AN ALL PAIRS SHORTEST PATH ALGORITHM WITH EXPECTED RUNNING TIME $O(n^2 \log n)$

Alistair Moffat

Department of Computer Science
University of Canterbury
New Zealand.

Tadao Takaoka

Department of Information Science
Ibaraki University
Japan.

Abstract. An algorithm is described that solves the all pairs shortest path problem for a non-negatively weighted graph. The algorithm has an average requirement on quite general classes of random graphs of $O(n^2 \log n)$ time, where n is the number of vertices in the graph.

1. Introduction.

Let N=(G,C) be a directed network, where G=(V,E) is a complete directed graph of n vertices and C:E -) R+ is a cost function on the edges of G. The cost of a path in N is the sum of the costs of the edges comprising the path; a shortest path between two vertices is a path of minimal cost. The all pairs shortest path (apsp) problem requires the tabulation of the function L, where for two vertices i,j in V, L(i,j) is the cost of a shortest path from vertex i to vertex j. When the weight function C is restricted to be non-negative the function L is defined for all i,j in V, and it is this case that is considered here.

Many algorithms to solve the apsp problem are known. For average case running time, where the edge costs are drawn independently from any arbitrary distribution, a sequence of successively faster algorithms has been given[8 9 3] with the current best upper bound being the $O(n^2 \log n \log^* n)$ expected time algorithm of Bloniarz. Here a hueristic due to Fredman is added to the algorithm of Spira; the result is an $O(n^2 \log n)$ average time algorithm that improves asymptotically the bound of Bloniarz.

2. The Spira paradigm.

Spira[8] introduced average time analysis to apsp algorithms in 1973. He improved an algorithm originally attributable to Dantzig[4]; by pre-sorting the edge lists and using a heap to enable paths to be efficiently found in increasing cost order he reduced the $O(n^3)$ bound of Dantzig's method to $O(n^2 \log^2 n)$ average time.

In Spira's method, to solve a single source problem from some vertex s, vertices are labelled and included into a solution set S in order of increasing shortest path distance from s. The vertices in S have their correct shortest path cost recorded in a vector D. Initially s is the only labelled vertex, with D[s]=0; in the course of the processing S is expanded until every vertex is included. Edges from any vertex in S are examined in increasing cost order with the aid of the sorted edge lists, and at any stage all edges that have been examined lead to vertices that have been labelled. For each labelled vertex c, if (c,t) is the shortest unconsidered edge from c then (c,t) is maintained as the "candidate" for c, with a numeric "weight" given by D[c]+C(c,t).

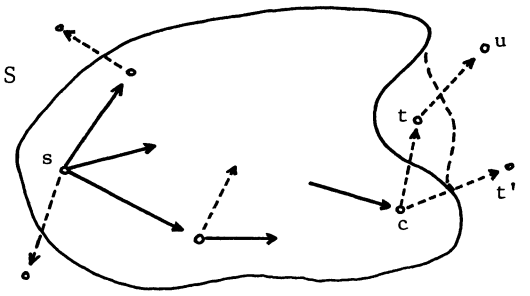

Figure 1. Spira's paradigm.

To allow efficient identification of the candidate of minimum weight, a heap data structure is used. Suppose that at some stage of the algorithm (c,t) with weight D[c]+C(c,t) is at the root of the heap. If t is not already labelled it can be given a final distance D[t] of D[c]+C(c,t) and included in S. In this case a candidate for t must be added into the heap; the candidate for t will be (t,u), the shortest edge from t, with weight D[t]+C(t,u). After this step it is certainly the case that t is labelled and that the edge (c,t) has been considered as part of a shortest path, so now candidate (c,t) is replaced in the heap by (c,t'), where t' is the next shortest edge from c. This next shortest edge is again found in O(1) time with the use of the sorted edge list.

The following program is for Spira's single source algorithm. The variable "heap" is a binary heap[1] of candidates, where the weight of candidate (c,t) is given by D[c]+C(c,t). A heap of n elements can be built in O(n) time; updates and insertions can be performed in O(logn) time, and the element of minimum weight can be identified in O(1) time.

0272-5428/85/0000/0101$01.00 © 1985 IEEE

```
procedure spira-sp( s ) ;
  begin
    S := {s} ; D[s] := 0 ;
    initialise the heap to (s,t), where t
      is the endpoint of the shortest edge
      from s ;
    while |S| < n do
      begin
        let (c,t) be at the root of the heap ;
        replace (c,t) by (c,t'), where t' is the
          endpoint of the next shortest edge
          from c, and rearrange the heap ;
        if t is not in S then
          begin
            S := S + {t} ;
            D[t] := D[c]+C(c,t) ;
            add into the heap (t,u), where u is
              the endpoint of the shortest
              edge from t, and rearrange the
              heap ;
          end ;
      end ;
  end {spira-sp} ;
```

Algorithm Spira-sp.

