u,v)$ to $E_v^*$.
                For all $t$ such that $P[v,t].opt = True$
                    Update$(u,v,t)$.
            If $(u \leadsto v)$ is a path that is not an edge then
                For all edges $(w,u) \in E_u^*$,
                    Update$(w,u,v)$.

Procedure Update$(x,y,z)$:
    If $P[x,y].weight + P[y,z].weight < P[x,z].weight$ then
        Set $P[x,z].weight := P[x,y].weight + P[y,z].weight$.
        Set $P[x,z].first := y$.
        Change the priority of $x, z$ in the heap to $P[x,z].weight$.
        Modify $P[x,z].heaptr$ accordingly.
```

Figure 2: The Hidden Paths Algorithm.

threads are integrated in a way which allows the use of intermediate results from one thread to reduce the work done by another. In a sense, the algorithm discovers the hidden "shortest path structure" of the graph by pruning away the unnecessary parts. We note that the Hidden Paths Algorithm actually constructs each path in reverse order, by adding edges to the tail of the path. This facilitates forward traversal on the constructed paths. It is simple to modify the algorithm to construct the paths in the standard fashion. An intuitive description of the algorithm is given below. A precise presentation of the Hidden Paths Algorithm can be found in Figures 1 and 2.

The Hidden Paths Algorithm maintains a heap containing for each ordered pair of vertices $u, v$ the best path from $u$ to $v$ found so far. The heap is ordered according to the following order on paths: $p \prec q$ iff $(\|p\|, |p|) < (\|q\|, |q|)$ according to lexicographic order. We say that $p \cong q$ if $(\|p\|, |p|) = (\|q\|, |q|)$. The heap will be initialized to contain for each pair $u, v$ a path of weight $\|(u,v)\|$ (recall that if $(u,v) \notin E$ then $\|(u,v)\| = \infty$). It will always be the case that the path at the top of the heap will be an optimal path.

At each phase, the Hidden Paths Algorithm removes a path, say $(u \leadsto v)$, from the top of the heap (a delete-min operation). This is the optimal path from $u$ to $v$. This path is now used to construct a set of new *candidate paths*. If a new candidate path $(w \leadsto t)$ is shorter, it replaces the current best path from $w$ to $t$ in the heap. This maintains the optimality of the path at the top of the heap.

The complexity of the algorithm is a function of the number of candidate paths created, so we do not want to create too many. If $(u \leadsto v)$ is an edge, then the candidate paths are all those paths of the form $(u, v \leadsto t)$ where $(v \leadsto t)$ has already been discovered to be optimal. If $(u \leadsto v)$ is a path which is not an edge, then the candidate paths are all those of the form $(w, u \leadsto v)$ where $(w, u)$ is an optimal edge which has already been found. Note that in the construction of candidate paths, the edge is always concatenated to the tail.

The following theorem shows that this limited set of candidate paths suffices for finding the shortest path between each pair of vertices in the graph. The analysis in Section 2.3 shows that the resulting complexity is $O(m^*n + n^2 \log n)$.

**Theorem 2.2** *The Hidden Paths Algorithm finds all the shortest paths in the graph. Furthermore, it discovers them in order of increasing weight.*

**Proof:** Let *Opt* be the set of optimal paths found so far. We prove the theorem by induction on the size of *Opt*. The inductive hypothesis is that at the beginning of each iteration,

1. If $p$ is the item at the top of the heap, then $p$ is an optimal path and *Opt* contains all those optimal paths $q$ such that $q \prec p$ (and possibly some paths $q$ such that $q \cong p$). In particular, all subpaths of paths in *Opt* are also in *Opt*.

2. For each pair of vertices $u$ and $v$ for which an optimal path has not yet been found, the heap contains a path of minimal weight among those containing only a single edge or of the form $(u, w \leadsto v)$ for $(u, w)$ and $(w \leadsto v)$ in *Opt*.

The inductive hypothesis holds trivially at the beginning of the first iteration, since at that time, $p$ is the shortest edge in the graph, *Opt* is empty, and the heap contains all the edges.

