Incremental Maintenance of Shortest Distance and Transitive Closure in First-Order Logic and SQL

CHAOYI PANG
CSIRO ICT Center/E-Health Research Center
GUOZHU DONG
Wright State University
and
KOTAGIRI RAMAMOHANARAO
University of Melbourne

Given a database, the view maintenance problem is concerned with the efficient computation of the new contents of a given view when updates to the database happen. We consider the view maintenance problem for the situation when the database contains a weighted graph and the view is either the transitive closure or the answer to the all-pairs shortest-distance problem (*APSD*). We give incremental algorithms for *APSD*, which support both edge insertions and deletions. For transitive closure, the algorithm is applicable to a more general class of graphs than those previously explored. Our algorithms use first-order queries, along with addition (+) and less-than (<) operations (*FO*(+, <)); they store $O(n^2)$ number of tuples, where n is the number of vertices, and have AC^0 data complexity for integer weights. Since *FO*(+, <) is a sublanguage of SQL and is supported by almost all current database systems, our maintenance algorithms are more appropriate for database applications than nondatabase query types of maintenance algorithms.

Categories and Subject Descriptors: D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement; G.2.2 [**Discrete Mathematics**]: Graph Theory; H.3.5 [**Information Storage and Retrieval**]: Online Information Services; H.2.8 [**Database Management**]: Database Applications; F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General; H.4.0 [**Information Systems Applications**]: General

Part of the results in this article appeared as Incremental *FO*(+; <) maintenance of all-pairs shortest paths for undirected graphs after insertions and deletions. In *Proceedings of the 7th International Conference on Database Theory* (Jerusalem, Israel, January 1999).

Authors' addresses: C. Pang, CSIRO ICT Center, cHRC, Level 20, 300 Adelaide Street, Brisbane, Queensland 4000, Australia; email: chaoyi.pang@csiro.au; G. Dong, Department of Computer Science and Engineering, Wright State University, 3640 Colonel Glenn Highway, Dayton, Ohio 45435-0001; email: (gdong@cs.wright.edu). K. Ramamohanarao, Department of Computer Science and Software Engineering, The University of Melbourne, 4th floor, ICT Building, 111 Barry Street, Carlton Victoria 3053, Australia; email: rao@cs.mu.oz.au.

General Terms: Algorithms, Languages, Performance

Additional Key Words and Phrases: Database view, first-order logic, incremental algorithm, maintenance, SQL

1. INTRODUCTION

Finding shortest-path information in a graph is one of the most commonly encountered problems in the study of transportation and communication networks and has applications in routing, scheduling, and computation of network flows, in the context of document formatting and so on. Since changes to such networks, caused by link failures or new links being added to the service, occur often, the incremental shortest-path problem is relevant and important for these applications.

Problem statement: Given a database, the view maintenance problem is concerned with the efficient computation of the new contents of a given view when updates to the database occur. We consider the view maintenance problem for the situation when the database contains a weighted graph and the view is either the transitive closure or the answer to the all-pairs shortest distance (*APSD*) problem.

Contributions:

(1) We give algorithms for incrementally maintaining the all-pairs shortest distance view in graphs after arc insertions and deletions. We do this by first considering the situation where arcs have positive distance ($APSD_{>0}$), and then extending to the situation where arcs have non-negative distance ($APSD_{\geq 0}$).[1]

(2) We show that the algorithms for maintaining the *APSD* view can also maintain transitive closure for a larger class of graphs, which have not been explored before.

While our algorithms only directly handle single arc insertions and deletions, they can also handle distance-modification on an arc, since each such change is equivalent to deleting an arc and then adding the same arc with the new distance.

Our maintenance algorithms only use first-order queries with addition "+" and less-than "<" operations, namely $FO(+, <)$. Our results are useful as follows:

—Since $FO(+, <)$ is supported by almost all current database systems, our incremental maintenance algorithms are more appropriate for database applications than non-*FO* incremental algorithms. Our algorithms can be executed on any relational database system, without the need for embedding in a host language. For example, in the Oracle database system, our algorithms

[1]The restriction of "positive distance" implies that each shortest walk between two nodes is also a shortest path between the same two nodes; the presence of negative-distance arcs can imply that shortest walks may not exist.

can be implemented in SQL and directly processed by an SQL statement executor on the server without requiring going through the PL/SQL engine. This avoids the overhead of going through extra processing engines.

In relational databases, the optimization of recursive queries is significantly harder than that of nonrecursive queries. The absence of recursive mechanisms in our algorithms (which use simple relational data structures and are written in relational queries) could make additional optimizations possible, as a large range of query optimization techniques can be applied.

—Theoretically, the time complexity of our algorithms is dominated by "+" and "<" operations with an upper bound of NC (the class of problems computable in polylogarithmic time with a polynomially many processors). When the distance of each arc in the graph is an integer, our algorithms have AC^0 data complexity [Grumbach and Su 1995]. (AC^0 is the class of problems that can be solved using polynomially many processors in constant time). To the best of our knowledge, no AC^0 incremental algorithms (even for non-*FO* algorithms) for the *APSD* problem, supporting both arc deletions and arc insertions, on general undirected graphs (and digraphs with certain properties) have been previously reported. Our algorithms are the first of this kind.

—From a practical perspective, the incremental algorithms are efficient, since each involves a couple of joins. The joins can be evaluated efficiently, by using indexing and the deleted edge to identify the tuples needed in the joins.

All our algorithms employ one common technique as the basis for the maintenance results for an arc deletion: They first delete a set of tuples whose existence in the relation depends on the deleted arc; this step may delete more than necessary. Then they correct the wrong deletions by doing the relational natural join of the result of the first step with the modified graph several times. This generalizes the technique used in Dong and Pang [1997] and Dong and Su [2000] for the maintenance of the transitive closure of acyclic digraphs and undirected graphs.

Organization: We define necessary notations in Section 2. Section 3 gives our incremental algorithm for the distance problem in undirected graphs. Section 4 considers the *APSD* algorithms in digraphs. Section 5 discusses applications of our results to the maintenance of transitive closure and shortest paths. Section 6 discusses complexity issues. Section 7 is devoted to related work. Section 8 gives some concluding remarks.

2. PRELIMINARIES

In this section we present some standard terminology of graph theory, together with some necessary new notations and definitions. We assume the reader is familiar with the first-order logic or the relational calculus.

A *directed graph* (*digraph*) is a pair $G = (V, E)$, where V is a finite set of nodes and $E \subseteq V \times V$ is a set of ordered pairs or *arcs*. When multiple graphs are present, we will use $V(G)$ and $E(G)$ to represent V and E of the underlying graph G. A digraph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.

