Incremental Algorithms for Minimal Length Paths*

Giorgio Ausiello†
Giuseppe F. Italiano‡
Alberto Marchetti Spaccamela§
Umberto Nanni¶

Abstract

We consider the problem of maintaining a solution to the All Pairs Shortest Paths Problem in a directed graph $G = (V, E)$ where edges may be dynamically inserted. In case of unit edge costs, we introduce a new data structure which is able to answer queries concerning the length of the shortest path (i.e., a path with minimal number of edges) from one vertex to another in constant time, and to trace out the shortest path from one vertex to another in time linear in the number of edges reported. The total time required to maintain the data structure under a sequence of at most $O(n^2)$ insertions of edges is $O(n^3 \log n)$ [1] in the worst case, where n is the total number of vertices in G. Our data structure can be extended to provide incremental algorithms for maintaining maximal length paths or shortest paths when the edge costs are integers in a fixed range. All the given algorithms improve the best previously known bounds for the same problems and are a factor of $\log n$ away from the best possible bounds.

*Work partially supported by the ESPRIT II Basic Research Actions Program of the European Communities under contract No. 3075 (Project ALCOM).

†Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Roma, Italy.

‡Department of Computer Science, Columbia University, New York, NY 10027 and Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Roma, Italy. Partially supported by NSF Grants CCR-86-05353, CCR-88-14977 and by an IBM Graduate Fellowship.

§Dipartimento di Matematica Pura e Applicata, Università di L'Aquila, L'Aquila, Italy.

¶Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Roma, Italy. Partially supported by Selenia S.p.A., Italy.

[1] All the logarithms are assumed to be to the base 2 unless explicitly specified otherwise.

1 Introduction

A dynamic data structure is able to update the solution of a problem after dynamic changes, rather than having to recompute it from scratch each time. Given their powerful versatility, it is not surprising that dynamic data structures are often more difficult to design than static ones. In the realm of graph problems, several dynamic data structures have been proposed to support insertions and deletions of edges and/or vertices in a graph, in addition to certain types of queries. In particular, much attention has been devoted to the dynamic maintenance of connectivity [7, 9, 10, 14, 23, 25], transitive closure [3, 16, 17, 18, 20, 22, 29], shortest paths [8, 24, 26, 29] and minimum spanning trees [6, 7, 10, 11, 26]. Dynamic algorithms for graphs are of theoretical as well as of practical interest [1, 4, 5, 15, 19, 30].

A fundamental problem in this area is the dynamization of the All Pairs Shortest Paths Problem. The problem consists of maintaining online during edge insertions a distance matrix D such that entry $D[i, j]$ contains the length of the shortest path from i to j, for any pair of vertices (i, j). Another way to look at this problem is to maintain an underlying directed graph under an arbitrary sequence of operations of the following three kinds.

- $add(x, y)$: insert an edge from vertex x to vertex y.
- $length(x, y)$: return the length of the shortest path from x to y, if one exists; return $+\infty$ otherwise.
- $minpath(x, y)$: return the shortest path from x to y, if one exists.

The constraint of the problem is that we would like to answer *length* queries in $O(1)$ time and to perform *minpath* operations in time linear in the length of the traced path.

This problem has been previously studied [8, 24, 26, 29] but the known solutions seem far from optimal. Even and Gazit [8] showed how to maintain a solution to the All Pairs Shortest Paths during edge insertions and deletions. Their data structure requires $O(n^2)$ time to be updated after an edge insertion and $O(mn + n^2 \log n)$ time after an edge deletion, where m is the current number of edges in the graph. Notice that the time required by an edge deletion is similar to the best known algorithm for computing all pairs shortest paths from scratch, due to Fredman and Tarjan [12]. Rohnert [24] and recently Yellin [29] gave similar bounds. The bounds achieved by these data structures do not benefit from an amortized analysis and the total time required to support a sequence of $O(n^2)$ edge insertions is $O(n^4)$.

In this paper, we consider the problem of maintaining a solution to the All Pairs Shortest Path Problem on a directed graph under edge insertions, with integer edge costs in a fixed range. In the special case of unit edge costs, we show how to maintain online a solution to the All Pairs Shortest Paths Problem by means of a data structure which needs a total of $O(n^3 \log n)$ time in the worst case to be maintained during any sequence of (at most $O(n^2)$) *add* operations, where n is the number of vertices in the graph. The total space required is $O(n^2)$. Since we show that there are sequences of edge insertions which require $\Omega(n^3)$ changes in the distance matrix D, our algorithm is only a factor of $\log n$ away from the best possible bound. In the remainder of this paper we refer to the All Pairs Shortest Paths Problem with unit edge costs as the All Pairs Minimal Length Paths Problem. Furthermore, we extend our data structure to deal with integer edge costs in the range $[1 \ldots n^\alpha], \alpha < 1$ and we show how to update the solution to the All Pairs Shortest Paths Problem during both edge insertions and edge cost decreases in a total of $O(n^{3+\alpha} \log n)$ worst-case time. The space required by our data structure is $O(n^2)$. Also in this case, there are sequences of edge insertions which require $\Omega(n^{3+\alpha})$ changes in the distance matrix D, and once again, our algorithm is only a factor of $\log n$ away from the best possible bound. As a side result, we show also how to achieve similar bounds while maintaining maximal length paths in directed acyclic graphs, when *add* operations do not introduce cycles. This is not a restriction, since the maximal length path problem in cyclic graphs is known to be NP-complete [13].

