Algorithmica (1995) 13: 426–441

All-Pairs Shortest Paths and the Essential Subgraph[1]

C. C. McGeoch[2]

Abstract. The *essential subgraph* H of a weighted graph or digraph G contains an edge (v, w) if that edge is uniquely the least-cost path between its vertices. Let s denote the number of edges of H. This paper presents an algorithm for solving all-pairs shortest paths on G that requires $O(ns + n^2 \log n)$ worst-case running time. In general the time is equivalent to that of solving n single-source problems using only edges in H. For general models of random graphs and digraphs G, $s = O(n \log n)$ almost surely. The subgraph H is optimal in the sense that it is the smallest subgraph sufficient for solving shortest-path problems in G. Lower bounds on the largest-cost edge of H and on the diameter of H and G are obtained for general randomly weighted graphs. Experimental results produce some new conjectures about essential subgraphs and distances in graphs with uniform edge costs.

Key Words. All-pairs shortest path, Single-source shortest path, Random graph, Random weighted graph, Graph diameter, Minimum spanning tree, Essential arcs.

1. Introduction.

Let $G = (V, E)$ be a complete weighted graph or digraph of n vertices. All edges have positive cost, and some costs may be infinite; let m denote the number of edges having finite cost. The cost of an edge from v to w is $c(v, w)$, and the distance between v and w in G is denoted $d(v, w)$. The *essential subgraph* $H = (V, E')$ contains an edge $(x, y) \in E$ whenever $d(x, y) = c(x, y)$ and there is no *alternate path* of equivalent cost. That is, edge (x, y) is in H when it is uniquely the shortest path in G between x and y. Note that H cannot be immediately found by comparing G and its distance matrix because of the difficulty of verifying the uniqueness property. Let s denote the number of edges in H.

This paper presents a new algorithm SHORT that solves the all-pair shortest-path (ASP) problem for G. The running time is equivalent to solving n single-source shortest-path (SSP) problems using only the edges in H. Any efficient implementation of Dijkstra's SSP algorithm may be used as a subroutine: with Fredman and Tarjan's [11] implementation, SHORT requires $O(ns + n^2 \log n)$ time, an improvement over their $O(nm + n^2 \log n)$ algorithm whenever $s = o(m)$. A recent algorithm by Karger *et al.* [17] also has $O(ns + n^2 \log n)$ running time.

For a general average-case model described in a later section, $s = O(n \log n)$ almost surely. The expected running time of $O(n^2 \log n)$ matches that of algorithms by Frieze and Grimmet [14], Hassin and Zemel [15], and Moffat and Takoaka

[1] Much of this research was carried out while the author was a Visiting Fellow at the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS).

[2] Department of Mathematics and Computer Science, Amherst College, Amherst, MA 01002, USA.

Received November 27, 1991; revised August 4, 1993. Communicated by D. S. Johnson.

[18]. However, these previous algorithms are only useful when their average-case models are known to hold for G.

SHORT represents a departure from standard approaches to the ASP problem. Rather than iterating over nodes to solve n SSP problems, the algorithm iterates over edges and solves the SSP problems incrementally: SHORT is efficient because each distance need only be computed once.

The next section surveys previous work on the all-pairs shortest-paths problem. Section 3 gives the algorithm and its worst-cast and average-case analyses. Some properties of the essential subgraph H are presented in Section 4. For example, the cost $c(r)$ of the largest-cost edge in H is a lower bound on the diameter of H (equivalently of G). A lower bound on $c(r)$ is obtained for a general family of edge cost distributions.

Section 5 describes a modification of SHORT that can randomly generate subgraphs H without first generating graphs G. Several experimental observations and conjectures are presented concerning graphs G having uniform edge costs. For example, it is only known analytically that s is between $n - 1$ and $27n \log_e n$ for this model: experiments suggest that $E[s] \approx 0.5n \log_e n$. Also, distances in uniform graphs appear to be distributed normally.

2. Previous Work. Floyd [9] describes an $O(n^3)$ ASP algorithm that can be implemented with small overhead and is most efficient for dense graphs or graphs with negative edge costs. Fredman [10] presents an $O(n^3(\log \log n)^{1/3}/(\log n)^{1/3})$ algorithm which is rather more complicated. Alon *et al.* [2] give a $o(n^3)$ algorithm for restricted types of graphs with running time depending on efficient matrix multiplication methods.

The standard approach to solving ASP when G is fairly sparse is to apply Dijkstra's [7] SSP algorithm n times, once for each node. Run-time improvements have been obtained through better data structures for Dijkstra's algorithm. Johnson [16] gives an algorithm requiring $O(m \log_{2+m/n} n)$ time per node. Fredman and Tarjan [11] use Fibonacci heaps to reduce Dijkstra's algorithm to $O(n \log n + m)$. Fredman and Willard [12] use Atomic Heaps to obtain an $O(m + n \log n/\log \log n)$ algorithm for a computational model in which operations such as address calculations are allowed. For graphs with integer edge costs and maximum cost C, Ahuja *et al.* [1] give an algorithm with $O(m + n\sqrt{\log C})$ running time for each node.

Frieze and Grimmet [14] present an ASP algorithm for digraphs which requires $O(n^2 \log n + nm)$ worst-case time. They also propose an algorithm that first extracts a subgraph $\tilde{G}$, containing at most $20n \log_2 n$ edges, and solves ASP on the subgraph. For a large class of distributions on edge costs, they show that with high probability the subgraph $\tilde{G}$ is sufficient for solving the ASP problem for G; if the distribution is uniform, then $\tilde{G}$ need have only $12n \log_2 n$ edges.

Hassin and Zemel [15] consider the case where G may be directed or undirected and the distribution on edge costs is uniform. They describe a subgraph $\hat{G}$ (different from $\tilde{G}$) that contains no more than $27n \log_e n$ edges and show that for this model $\hat{G}$ is almost surely sufficient to compute ASP for G.