A sequence $u_0u_1 \ldots u_n$ $(n > 0)$ of nodes in G is a *walk* from u_0 to u_n if (u_{i-1}, u_i) is in E for each $i \in [1..n]$; the sequence is a *path* if it is a walk and $u_i \neq u_j$ whenever $0 \leq i < j \leq n$. We say $u_iu_{i+1} \ldots u_{j-1}u_j$ is a *subpath* of $u_0u_1 \ldots u_n$ for all $0 \leq i < j \leq n$; when $i \neq 0$ or $j \neq n$, we say the subpath is *strict*. The sequence $u_0u_1 \ldots u_n$ $(n > 0)$ is a *cycle* if $u_0 = u_n$ and $u_i \neq u_j$ for all $0 \leq i < j \leq n$ such that $(i, j) \neq (0, n)$. An arc on a cycle is called a cyclic arc. G is *acyclic* if it contains no cycles. Let TC_G be the transitive closure of G, i.e., $TC_G = \{(x, y) \mid$ there exists a path from x to y in $G\}$. Since arcs of the form (u, u) in a digraph contribute only in a trivial way to its transitive closure, we will only consider digraphs without such arcs. A *strongly connected component* (*SCC*) of G is a maximal subset X of V such that $(x, y) \in TC_G$ for each pair of nodes $x, y \in X$.

We will consider weighted graphs G, where each arc e has a distance (denoted as $dist(e)$). A distance can either be an integer or a floating point number. $dist(e)$ may represent a cost of some type associated with the arc e.

The *distance* of a path or a walk p, denoted as $dist(p)$, is the sum of $dist(e)$ over all arcs e on p. The *length* of p, denoted as $length(p)$, is the number of arcs on p. The *shortest-distance* (*shortest-length*, respectively) path (or walk) from node x to node y is the path (or walk) from x to y with minimum distance (length, respectively). A shortest path is a path that has the shortest distance.

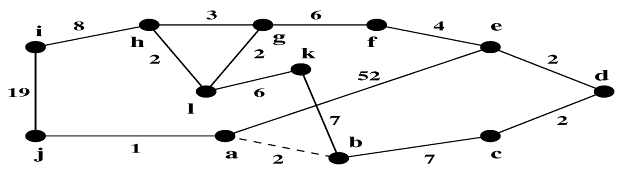
We now introduce some relations for shortest-distance paths, SP_G, and for shortest-length paths, SPD_G, for digraph G; these relations will play a key role in our maintenance algorithms. SP_G is defined as: $SP_G(x, y, d)$ is true if and only if the shortest path from x to y in G has distance d. We define $SP_G(u, u, 0)$ for each node u of G and $SP_G(x, y, \infty)$ if there is no path from x to y. SPD_G is defined as: $SPD_G(x, y, l, d)$ is true if and only if (i) the shortest paths from node x to node y in G have distance d and (ii) one shortest path from x to y in G has length l and (iii) no shortest path from x to y in G has length less than l. When there is no path from x to y in G, we assume $SPD_G(x, y, \infty, \infty)$ holds.

The *all-pair shortest-distance* (*APSD*) problem is to find the shortest distance between every pair of nodes in a digraph. A restricted form of the *APSD* problem is the $APSD_{>0}$ ($APSD_{\geq 0}$, respectively) problem which restricts each arc in the graph to a positive (non-negative, respectively) distance.

An *undirected graph* G is a pair (V, E) such that E is symmetric (in the sense that $(b, a) \in E$ holds whenever $(a, b) \in E$ holds.) For undirected graphs, a node (an arc, respectively) is called a vertex (an edge, respectively); concepts such as *walk*, *path*, and transitive closure can be defined in a similar way as in a digraph.

While paths and walks in undirected graphs are not ordered in general, we sometimes need to emphasize an ordering of the vertices. For example, we will say a path p is *from vertex u to vertex v* to imply that the vertices of p are ordered in the form of $u \ldots v$, and we say a path p from u to v *goes through edge $e = (a, b)$ in the order of ab* to imply that p has the form of $p = u \ldots ab \ldots v$.

Example 2.1. In the graph G shown in Figure 1(1), "*ihlghlk*" is a walk; "*kbcd*" and "*klgfed*" are paths; "*kbcd*" is a shortest path between k and d; the shortest distance between k and d is 16.



(1)

	a	b	c	d	e	f	g	h	i	j	k	l
a		43	45	43	41	37	31	28	20	1	36	30
b			7	9	11	15	15	15	23	42	7	13
c				2	4	8	14	17	25	44	14	16
d					2	6	12	15	23	42	16	14
e						4	10	13	21	40	18	12
f							6	9	17	36	14	8
g								3	11	30	8	2
h									8	27	8	2
i										19	16	10
j											35	29
k												6

(2)

	a	b	c	d	e	f	g	h	i	j	k	l
a		2	9	11	13	17	17	17	20	1	9	15
b			7	9	11	15	15	15	22	3	7	13
c				2	4	8	14	17	25	10	14	16
d					2	6	12	15	23	12	16	14
e						4	10	13	21	14	18	12
f							6	9	17	18	14	8
g								3	11	18	8	2
h									8	18	8	2
i										19	16	10
j											10	16
k												6

(3)

Fig. 1. Shortest distances.

Most of our incremental maintenance algorithms will exploit the following property, which relates the shortest paths with the auxiliary relation to be maintained. A digraph G is said to be *sp-bound* for arc e under *SP* if no two consecutive shortest paths $p_{xy} = xu_1 \dots u_i y$ and $p_{yz} = yv_1 \dots v_j z$ of *SP* share the arc e. A digraph G is *sp-bound* under *SP* if and only if it is *sp*-bound for each arc $e \in G$ under *SP*. The notion of *sp*-bound can be extended to other relations such as *TC* and *SPD*. This notion allows us to handle many types of graphs: An acyclic digraph G is *sp*-bound under TC_G [Dong and Pang 1997]. Moreover, as will be seen later, (1) each undirected graph G is *sp*-bound under SP_G (but not TC_G) if each edge of G has a positive distance. (2) Each undirected graph G is *sp*-bound under SPD_G (but not SP_G) if each edge of G has a non-negative distance. (3) There exist special *sp*-bound non-acyclic digraphs under SP_G (or SPD_G).

We will use $G(a, b)$ or $G(e)$ to denote "arc $e = (a, b)$ is in G". We use G_{+e} to denote the graph that results from inserting e to G, and use G_{-e} to denote the graph that results from deleting e from G.

We use $\min\{l_1, l_2, \dots, l_k\}$ to denote the minimum among the numbers in the set $\{l_1, l_2, \dots, l_k\}$. Since the min operator is only applied to a bounded set at a time in our algorithms, it can be replaced by expressions using *less-than* ($<$). For example, $min\{a, b, c\}$ can be expressed as $[((a \leq b) \wedge (a \leq c)) \rightarrow (min\{a, b, c\} = a)] \vee [((b \leq a) \wedge (b \leq c)) \rightarrow (min\{a, b, c\} = b)] \vee [(c \leq a) \wedge (c \leq b) \rightarrow (min\{a, b, c\} = c)]$.

For each binary relation R, we let $\widehat{R} = R \cup \{(u, u) | u \text{ is a vertex in } R\}$.