Assume that the inductive hypothesis holds at the beginning of an iteration, and let $p = (u \leadsto v)$ be the item at the top of the heap. Then we know that $p$ is an optimal path. The construction of candidate paths in Step 2 of the algorithm (see Figure 2) ensures condition 2 above.

It remains only to prove that if $q$ is the next item on top of the heap, then $q$ is optimal and every smaller optimal path has been found. This can be false only if there exists another optimal path $r$, such that $r \prec q$, and $r$ has not yet been placed into the heap. Let $r$ be the smallest such path. Since all edges were placed in the heap during the initialization step, $r$ must be a path of length at least two. Assume $r = (x, y \leadsto w)$. But $(x, y) \prec r$ and $(y \leadsto w) \prec r$ and both are necessarily optimal. Since $r$ is the minimal optimal path not yet in *Opt*, both $(x, y)$ and $(y \leadsto w)$ must be in *Opt*, and therefore $r$ must have been constructed and inserted into the heap. This is a contradiction. Therefore, there can be no optimal paths not in *Opt* which precede $q$ under $\prec$, and $q$ must be optimal. $\square$

Note that this algorithm works with minimal changes for undirected graphs.

## 2.3 Running Time Analysis

Let us count the number of heap operations taken by the algorithm. It should be clear that the cost of all other operations is subsumed by the cost of the heap operations. During the initialization step, a heap of size $O(n^2)$ is created. Steps 1 and 2 are iterated at most $n(n-1)$ times, since in each iteration an optimal path is deleted. Therefore, the algorithm executes at most $n(n-1)$ delete-min operations.

It remains to count only the time taken by Step 2. During the execution of the algorithm, the only candidate paths which are created are those of the form $(u, v \leadsto w)$ where both $(u, v)$ and $(v \leadsto w)$ are optimal. For each optimal edge $(u, v)$, there are at most $n-1$ such paths (one for each $w$). The total number of candidate paths created is therefore $O(m^*n)$. There is at most one priority change operation associated with each candidate path.

The complexity of the algorithm depends on the implementation of the heap (see Table 1). Using a standard heap implementation, we get a total complexity of $\Theta(m^*n \log n)$. Using Fibonacci heaps (see [8]), insertions and priority change operations take constant amortized time, and we therefore get a complexity of $\Theta(n^2 \log n + m^*n)$.

The Hidden Paths Algorithm is also very simple and easy to implement, and thus provides a practical substitute for Dijkstra's algorithm.

## 2.4 $m^*$ in a Random Graph

To predict the behavior of the Hidden Paths Algorithm in practice, we need to study the quantity $m^*$ for "typical" graphs. It is easy to construct graphs for

| Operation | # of ops | cost for reg. heap | cost for Fib. heap |
|---|---|---|---|
| create | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ |
| priority change | $O(m^*n)$ | $O(m^*n\log n)$ | $O(m^*n)$ |
| delete-min | $O(n^2)$ | $O(n^2\log n)$ | $O(n^2\log n)$ |
| Total | — | $O(m^*n\log n)$ | $O(m^*n+n^2\log n)$ |

Table 1: The running time of the Hidden Paths Algorithm

which $m^*=O(n)$, while $m$ is $\Theta(n^2)$. It is also easy to construct graphs for which $m^*=m$. In this section we note that for a large class of probability distributions on random graphs, $m^*=O(n\log n)$.

Consider a distribution $F$ on non-negative edge weights, which does not depend on $n$, such that $F'(0)>0$. In particular, the uniform distribution on $[0,1]$ and the exponential distribution with mean $\lambda$ both satisfy these conditions. Frieze and Grimmet [9] prove the following result, along with some generalizations:

**Theorem 2.3 (Frieze and Grimmet)** *Let $G$ be a complete directed graph, whose edge weights are chosen independently according to $F$. Then with probability $1-O(n^{-1})$, the diameter of $G$ is $O(\log n/n)$, and hence $m^*(G)=O(n\log n)$.*

Similar results are derived in a different context by Luby and Ragde [15]. Hassin and Zemel prove the above theorem for both directed and undirected graphs, when the edge weights are uniformly distributed. The constant factors given by these analyses are small. In fact, empirical studies by McGeoch [16] indicate that when the edge weights are uniformly distributed, $m^*(G)$ grows approximately as $0.5n\ln n+.3n$.