We remark that previously proposed data structures for the same problem [8, 24, 26, 29] do not seem to benefit from unit edge costs and the total time they require to support a sequence of $O(n^2)$ *add* operations is $O(n^4)$. Therefore, our data structure improves almost an order of magnitude on this time.

2 The Data Structure

We assume that the reader is familiar with the standard graph theoretical terminology as contained for instance in [2, 27].

Given a digraph $G = (V, E)$ and a vertex $x \in V$, a *tree of forward paths rooted at* x is a tree $T_f(x) = (X, S)$ rooted at x such that *(i)* X is the set of descendants of x in G, and *(ii)* $S \subseteq E$. Similarly, a *tree of backward paths* rooted at x is a tree $T_b(x) = (Y, S)$ rooted at x such that *(i)* X is the set of ancestors of x in G, and *(ii)* $S \subseteq E^R$, where $E^R = \{(y, x)/(x, y) \in E\}$. A tree of forward (backward) paths $T = (X, S)$ rooted at x is said to be of *minimal length* (or equivalently it is said to be a *tree of forward (backward) minimal paths*) if for each vertex $v \in X$, the depth of v in T coincides with the length of the minimal path from x to v. In a similar vein given a directed acyclic graph (from now on referred to as a *dag*) G, a tree of forward (backward) paths $T = (X, S)$ rooted at x is said to be of *maximal length* (or equivalently it is said to be a *tree of forward (backward) maximal paths*) if for each vertex $v \in X$, the depth of v in T coincides with the length of the maximal path from x to v. Figure 1 exhibits a digraph G with the trees of forward and backward paths of minimal length rooted at one of its vertices.

We now present a data structure which maintains online a solution to the All Pairs Minimal Length Paths Problem. Our data structure supports each *length* operation in $O(1)$ time, and each *minpath* operation in $O(k)$ time, where k is the length of the minimal length path traced. The overall time required to maintain the data structure during a sequence of (at most $O(n^2)$) *add* operations is $O(n^3 \log n)$. The space complexity is $O(n^2)$.

In addition to maintaining and updating the $n \times n$ distance matrix D, we associate to each vertex $v \in V$ the two sets *DESC(v)* and *ANC(v)* of its descen-

dants and ancestors. In order to retrieve information about minimal length paths, $DESC(v)$ is organized as a tree of forward minimal paths rooted at v and $ANC(v)$ as a tree of backward minimal paths rooted at v. Moreover, we use two $n \times n$ matrices of pointers, named *FORWARD* and *BACKWARD* defined as follows. $FORWARD[u,v]$ contains a pointer to vertex v in the tree of minimal forward paths $DESC(u)$ if vertex v is a descendant of vertex u. Otherwise, $FORWARD[u,v]$ contains a special *null* pointer. Similarly, $BACKWARD[u,v]$ contains a pointer to vertex v in the tree of minimal backward paths $ANC(u)$ if vertex v is an ancestor of vertex u. Otherwise, $BACKWARD[u,v]$ contains a special *null* pointer. All these data structures can be initialized in $O(n^2)$ time.

With these data structures, *length* operations are easily performed in $O(1)$ time, as the following pseudo-code shows.

function *length* (*vertices* : x, y) : integer;
begin return($D[x,y]$) **end;**

A minimal path from u to v can be returned by first examining the entry $FORWARD[u,v]$. If it is a *null* pointer, then there is no path from u to v and therefore the value $+\infty$ must be returned. Otherwise $FORWARD[u,v]$ allows us to locate v in $DESC(u)$, the tree of forward minimal paths rooted at u. If reversal pointers to the parent for each vertex in such trees are maintained, a bottom-up traversal from v to the root u in $DESC(u)$ takes at most $O(k)$ time to return a minimal length path from u to v in G, where $k \leq n$ is the length of the achieved path.

```
procedure minpath (vertices : x, y ;
                   list of vertices : Π);
pointer to vertices : p ;
begin
  Π := ∅ ; p := FORWARD[x, y] ;
  while p ≠ null do begin
    insert into Π the vertex pointed to by p ;
    p := parent(p) ;
  end ;
end ;
```

We now show how to update the distance matrix D, the two matrices *FORWARD* and *BACKWARD*, and the $2n$ trees $DESC(v)$ and $ANC(v)$ for any v, because of the insertion of a new edge (i,j). Denoting by $d(x,y)$ the length of a minimal path from vertex x to vertex y in the graph G (hereafter referred to also as minimal length function), the following properties hold:

$$d(x,x) = 0, \forall x \in V. \quad (1)$$

$$d(x,y) = \min_{u \in V}\{d(x,u) + d(u,y)\}, \forall x,y \in V. \quad (2)$$

Furthermore, if $d_{new}(x,y)$ denotes the minimal length function after the insertion of an edge (i,j) and $d_{old}(x,y)$ denotes the minimal length function before the insertion of (i,j), then the following property must be true:

$$d_{new}(x,y) = \min\{d_{old}(x,y), d_{old}(x,i) + 1 + d_{old}(j,y)\}, \forall x,y \in V. \quad (3)$$

Equation (3) states that the insertion of an edge (i,j) can introduce new shorter paths only from ancestors x of vertex i (for which $d(x,i) < +\infty$) to descendants y of vertex j (for which $d(j,y) < +\infty$). In this case, the trees of minimal forward paths might have to be updated for vertices in $ANC(i)$ and the trees of minimal backward paths might have to be updated for vertices in $DESC(j)$.