3. THE *APSD* PROBLEM IN UNDIRECTED GRAPHS

We now discuss the $APSD_{>0}$ ($APSD_{\geq 0}$, respectively) problem, for undirected graphs where each edge has a positive (non-negative, respectively) distance.

3.1 Incremental Algorithm for The $APSD_{>0}$ Problem

In this section, for an undirected graph G where each edge has a positive distance, we give incremental maintenance algorithms to find the shortest distance between every pair of vertices after single-edge insertion and deletion. The technique for edge insertion is fairly simple and similar techniques have been used previously. The technique for edge deletion is new and much more involved.

The algorithms in this section use the following lemma.

LEMMA 3.1. *Let G be an undirected graph.*

(1) *If path $p = uw_1 \ldots w_k v$ ($u = w_0$, $v = w_{k+1}$) is a shortest path between u and v, then any strict subpath $q = w_i \ldots w_j$ of p for $0 \leq i < j \leq k+1$ is a shortest path between w_i and w_j.*

(2) *If $dist(g) > 0$ for each edge $g \in G$, then for each non-path walk between two vertices there exists a shorter path between the same vertices.*

PROOF. (1) Otherwise, let $q = w_i \ldots w_j$ be a strict subpath of p which is not a shortest path between w_i and w_j. Let a shortest path between w_i and w_j be $q_1 = w_i v_1 \ldots v_h w_j$. Then $dist(q_1) < dist(q)$. Let $q_2 = uw_1..w_i v_1..v_h w_j..w_k v$. It follows that $dist(q_2) < dist(p)$. That is contradictory to p being a shortest path, and so (1) holds.

(2) Let $q = u_0 u_1 \ldots u_k$ be a non-path walk. Then there exist $0 \leq i < j \leq k$ such that $u_i = u_j$ and the distance of subwalk $u_i \ldots u_j$ of q is positive. Now, $q' = u_0..u_i u_{j+1}..u_k$ has a shorter distance than q. Repeating this process if q' is not a path, we will ultimately obtain a path from u_0 to u_k that is shorter than q. □

The converse of Lemma 3.1(1) does not hold. For example, let G be $\{(a, b, 1), (b, c, 1), (a, c, 1)\}$. Then $SP_G(a, b, 1) \wedge SP_G(b, c, 1) \wedge SP_G(a, c, 1)$ holds. The path $a\ b\ c$ is not a shortest path in G, but the subpaths $a\ b$ and $b\ c$ are shortest paths.

3.1.1 *Inserting Edge e in G.*

The technique for edge insertion is simple. Similar techniques have been used in many recursive algorithms [Shmueli and Itai 1984; Even and Gazit 1985; Blakeley et al. 1986; La Poutre and van Leeuwen 1987; Buchsbaum et al. 1990; Lin and Zhang 1990; Ausiello et al. 1992; Harrison and Dietrich 1992; Ramalingam 1996] and in the first-order algorithms of Dong and Topor [1992], and Patnaik and Immerman [1994].

THEOREM 3.2. *We can construct a formula ψ of $FO(+, <)$ from SP_G and $e = (a, b)$ such that $SP_{G_{+e}} \equiv \psi$, provided that G_{+e} is an undirected graph with $dist(g) > 0$ for each edge g.*

PROOF. Let

$$\begin{aligned}\psi(x, y, d) \equiv\ & (\exists d_0, d_1, d_2, d_3, d_4)\,(SP_G(x, a, d_1) \wedge SP_G(b, y, d_2) \vee \\ & SP_G(x, b, d_3) \wedge SP_G(a, y, d_4)) \wedge SP_G(x, y, d_0) \wedge \\ & (d = min\{dist(e) + d_1 + d_2, dist(e) + d_3 + d_4, d_0\}).\end{aligned}$$

Observe that the *min* is applied to a set of three numbers. As noted in Section 2, it can be replaced by formulas using $<$. Therefore, $\psi \in FO(+, <)$.

We need to show $SP_{G_{+e}} \equiv \psi$. To show that $SP_{G_{+e}} \Rightarrow \psi$, let (x, y, d) be any given tuple of $SP_{G_{+e}}$, Two cases arise: (1) Edge e is not on any shortest path between x and y. Then $SP_G(x, y, d)$ holds and so $\psi(x, y, d)$ is true. (2) Edge e is on a shortest path between x and y. Let us assume the shortest path has the form of $x \dots w \dots ab \dots v \dots y$; the situation when the shortest path has the form of $x \dots w \dots ba \dots v \dots y$ is similar. By Lemma 3.1, since path $x \dots w \dots ab \dots v \dots y$ is the shortest path in G_{+e}, paths $q_1 = x \dots w \dots a$ and $q_2 = b \dots v \dots y$ are shortest paths and they clearly do not pass through edge e. Therefore, $d = dist(q_1) + dist(q_2) + dist(e)$ and $\psi(x, y, d)$ is true.

To show that $\psi \Rightarrow SP_{G_{+e}}$, let (x, y, d) be a tuple such that $\psi(x, y, d)$ is true. Since ψ is selecting the smallest distance d of the shortest paths from x to y that passes through e and the shortest paths from x to y that do not pass through e. this d is clearly the shortest distance from x to y. Hence $SP_{G_{+e}}(x, y, d)$ holds. □

The SQL algorithm for edge insertion is presented in Table 1.

Example 3.3. Consider the graph G shown in Figure 1(1):

$$G = \{(a, j, 1), (a, e, 52), (b, c, 7), (b, k, 7), (c, d, 2), (d, e, 2), (e, f, 4), (f, g, 6), (g, h, 3), (g, l, 2), (h, i, 8), (h, l, 2), (i, j, 19), (k, l, 6)\}.$$

The shortest paths between vertices of G are given in Figure 1(2). Consider inserting the edge $\varepsilon = (a, b, 2)$ $(dist(\varepsilon) = 2)$. The shortest paths of $G_{+\varepsilon}$ computed by the algorithm are shown in Figure 1(3). The shaded numbers of Figure 1(2) are the ones that are modified after insertion. For instance, $SP_{G_{+\varepsilon}}(a, c, 9)$ holds since $9 = min\{2 + 7, 45\}$, $G(a, b, 2)$, $SP_G(b, c, 7)$ and $SP_G(a, c, 45)$.

3.1.2 *Deleting Edge e from G.* The main result in the section is for edge deletion in undirected graphs. Our maintenance algorithm uses one property (Lemma 3.5): any shortest path of G_{-e} can be regenerated through the join of two shortest paths, which do not pass through edge e, with one edge in G_{-e}.

THEOREM 3.4. *We can construct a formula ψ of $FO(+, <)$ from SP_G and e such that $SP_{G_{-e}} \equiv \psi$, provided that G is an undirected graph where each edge has a positive distance.*

To construct ψ, we need some lemmas, including the following key lemma, which proves that G is *sp*-bound under SP_G.