For both of these algorithms, in the (low-probability) event that the subgraph is not sufficient for the computation, an unsuccessful result is detected and a conventional ASP algorithm is applied to the original graph G (an erroneous computation can be detected in $O(m)$ time). The average time works out to $O(n^2 \log n)$, an improvement over earlier algorithms by Spira [22] ($O(n^2 \log^2 n)$) and Bloniarz [5] ($O(n^2 \log n \log^* n)$) for graphs with random edge costs.

The performance of the $O(n^2 \log n)$ algorithms depends on the fact that the computation is carried out on some small subgraph of G. However, the analyses rely upon *a priori* assumptions about the structure of G: the subgraphs $\tilde{G}$ and $\hat{G}$ are claimed to be sufficient only when G obeys the probabilistic models. In contrast, SHORT builds the subgraph H on the fly rather than extracting it beforehand. When G is not known to conform to the random models, SHORT would be preferred because there is no possibility of an incorrect computation needing to be redone. Furthermore, H is the optimal subgraph needed for the computation, and therefore smaller than either $\tilde{G}$ or $\hat{G}$ when G does obey the random models. Experiments in Section 5 suggest that $s \approx 0.5\, n \log_e n$ for undirected graphs with uniform edge costs, whereas $27n \log_e n$ edges are required for Hassin and Zemel's algorithm.

Moffat and Takaoka [18] give a hybrid of Spira's and Bloniarz's algorithms that solves n single-source problems, each in $O(n \log n)$ expected time and $O(n^2)$ worst-case time, in addition to a one-time $O(n^2 \log n)$ presorting phase. The performance of their algorithm depends upon careful treatment of the candidate vertices in each step of a tree-growing process. Their analysis requires only that the distribution on edge costs be independent of the edge endpoints; but, again, SHORT (or a simpler algorithm with good worst-case performance) would be preferred in practice if the average-case model is not known to hold.

Karger *et al.* [17] have recently developed a HIDDEN PATHS algorithm that has time and space requirements identical to those for SHORT. The main data structure in their algorithm is a heap containing both edges and paths in G, extracted in order by cost, such that each item extracted is guaranteed to be optimal. As an edge or path is extracted, certain new paths are inserted into the heap and certain heap elements are updated. Their analysis depends on bounding the number of insertions and updates, which depends on s (called m^* in their paper).

Although both SHORT and HIDDEN PATHS iterate over edges instead of nodes, they appear to be distinct algorithms. It can be shown, for example, that the two algorithms would discover and report distances in different order. In practice the choice between SHORT and HIDDEN PATHS may depend upon the particular application: for instance, the n shortest-path trees are constructed as a by-product of SHORT but not of HIDDEN PATHS.

There appears to be very little previous work concerning the structure of the essential subgraph H. Robert [20] considered a generalization of the shortest-paths problem to computations on Q-semirings and used the term *essential arcs* in a different but equivalent definition of edges of H. He showed that the essential subgraph is unique for any graph G and that for every vertex pair a shortest path between them comprising only essential edges exists.

3. Algorithm SHORT.

It is helpful to think of SHORT as an algorithm for constructing H. The construction method sketched below works for either graphs or digraphs.

```
Algorithm SHORT
Initialize $H$ to empty
Initialize Heap to contain all edges of G
Loop:
    Extract min-cost edge (v, w, c) from Heap
    If $H$ contains no alternate path from $v$ to $w$ of length $c$ or less,
      then insert (v, w, c) in $H$
      else discard (v, w, c)
until Heap is empty.
```

We first demonstrate that SHORT builds H correctly. The algorithm costructs a sequence of subgraphs $H_1 \subseteq H_2 \subseteq \cdots H_i \cdots \subseteq H_m \subseteq H_*$, where H_i is the subgraph present at the beginning of the ith iteration and H_* denotes the final product (after the mth iteration).

THEOREM 3.1. *Edge (v, w), with cost $c = c(v, w)$, is in H_* if and only if the edge is in H.*

PROOF. Suppose edge (v, w, c) is considered during the ith iteration of SHORT. Let $d_i(v, w)$ be the shortest distance from v to w using only edges in H_i. If v and w are not connected in H_i, then $d_i(v, w) = \infty$.

We assume by induction that the subgraph H_i has been constructed correctly. For the initial step, note that the minimum-cost edge of G must be in H, and this edge will certainly be inserted in H_1 by the algorithm.

Suppose that $d_i(v, w) \le c(v, w)$. Since $(v, w) \notin H_i$ there must be an alternate path in H_i. Since $H_i \subseteq G$, there must also be an alternate path in G.

Now suppose that $d_i(v, w) > c(v, w)$. There cannot be an alternate path in G such that $d(v, w) \le c(v, w)$. Suppose there were such a path p. Each edge in p must cost strictly less than $c(v, w)$, and so each of these edges is either in H_i or can be replaced by an alternate path in H_i. This would imply that $d_i(v, w) \le d(v, w) \le c(v, w)$, contradicting our assumption. □

3.1. The Search Procedure. Considering edges in increasing order by cost ensures that each edge need only be examined once in the construction of H, since no later, costlier edge could be part of a shorter path. In this section we present a Search procedure that returns a decision *accept* or *reject* depending upon whether an alternate path exists in the partially built subgraph H_i. The procedure is implemented such that each distance need only be computed once.

It is helpful to recall Dijkstra's algorithm for the SSP problem (see [7], [21], or [23]). A shortest-path tree $T(v)$ rooted at v is a subtree of G such that, for each node w, the shortest path from v to w in G is also the shortest path in $T(v)$. Dijkstra's algorithm builds $T(v)$ by maintaining a heap of *fringe* vertices which are not in $T(v)$ but are adjacent to vertices in $T(v)$. Vertices are extracted from the fringe heap and added to the tree $T(v)$ in increasing order of distance from v. Once x is inserted in $T(v)$, each edge incident on x must be *processed*: new fringe vertices may be *inserted* in the heap, some current fringe vertices may require a *decrease-key* operation if a shorter distance from v is discovered, and some vertices are ignored because they are already tree vertices. We assume the existence of a procedure Dijkstra-Process(v, x, y) that operates on edge (x, y) when vertex x is added to $T(v)$.