We now describe how to update trees of minimal forward paths because of the insertion of an edge (i,j). A crucial role in this update is played by the two trees $ANC(i)$ and $DESC(j)$: in order to guarantee a correct update of the forward trees, it is sufficient that the tree $DESC(x)$ for each vertex x in $ANC(i)$ is updated by taking into account $DESC(j)$.

However, to improve the time required for updating the data structure we need a more sophisticated approach. Our algorithm can be described as follows. We start from vertex i and update both the i-th row of the matrix D and the tree $DESC(i)$ by visiting the tree $DESC(j)$. The details of this update will be given later on. At the end of the update of $DESC(i)$, a tree T contained in $DESC(j)$ is defined which includes only vertices v for which the distance from i to v decreased due to the insertion of edge (i,j). This tree T is given to the children of i in $ANC(i)$ which will perform the updates of their forward trees of minimal paths in a recursive fashion. In other words, each vertex x in $ANC(i)$ starting from vertex i receives a tree T which is a (possibly pruned) copy of $DESC(j)$. After receiving T, vertex x updates the tree of forward minimal paths stored in it by taking into account T. At the same time, x performs a

further pruning of this copy and pass it to its children in $ANC(i)$ (i.e., vertices u for which there is an edge (u,x) in the graph).

The update of $DESC(x)$ for each vertex x in $ANC(i)$ is carried out as follows. Vertex x visits the tree T it receives, starting from the root of T. The subsequent update of $DESC(x)$ and the pruning of T are subject to the following two rules.

(i) When a vertex y in T for which $D[x,i]+1+D[j,y] < D[x,y]$ is reached, then a shorter path from x to y has been found. In this case, vertex y is inserted into $DESC(x)$ either as a child of i if $y=j$ or as a child of the same parent it has in $DESC(j)$ otherwise. By inserting a vertex y into a tree, we mean that any previous occurrence of y in the tree is implicitly deleted. With the help of the *FORWARD* and *BACKWARD* matrices this task can be accomplished in constant time and therefore will not affect the asymptotic time spent during the update. At the end of this step, all the children of y in T are examined in a recursive fashion.

(ii) When a vertex y in T for which $D[x,i]+1+D[j,y] \geq D[x,y]$ is reached, then no shorter path from x to y was introduced by the insertion of the edge (i,j). In this case no update is performed in $DESC(x)$ and the tree T is pruned by cutting the subtree rooted at y.

After vertex x finishes to update $DESC(x)$, the pruned copy of T is passed to the children of x in $ANC(i)$. In the following procedure the implementation details relative to the pruning of the tree and the definition of the parameter T are not completely specified. In the full paper we will show how the parameter passing can be implemented efficiently without affecting the claimed time bounds.

```
      procedure UpdateForward (vertices : x, i, j ;
                               tree : T) ;
      vertices : y, w; queue : Q;
      begin
1.      Q := {j};
2.      while Q ≠ ∅ do begin
3.        remove an item y from the queue Q;
4.        if D[x,i] + 1 + D[j,y] < D[x,y] then
            begin
5.            if y = j then
6.              insert j in DESC(x) as a
                child of i
              else
7.              insert y in DESC(x) as a
                child of parent(FORWARD[j,y]);
8.            D[x,y] := D[x,i] + 1 + D[j,y];
9.            for each w child of y in DESC(j)
                do insert w into the queue Q;
            end
          else
10.         delete the edge (parent(y), y) from
            the tree T;
        end;
11.     if T ≠ ∅ then for each edge (x,u) in ANC(i)
          do UpdateForward(u, i, j, T) ;
      end;
```

The updates of the trees of minimal backward paths can be carried out in a completely analogous fashion. In particular, the procedure *UpdateBackward* which updates the trees $ANC(y)$ for each y descendant of vertex j, can be obtained by interchanging the roles of the trees *DESC* and *ANC* and of the matrices *FORWARD* and *BACKWARD*.

3 Correctness and Time Complexity

In this section, we analyze the time complexity of the data structure and prove its correctness. Again we denote by $d(x,y)$ the length of a minimal path from x to y in the graph. Before stating our results, we need few technical lemmas.

Lemma 1 *After each operation and for any pair of vertices x and y in the graph, $D[x,y]$ equals the depth of vertex y in the tree* DESC(x), *if y is a descendant of x. Otherwise, y is not in* DESC(x) *and $D[x,y] = +\infty$.*

Proof : By induction on the number of operations performed. At the beginning, the lemma is true since $D[x,y] = +\infty$ for $x \neq y$, $D[x,x] = 0$ and $DESC(x) = \{x\}$, for any $x \in V$. Assume now that the lemma hold before the i-th operation. If the i-th operation is either a *length* or a *minpath* operation, then the lemma still holds afterwards because these two operations do not modify the data structure.