**Corollary 2.4** *If the edge weights of $G$ are chosen independently according to $F$, then with high probability the running time of the Hidden Paths Algorithm is $O(n^2\log n)$.*

This time bound is an improvement over earlier algorithms by Spira [18] and Bloniarz [1], and matches the performance of the algorithm of Frieze and Grimmet. However, the Hidden Paths Algorithm can be effectively used in any situation where $m^*$ is significantly less than $m$, whereas the algorithm of Frieze and Grimmet is designed specifically for random graphs.

### 2.5 Generalizations

The Hidden Paths Algorithm can easily be transformed to one that finds the $k$ shortest paths in the graph. The revised algorithm will initialize the heap to contain only the actual edges in the graph. Then, when comparing candidate paths to existing paths (in Procedure Update), the algorithm will sometimes have to do an insert rather than a priority change operation. The running time for this algorithm (using Fibonacci heaps) is $O(m+k\log n+k^2)$.

Finally, we note that the Hidden Paths Algorithm considers some unnecessary candidate paths. One possible improvement, developed independently by us and by Jakobsson [10], creates only candidate paths every subpath of which is optimal. More specifically, a path $p=(u,v\rightsquigarrow w,t)$ is made a candidate path iff $(u,v\rightsquigarrow w)$ and $(v\rightsquigarrow w,t)$ are already known to be optimal. This will clearly reduce the number of candidate paths formed, but there does not seem to be a simple expression for the reduced running time achieved by this algorithm.

## 3 A Lower Bound

Spira and Pan have shown a lower bound of $\Omega(n^2)$ in the algebraic decision tree model [19]. However, many algorithms for the shortest paths problem use edge weights only to compute and compare the weights of paths. It therefore seems reasonable to consider a more restricted decision tree model for the shortest paths problem.

**Definition 3.1** *A* path-comparison-based *all-pairs shortest paths algorithm $\mathcal{A}$ accepts as input a graph $G$ and a weight function. The algorithm $\mathcal{A}$ can perform all standard operations. However, the only way it can access the edge weights is to compare the weights of two different paths.*

We can think of the algorithm as accessing the path weights only through a black-box path length comparator. The algorithm must output a reasonable encoding of the shortest paths in $G$.[1]

[1]We require that the output have no information as to path weights. For example, the graph itself is not a reasonable encoding of the solution.

It should be noted that the algorithms of Floyd, Dijkstra, and Spira, as well as the Hidden Paths Algorithm, all fit into this path-comparison-based model. On the other hand, Fredman's $o(n^3)$ algorithm [7] is not path-comparison-based because it adds weights of edges which do not form a single path. Fredman's algorithm conforms to the more general algebraic decision tree model.

We show a lower bound of $\Omega(n^3)$ on the running time of any path-comparison-based shortest paths algorithm running on a graph of $\Theta(n^2)$ edges. We then show how the construction can be modified to yield a lower bound of $\Omega(mn)$ for graphs with $m$ edges.

To show the $\Omega(n^3)$ lower bound, we construct a directed graph of $3n$ vertices on which any path-comparison-based shortest paths algorithm must perform $\Omega(n^3)$ comparisons. The directed graph $G$, with weight function $||\cdot||$, has $\Omega(n^3)$ paths. We show that if $\mathcal{A}$ fails to examine one of these paths then we can modify the weight function to make that path optimal without $\mathcal{A}$ being able to detect it.

The graph $G$ is a directed tripartite graph on vertices $u_i, v_j$, and $w_k$ where $i, j$ and $k$ range from 0 to $n-1$. The edge set for $G$ is $\{(u_i, v_j)\} \cup \{(v_j, w_k)\}$. Therefore, the only paths are individual edges and paths $(u_i, v_j, w_k)$ of length two.