The Search procedure constructs n single-source trees incrementally, maintaining the property that no *path* of length greater than c is built in any tree before all *edges* of cost c have been considered. (When a path and an edge have equal cost, we insert the path and discard the edge.) Therefore no tree need ever be modified, since large edges cannot affect short paths. When vertex w is added to the tree for $T(v)$, the correct distance $d(v, w)$ is also recorded in a distance matrix. We show that the procedure only needs access to the edges of H_i during the ith search.

When called with parameters (v, w, c), Search adds more vertices to the tree $T(v)$ rooted at v, and updates the fringe heap, until one of the following *stopping conditions* occurs:

1. An alternate path from v to w having length at most c is found in H_i. In this case the procedure returns *reject.*
2. There are no more edges in the fringe heap, indicating that v and w are not connected in H_i. The procedure returns *accept.*
3. All vertices with distance at most c from v have been examined. The distance to w in H_i must therefore be greater than c. The procedure returns *accept.*

The partially built tree and fringe heap are saved between calls to Search. The next time Search is called for source vertex v, the procedure continues constructing $T(v)$. Since some edges may have been added to H_i in the interim, it is necessary to update the heap in a *Restore* operation.

The procedure is sketched below. Entries in the Distance matrix are initialized to infinity and contain distances once they are known. Each tree T(v) is initialized to contain no edges. Each element of the Fringe(v) heap has three components: the name of the fringe vertex, the distance from v to that vertex, and the tree_neighbor through which T(v) and the fringe vertex are to be connected. Each Fringe(v) heap initially contains the source vertex v, with tree neighbor NULL and distance 0. The Find_Min operation returns a minimum element without modifying the heap, while Extract_Min removes the minimum element.

```
**Procedure Search(H, v, w, c)**
*Stopping condition 1:*
If Distance[v,w] $\le c$ then return(reject)
*Restore step:*
For each vertex $z$ in $T(v)$,
    For each edge (z,y) added to H since the last search from v,
      Dijkstra-Process(v, z, y)
  Loop:
    *Stopping condition 2:*
    If Fringe(v) is empty then return(accept)
    *Stopping condition 3:*
    node = Find_Min(Fringe(v))
    If node.distance > c then return(accept)
    *Construct T(v):*
    node = Extract_Min(Fringe(v))
    x = node·name
    Distance[v, x] = node.distance
    Insert (node·tree_neighbor, x) into T(v)
    For each edge (x, y) in H, Dijkstra-Process(v, x, y)
    *Stopping condition 1:*
    If x = w then return(reject)
  Endloop
```

Note that some trees may not be completed after the *m*th call to Search. A postprocessing step is necessary that calls Search once for each source node v and forces the procedure to reach Stopping Condition 2. With this convention, the correctness of the procedure in constructing H, the Distance matrix, and the shortest-path trees follows from the observations below:

1. It is straightforward to show that Dijkstra's algorithm correctly builds single-source shortest-path trees of G when restricted to edges in the (completed) subgraph H.
2. Suppose edge (v, w, c) is considered during the ith call to Search. At the beginning of this call the procedure has access to H_i, a subgraph of H. The Restore operation ensures that Dijkstra's algorithm considers *all* edges of H_i. With only H_i available, the procedure can correctly add a node x to $T(v)$ whenever $d_i(v, x) \leq c$, because later edges, of cost c or more, cannot invalidate this distance. The stopping conditions ensure that this bound is enforced. The postprocessing step ensures that all trees eventually contain n nodes.
3. If tree construction is correct, any distance recorded in the Distance matrix is also correct.
4. The decision *accept/reject* is also returned correctly. By Theorem 3.1, there is an alternate path in H_i if and only if there is one in H. Suppose there is an alternate path in H_i. If w is already in $T(v)$ at the beginning of the procedure, then Stopping Condition 1 at the top is invoked. Otherwise, the procedure will add nodes to $T(v)$ until w is discovered, and will reach Stopping Condition 1 inside the loop. If there is no alternate path in H_i, then the procedure will eventually reach either Stopping Condition 2 or 3.

3.2. Analysis of SHORT. We first consider the total cost of all calls to Search. For each v, the subroutine Dijkstra-Process processes each edge of H exactly once (either during the Restore operation or in the loop). There are at most n *inserts*, *deletes*, and s decrease-key operations on the fringe heap. These operations can be performed in $O(s + n \log n)$ worst-case time using Fredman and Tarjan's [11] Fibonacci heaps. There is only constant extra cost per edge for time-stamping and locating edges during the Restore operation, since these edges are easily found at the back of the adjacency lists of H_i.

Each call to Search also involves a lookup in the Distance matrix. The total cost of all calls to Search is therefore $O(n(n \log n + s) + m)$. The heap operations in the main loop of SHORT require $O(m \log m)$ time. The total cost is therefore $O(n^2 \log n + ns)$.

Note that H, the distance matrix, and the shortest-path trees may be completed well before the mth edge is examined; the algorithm can easily detect this fact and stop early. This does not change the asymptotic running time, but it could produce a considerable speedup in practice.

The SEARCH procedure requires $O(s)$ space to store H, $O(n^2)$ space to store the shortest-path trees, and $O(n^2)$ space for the distance matrix. Note that any distance already known must be no more than the current edge cost: therefore the comparison in the first line of Search could be replaced by a test of whether w is already in T(v). Furthermore, the Restore operation needs to know which vertices are currently in T(v), but not necessarily its structure. If distances can be reported on-the-fly as they are discovered, then considerable (although constant-factor) space savings could be realized by replacing Distance with a bit matrix and replacing each T(v) with a list of vertices.