If the i-th operation is $add(i,j)$ then we show that after performing this operation for any vertex x the x-th row of matrix D still satisfies the lemma. We

proceed now by using a second induction, this time on the number of elementary operations performed onto $DESC(x)$. The basis of the second induction is satisfied since if j is inserted into $DESC(x)$, it is inserted on line 6. of procedure *UpdateForward* as a child of i. This implies that j is correctly inserted at depth $D[x,j] = D[x,i] + 1$ in $DESC(x)$ because of the first induction step. Assume now that all the insertions in the tree $DESC(x)$ before inserting a new vertex $y \neq j$ in it satisfy the lemma. When y is to be inserted in $DESC(x)$ it is inserted as a child of v, with v being the parent of y in the tree $DESC(j)$ (line 7. of procedure UpdateForward). Furthermore, v must have been inserted in $DESC(x)$ by procedure *UpdateForward* (otherwise vertex y will not be considered for possible insertion in $DESC(x)$ because of line 10.). As a consequence of the second induction step $D[x,v] = D[x,i] + 1 + D[j,v]$ equals the depth of v in the tree $DESC(x)$. Since $D[x,y]$ will be set to $D[x,i]+1+D[j,y]$ (line 8.) and $D[j,y] = D[j,v]+1$ because of the first induction step on tree $DESC(j)$, then $D[x,y] = D[x,i]+1+D[j,v]+1$ will be equal to the depth of y in the tree $DESC(x)$. Since for vertices y not in $DESC(x)$ after the update the entry $D[x,y]$ is not changed, this concludes the two induction steps and gives the lemma. □

Lemma 1 shows that if the distance matrix D is correctly updated, then procedure *minpath* correctly returns a path with minimal number of edges.

Lemma 2 *Assume that $D[x,y] = d(x,y)$ for any pair of vertices x and y in the graph. Then for any pair of vertices j and v such that v is reachable from j, and for any vertex y contained in the subtree of* DESC(j) *rooted at v, after each operation the following equality holds.*

$$D[j,y] = D[j,v] + D[v,y]. \tag{4}$$

Proof : We proceed by contradiction. If $D[j,y] > D[j,v] + D[v,y]$, then since $d(x,y) = D[x,y]$ for any pair of vertices x and y, we have $d(j,y) > d(j,v)+d(v,y)$ which contradicts the fact that $d(j,y)$ is the length of a minimal path from j to y. If $D[j,y] < D[j,v] + D[v,y]$, then $d(v,y) = D[v,y] > D[j,y]-D[j,v]$. Since by lemma 1 and by the hypothesis $D[x,y] = d(x,y)$ the path from v to y in $DESC(j)$ is a path in the graph of length $D[j,y] - D[j,v]$, this contradicts the fact that $d(v,y)$ is the length of a minimal path from v to y. Since it cannot be neither $D[j,y] > D[j,v] + D[v,y]$ nor $D[j,y] < D[j,v] + D[v,y]$, then $D[j,y] = D[j,v] + D[v,y]$. □

Lemma 3 *Assume that $D[x,y] = d(x,y)$ for any pair of vertices x and y in the graph before inserting an edge from i to j. Then given any pair of vertices u and v such that $d(u,v) \leq d(u,i)+1+d(j,v)$, for any vertex y contained in the subtree of* DESC(j) *rooted at v and for any vertex x contained in the subtree of* ANC(i) *rooted at u, the following two inequalities hold.*

$$d(u,y) \leq d(u,i) + 1 + d(j,y) \tag{5}$$

$$d(x,v) \leq d(x,i) + 1 + d(j,v) \tag{6}$$

Proof : We prove only inequality (5). The proof for inequality (6) is completely analogous and therefore it has been omitted. Consider any vertex y in the subtree of $DESC(j)$ rooted at v. Due to equation (2)

$$d(u,y) \leq d(u,v)+d(v,y) \leq d(u,i)+1+d(j,v)+d(v,y). \tag{7}$$

Since y is in the subtree of $DESC(j)$ rooted at v, then by lemma 2 $D[j,y] = D[j,v] + D[v,y]$. Since $D[x,y] = d(x,y)$ for any pair of vertices x and y in the graph, this is equivalent to say

$$d(j,y) = d(j,v) + d(v,y). \tag{8}$$

Combining (7) and (8) gives inequality (5). □

The correctness of our approach hinges on the following theorem.

Theorem 1 *After each operation, for the data structure the following equality holds*

$$D[x,y] = d(x,y) \qquad \forall x,y \in V. \tag{9}$$

That is, given any two vertices x and y, the entry (x,y) of the matrix D correctly gives the length of the minimal path from x to y.

Proof : We proceed by induction on the number of operations performed. Since *minpath* and *length* operations do not modify the data structure, we consider only *add* operations.

The basis of the induction is true since at the beginning the graph contains no edges, $D[x,y] = d(x,y) = +\infty$ for any two vertices x and y ($x \neq y$), and $D[x,x] = d(x,x) = 0$ for each vertex x.