To define the weight function, we work in base $n+1$ notation, generalized to allow negative digits. Define

$$[a_r, \ldots, a_0]_b = \sum_{i=0}^{r} a_i b^i$$

The edge weights are

$$\begin{aligned} ||(u_i, v_j)|| &= [1, 0, i, 0, j, 0, 0]_{n+1} \\ ||(v_j, w_k)|| &= [0, 1, 0, k, 0, -j, 0]_{n+1} \end{aligned}$$

and thus

$$||(u_i, v_j, w_k)|| = [1, 1, i, k, j, -j, 0]_{n+1}.$$

The following lemma is an immediate consequence:

**Lemma 3.1** *Let $<$ denote a lexicographic ordering on tuples of integers, with the leftmost integer being the most significant. For all $i, j, j', k, k'$ :*

1. $||(u_i, v_j)|| < ||(u_{i'}, v_{j'})||$ *iff* $(i, j) < (i', j')$
2. $||(v_j, w_k)|| < ||(v_{j'}, w_{k'})||$ *iff* $(k, -j) < (k', -j')$
3. $||(v_j, w_k)|| < ||(u_i, v_{j'})||$
4. $||(u_{i'}, v_{j'})|| < ||(u_i, v_j, w_k)||$
5. $||(u_i, v_j, w_k)|| < ||(u_{i'}, v_{j'}, w_{k'})||$ *iff* $(i, k, j) < (i', k', j')$.

Thus the unique optimal path from $u_i$ to $w_k$ goes through $v_0$, and has weight $[1, 1, i, k, 0, 0, 0]_{n+1}$. Define $L$ to be this set of optimal paths.

Consider giving $(G, ||\cdot||)$ as input to $\mathcal{A}$, and suppose that $\mathcal{A}$ runs correctly. It must therefore output the set of optimal paths $L$. Suppose further that a particular path $p^* = (u_{i^*}, v_{j^*}, w_{k^*})$ was never one of the operands in any comparison operation which $\mathcal{A}$ performed. We produce a weight function $||\cdot||'$ in which $p^*$ is the unique shortest path from $u_{i^*}$ to $w_{k^*}$, but the ordering by weight of all the other paths remains the same. If we run $\mathcal{A}$ on $(G, ||\cdot||')$, all path comparisons not involving $p^*$ give the same result as they did using $||\cdot||$. Therefore, since $\mathcal{A}$ never performed a comparison involving $p^*$ while running on $||\cdot||$, we deduce that $\mathcal{A}$ still outputs $L$, which is now incorrect.

The weight function $||\cdot||'$ is $||\cdot||$ with the following modifications. For $j \le j^*$, we decrease

$$||(u_{i^*}, v_j)||' = [1, 0, i, 0, 0, j, j]_{n+1}.$$

Then we decrease

$$||(v_{j^*}, w_{k^*})||' = [0, 1, 0, k, 0, -j^*, -n]_{n+1}.$$

Thus

$$||(u_{i^*}, v_{j^*}, w_{k^*})|| = [1, 1, i^*, k^*, 0, 0, j^* - n]_{n+1}.$$

**Lemma 3.2** *In $G$, the conditions of Lemma 3.1 continue to hold for $||\cdot||'$, except that the single path $(u_{i^*}, v_{j^*}, w_{k^*})$ directly precedes $(u_{i^*}, v_0, w_{k^*})$ in the ordering. Thus under $||\cdot||'$ the path $p^*$ is optimal.*

We have therefore proved the following:

**Theorem 3.3** *There exists a directed graph of $3n$ vertices on which any path-comparison-based shortest paths algorithm must perform at least $n^3/2$ path weight comparisons.*

Note that if the path which $\mathcal{A}$ did not check was $(u_{i^*}, v_0, w_{k^*})$, we can apply the above construction with $j^* = 1$ and the algorithm will fail, because the only comparison which has a different result is the one between $(u_{i^*}, v_0, w_{k^*})$ and $(u_{i^*}, v_1, w_{k^*})$, which by hypothesis was not performed since $(u_{i^*}, v_0, w_{k^*})$ was never examined. Note also that since all edge weights are polynomial in $n$, the input graph $G$ is not intractable merely because of an unusually large input size.