We now consider the average-case running time. Let $G(i)$ be an unweighted graph of size i chosen at random (with arbitrary distribution function). Let $p(i)$ be the probability that a graph of size i will *fail* to have a given property. Then the property holds *almost surely* if the infinite sum of the probabilities $p(1), p(2), \ldots$ is finite. This is a stronger condition than holding *in probability*.

Frieze and Grimmet [14] consider directed graphs with edge costs drawn independently from a fixed distribution F such that $F'(0) > 0$. The subgraph $\tilde{G}$ contains an edge (v, w) of G if w is among the p-nearest neighbors of v, where $p = \min\{n - 1, 20 \log_2 n\}$. Their proof that $\tilde{G}$ is almost surely sufficient for computing ASP on G implies immediately that $H \subseteq \tilde{G}$, since every edge of H is necessary for the shortest-path computation. Therefore, for random weighted digraphs, $s \leq 20n \log_2 n$ almost surely, and the running time of SHORT is $O(n^2 \log n)$ almost surely.

Hassin and Zemel [15] consider directed and undirected graphs with the uniform distribution on edge weights. Their subgraph $\hat{G}$ is constructed by extracting all edges of weight less than $27 \log_e n/n$. They show that $\hat{G}$ is almost surely sufficient for computing shortest paths in G. For random uniform graphs and digraphs, SHORT has $O(n^2 \log n)$ time almost surely.

4. Properties of Essential Subgraphs.

Very little previous work has appeared concerning the structure of the essential subgraph H. Robert [20] shows that the essential subgraph is unique and that for every pair of vertices there is a shortest path between them comprising only edges in H. This immediately implies that H is sufficient for computing distances in G; and certainly every edge of H is necessary. In this sense H is the optimal subgraph for distance computations on G.

It is easy to show that H is exactly the union of n SSP trees and that H must contain a minimum spanning tree of G. Furthermore, since (by the construction method) the adjacency lists of H are ordered by increasing cost, nearest neighbors can be found in constant time per node. In some applications it may be cost-efficient to precompute H and to use it to answer queries regarding distances, paths, neighbors, and subtrees of G.

Also, H can be useful for maintaining distances under edge-cost perturbations in G. In general the problem of edge maintence appears to be very difficult: the best known algorithms require $O(n^2 \log n + nm)$ worst-case time for each perturbation, which is no better than starting from scratch. Some improvements have been achieved for special classes of graphs and updates (see [3] and [19]).

Consider the problem of maintaining H when an edge of G is modified. If only cost *decreases* occur, then the only possible effect on H is that some essential edges may become nonessential; it is only necessary to recompute distances in H to find and remove those edges. Therefore H can be maintained without access to the edge set of G. If the new subgraph H' has s' edges, then the recomputation can be done in $O(n^2 \log n + ns')$ time. If cost increases are allowed, then some previously nonessential edges may become essential, and the edges set of G must be available for proper maintenance of H.

Lower Bounds on Diameters. For a given graph G, let r denote the rank in G of the largest-cost edge in H. This is the last edge that SHORT inserts in H. Let $c(r)$ denote the cost of this edge.

A *randomly weighted graph* G_F has positive edge costs drawn independently and identically distributed according to some fixed probability distribution F, and the distribution is independent of the edge endpoints. These graphs are complete but some edges may have infinite cost. A *uniform graph* G_u is a randomly weighted graph where the edge-cost distribution is uniform on $(0, 1]$.

Frieze and Grimmet [14] give an upper bound on the diameter of randomly weighted digraphs in terms of the edge-cost distribution F; when F is uniform the diameter is at most $12 \log_e n/n$ with probability at least $1 - O(n^{-2})$. Hassin and Zemel [15] give upper and lower bounds of $27 \log_e n/n$ and $\log_e n/n$ for the diameter of uniform graphs. The following result generalizes their lower bound to randomly weighted graphs. For a given edge-cost distribution function $F: R^+ \to [0, 1]$ the lower bound is stated in terms of the inverse function $F^{-1}: [0, 1] \to R^+$.

THEOREM 4.1. *Let G_F be a randomly weighted graph with distribution F on edge costs. Let $p_0 = a \log_e n/n$ for some $a < 1$. Then $F^{-1}(p_0) \le c_r$ almost surely.*

PROOF. A *probabilistic graph* G_p is one in which each edge appears independently with probability p. Weide [24] demonstrates the following relationship between randomly weighted graphs and probabilistic graphs: Suppose there is an optimization problem on randomly weighted graphs G_F, for which the objective function (to be minimized) is the maximum edge cost c_k in a feasible solution. Let K_n denote the set of feasible solutions. Suppose it is known that, almost surely, a random probabilistic graph with edge probability at most p_0 does *not* contain a member of K_n as a subgraph. If F denotes the distribution on edge costs in G, then $F^{-1}(p_0) \leq c_k$ almost surely.

The following optimization problem meets our needs: find the subtree K which spans G and for which the cost c_k of the largest-cost edge in K is minimized over all possible spanning subtrees. Certainly any feasible solution K must be connected. Erdös and Rényi [8] (or see [6]) show that a probabilistic graph with edge probability at most $p_0 = a \log_e n/n$ is almost surely not connected. Therefore $F^{-1}(p_0) \leq c_k$ almost surely. Since H must also span G we have $c_k \leq c(r)$. □

This lower bound on $c(r)$ immediately gives a lower bound on the diameter of G, since for any graph G we have $c(r) \leq DIAM(H) = DIAM(G)$.

This also gives a lower bound on the diameter of a minimum spanning tree (MST) of G, since $DIAM(G) \leq DIAM(MST(G))$. It appears that the best known upper bound on the MST diameter is Frieze's bound of 1.2 for the *total weight* of the tree [13] (or see [6]). Finding tighter bounds on the MST diameter is an open problem. Some experimental results are presented in the next section.