Suppose equality (9) holds before the insertion of an edge (i,j). Let us denote by $D_{new}[x,y]$ and

by $d_{new}(x,y)$ respectively the values of $D[x,y]$ and $d(x,y)$ after inserting the edge (i,j). We want to prove that given $D[x,y] = d(x,y)$

$$D_{new}[x,y] = d_{new}(x,y) \qquad \forall x,y \in V. \quad (10)$$

The case $D_{new}[x,y] < d_{new}(x,y)$ cannot happen since by lines 4. and 8. in procedure *UpdateForward* and by the induction step :

$$\begin{aligned} D_{new}[x,y] &\geq \min\{D[x,y], D[x,i]+1+D[j,y]\} \\ &= \min\{d(x,y), d(x,i)+1+d(j,y)\} \\ &= d_{new}(x,y). \end{aligned}$$

Suppose now there exists a pair of vertices $x,y \in V$ such that $D_{new}[x,y] > d_{new}(x,y)$. This can happen if and only if x is an ancestor of i, y is a descendant of j and y was not inserted in the queue Q during the execution of the procedure *UpdateForward*(x,i,j,T). As a consequence, there exist two vertices u and v such that x is contained in the subtree of $ANC(i)$ rooted at u and y is contained in the subtree of $DESC(j)$ rooted at v, and for which procedure *UpdateForward*(u,i,j,T) pruned at v the tree considered because the condition on line 4. was not satisfied. Hence, $D[u,v] < D[u,i]+1+D[j,v]$ or, what is the same for the induction step, $d(u,v) < d(u,i)+1+d(j,v)$. Using this inequality yields

$$\begin{aligned} d(x,y) &\leq d(x,u)+d(u,v)+d(v,y) \\ &< d(x,u)+d(u,i)+1+d(j,v)+d(v,y). \end{aligned}$$

Because of the induction step

$$d(j,v)+d(v,y) = D[j,v]+D[v,y]$$

and by lemma 2

$$D[j,v]+D[v,y] = D[j,y].$$

Applying again the induction step to the last equality, we derive $d(j,v)+d(v,y) = d(j,y)$. Similarly, we obtain $d(x,u)+d(u,i) = d(x,i)$. Therefore, $d(x,y) < d(x,i)+1+d(j,y)$. Applying equality (3), we get $d_{new}(x,y) = \min\{d(x,y), d(x,i)+1+d(j,y)\} = d(x,y)$. Thus, $D_{new}[x,y] > d_{new}(x,y) = d(x,y) = D[x,y]$. This is clearly a contradiction since entries in the matrix D can only be decreased by procedure *UpdateForward*.

Since it cannot be either $D_{new}[x,y] < d_{new}(x,y)$ or $D_{new}[x,y] > d_{new}(x,y)$ for any pair of vertices, equality (10) must hold and therefore the theorem is proven. □

Before analyzing the time and space complexity of the data structure, we need some new terminology. During the insertion of an edge (i,j) in the graph G, for any x ancestor of i let us denote by $T(x)$ the set of vertices in $DESC(j)$ visited by *UpdateForward* while updating the tree $DESC(x)$. Note that as a special case, $T(x)$ might be empty. We say that a pair of vertices $< x,y >$ is *benign for edge* (i,j) if x is an ancestor of i, y is in $T(x)$, and before the insertion of (i,j) $D[x,y] > D[x,i]+1+D[j,y]$ (i.e., (i,j) introduces a new shorter path from x to y). Furthermore, we say that a pair of vertices $< x,y >$ is *malign for edge* (i,j) if x is an ancestor of i, y is in $T(x)$, y is visited while updating $DESC(x)$ and before the insertion of (i,j) $D[x,y] \leq D[x,i]+1+D[j,y]$ (i.e., (i,j) introduces no new shorter path from x to y). Denoting by $B_{i,j}$ the set of pairs which are benign for (i,j) and by $M_{i,j}$ the set of pairs which are malign for (i,j), the following fact is a consequence of the definition of malign pairs.

Fact 1 *The total time required by procedure* add *to insert an edge* (i,j) *is* $O(|B_{i,j}|+|M_{i,j}|)$.

We now characterize a property of benign and malign pairs.

Lemma 4 *For any pair* $< x,y >$ *in* $M_{i,j}$, *denote by* π_x *the path from* x *to* i *in* ANC(i) *and by* π_y *the path from* j *to* y *in* DESC(j). *For any vertex* $u \neq x$ *in* π_x *and for any vertex* $v \neq y$ *in* π_y *then the pairs* $< u,y >$ *and* $< x,v >$ *are benign for* (i,j).

Proof : We proceed by contradiction. Consider first a malign pair $< x,y >$ and assume that there exists a vertex $u \neq x$ in π_x such that also the pair $< u,y >$ is malign. This means that vertices in the subtree rooted at y in $DESC(j)$ are deleted from $T(u)$ on line 10. in procedure *UpdateForward*. Since $T(x) \subseteq T(u)$ because of line 11. in procedure *UpdateForward* then $y \notin T(x)$, which is clearly a contradiction. To prove the second part of the lemma, consider a malign pair $< x,y >$ and assume that there exists a vertex $v \neq y$ in π_y such that also the pair $< x,v >$ is malign. This means that the vertices in the subtree rooted at v in $DESC(j)$ are not visited while updating $DESC(x)$, because of line 10. in procedure *UpdateForward*. Therefore $y \notin T(x)$, again a contradiction. □

The following theorem proves the correctness and analyze the complexity of our approach.