LEMMA 3.5. *Let G be a graph where each edge has a positive distance. Suppose there exists a shortest path $p_1 = u \dots ab \dots v$ from vertex u to vertex v of G, which goes through edge $e = (a, b)$ in the order of ab (as shown in Figure 2(I)). Let $p_2 = uw_1 \dots w_k v$ ($u = w_0$ and $v = w_{k+1}$) be a shortest path between u and v in G_{-e}.*

a. If a shortest path of G from u to w_i (w_i is on p_2, $1 \le i \le k+1$) goes through edge e, then it goes through edge e in the order of ab.

b. Similarly, if a shortest path of G from w_i (w_i is on p_2, $0 \le i \le k$) to v goes through edge e, then it goes through edge e in the order of ab.

Table I. SQL Queries for Maintenance after Insertion

```
% TABLE Graph(Start,Tail,Dist): A tuple (s,t,d) is in Graph if the distance
% of edge (s,t) is d; for each vertex x, the tuple (x,x,0) is also in Graph.
% TABLE ShortDis(Start,Tail,Dist): A tuple (s,t,d) of ShortDis means that
% the shortest distance between vertex s and t is d; for each vertex x,
% (x,x,0) is in ShortDis.
% TABLE Susp: tuples which may need to be modified when graph G changes.
% TABLE Graph, ShortDis and Susp are symmetric.
%
% When inserting edge e=(a,b) with dist(e)>0, the shortest distances formed
% by the following paths could be affected:
%   (1) any path from x through (a,b) to y; such paths are stored in Susp1
%   (2) any path from x through (b,a) to y; such paths are stored in Susp2
    INSERT INTO Susp1(Start,Tail,Dist)
    SELECT X.Start, Y.Tail, dist(e)+X.Dist+Y.Dist
    FROM ShortDis X, ShortDis Y
    WHERE X.Tail=a AND Y.Start=b AND
        ((dist(e)+X.Dist+Y.Dist <
            (SELECT Z.Dist  FROM ShortDis Z
             WHERE Z.Start=X.Start AND Z.Tail=Y.Tail)
         OR (NOT EXISTS (SELECT *  FROM ShortDis Z
             WHERE Z.Start=X.Start AND Z.Tail=Y.Tail));
    INSERT INTO Susp2(Start,Tail,Dist)
    SELECT X.Start, Y.Tail, dist(e)+X.Dist+Y.Dist
    FROM ShortDis X, ShortDis Y
    WHERE X.Tail=b AND Y.Start=a AND
        ((dist(e)+X.Dist+Y.Dist <
            (SELECT Z.Dist  FROM ShortDis Z
             WHERE Z.Start=X.Start AND Z.Tail=Y.Tail)
         OR (NOT EXISTS (SELECT * FROM ShortDis Z
             WHERE Z.Start=X.Start AND Z.Tail=Y.Tail));

% TABLE Susp = UNION of Susp1 and Susp2. For each (start,tail,_) in Susp,
% there is only one dist such that (start,tail,dist) in Susp (see Lemmas
% in Section 3.1)
    INSERT INTO Susp(Start,Tail,Dist)
    SELECT *
    FROM (Susp1 UNION Susp2);

% Remove tuples of ShortDis that no longer express shortest distance.
    DELETE FROM ShortDis (Start,Tail,Dist) WHERE (Start,Tail) IN
      (SELECT Start,Tail  FROM Susp);

% Re-insert the short distance triples with new distances to get result:
INSERT INTO ShortDis (Start,Tail,Dist) SELECT * FROM Susp;
```

c. *There exists a vertex w_i ($1 \leq i \leq k+1$) such that for subpaths $p_{21} = uw_1 \dots w_{i-1}$ and $p_{22} = w_i \dots v$ of p_2, no shortest path of G among vertices $u, w_1, \dots, w_{i-1}$, and no shortest path of G among vertices $w_i, w_{i+1}, \dots, v$, goes through edge e.*

Proof. Since (a) and (b) are dual, we will prove (a) and (c) only.

For (a), we show that, for any vertex w_i on the path p_2, if there exists a shortest path between u and w_i that goes through e then the shortest path

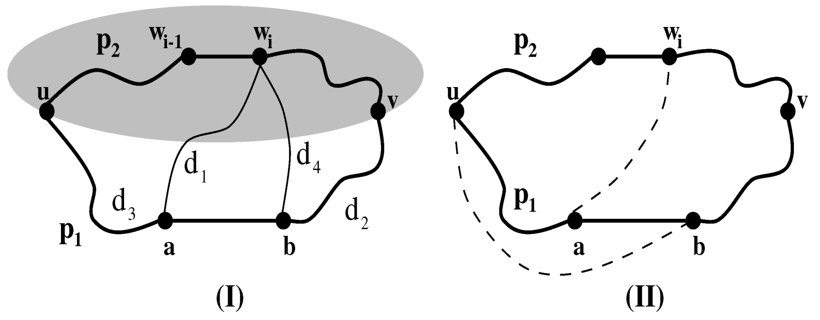


Fig. 2. Regenerating shortest paths after an edge deletion.

cannot go through e in the order of $u \ldots ba \ldots w_i$; the situation of Figure 2(II) cannot happen. Assume otherwise. Then there exists a shortest path between u and w_i (for some $0 < i \leq k+1$) that has the form of $u \ldots ba \ldots w_i$ (expressed in dashed line).

Let us denote the path, shown by the dashed line (the lower solid line, respectively), between u and b by $path_{dash}(u, b)$ ($path_{solid}(u, b)$, respectively), and denote the path, shown by the dashed line, between a and w_i by $path_{dash}(a, w_i)$.

By the hypotheses and Lemma 3.1, the following holds:

$$\begin{aligned}&SP_G(a, b, dist(e)) \wedge SP_G(u, a, dist(path_{dash}(u, b)) + dist(e))\\ &\quad\wedge SP_G(u, a, dist(path_{solid}(u, b)) - dist(e)) \wedge SP_G(u, b, dist(path_{solid}(u, b)))\\ &\quad\wedge SP_G(u, b, dist(path_{dash}(u, b))).\end{aligned}$$

The formula says the following: the shortest paths between a and b have distance $dist(e)$; the shortest paths between u and a have a distance of $dist(path_{dash}(u, b)) + dist(e)$ and of $dist(path_{solid}(u, b)) - dist(e)$; the shortest paths between u and b have a distance of $dist(path_{solid}(u, b))$ and of $dist(path_{dash}(u, b))$. Hence

$$\begin{aligned}dist(path_{dash}(u, b)) + dist(e) &= dist(path_{solid}(u, b)) - dist(e)), \textit{ and}\\ dist(path_{solid}(u, b)) &= dist(path_{dash}(u, b)).\end{aligned}$$

It follows that $dist(e) = 0$, contradictory to the positive edge-distance assumption.