Of course it is possible to use the distribution function F to translate the lower bound on $c(r)$ into a lower bound on r. We can also obtain a lower bound on r that is independent of F: a random graph with $(n/2) \log_e n + \omega(n)$ edges is almost surely disconnected when $\omega(n) \to -\infty$ [6]. SHORT must examine (but possibly discard) at least this many edges when constructing H, because H must span G and edge costs are independent of vertices. This lower bound holds for any ASP algorithm that considers edges in random order.

5. Experimental Results.

Let G be a random graph with edge costs drawn uniformly and independently from $(0, 1]$. Very little is known analytically about distances in random uniform graphs, or about the structure of H for this model. This section describes experiments concerning distances, the essential subgraph, and related properties for uniform graphs G.

The algorithm SHORT can be easily modified to generate subgraphs H having appropriate distributional properties for uniform graphs G, without first generating the graphs. The heap function in the main loop that *extracts* edges of G by increasing cost need only be replaced by a function that randomly *generates* edges with uniform costs assigned in ascending order. Bentley and Saxe [4] give an algorithm for generating ordered uniform variates from $(0, 1]$ that requires $O(1)$ storage and $O(1)$ time per variate.

To save space, the generation program implemented here stores H but not G, the distance matrix, or the shortest-path trees. Since the distance matrix is not available, every invocation of **Search(v, w, c)** causes a (possibly redundant) search from v to w in H. This extra search cost would be prohibitively expensive if all m

edges were generated. However, by the analytical bounds on $c(r)$ we know that H will probably be completed after $27n \log_e n$ edges are generated. A *stopping criterion* is needed to allow the algorithm to terminate when H is complete.

One strategy would be to guess that H is finished after $27n \log_e n$ edges have been considered. However, this would require extra computation to verify correctness and also to deal with the (low probability) event of a wrong guess.

Instead, the stopping criterion used in the experiments exploits the fact that the diameter of the MST is an upper bound on $c(r)$. The main loop of the generation algorithm, while constructing H, also uses Kruskal's method to build an MST of G. Once the MST is found its diameter is computed in $O(n)$ time. Edges with cost greater than the MST diameter are not generated. Although there is no useful analytical upper bound on the diameter of the MST of a uniform graph, the approach works well in practice, as is discussed below.

The experiments were performed on a Sun SPARCstation ELC. The Sun Unix generator `drand48` provides a stream of random uniform variates; the method of Bentley and Saxe [4] was used to generate random edge costs in ascending order.

The random number generators passed graphical and empirical tests that: edge endpoints were generated randomly and independently; edge costs were uniformly distributed on $(0, 1]$; and edge costs were independent of edge vertices.

Experiments were run for 50 trials each at the problem sizes $n = 200, 400, \ldots, 1400$; some detailed experiments producing very large data sets were performed at $n = 200$. About 3 days of CPU time were required to generate all the data.

Largest Edge in H. Let the random variate R be an estimator of r, the rank in G of the largest-cost edge in H, and let variate C be the corresponding estimator of its cost $c(r)$. From the analytical results cited previously, we expect that $(n/2) \log_e n - \varepsilon \leq R$ and that $(1 - \varepsilon) \log_e n/n \leq C \leq 27 \log_e n/n$ when n is large enough.

Experiments suggest that these asymptotic bounds also hold for n below 1400. The function form $cn \log_e n$ provides an excellent fit to R. The mean of the ratio $R/(n \log_e n)$ over all observations was 1.134; at $n = 1400$ (possibly a better indicator of the asymptotic ratio), the observed mean was 1.122. In all experiments the ratio was never observed to be outside the range [0.862, 1.563].

Both functional forms $cn \log_e n/m$ and $c \log_e n/n$ (which differ only in lower-order terms) provide very good fits to the observed values for C, and one is not noticeably better than the other. The mean, over all observations, of $C/(\log_e n/n)$ was 2.270, the mean at $n = 1400$ was 2.237, and the ratio was never observed to be outside the range [1.774, 3.147].

CONJECTURE. *The limit as $n \to \infty$ of $r/(n \log_e n)$ is a constant near* 1.1; *the asymptotic ratio $c(r)/(\log_e n/n)$ is near* 2.2.

Size of H. The number of edges in H is denoted s. Certainly $n - 1 \leq s$ for any G, and it is easy to verify analytically that $s \leq r \approx m \cdot c(r) \leq 13.5n \log_e n$ for

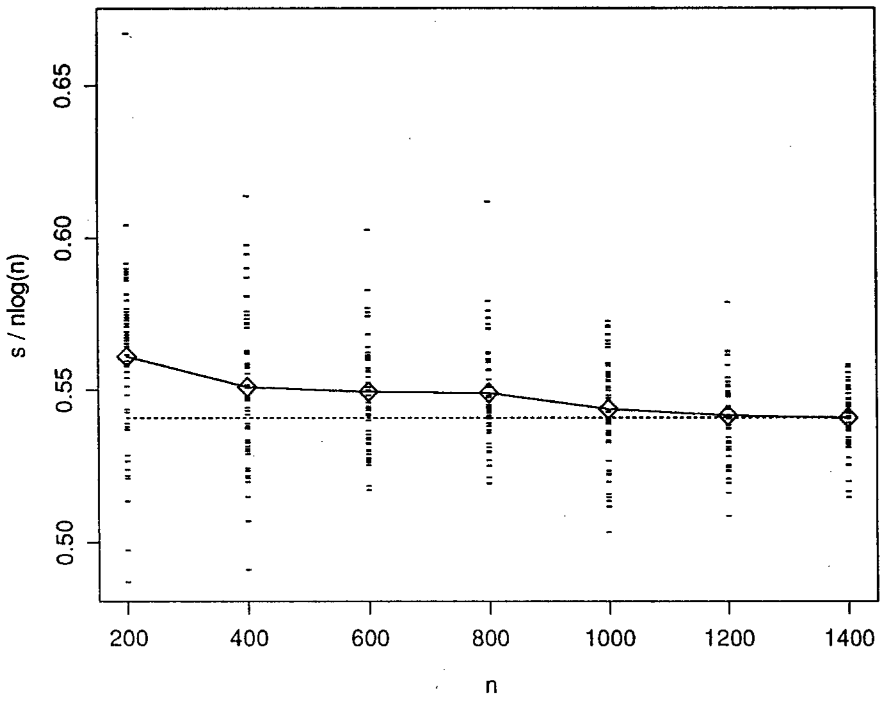

Fig. 1. Size of H.

uniform graphs. Therefore we have a $\Theta(\log n)$ gap between known upper and lower bounds on s.