Correctness of Spira-sp.
The following argument was given by Bloniarz to show that the algorithm is correct. The constraints on the vertices of S are such that, for all c in S, if (c,t) is the heap entry for c, then

$L(s,c) = D[c]$,
$L(s,c) \leq L(s,u)$ for all u in V-S, and
$C(c,t) \leq C(c,u)$ for all u in V-S.

Lemma 1[8] [3]. Suppose (c0,t0) in the heap is such that

$D[c0] + C(c0,t0) = \text{minimum}\{ D[c] + C(c,t) : (c,t) \text{ in heap } \}$.

Then if t0 is in V-S,

$L(s,t0) = D[c0] + C(c0,t0)$, and
$L(s,t0) \leq L(s,u)$ for all u in V-S .

The lemma shows that the constraints on S are maintained as more and more vertices are added, and when |S|=n the first constraint means that the vector D contains the required shortest path costs. Only when an unlabelled candidate is drawn as the minimum cost element in the heap will S be expanded; to see that |S| will ultimately reach n (for a complete graph) and the algorithm terminate, observe that every iteration of the while loop irrevocably "consumes" one edge from one of the sorted edge lists, and so at most n(n-1) iterations of this loop can be required before all edges have been examined and thus all vertices labelled.

Random graphs.
Spira used as his class of randomness graphs in which the cost on each of the n(n-1) edges is drawn independently from the same random distribution. The distribution itself is arbitrary. Bloniarz redefined this class of graphs, widened it to a class he called "endpoint independent" graphs, and showed that the average running time of Spira's algorithm on this wider class of random graphs is still $O(n^2 \log^2 n)$. The primary property of an endpoint-independent measure Bloniarz describes thus: "Suppose a particular edge is selected from the sorted edge list by virtue of either its position on the list or the value of its cost. If P [the probability measure] is endpoint-independent, then the endpoint of this edge is independent of the edge's selection; every endpoint is equally likely"[3].

For the analysis of the new algorithm this description of the classes of graph to be processed will also be used - the claimed running time of the new algorithm will be achieved when the graph is such that each edge list when sorted by cost is a random permutation of the edge list when sorted by destination. Note that this requires that edges in the edge list of equal cost be stored in a random order within the overall sorted order[2]. Then when the next element of any edge list is taken its destination is equally likely to be any of the n-1 other vertices in the graph, and the repeated examination of the destinations of edges as any edge list is scanned can be taken to be a sequence of independent trials.

A number of types of edge cost assignments meet this endpoint independence requirement. The simplest situation is when each edge in the graph is independently assigned a value from any single distribution of any sort; but also within the scope of the definition is the situation in which each source has associated with it some different random distribution function, and edges are assigned costs independently drawn from the distribution of the corresponding source.

Analysis of Spira-sp.
At the j'th stage, when |S|=j, the heap contains j candidates. The minimum cost candidate is then drawn in an attempt to label a new vertex; because of the endpoint independence the destination of the minimum cost candidate is equally likely to be any of the n vertices, and there are only n-j that are unlabelled. The probability of drawing an unlabelled vertex as the minimum cost candidate is thus (n-j)/n. The process is continued with |S|=j until an unlabelled vertex is drawn; the expected number of such drawings, until S can be expanded, is n/(n-j). The total number of drawings from the heap for one single source problem is expectedly sigma(j=1 to n-1)(n/(n-j)), as S expands from one element to n elements. This sum is O(nlogn). Each drawing requires a heap operation, so that over all sources of an apsp problem the time required is $O(n^2 \log^2 n)$, which dominates the $O(n^2 \log n)$ requirement of the presort.

Fredman's Modification.
Fredman[6] introduced the concept of heap cleaning to show that Spira's algorithm could be modified to require only $O(n^2 \log n)$ comparisons on average. He suggested cleaning the heap at stages n/2, 3n/4, 7n/8, ..., n-2, n-1, totalling* logn cleaning steps in all. The heap cleaning, or removal and replacement of "dirty" candidates that are already labelled, is to improve the

---

* The function ln is used for natural logarithms. All other logarithms are binary.

probability of an unlabelled candidate being drawn at the root the heap.