For (c), we illustrate the situation with Figure 2(I). Let i be the smallest integer among $[1 \ldots k+1]$ such that a shortest path between u and w_i uses edge e and *every* shortest path between u and w_{i-1} does not use edge e. Since no shortest path from u to u uses e and edge e is on a shortest path between u and v, i exists.

By the way vertex w_i is chosen, subpath $p_{21} = uw_1 \ldots w_{i-1}$ of p_2 is the shortest path between u and w_{i-1} both in G and G_{-e}, and there is no shortest path of G among vertices $u, w_1, \ldots, w_{i-1}$ using edge e. By Lemma 3.1 and Lemma 3.5(a),

$$SP_G(u, w_i, d_3 + d_4 + dist(e)) \wedge SP_G(u, a, d_3) \wedge SP_G(b, w_i, d_4) \qquad (1)$$

holds. We now prove that e does not lie on any shortest paths among vertices $w_i, \ldots, w_k, v$ in G. Otherwise, according to Lemma 3.1 and Lemma 3.5(b),

$$SP_G(w_i, v, d_1 + d_2 + dist(e)) \wedge SP_G(w_i, a, d_1) \wedge SP_G(b, v, d_2) \tag{2}$$

holds. From (1) and (2), since a shortest path between two vertices is not longer than any other paths between the same vertices, we have

$$d_3 + d_4 + dist(e) \leq d_3 + d_1, \qquad and \qquad d_1 + d_2 + dist(e) \leq d_4 + d_2.$$

It follows that $dist(e) \leq 0$, contradictory to the positive edge-distance assumption. This implies that edge e does not lie on any shortest path among $w_i, \ldots, w_k, v$ and therefore, no shortest path among $w_i, \ldots, w_k, v$ goes through edge e. □

Example 3.6. We now illustrate the last lemma by considering undirected graph $G_{+\varepsilon}$ in Example 3.3. The shortest path between vertex j ($u = j$ in the Lemma) and vertex d ($v = d$ in the Lemma) is $p_1 = j\ a\ b\ c\ d$. Path $p_2 = j\ i\ h\ g\ f\ e\ d$ is the shortest path between j and d not going through edge ε. In Lemma 3.5(c), we can let the chosen vertex be h ($w_i = h$ in the Lemma). Therefore, $path(j, i) = j\ i$ is the shortest path between j and i and $path(h, d) = h\ g\ f\ e\ d$ is the shortest path between h and d (no shortest paths among i and j and among h, g, f, e, d go through edge (a, b)) according to Lemma 3.5(c).

Lemma 3.5(a) and (b) imply an "ordering" on the vertices of G. They suggest that if one shortest path from an earlier vertex to a later vertex on path p_2 goes through edge e in one "direction" then every shortest path from any earlier vertex to any later vertex on p_2 goes through edge e in the same "direction", so long as the shortest path passes through edge e.

Given an undirected graph G where each edge has a positive distance and an edge $e = (a, b) \in G$, the tuples of SP_G can be classified into two kinds: Γ, the set of tuples (x, y, d) such that there is a shortest path through edge e between x and y in G, and $\Omega = SP_G - \Gamma$. Lemma 3.5 implies that each shortest path of G_{-e} that is not in Ω can be regenerated, through at most two join operations, by those shortest paths that do not go through edge e in G (i.e., in Ω). In order to give a full description, we introduce some formulas.

For edge $e \in G$, let $\Gamma_e^G(x, y, d)$ be a formula stating the fact that there is a shortest path between x and y in G using edge $e = (a, b)$, that is,

$$\begin{aligned}\Gamma_e^G(x, y, d) \ \equiv\ & (\exists d_0, d_1, d_2, d_3, d_4)\, SP_G(x, y, d) \wedge [SP_G(x, a, d_1) \\ & \wedge SP_G(b, y, d_2) \wedge (d = d_1 + d_2 + dist(e)) \vee SP_G(x, b, d_3) \\ & \wedge SP_G(a, y, d_4) \wedge (d = d_3 + d_4 + dist(e))].\end{aligned}$$

Let $\Omega_e^G(x, y, d)$ be defined by

$$\Omega_e^G(x, y, d) \ \equiv\ SP_G(x, y, d) \wedge \neg\Gamma_e^G(x, y, d).$$

Then $\Omega_e^G(x, y, d)$ states that the shortest paths between x and y in G have distance of d but no such shortest paths use edge e. With the newly defined formulas, Lemma 3.5(c) can be rewritten into the following form:

Lemma 3.5(d): Let G be a graph where $dist(g) > 0$ for each edge g. Suppose there exists a shortest path $p_1 = u \ldots ab \ldots v$ from vertex u to vertex v of G that goes through edge $e = (a, b)$ in the order of ab (as in Figure 2(I)). Let $p_2 = uw_1 \ldots w_k v$ ($u = w_0$ and $v = w_{k+1}$) be a shortest path in G_{-e}. Then there exists a vertex w_i ($1 \leq i \leq k+1$) such that for subpaths $p_{21} = uw_1 \ldots w_{i-1}$ and $p_{22} = w_i \ldots v$ of p_2, $\Omega_e^G(u, w_{i-1}, dist(p_{21})) \wedge G_{-e}(w_{i-1}, w_i) \wedge \Omega_e^G(w_i, v, dist(p_{22}))$ holds. □

Let $\Phi_e^G(x, y, d)$ be the following:

$$(\exists d_1, d_2)(\exists w_1, w_2)\ \Omega_e^G(x, w_1, d_1) \\ \wedge \widehat{G_{-e}}(w_1, w_2) \wedge \Omega_e^G(w_2, y, d_2) \wedge (d = d_1 + dist(w_1, w_2) + d_2).$$

LEMMA 3.7. *Given an undirected graph G satisfying $dist(e) > 0$ for each edge $e \in G$, $SP_{G_{-e}}$ is a subset of Φ_e^G.*

PROOF. Suppose $SP_{G_{-e}}(x, y, d)$ holds and we prove $\Phi_e^G(x, y, d)$ is true.

Since $SP_{G_{-e}}(x, y, d)$ holds, there exists a shortest path $p = xw_1 \ldots w_k y$ in G_{-e} such that $SP_{G_{-e}}(x, y, dist(p))$. Let $x = w_0$ and $y = w_{k+1}$. Two cases arise: (a) In G, none of the shortest paths between x and y uses edge e. Then $\Omega_e^G(x, y, d)$ is true. Since $\widehat{G_{-e}}(y, y) \wedge \Omega_e^G(y, y, 0)$ is true, $\Omega_e^G(x, y, d)$, $\widehat{G_{-e}}(y, y)$ and $\Omega_e^G(y, y, 0)$ are all true. Therefore, $\Phi_e^G(x, y, d)$ holds. (b) Otherwise. There is a shortest path between x and y in G using edge e. From Lemma 3.5(d), there exists w_i such that

$$\Omega_e^G(x, w_{i-1}, d_1) \wedge G_{-e}(w_{i-1}, w_i) \wedge \Omega_e^G(w_i, y, d_2) \wedge (d = d_1 + dist(w_{i-1}, w_i) + d_2)$$

is true. Therefore, $\Phi_e^G(x, y, d)$ holds. □

Now, Theorem 3.4 can be easily proved from Lemma 3.7.