Theorem 2 *There exists a data structure which correctly supports each* length *operation in $O(1)$ time and each* minpath *operation in $O(k)$ time, where k is the length of the traced (minimal) path. The total time required to maintain the data structure during the insertion of at most $O(n^2)$ edges is $O(n^3 \log n)$, where n is the number of vertices in the graph. The space complexity of the data structure is $O(n^2)$.*

Proof : The correctness of *length* and *add* operations derives from theorem 1, while the data structure correctly supports *minpath* operations because of theorem 1 and lemma 1. The space required is $O(n^2)$ since three $n \times n$ matrices and $2n$ trees, each of size at most n, are used. The time required by procedure *length*(x, y) is $O(1)$ since it accesses entry (x, y) of the distance matrix D. As previously noticed, procedure *minpath*(x, y) takes $O(k)$ to return a minimal path from x to y (if one exists), where k is the length of the achieved minimal path.

We have now to compute the total time required to maintain the data structure during any sequence S of edge insertions. Because of fact 1, this time is bounded by the total number of benign pairs plus the total number of malign pairs during the insertions of edges:

$$O(\sum_{(i,j)\in S} (|B_{i,j}| + |M_{i,j}|)). \quad (11)$$

Since for each pair $< x, y >$ in $B_{i,j}$ the entry (x, y) of the distance matrix D is decreased at least by 1 during the insertion of edge (i, j) and there can be at most $O(n^2)$ of such different pairs, the total time spent because of benign pairs is

$$\sum_{(i,j)\in S} |B_{i,j}| \leq O(n^3). \quad (12)$$

In order to bound the total time spent because of malign pairs, we use an amortization argument based upon the credit technique by Tarjan [28]. Our credit policy is the following:

When inserting edge (i, j), for each benign pair $< x, y > \neq < i, j >$ in $B_{i,j}$ give $2\frac{n}{d(x,y)-1}$ credits both to x and y, where $d(x, y) > 1$ is the minimum distance from x to y after the insertion of (i, j). Furthermore, give $2n$ credits both to i and j.

Denote by $\Phi_{i,j}$ the number of credits given during the insertion of edge (i, j). Since we give credits to a pair of vertices if and only if it is a benign pair, their distance strictly decreases. Therefore, the total number of credits given during any sequence S of edge insertions is

$$\sum_{(i,j)\in S} \Phi_{i,j} \leq O(\sum_{x\in V} \sum_{y\in V} \sum_{d=1}^{n} \frac{n}{d}) \leq O(n^3 \log n). \quad (13)$$

To complete our proof, we have to show that the credits given are enough to pay for the work done because of malign pairs. We will show something stronger, namely that all the credits $\Phi_{i,j}$ given to benign pairs while inserting edge (i, j) are enough to pay for the work done because of malign pairs during the insertion of the same edge (i, j).

Consider any malign pair $< x, y >$ in $M_{i,j}$. Denote by π_x the path in $ANC(i)$ from x to i and by π_y the path in $DESC(j)$ from j to y. Let $d_x \geq 0$ and $d_y \geq 0$ be respectively the lengths of π_x and π_y. Because of lemma 4, for any vertex $u \neq x$ in π_x and $v \neq y$ in π_y, the pairs $< u, y >$ and $< x, v >$ are benign for (i, j). The total number of credits given to vertex x because of vertices in π_y is

$$2 \sum_{h=d_x}^{d_x+d_y-1} \frac{n}{h} \geq 2n \log_e(1 + \frac{d_y}{d_x}).$$

Similarly, the total number of credits given to vertex y because of vertices in π_x is

$$2 \sum_{h=d_y}^{d_x+d_y-1} \frac{n}{h} \geq 2n \log_e(1 + \frac{d_x}{d_y}).$$

As a consequence, at least $2n \log_e((1 + d_y/d_x)(1 + d_x/d_y)) = 2n \log_e(2 + d_y/d_x + d_x/d_y) \geq 2n$ credits are given to both x and y during the insertion of edge (i, j). The credits given to both x and y can henceforth pay for the work due to all the malign pairs of the form $< x, . >$ and $< ., y >$ which are at most $2n$. Since this argument can be repeated for any malign pair, it proves that $\Phi_{i,j} \geq |M_{i,j}|$. Namely, all the credits given during operation *add*(i, j) are able to pay for the overall work due to malign pairs during the same operation.

To conclude our proof, we notice that because of equations (11), (12) and (13), the total time spent while updating the data structure during any sequence S of edge insertions is $O(\sum_{(i,j)\in S} |B_{i,j}| + |M_{i,j}|) \leq O(n^3) + \sum_{(i,j)\in S} \Phi_{i,j} \leq O(n^3 \log n)$. □

The bound given in theorem 2 is only a factor of $\log n$ away from the best possible bound for the problem of maintaining online a solution to the All Pairs Minimal Length Paths Problem. In fact we show now that there exist sequences of *add* operations which force as many as $\Omega(n^3)$ changes in the distance matrix D.