Let S be the random variate estimating s in each trial. Figure 1 shows the ratio $S/n \log_e n$ for 50 trials at each n. The solid line connects means $\bar{S}$ for each n. At $n = 1400$ we have $\bar{S}/n \log_e n = 0.541$, which is marked on the graph by a dashed horizontal line.

This figure illustrates the difficulty of extending observations taken at finite n to conjectures about asymptotic behavior. It is impossible to tell whether $\bar{S}/n \log_e n$ is approaching a constant or whether $\bar{S}$ is asymptotically $o(n \log_e n)$. However, the data provides even less support for a conjecture of cn or even $cn \log \log n$ asymptotic growth.

Further evidence that $\bar{S}$ grows as $cn \log n$ appears when we compare S and R. Figure 2 shows the observed ratio R/S plotted against S in each trial. The horizontal dashed line markes the mean ratio 2.069 observed for all trials. This and related analyses give clear indications that the ratio R/S is constant in n, and it is known analytically that $R = \Omega(n \log n)$.

CONJECTURE. *For random uniform graphs* $s = \Omega(n \log n)$. *Asymptotically the ratio* $s/(n \log_e n)$ *is near* 0.5. *The ratio* r/s *is constant in* n.

It appears, then, that about half of the r edges that are considered during the

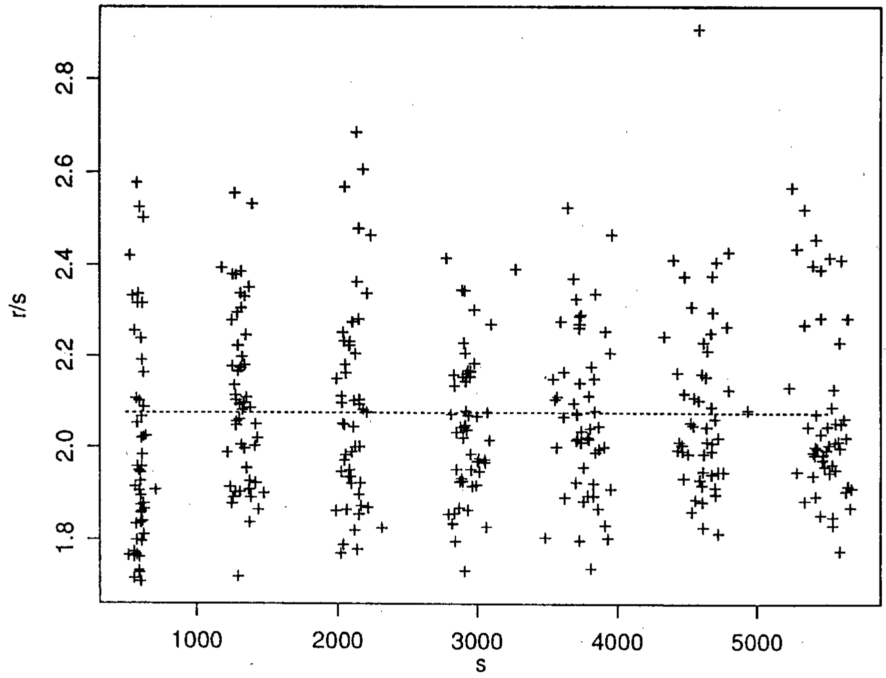

Fig. 2. R versus S.

construction of H actually become essential edges. Any ASP algorithm that examines edges greedily considers only twice as many edges as is absolutely necessary.

Diameter of the MST Let D denote the diameter of the MST of G observed in a single trial, and let $L = m \cdot D$. We work with L rather than D because some types of data analysis are easier (L increases in n while D approaches a constant limit) and because we are interested in comparing L with R.

The variate L appears to grow as $l(n) = cn \log_e^2 n$ for a constant c. A graph of $L/l(n)$ is similar in appearance to Figure 1: possibly the function $l(n)$ overestimates asymptotic growth in $\bar{L}$. Nevertheless, within the range of problem sizes studied, this is the best functional fit found to within a $\log \log n$ factor: the observations L are clearly growing faster than $l(n)/\log \log n$ and clearly growing more slowly than $l(n) \log \log n$. The observed constant c was 0.564 on average and was never observed to be outside the range [0.396, 0.857].

Conjecture. *The diameter of the MST of a uniform graph grows approximately as $d(n) = l(n)/m = 0.56n \log_e^2 n/m \approx 1.1 \log_e^2 n/(n-1)$.*

As noted earlier, the diameter of the MST provides the stopping criterion for the random generation algorithm. Since r is about $1.1n \log_e n$ it appears that the bound $l(n)$ is off by only a factor of $\log n$. These results also suggest that the diameter of the MST and the diameter of G are separated by a $\log n$ factor.