Proof of Theorem 3.4 Let $\psi(x, y, d) \equiv \Phi_e^G(x, y, d) \wedge (\forall d')[\Phi_e^G(x, y, d') \rightarrow (d \leq d')]$ Clearly, formula ψ is in $FO(+, <)$. From Lemma 3.7, we know $SP_{G_{-e}} \subseteq \psi$. On the other hand, suppose $\psi(x, y, d)$ holds. By the definition of $SP_{G_{-e}}$, there exists d' such that $SP_{G_{-e}}(x, y, d')$ holds. Since $SP_{G_{-e}} \subseteq \psi$, $\psi(x, y, d')$ holds. By the definition of ψ, there is at most one d'' for tuple (x, y) such that $\psi(x, y, d'')$ holds. So $d = d'$. Thus $\psi \subseteq SP_{G_{-e}}$. □

Example 3.8. For a simple illustration of the Theorem, let us consider graph $G_{+\varepsilon}$ of Example 3.3. Suppose we delete edge $\varepsilon = (a, b)$. $\Gamma_\varepsilon^{G_{+\varepsilon}}(j, d, 12)$ is true since the shortest path $j\ a\ b\ c\ d$ ($SP_{G_{+\varepsilon}}(j, d, 12)$ holds) between j and d uses edge (a, b). The shortest path between j and d in $G_{+\varepsilon} - (a, b)$ is path $j\ i\ h\ g\ f\ e\ d$. That is, $SP_G(j, d, 42)$ holds. $\Phi_\varepsilon^{G_{+\varepsilon}}(j, d, 42)$ is true since $\Omega_\varepsilon^{G_{+\varepsilon}}(j, i, 19)$, $G(i, h)$ and $\Omega_\varepsilon^{G_{+\varepsilon}}(h, d, 15)$ all hold and the distance of edge (i, h) is 8.

Lemma 3.7 reveals why we have such an incremental maintenance result for undirected graphs. It says that the distance of a shortest path in G_{-e} is bounded by the distances of (three) shortest paths in G. A similar result holds for acyclic digraphs, but not for general digraphs (see Section 4).

The algorithm for the maintenance under deletion is presented in Table II.

Table II. Maintenance Queries for Deletion

```
% TABLE Graph(Start,Tail,Dist): A tuple (s,t,d) is in Graph if the distance
% of edge (s,t) is d; for each vertex v, tuple (v,v,0) is in Graph.
% TABLE ShortDis(Start,Tail,Dist): A tuple (s,t,d) of ShortDis means that the
% shortest distance between vertex s and vertex t is d; for each vertex x,
% tuple (x,x,0) is in ShortDis.

% TABLE Susp: tuples that may need to be modified when graph G changes.
% When deleting edge e=(a,b) where dist(e)>0, the shortest distances formed
% by the following paths could be affected: (1) any path from x through (a,b)
% to y, (2) any path from x through (b,a) to y.
    INSERT INTO Susp(Start,Tail,Dist)
    SELECT A.Start, A.Tail, A.Dist
    FROM ShortDis A, ShortDis X, ShortDis Y
    WHERE A.Start=X.Start AND A.Tail=Y.Tail AND
            ((X.Tail=a AND Y.Start=b AND dist(e)+X.Dist+Y.Dist=A.Dist) OR
             (X.Tail=b AND Y.Start=a AND  dist(e)+X.Dist+Y.Dist=A.Dist));

% TABLE Trust stores the shortest distance tuples that do not use the
% deleted edge e=(a,b).
    INSERT INTO Trust(Start,Tail,Dist)
    SELECT A.Start, A.Tail, A.Dist
    FROM ShortDis A
    WHERE NOT EXISTS (SELECT * FROM Susp X
                      WHERE X.Star=A.Star AND X.Tail=A.Tail);

% Remove edge e=(a,b) from TABLE Graph(Start,Tail,Dist).
    DELETE FROM Graph(Start,Tail,Dist)
    WHERE Start=a AND Tail=b;

% TABLE Temp: Stores the new tuples (u,v,d) expressing that there exists a
% path between vertex u and v with distance d; d may not be the shortest.
    INSERT INTO Temp(Start,Tail,Dist)
    SELECT A.Start, B.Tail, dist(G.Star,G.Tail)+X.Dist+Y.Dist
    FROM TRUST A, Graph G, TRUST B
    WHERE A.Tail=G.Star AND G.Tail=B.Star AND
            (EXISTS (SELECT * FROM Susp X
                     WHERE X.Star=A.Star AND X.Tail=B.Tail));

% The result:
    DELETE FROM ShortDis;
    INSERT INTO ShortDis(Start,Tail,Dist)
    (SELECT * FROM Trust)
    UNION
    (SELECT A.Start, A.Tail, MIN(A.Dist)
    FROM Temp A
    GROUP BY A.Start, A.Tail);
```

3.2 Incremental Algorithm for $APSD_{\geq 0}$

We now discuss how to extend the maintenance results for $APSD_{>0}$ to $APSD_{\geq 0}$.

The following example shows that the incremental maintenance algorithms for $APSD_{>0}$ cannot be used to solve the $APSD_{\geq 0}$ problem on the deletion of 0-distance edges. (One can prove, similarly to the proofs of results in

Section 3, that they work on non-negative edge insertions and positive edge deletions.)

Example 3.9. Let G be a connected undirected graph where all edges have 0 distance. Whenever an edge is deleted, each tuple of SP_G will also be deleted by our algorithm, since the shortest path between every two vertices in the graph has the same distance as that of a walk passing the deleted edge.

This example shows that G is not *sp*-bound under SP_G if there are 0-distance edges. When each edge of G has a positive distance, each tuple of SP_G corresponds to paths of G. When G has 0-distance edges, a tuple of SP_G may correspond to walks of G. This is why the algorithm on $APSD_{>0}$ for edge deletion does not work for 0-distance-edge deletions.

To solve this problem, we will maintain SPD_G instead of SP_G. The addition of the length attribute allows us to map each tuple of SPD_G to a path. We can apply the algorithms for $APSD_{>0}$ twice to maintain SPD_G, once for distance and once for length, in a nested way. The details of the algorithms are given in Pang et al. [1999] and Pang [1999].

4. THE *APSD* PROBLEM IN DIGRAPHS

We now consider the incremental maintenance of the *APSD* problem in digraphs without negative cycles.[2] We first extend sp-boundness to ρ sp-boundness and show that the ρ sp-bound digraphs form quite a large family of diagraphs generalizing several previous families; then we give the incremental algorithm for maintaining *APSD* after deleting an ρ *sp-bound* arc.

The notion of ρ sp-boundness is useful since all digraphs in this family can be maintained in a similar way as those acyclic digraphs and undirected graphs.