Spira's algorithm is allowed to begin normally. When the size of S reaches n/2 (figure 2) the normal processing is suspended and a heap cleaning stage carried out. The first step in a cleaning stage is to mark as purged all labelled vertices. Next, the label of each candidate in the heap is examined, and all candidates that are already labelled are replaced by unlabelled candidates, found by scanning the edge lists and skipping all edges with purged destinations. While replacing these dirty candidates, the heap will lose its heap property, but during this second step this is permitted. Finally, the heap is completely rebuilt, taking O(n) time.

Once the heap has been cleaned, the normal Spira type processing is resumed, but with the additional rule that whenever an edge with a purged destination is encountered it should be skipped, meaning that none of the n/2 vertices labelled in the first phase will reappear as candidates. The alternation of normal processing and cleaning continues, with heap cleaning taking place when |S|=n/2, 3n/4, and so on. Each heap cleaning step will require some amount of time to purge vertices and to find clean candidates, and O(n) comparisons for the heap rebuilding. Over each single source problem the logn heap rebuildings will require O(nlogn) comparisons.

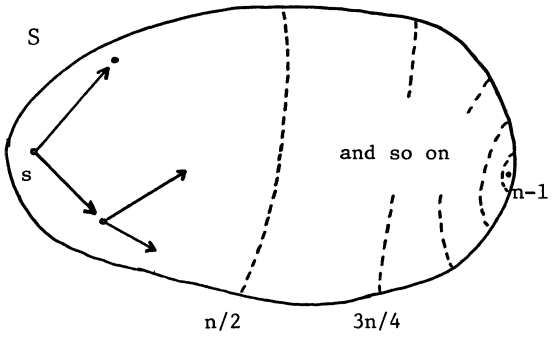

Figure 2. Fredman's cleaning stages.

Having logn cleanings ensures that the probability of drawing an unlabelled candidate at the root of the heap is always greater than 1/2, since the destination of a candidate will always be among the set of vertices that were unlabelled at the most recent heap cleaning operation, and between any cleaning operation and the next the number of unlabelled vertices halves. Thus the number of heap operations required before S expands is always expectedly less than 2, and since each heap operation will require O(logn) comparisons, for the apsp problem an average of $O(n^2 \log n)$ comparisons will be sufficient.

However, as was noted by Fredman, the running time is much higher. By the time the heap is cleaned with |S|=n-1, each edge list pointer will have expectedly moved to a position midway along the corresponding edge list, as the last remaining vertex will be the candidate of all n-1 labelled vertices. So for a single source problem the total edge scanning effort will be $\Theta(n^2)$; or $\Theta(n^3)$ for an apsp problem.

3. An $O(n^2 \log n)$ Algorithm.

The important observation that makes the new algorithm possible is that Fredman's scanning for a good candidate, required to make the heap clean, is expensive only in the closing stages. Even when there are as few as n/logn vertices remaining unlabelled, scanning for a good candidate in an edge list will expectedly require the skipping of only logn edges. This observation leads to the rule that the heap cleaning must be ceased at the critical point of |S|=n-n/logn; in the concluding stages, as the last n/logn vertices are labelled, the heap must be allowed to become dirty. Stopping the cleaning when |S|=n-n/logn means that there will only be loglogn cleaning stages.

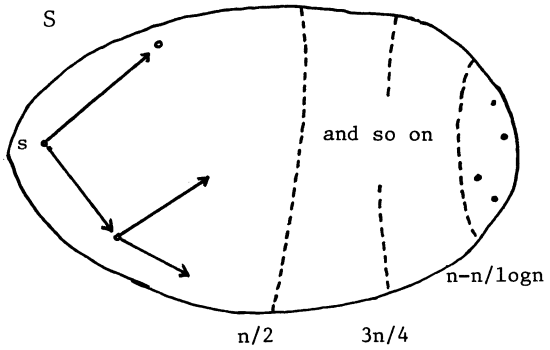

Figure 3. Revised cleaning stages.

As the last n/logn vertices are labelled it becomes increasingly unlikely that a good candidate will be drawn from the heap. But n/logn is smaller than n by a factor of logn, and so to label these n/logn vertices and still retain an O(n) bound on the total number of heap operations there are now O(logn) heap operations available per labelling.