Consider the graph $G = (V, E)$ given in figure 2, with vertex set

$$V = \{s_1, s_2, \ldots, s_{\frac{n}{3}}, x_1, x_2, \ldots, x_{\frac{n}{3}}, t_1, t_2, \ldots, t_{\frac{n}{3}}\},$$

where n is a multiple of 3, and edges (s_i, x_1), $i = 1, 2, \ldots, \frac{n}{3}$, (x_i, x_{i+1}), $i = 2, \ldots, \frac{n}{3} - 1$, and $(x_{\frac{n}{3}}, t_i)$, $i = 1, 2, \ldots, \frac{n}{3}$.

If now the $(n/3 - 1)$ edges

$$(x_1, x_i), \quad i = 2, \ldots, \frac{n}{3},$$

are inserted into G in this order, the changes in the distance matrix D can be computed as follows. The insertion of edge (x_1, x_i), $2 \leq i \leq \frac{n}{3}$, creates new shorter paths from vertex s_j, $1 \leq j \leq \frac{n}{3}$, and from vertex x_1 to vertices x_h, $i \leq h \leq \frac{n}{3}$, and t_k, $1 \leq k \leq \frac{n}{3}$. This gives a total of $(n/3+1)(2n/3-i+1)$ changes required in the distance matrix D. After inserting all the edges, the total number of updates required in D is therefore

$$\sum_{i=2}^{n/3} (\frac{n}{3} + 1)(\frac{2}{3}n - i + 1),$$

which is $\Omega(n^3)$.

We notice that the worst-case per operation time complexity of our data structure is $O(n^2)$, and this is also the total number of updates which may be required in the matrix D during the insertion of a new edge.

4 Extensions

The results obtained for the online maintenance of a solution to the All Pairs Minimal Length Path Problem can be generalized in two directions.

First, the data structure can deal with integer edge costs in the range $[1 \ldots n^\alpha], \alpha < 1$ while updating a solution to the All Pairs Shortest Paths Problem during insertions of edges and edge cost decreases in a total of $O(n^{3+\alpha} \log n)$ worst-case time. The space required by the data structure is $O(n^2)$. Also in this case, there are sequences of edge insertions which require $\Omega(n^{3+\alpha})$ changes in the distance matrix D. Once again, our algorithm is only a factor of $\log n$ away from the best possible bound.

Second, a slight modification of the algorithm proposed is able to perform *maxpath* operations (i.e., queries about maximal length paths) on directed acyclic graphs where the insertions of new edges do not introduce cycles. This is not a significant restriction, since the longest path problem is known to be NP-complete for arbitrary graphs [13], while a polynomial time algorithm exists for directed acyclic graphs [21].

Acknowledgments

The second author is grateful to Zvi Galil and Moti Yung for stimulating discussions on this topic during the Spring 89 Seminars on Advanced Algorithms held at Columbia University.

References

[1] R. Agrawal, A. Borgida, and H. V. Jagadish, Efficient management of transitive relationships in large data and knowledge bases, *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, 1989.

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

[3] G. Ausiello, A. Marchetti Spaccamela, and U. Nanni, Dynamic maintenance of paths and path expressions in graphs, *Proc. 1st Int. Joint Conf. ISSAC 88 (Int. Symp. on Symbolic and Algebraic Computation) and AAECC 6 (6th Int. Conf. on Applied Algebra, Algebraic Algorithms and Error Correcting Codes)*, 1988.

[4] M. Burke, and B. G. Ryder, Incremental iterative data flow analysis algorithms, Technical Report LCSR-TR-96, Department of Computer Science, Rutgers University, 1987.

[5] M. D. Carrol and B. G. Ryder, Incremental data flow analysis via dominator and attribute updates, *Proc. 15th Annual ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages*, 1988, 274–284.

[6] F. Chin and D. Houk, Algorithms for updating minimum spanning trees, *J. Comput. System. Sci.* 16 (1978), 333-344.

[7] D. Eppstein, G. F. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook, M. Yung, Maintenance of a minimum spanning forest in a dynamic planar graph, These Proceedings.

[8] S. Even, and H. Gazit, Updating distances in dynamic graphs, *Methods of Operations Research* 49 (1985), 371–387.

[9] S. Even, and Y. Shiloach, An on-line edge deletion problem, *J. Assoc. Comput. Mach.* 28 (1981), 1–4.

[10] G. N. Frederickson, Data structures for on-line updating of minimum spanning trees, *SIAM J. Comput.* 14 (1985), 781–798.

[11] G. N. Frederickson and M. A. Srinivas, On-line updating of degree-constrained minimum spanning trees, *Proc. 22nd Annual Allerton Conference on Communication, Control and Computing*, 1984.

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

[13] M. R. Garey, D. S. Johnson, *Computers and intractability: a guide to the theory of NP-Completeness*, W. H. Freeman, 1979.

[14] D. Harel, On-line maintenance of the connected components of dynamic graphs, Unpublished manuscript, 1982.

[15] N. Horspool, Incremental generation of LR parsers, Technical Report, Department of Computer Science, University of Victoria, 1988.

[16] T. Ibaraki, and N. Katoh, On-line computation of transitive closure for graphs, *Inform. Process. Lett.* 16 (1983), 95–97.

[17] G. F. Italiano, Amortized efficiency of a path retrieval data structure, *Theoret. Comput. Sci.* 48 (1986), 273–281.

[18] G. F. Italiano, Finding paths and deleting edges in directed acyclic graphs, *Inform. Process. Lett.* 28 (1988), 5–11.