4.1 The ρ sp-bound Diagraphs

We now extend the notion of sp-bound diagraphs. The extended definition describes a class of digraphs where *APSD* can be incrementally maintained in $FO(+, <)$. Intitutively, a ρ sp-bound digraph is a digraph in which, for each shortest path, there exist at most ρ number of sequencing subpaths that share a same arc.

Definition 4.1. (ρ *sp*-bound) Let ρ be a fixed positive integer. A digraph G is said to be ρ *sp*-bound for arc e under SP_G if for any shortest path $p_{xy} = xu_1 \ldots u_{ky}$ of $SP_{G_{-e}}$, there exist at most $\rho + 1$ distinct nodes $u_{i_1}, u_{i_2}, \ldots, u_{i_{\rho+1}}$ on p_{xy} such that $i_1 < i_2 \ldots < i_{\rho+1}$ and, for each $j = 1, \ldots, \rho$, there exists a shortest path of SP_G from u_{i_j} to $u_{i_{j+1}}$ that uses arc e. A digraph G is ρ *sp*-bound under SP_G if it is ρ *sp*-bound under SP_G for each arc $e \in G$. In the place of SP_G, we can also use other path-based relations such as TC_G and SPD_G; when TC_G is used we do not require the path p_{xy} to be a shortest path.

We note that the original notion of sp-bound is the same as that of 1 sp-bound under TC_G. Moreover, if a graph G is ρ sp-bound under TC_G then G is ρ

[2]A cycle is said to be *negative* if its distance is negative. Note that the presence of negative cycles implies that shortest walks can be unbounded.

sp-bound under SP_G and under SPD_G; if G is ρ sp-bound under SP_G then G is ρ sp-bound under SPD_G.

LEMMA 4.2. *A digraph G is $\rho + 2$ sp-bound under SPD_G (or TC_G or SP_G) if each SCC is ρ sp-bound under SPD_G (or TC_G or SP_G).*

PROOF. Suppose G is a graph such that each SCC of G is ρ *sp*-bound under SPD_G. Let $e = (a, b)$, $p_{xy} = xu_1 \dots u_k y$ be a shortest path of $SP_{G_{-e}}$, and $u_{i_1}, u_{i_2}, \dots, u_{i_m}$ be nodes on p_{xy} such that, for each $j = 1, \dots, \rho + 1$, there exists a path from u_{i_j} to $u_{i_{j+1}}$ that uses arc e. Clearly, $a, b, u_{i_2}, \dots, u_{i_{m-1}}$ are in a common SCC, and $u_{i_2} u_{i_2+1} \dots u_{i_{m-1}}$ is a shortest path within that SCC. It follows from the assumption that $m - 2 \le \rho + 1$, and hence $m \le \rho + 3$. □

We now show that the class of ρ sp-bound diagraphs is quite general. We do this by showing that two well known families of graphs are ρ sp-bound, and give an example family of ρ sp-bound graphs that is not any of previously known families.

PROPOSITION 4.3. *(a) An acyclic graph G is 1 sp-bound ($\rho = 1$) under TC_G, SP_G and SPD_G. (b) A digraph without arc-join cycles is 2 sp-bound under TC_G, SP_G and SPD_G.*[3]

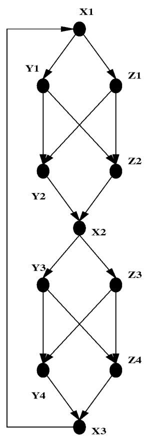
PROOF. Let G be a digraph and $e = (a, b)$ be an arc of G. We consider ρ sp-boundness under the most basic path relation, namely TC_G.

For (a), suppose G is acyclic, but not 1 *sp*-bound. Then there exist a path $p_{xy} = xu_1 \dots u_k y$ in G_{-e} and three nodes $u_{i_1}, u_{i_2}, u_{i_3}$ such that there exist paths from u_{i_j} to $u_{i_{j+1}}$ through e for $j = 1, 2$. Then u_{i_2} is in a cycle with a, b, contradicting the acyclicity assumption.

For (b), suppose G is a diagraph without arc-join cycles, but G is not 2 *sp*-bound. Then there exist a path $p_{xy} = xu_1 \dots u_k y$ in G_{-e} and four nodes $u_{i_1}, u_{i_2}, u_{i_3}, u_{i_4}$ such that $i_j < i_{j+1}$ and there exist paths $p_{ij} = u_{i_j} \dots ab \dots u_{i_{j+1}}$ in G from u_{i_j} to $u_{i_{j+1}}$ through e for $j = 1, 2, 3$. We now have two walks going through e: the first is the concatenation of the tail half of p_{12} (starting from ab) and the head half of p_{23} (ending at a) excluding u_{i_2}, and the second is the concatenation of the tail half of p_{23} (starting at ab) and the head half of p_{34} (ending at a) excluding u_{i_3}. Let p_1 denote the first walk and p_2 the second. By removing arcs if necessary, we can assume that p_1 and p_2 are cycles and they both include ab as their beginning and a as their ends. Two cases arise. Case (i): p_1 and p_2 are not identical. Note that they both go through the arc e. Case (ii): p_1 and p_2 are identical. Note that p_1 ($= p_2$) cannot have the form $ab \dots u_{i_2} \dots u_{i_3} \dots a$, since $u_{i_2} \dots u_{i_3} \dots a$ is a subpath of $u_{i_2} \dots ab \dots u_{i_3}$. Hence p_1 ($= p_2$) has the form $ab \dots u_{i_3} \dots u_{i_2} \dots a$. Let p' be the subpath $u_{i_3} \dots u_{i_2}$ of p_1. Concatenating p' with the subpath $u_{i_2} u_{i_2+1} \dots u_{i_3-1} u_{i_3}$ of p_{xy} we get a new cycle p_3. (We remove arcs if necessary, similar to above.) Then p_1 and p_3 are two cycles and they share all arcs of p'. In all cases, G has arc-join cycles, contradictory to the assumption. □

Example 4.4. Consider the digraphs G shown in Figure 3. The general form of G is the digraph G_k shown in Figure 3.3 with $V(G_k) = X \cup Y \cup Z$ where

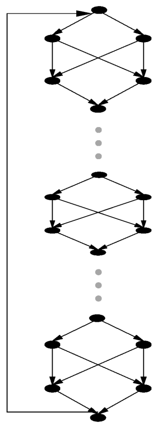
[3]Two cycles are said to be *arc-join cycles* if they share a common arc.



(1)

	X1	Y1	Z1	Y2	Z2	X2	Y3	Z3	Y4	Z4	X3
X1	0	1	1	2	2	3	4	4	5	5	6
Y1	6	0	7	1	1	2	3	3	4	4	5
Z1	6	7	0	1	1	2	3	3	4	4	5
Y2	5	6	6	0	7	1	2	2	3	3	4
Z2	5	6	6	7	0	1	2	2	3	3	4
X2	4	5	5	6	6	0	1	1	2	2	3
Y3	3	4	4	5	5	6	0	7	1	1	2
Z3	3	4	4	5	5	6	7	0	1	1	2
Y4	2	3	3	4	4	5	6	6	0	7	1
Z4	2	3	3	4	4	5	6	6	7	0	1
X3	1	2	2	3	3	4	5	5	6	6	0

(2)



(3)

Fig. 3. (1) Digraph G_3; (2) SP_{G_3}; (3) Digraph G_k.