We can adapt this proof to show an $\Omega(mn)$ lower bound on graphs of $m$ edges. Assume without loss of generality that $m \geq 4n$ and that $2n$ divides $m$. We perform the same construction, but of the middle vertices we use only $v_1, \ldots, v_{m/2n}$, connecting each of them to all the vertices $u_i$ and $w_k$. This requires $m$ edges and creates $mn/2$ paths.

**Theorem 3.4** *There exists a directed graph with $2n+m/2n$ vertices and $m$ edges, on which any path-comparison-based shortest paths algorithm must perform at least $mn/2$ path weight comparisons.*

We can in fact show a stronger lower bound, namely that even the shortest paths *verification* problem requires $\Omega(mn)$ time for path-comparison-based algorithms. A verification algorithm $\mathcal{A}$ accepts as input a graph, a weight function (which we again think of as a black-box comparator), and an encoding $L$ which describes, for each pair of vertices, a path between them. Note that the standard description of shortest paths can be encoded in $O(n^2)$ space, so the input size imposes no non-trivial lower bound. The algorithm $\mathcal{A}$ accepts its input if and only if each path in $L$ is a shortest path.

To show this bound, we use the same construction as before. We set the encoding of shortest paths to be $L = \{(u_i, v_0, w_k) \mid i, k = 0 \ldots n-1\}$, and provide $(G, ||\cdot||, L)$ as input to $\mathcal{A}$.

**Corollary 3.5** *Any path-comparison-based algorithm for verification of shortest paths requires time $\Omega(mn)$ on $G$.*

If we add edges from each $u_i$ to each $w_k$ (thus producing $\Omega(n^2)$ edges), and set $||(u_i, w_k)|| = ||(u_i, v_0, w_k)||$, we can similarly deduce that

**Corollary 3.6** *Any path-comparison-based algorithm for verifying that all the edge weights satisfy the triangle inequality requires time $\Omega(n^3)$ on graphs of $n$ vertices.*

The construction of Theorem 3.3 can be applied to randomized algorithms for the shortest paths problem. For suppose that the expected number of comparisons performed by such an algorithm is $o(n^3)$. Then the probability that a randomly selected path is checked by the algorithm goes to 0 as $n$ goes to $\infty$. Thus if we took the graph $G$ and selected a single path $(u_{i^*}, v_{j^*}, w_{k^*})$ uniformly at random and applied the $||\cdot||'$ construction, the algorithm would detect our modification with probability approaching 0. We thus have

**Theorem 3.7** *If a randomized path-comparison-based algorithm performs $o(mn)$ expected comparisons on graphs with $m$ edges and $n$ vertices then there is a graph on which the algorithm will almost surely fail to be correct.*

More precisely, if the algorithm performs $r$ expected comparisons then its probability of success is $O(r/mn)$. Modifications of the above corollaries now follow in a similar fashion for randomized algorithms.

We conjecture that these lower bounds hold for undirected graphs as well, but this remains to be proved.

## 4 Generalizations

Many shortest paths algorithms in fact solve a much more general problem. In particular, we consider the following generalized shortest paths problem: Given a graph $G$, and a *generalized weight function* $||\cdot||$ which maps every path $p$ to a weight $||p||$ in the reals, find for each pair of vertices a path between them of minimum weight. To make this problem tractable, we impose restrictions on the weight function. Generalized weight functions have also been studied by Knuth in [13].