[19] H. V. Jagadish, A compression technique to materialize transitive closure, *ACM Trans. on Database Systems*, to appear.

[20] J. A. La Poutré, and J. van Leeuwen, Maintenance of transitive closure and transitive reduction of graphs, *Proc. Workshop on Graph-Theoretic Concepts in Computer Science*, 1988, 106–120.

[21] E. L. Lawler, *Combinatorial optimization: networks and matroids*, Holt, Rinehart and Winston, 1976.

[22] F. P. Preparata, and R. Tamassia, Fully dynamic techniques for point location and transitive closure in planar structures, *Proc. 29th Annual Symp. on Foundations of Computer Science*, 1988, 558–567.

[23] J. H. Reif, A topological approach to dynamic graph connectivity, *Inform. Process. Lett.* 25 (1987), 65–70.

[24] H. Rohnert, A dynamization of the all pairs least cost path problem, *Proc. 2nd Annual Symp. on Theoretical Aspects of Computer Science* , 1985, 279–286.

[25] D. D. Sleator, and R. E. Tarjan, A data structure for dynamic trees, *J. Comput. System Sci.* 24 (1983), 362–381.

[26] P. M. Spira and A. Pan, On finding and updating spanning trees and shortest paths, *SIAM J. Comput.* 4 (1975), 375–380.

[27] R. E. Tarjan, *Data structures and network algorithms*, CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 44, SIAM, 1983.

[28] R. E. Tarjan, Amortized computational complexity, *SIAM J. Alg. Disc. Meth.* 6 (1985), 306–318.

[29] D. M. Yellin, A dynamic transitive closure algorithm, Research Report, IBM Research Division, T. J. Watson Research Center, 1988.

[30] D. M. Yellin, and R. Strom, INC: a language for incremental computations, *Proc. ACM SIGPLAN '88 Conf. on Programming Language Design and Implementation*, 1988, 115–124.

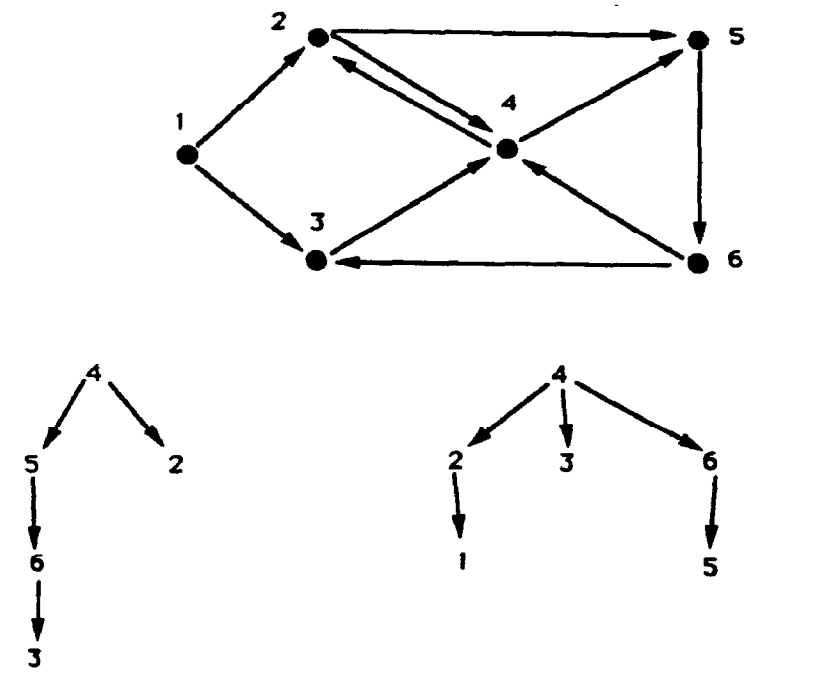

Figure 1

(a) A graph G=(V,E);
(b) Tree of forward paths of minimal length rooted at vertex 4;
(c) Tree of backward paths of minimal length rooted at vertex 4.

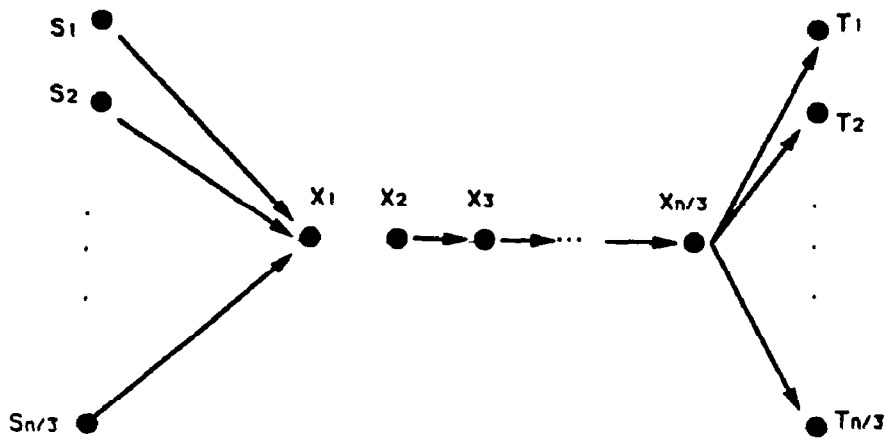

Figure 2