From these ideas the algorithm follows.

```
procedure fast-sp( s ) ;
  begin
    S := {s} ; D[s] := 0 ; purged := {s} ;
    initialise the heap to (s,t), where t
      is the endpoint of the shortest edge
      from s ;
    for k := 1 to loglog(n) do
      begin
        expand-soln( n - (n div 2**k) ) ;
        clean-up-heap ;
      end ;
    expand-soln( n ) ;
  end {fast-sp} ;
```

```
procedure expand-soln( stopat ) ;
  begin
    while |S| < stopat do
      begin
        let (c,t) be at the root of the heap ;
        replace (c,t) by (c,t'), where t' is the
          endpoint of the next shortest edge
          from c such that t' is unpurged,
          and rearrange the heap ;
        if t is not in S then
          begin
            S := S + {t} ;
            D[t] := D[c]+C(c,t) ;
            add into the heap (t,u), where u is
              the destination of the shortest
              edge from t such that u is
              unpurged, and rearrange the heap ;
          end ;
      end ;
  end {expand-soln} ;

procedure clean-up-heap ;
  begin
    purged := S ;
    for each element (c,t) in the heap do
      if t is in S then
        replace (c,t) by (c,t'), where t'
          is the next closest unpurged
          destination vertex from c ;
    completely rebuild the heap ;
  end {clean-up-heap} ;
```

Algorithm Fast-sp.

The function loglog(n) should return 0 for n<4, and ceiling(log(log(n))) otherwise. This definition means that for n=4
logn <= 2**loglog(n) < 2logn.

Correctness of Fast-sp.
The correctness of the algorithm follows from the correctness of Spira's algorithm. In the edge scanning, only edges leading to labelled vertices will be skipped, and these would also have been discarded as useless by Spira's algorithm if taken from the heap. To make this clear, to the heap manipulation routines should be added a tie breaking rule that says that if two candidates have equal weight, the candidates of v1 and v2 say, then the candidate returned will be that of the smaller of v1 and v2, using the index of the labelling vertex as a secondary key. Then the sequence of edges used for labelling will be identical for both algorithms, and the shortest path tree generated by the new algorithm will be identical to the shortest path tree of Spira's algorithm.

Analysis of Fast-sp.
The processing consists of loglogn+1 phases. During the first phase the solution set is expanded from 1 entry to n/2 entries, during the loglogn'th, from n-2n/logn to n-n/logn entries, and in the final loglogn+1'th phase S grows to include all of the remaining n/logn vertices. The dominant components in the running time are the number of update and insert operations on the heap (hops), and the total number of edges inspected (scans) during the course of the algorithm. Only O(nloglogn) time per source will be spent on the heap rebuildings.

The heap will be cleaned loglogn times; to find the good candidates with which the heap will be rebuilt requires some scanning effort. At the end of the k'th phase there are n/2**k unlabelled vertices, so that expectedly 2**k edges need to be examined before a clean candidate is found. This must be done for at most n heap entries, so the scanning effort for the k'th rebuilding is expectedly (2**k)n.

During the k'th of the first loglogn phases n/2**k vertices will be added to the solution set. Each successful labelling of a vertex involves one addition to the heap, and to achieve each successful labelling there will be some number of update operations caused by useless candidates, the number depending on the probability that the minimum weight candidate leads to an unlabelled destination. But because of the heap cleaning, this probability is never less than 1/2, so that during the whole phase the number of heap operations is expectedly less than 3n/2**k.

Each of these heap operations has associated with it some edge scanning. During the k'th phase there are n/2**(k-1) unpurged vertices so that the expected number of edges skipped for each heap operation is 2**(k-1), and the total edge scanning requirement caused by the k'th phase is expectedly 3n/2.

Thus the total cost of the first loglogn phases is given by

$$\begin{aligned} \text{hops} &= \sum_{k=1}^{\log\log n} [\, 3n/2^k \,] \\ &< 3n \\ \text{scans} &= \sum_{k=1}^{\log\log n} [\, 2^k n + 3n/2 \,] \\ &< 4n\log n + O(n\log\log n) \quad . \end{aligned}$$

During the last phase there are n/logn unpurged candidates, and at any stage during the last phase a heap operation will require a corresponding scanning effort of expectedly logn edges. During the phase every element in the heap will have as its candidate one of the n/logn unpurged destinations; moreover, all of the n/logn possible destinations of the minimum element as it is drawn from the heap are equally likely. Thus the situation is again a coupon collector problem[5], and the expected number of heap operations in the final phase is given by

$$\begin{aligned} \text{hops} &= n/\log n + (n/\log n)\ln(n/\log n) \\ &< 2n \quad , \end{aligned}$$

and the scanning effort by

$$\text{scans} < 2n\log n \quad .$$

Counting all loglogn+1 phases,

$$\begin{aligned} \text{hops} &< 3n + 2n \\ &= O(n) \quad , \\ \text{scans} &< 4n\log n + 2n\log n + O(n\log\log n) \\ &= O(n\log n) \quad . \end{aligned}$$

Thus the total expected running time for one source is given by

$$\begin{aligned} \text{time} &= \text{scans}*O(1) + \text{hops}*O(\log n) \\ &= O(n\log n) \quad , \end{aligned}$$

and the total expected running time for the all pairs problem, even including $O(n^2 \log n)$ time for the presort, is, as claimed, $O(n^2 \log n)$.