**Definition 4.1** *Consider a weight function $||\cdot||$:*

- *it is* consistent *if for all $u, v, w$,*

$$||(v \rightsquigarrow w)|| \leq ||(v \rightsquigarrow' w)||$$

*implies*

$$||(u \rightsquigarrow v \rightsquigarrow w)|| \leq ||(u \rightsquigarrow v \rightsquigarrow' w)||,$$

*and similarly for $||(u \rightsquigarrow v)|| \leq ||(u \rightsquigarrow' v)||$.*

- *it is* monotone *if for all $u, v, w$*

$$||(u \rightsquigarrow v \rightsquigarrow w)|| \geq \max(||(u \rightsquigarrow v)||, ||(v \rightsquigarrow w)||) \ .$$

- *it is* acyclic *if for all $u, v$ there exists a simple shortest path from $u$ to $v$.*

**Fact 4.1** *Any consistent monotone weight function is also acyclic.*

We argue that consistency is the major defining characteristic of the shortest paths problem, for it ensures that the shortest path between two vertices can be constructed from other shortest paths. The property of acyclicity is also basic, since it ensures that all shortest paths have bounded length. We shall therefore restrict our attention to consistent acyclic weight functions.

The standard shortest paths weight function is consistent. It is acyclic if there are no negative weight cycles. It is monotone if all path weights are non-negative. An example of a consistent, monotone, non-standard weight function is one which assigns to every path a weight equal to the weight of the maximal edge on the path. Solving single source shortest paths under this weight function is referred to as the *bottleneck path problem* in [20].

An algorithm for a generalized shortest paths problem receives as input the graph and a black box for the weight function. We assume that the black box takes constant time to compute the weight of any path. The following can be shown:

**Lemma 4.2** *Floyd's algorithm works on any consistent and acyclic weight function.*

**Lemma 4.3** *Dijkstra's algorithm and the Hidden Paths Algorithm work on any consistent monotone weight function.*

Our proof in Section 3 can be adapted to the situation of generalized weight functions, even for non-path-comparison-basedalgorithms. We show a lower bound for the class of consistent monotone weight functions, even on undirected graphs.

We consider a modified version of the above construction. Use the same graph $G$, with middle vertices $v_1 \dots v_{m/2n}$ but with undirected edges, and let $||\cdot||$ be defined as follows:

$$\begin{aligned} ||(u_i, v_j)|| &= 2 \\ ||(v_j, w_k)|| &= 2 \\ ||(u_i, v_j, w_k)|| &= 4. \end{aligned}$$

All other paths have length 5. Suppose as before that some path $(u_{i^*}, v_{j^*}, w_{k^*})$ does not have its weight queried. Change the weight of this path to be 3. It is simple to verify that the modified weight function remains consistent and monotone. This shows

**Theorem 4.4** *Any algorithm to solve the generalized shortest paths problem for consistent monotone weight functions requires $\Omega(mn)$ path weight queries.*

Corollaries parallel to those of Section 3 also hold. Thus any subcubic solution to the standard shortest paths problem must take advantage of the additivity of path weights.

## 5 Conclusion

We have produced a new algorithm, the Hidden Paths Algorithm, and identified a new measure $m^*$, the number of edges that participate in shortest paths. The Hidden Paths Algorithm runs in time $O(m^*n + n^2 \log n)$. The question arises: are there finer measures of the shortest-paths difficulty of a graph? In particular, the Hidden Paths Algorithm essentially runs in time proportional to the number of candidate paths formed. The improved algorithm mentioned in Section 2.5 forms a smaller number of such paths than the Hidden Paths Algorithm. Is there a simple measure for this quantity?

The expected value of $m^*$ has been shown to be significantly less than $m$ in the case of uniformly and independently distributed edge weights. This suggests that there are many situations in which the Hidden Paths Algorithm will be significantly faster than Dijkstra's algorithm. One can think of the optimal edges as forming a *certificate* of the shortest path structure of the graph, that must be revealed. The philosophy of the Hidden Paths Algorithm is thus similar to recent algorithms for connectivity, which work by first finding a sparse subgraph (or certificate) with the same connectivity [2, 17].