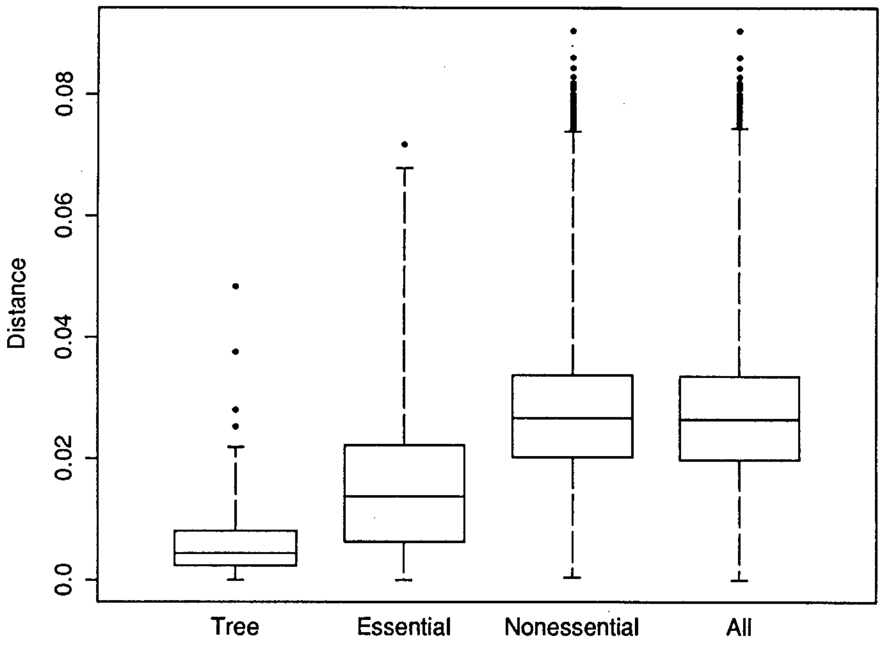

Fig. 3. Distribution of distances.

Distribution of Distances in G. Distances in G are of three types, each having distinct distributional properties. The s *essential distances* correspond to costs on essential edges. The $n-1$ *tree distances* correspond to edges in the MST of G, which is a subset of the essential subgraph. Each of the $m-s$ *nonessential distances* is equivalent to the sum of two or more essential distances.

Figure 3 shows the distribution of distances by type in three random trials at $n = 200$. In each data set the horizontal bar marks the median, the box ends mark the quartiles (containing 50% of the points) and the whiskers mark twice the interquartile range. In each trial there were 199 MST edges, 554.67 essential edges on average, 19,345.33 nonessential edges on average, and 19,900 total edges. Note that since there are many more nonessential edges than essential edges, the distributional properties of the former tend to dominate those of the entire set. Essential edges tend to be smaller than nonessential edges (as is expected), but there is considerable overlap in the distributions of the two types. Tree edges are even more highly concentrated near the bottom end of the range.

Figure 4 gives a more detailed view of the distribution of tree and essential edges, averaged over the three trials at $n = 200$. The 1500 smallest edges of G are shown by rank in groups of 100. For each group the average number of edges in that group that are essential and the average number that are MST edges are recorded. For example, of the smallest 100 edges of G, all 100 are essential edges and all 100 are tree edges (in three trials). Of the 401st–500th largest edges, on average 70 are essential and 2 are tree edges. The average number of nonessential edges in each group can be obtained by subtracting the number of essential edges from 100 (or, equivalently, by flipping the Essential curve top-to-bottom).

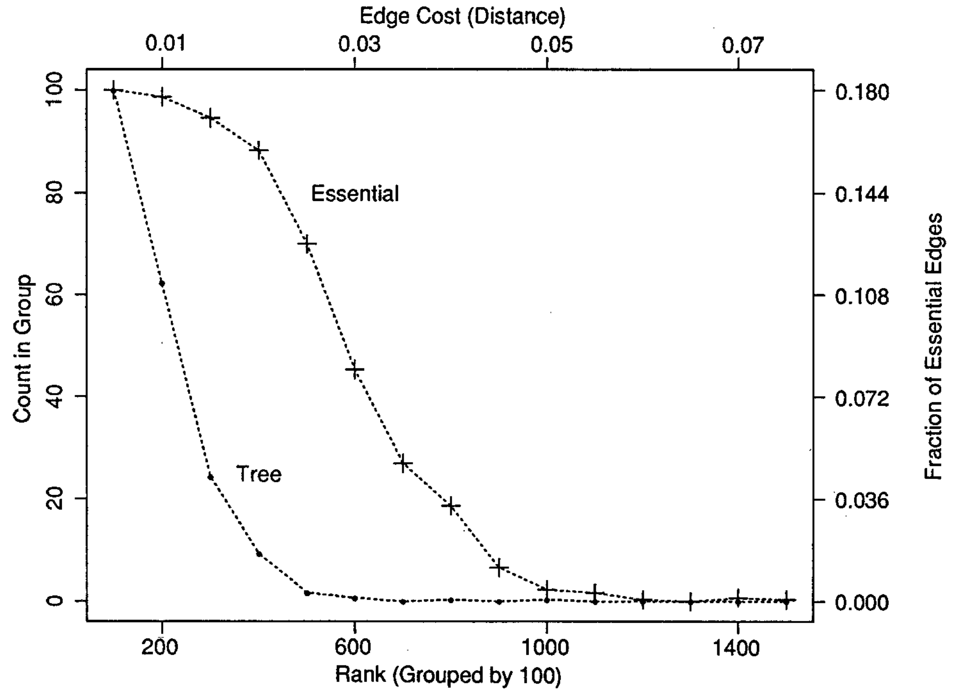

Fig. 4. Tree and essential distances.

The data in Figure 4 can also be read as an empirical density plot of essential and tree *distances*, since, for practical purposes, edge ranks and edge costs are related by $c = r/(m + 1)$ and for these edge types cost equals distance. The top axis marks edge costs for these groups. The right-side axis, showing the fraction of essential edges in each group, is scaled for the Essential curve by dividing the values on the Count axis by $S = 554.67$: for example, about 18% of essential edges come from ranks 0–100.

Theoretical characterization of these distributions remains an open problem. Preliminary analysis for these and other trials suggests that at least 80% of essential edges come from the range $(0, R/2]$, and that the average essential distance is about one-fourth the maximum essential distance (C).

Figure 5 shows an empirical probability density plot of *all* distances in G for the three trials at $n = 200$.