Continuous cleaning.

The worst case running time of algorithm Fast-sp is $O(n^3 \log n)$, a bound shared with the algorithms of Spira and Bloniarz. The following modification has the effect of reducing the worst case behaviour of Fast-sp to $O(n^3)$ while retaining the $O(n^2 \log n)$ average case running time.

The algorithm as given uses discrete cleaning stages to remove dirty candidates from the heap. It is also possible to return to Dantzig's original requirement, that all candidates be clean, and replace the dirty candidates immediately after each labelling, keeping the heap continuously clean. Each labelling will, by the endpoint independence, expectedly require the replacement of $j/(n-j)$ of the candidates, where $j=|S|$. The concept of the critical point is still important, and this policy can only be followed while $|S|<n-n/\log n$; during the last phase the heap must still be allowed to become dirty. Then since $j<n-n/\log n$, expectedly no more than $\log n$ candidates will need to be replaced per labelling. With a suitable data structure to record the vertices v that have t as their candidate, and a record of heap locations, this can be done in $O(\log n)$ expected time, since the position of each affected candidate is random in the heap, the weights of revised elements will always increase, moving them away from the root, and the average number of heap levels below a random position is $O(1)$. The scanning effort of this long first phase will also be $O(n \log n)$, since the final phase will begin in exactly the same configuration as if the heap cleaning had been discrete, meaning that the previous analysis applies. Thus, the apsp average time is retained at $O(n^2 \log n)$.

The worst that can happen is that every candidate will be replaced at every labelling, but provided that the updates are processed in bottom up order the complete heap rebuilding will take only $O(n)$ time, totalling $O(n^2)$ time for the long first phase until $|S|=n-n/\log n$. During the last phase there are only $n^2/\log n$ edges that will not be skipped, and even allowing an $O(\log n)$ time heap operation for each to enter the heap, the worst case for the last phase is also $O(n^2)$. The overall scanning effort is bounded by the number of edges, and is $O(n^2)$ per source, so that with continuous cleaning until $|S|=n-n/\log n$, the worst case running time for the apsp problem will be $O(n^3)$.

4. Summary.

By identifying a critical point when $|S|=n-n/\log n$, beyond which it becomes too expensive to clean the heap but before which it is costly to not clean the heap, it has been possible to develop an algorithm for the all pairs shortest path problem that requires $O(n^2 \log n)$ time on average, and $O(n^3)$ worst case time. This algorithm is an asymptotic improvement over the $O(n^2 \log n \log^* n)$ algorithm of Bloniarz, but solves the same general families of graphs; the algorithm also includes entirely the narrow class of probability distribution for which Frieze and Grimmett[7] obtained $O(n^2 \log n)$ running time.

References.

[1] Aho A.V., Hopcroft J.E., Ullman J.D., "The Design and Analysis of Computer Algorithms", Addison-Wesley, Reading, Massachussetts, 1974.

[2] Bloniarz P.A., Meyer A., Fischer M., "Some observations on Spira's shortest path algorithm", Tech. Rep. 79-6, Computer Science Department, State University of New York at Albany, 1979.

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

[4] Dantzig G.B., "On the shortest route through a network", Management Sci., 6(1960), 187-190.

[5] Feller W.H., "An Introduction to Probability Theory and Its Applications", John Wiley, New York, 1963.

[6] Fredman M.L., "On the decision tree complexity of the shortest path problem", 16th Conf. Found. Comp. Sci., 1975, 98-99.

[7] Frieze A.M., Grimmett G.R., "The shortest path problem for graphs with random arc lengths", Tech. Rep. S-83-02, Dept. Comp. Sc. and Stat., Queen Mary College, University of London, 1983.

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

[9] Takaoka T., Moffat A.M., "An $O(n^2 \log n \log\log n)$ expected time algorithm for the all shortest distance problem", Lecture Notes in Computer Science Vol. 88 (ed P. Dembinski), Springer-Verlag, Berlin, 1980, 643-655.