We have shown a lower bound of $\Omega(mn)$ on the running time of comparison based algorithms for all-pairs shortest paths. It is of particular interest that the construction and verification algorithms have the same worst case complexity. Compare this to the situation for the minimum spanning tree problem, where there is a linear-time algorithm to verify a minimum spanning tree [14], although no algorithm is known that finds one in linear-time. The comparison based lower bound shows that any improvement in the worst case complexity of shortest paths algorithms, such as Fredman's $o(n^3)$ algorithm, must take advantage of the numerical aspects of the problem, in addition to the ordering of path weights.

The obvious open problem arising from the lower bound is to extend the construction to the case of undirected graphs. Another goal would be to decrease the gap in the algebraic decision tree complexity, between Spira and Pan's $\Omega(n^2)$ lower bound, and Fredman's $O(n^{5/2})$ upper bound. Also, our lower bound would be strengthened if we could show that it held for all graphs of a certain structure and varying weights rather than for a single graph.

## Acknowledgements

We would like to thank Mike Luby and Cathy McGeoch for pointing out references for path lengths in graphs with random edge weights.

## References

[1] P. A. Bloniarz. "A shortest-path algorithm with expected time $O(n^2 \log n \log^* n)$". Technical Report 80-3, Department of Computer Science, State University of New York at Albany, August 1980.

[2] J. Cheriyan and R. Thurimella. "Algorithms for parallel $k$-vertex connectivity and sparse certificates". In *Proceedings of the 23rd ACM Symposium on Theory of Computing*, 1991.

[3] E. W. Dijkstra. "A note on two problems in connection with graphs". *Numerical Mathematics*, 1, 1959.

[4] T. Feder and R. Motwani. "Clique partitions, graph compression and speeding-up algorithms". In *Proceedings of the 23rd ACM Symposium on Theory of Computing*, 1991.

[5] R. W. Floyd. "Algorithm 97: Shortest path". *Communications of the ACM*, 5, 1962.

[6] G. N. Frederickson. "Planar graph decomposition and all pairs shortest paths". *Journal of the ACM*, 38(1), 1991.

[7] M. L. Fredman. "New bounds on the complexity of the shortest path problem". *SIAM Journal of Computing*, 5, 1976.

[8] M. L. Fredman and R. E. Tarjan. "Fibonacci heaps and their uses in improved network optimization algorithms". *Journal of the ACM*, 36, 1986.

[9] A. M. Frieze and G. R. Grimmet. "The shortest-path problem for graphs with random arc-lengths". *Discrete Applied Mathematics*, 10, 1985.

[10] H. Jakobsson. "Mixed-approach algorithms for transitive closure". In *Proceedings of the 10th ACM Symposium on Principles of Database Systems*, 1991.

[11] D. B. Johnson. "Efficient algorithms for shortest paths in sparse networks". *Journal of the ACM*, 24(1), 1977.

[12] L. R. Kerr. *The Effect of Algebraic Structure on the Computational Complexity of Matrix Multiplications.* PhD thesis, Cornell University, 1970.

[13] D. E. Knuth. "A generalization of Dijkstra's algorithm". *Information Processing Letters*, 6, 1977.

[14] J. Komlos. "Linear verification for spanning trees". *Combinatorica*, 5(1), 1985.

[15] M. Luby and P. Ragde. "A bidirectional shortest-path algorithm with good average case behavior". *Algorithmica*, 4, 1989.

[16] C. C. McGeoch. "A new all-pairs shortest-path algorithm". Technical Report 91-30, DIMACS, 1991.

[17] H. Nagamochi and T. Ibaraki. "Linear time algorithms for finding a sparse $k$-connected spanning subgraph of a $k$-connected graph". *Algorithmica, to appear*, 1991.

[18] 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 Journal of Computing*, 2, 1973.

[19] P. M. Spira and A. Pan. "On finding and updating shortest paths and spanning trees". In *Conference Record, IEEE 14th Annual Symposium on Switching and Automata Theory*, 1973.

[20] R. E. Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics.* Society for Industrial and Applied Mathematics, 1983.