Quantile–quantile plots at this and other values of n suggest that distances are fairly well described by normal curve, although some skewing toward the right can be observed. A normal curve with mean 0.026 and standard deviation 0.01 is superimposed on the plot. The center 0.026 corresponds to the observed median rather than the mean 0.0271, which provides a better fit to this skewed data set. Note that the normal fit is only descriptive: there is no analytical argument to suggest that the underlying distribution of distances is normal. Indeed a chi-square distribution might be more appropriate since distances are bounded below by zero; however this normal curve represents the best empirical fit found over a small set of candidate distributions (including chi-square).

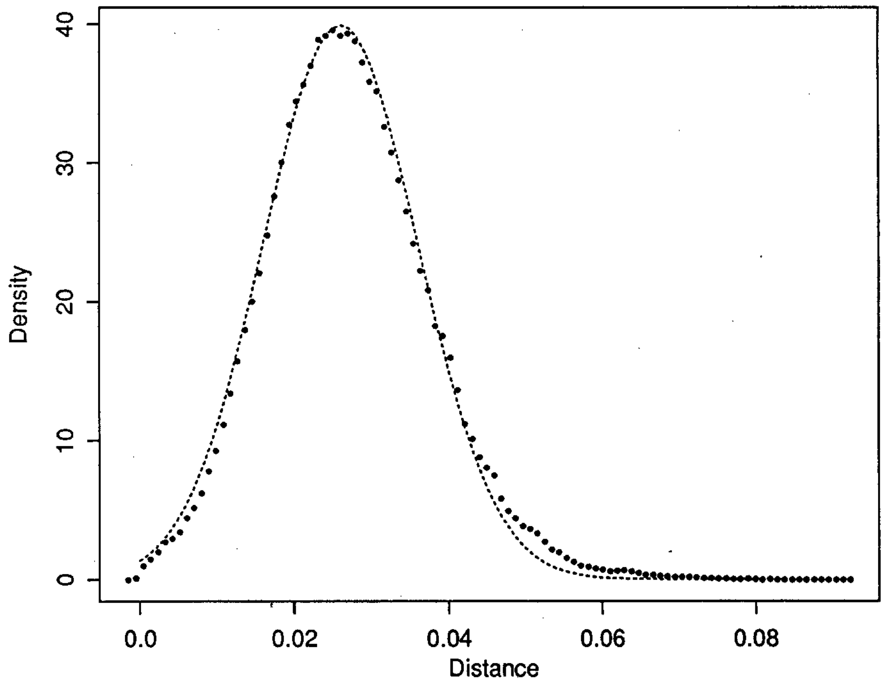

Fig. 5. Density plots of distances.

CONJECTURE. *Distances in uniform graphs have approximately a normal distribution. The average distance is about one-third the maximum distance (the graph diameter).*

Acknowledgments. Thanks go to Jon Bentley, David Johnson, Lyle McGeoch, and Peter Shor for several stimulating discussions. Bob Tarjan suggested that the diameter of the minimum spanning tree could be used as a stopping function for the generation algorithm. Two anonymous referees gave useful suggestions for improving the presentation.

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** (1990), 213–223.

[2] N. Alon, Z. Galil, and O. Margalit. On the exponent of the all pairs shortest path problem. *Proc. 32nd FOCS*, 1991, pp. 569–575.

[3] G. Ausiello, G. F. Italiano, A. M. Spaccamela, and U. Nanni. Incremental algorithms for minimal length paths. *Proc. First SODA*, 1990 pp. 12–21.

[4] J. L. Bentley and J. B. Saxe. Generating sorted lists of random numbers. *ACM Trans. Math. Software*, **6**(3) (1980), 359–364.

[5] P. A. Bloniarz. A shortest-path algorithm with expected time $o(n^2 \log n \log^* n)$. *SIAM J. Comput.*, **12**(3) (1983), 588–600.

[6] B. Bollobás. *Random Graphs*. Academic Press, New York, 1985.

[7] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, **1** (1959), 269–271.

[8] P. Erdös and A. Rényi. On random graphs, I. *Publ. Math. Debrecen*, **6** (1959), 290–297.

[9] R. W. Floyd. Algorithm 97: Shortest path. *Comm. ACM*, **5** (1962), 345.

[10] M. L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, **5** (1976), 83–89.

[11] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. Assoc. Comput. Mach.*, **34**(3) (1987), 596–615.

[12] M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Proc. 31st FOCS*, October 1990, pp. 719–725.

[13] A. M. Frieze. On the value of a random minimum spanning tree problem. *Discrete Appl. Math.*, **10** (1985), 47–56.

[14] A. M. Frieze and G. R. Grimmet. The shortest-path problem for graphs with random arc-lengths. *Discrete Appl. Math.*, **10** (1985), 57–77.

[15] R. Hassin and E. Zemel. On the shortest paths in graphs with random weights. *Math. Oper. Res.*, **10**(4) (1985), 557–564.

[16] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. Assoc. Comput. Mach.*, **24** (1977), 1–13.

[17] D. R. Karger, D. Koller, and S. J. Phillips. Finding the hidden path: time bounds for all-pairs shortest paths. *SIAM J. Comput.*, **22**(6) (1993), 1199–1217.

[18] A. Moffat and T. Takoaka. An all pairs shortest path algorithm with expected time $o(n^2 \log n)$. *SIAM J. Comput.*, **16**(6) (1987), 1023–1031.

[19] K. V. S. Ramaro and S. Venkatesan. On finding and updating shortest paths distributively. *J. Algorithms*, **13** (1992), 235–257.

[20] P. Robert. An algorithm for finding the essential sets of arcs of certain graphs. *J. Combin. Theory*, **10** (1971), 288–298.

[21] R. Sedgewich. *Algorithms*. Addison-Wesley, Reading, MA, 1988.

[22] P. M. Spira. A new algorithm for finding all shortest paths in a graph of positive arcs in average time $O(n^2 \log n)$. *SIAM J. Comput.*, **2**(1) (1973) 28–32.

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

[24] B. W. Weide. Random graphs and graph optimization problems. *SIAM J. Comput.*, **9**(3) (1980), 552–557.