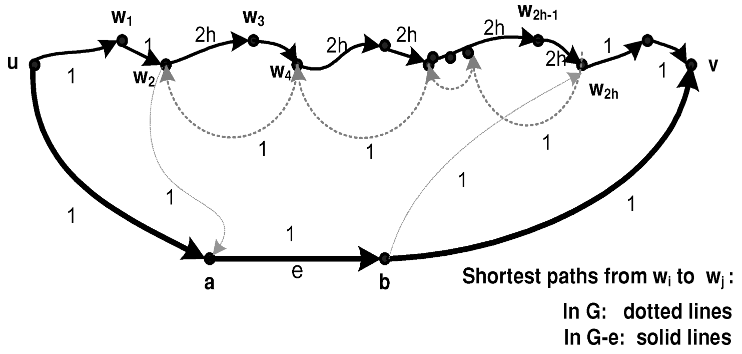


Fig. 4. G is 2-path But Not ρ *sp*-bound ($\rho < h$).

$|X| = k$, $|Y| = |Z| = 2(k-1)$ and $|V(G_k)| = 5k-4$; Figure 3.1 shows G_3. In digraph G_k, the number of paths from X_1 to X_k is 4^{k-1}, an exponential number of $|V(G_k)|$.

Observe that G_k is 1 sp-bound under SP_G, but it is not acyclic, and it is not an ℓ-path graph [Dong and Su 1995] for $\ell < 4^{k-1}$.[4] Consequently, the family $\{G_i \mid i = 1, 2, \dots\}$ is a family of 1 sp-bound non-acyclic diagraphs, but it is not an ℓ-path family for any ℓ.

We note that the k-path digraphs and ρ *sp*-bound digraphs are not comparable with each other in general. Under a certain condition, k-path digraphs are ρ *sp*-bound digraphs. The incomparability is established by combining Examples 4.4 and 4.5. The special condition is considered after the next example.

Example 4.5. For any given $h > 1$, there exists a 2-path digraph G that is not ρ *sp*-bound for $\rho < h-1$. Indeed, let G be the digraph expressed in Figure 4.

[4]A digraph is ℓ-path if there exist at most ℓ distinct paths between any pair of nodes.

The shortest paths from w_i to w_j $(i < j)$ are depicted in solid line (in dotted line, respectively) in G_{-e} (in G, respectively). Clearly, for shortest path $p_{uv} = u \ldots w_1 \ldots w_i \ldots w_{2h} \ldots v$ of G_{-e}, there exist h distinct nodes $w_2, w_4, \ldots, w_{2h}$ on p_{uv} such that, for each $j = 2, \ldots, 2h-2$, there exists a shortest path of SP_G from w_j to w_{j+2} that uses arc e. Therefore, G is not *sp*-bound for $\rho < h - 1$.

4.2 The Incremental Algorithms

We now consider the *APSD* problem for digraphs without negative cycles. We will consider how to maintain SPD_G, since the answer to the *APSD* problem can be derived from SPD_G.[5]

After insertions or deletions, we need to ensure that the new graphs still contain no negative cycles. Clearly, the deletion of an arc will not introduce negative cycles. We can check if an inserted arc introduces negative cycles in $FO(+, <)$ as follows:

LEMMA 4.6. *Let G be a digraph without negative cycles and $e = (a, b)$ be an arc not in G. Then G_{+e} has negative cycles if and only if there exist b, a, k, d such that $SPD_G(b, a, k, d) \wedge (d + dist(e) < 0)$ holds.*

4.2.1 *Inserting an Arc $e = (a, b)$ into G.*

The next theorem can be proved similarly to Theorem 3.2.

THEOREM 4.7. *Let G be a digraph and $e = (a, b)$ an arc such that G_{+e} contains no negative cycles. We can construct a formula φ of $FO(+, <)$ from SPD_G and e such that $SPD_{G_{+e}} \equiv \varphi$.*

4.2.2 *Deleting an Arc $e = (a, b)$ from G.*

The main result of this section is:

THEOREM 4.8. *Suppose G is a diagraph without negative cycles and G is ρ sp-bound under SPD_G, and e is an arc of G. We can construct a formula ψ of $FO(+, <)$ from SPD_G, e and G so that $SPD_{G_{-e}} \equiv \psi$.*

PROOF. Intuitively, we will construct the formula ψ for computing $SPD_{G_{-e}}$ by exploiting the ρ sp-boundness; ψ will join shortest paths that do not use e, and some additional work helps to ensure the resulting pathes are shortest.

Formally, let $\Gamma_e^G(x, y, k, d)$ express those paths of SPD_G using arc $e = (a, b)$, and let $\Omega_e^G(x, y, k, d)$ express those paths of SPD_G not using e:

$$\begin{aligned}\Gamma_e^G(x, y, k, d) \equiv\ & SPD_G(x, y, k, d) \wedge SPD_G(x, a, k_1, d_1)\\ & \wedge SPD_G(b, y, k_2, d_2) \wedge (k = k_1 + k_2 + 1)\\ & \wedge (d = d_1 + d_2 + dist(e)).\\ \Omega_e^G(x, y, k, d) \equiv\ & SPD_G(x, y, k, d) \wedge \neg\Gamma_e^G(x, y, k, d).\end{aligned}$$

Let the tuples of Φ_e^G denote walks formed by concatenating the paths of Ω_e^G and $\widehat{G_{-e}}$. We then extract tuples expressing the shortest paths from Φ_e^G into

[5]For special cases, e.g. when each arc has positive distance or the underlying digraph is acyclic, we can maintain SP_G (instead of SPD_G), in ways similar to Section 3.

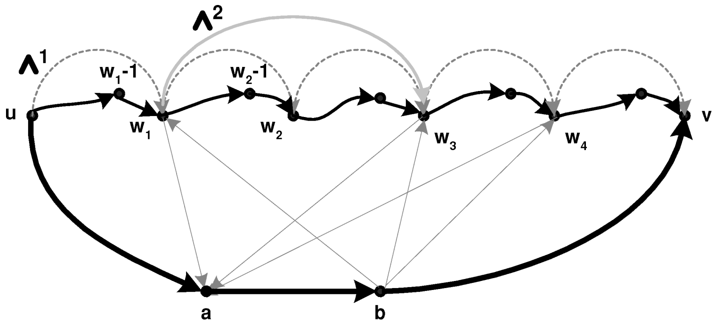


Fig. 5. An *sp*-bound Digraph.

MP_e^G, pull out the shortest paths with minimal length from MP_e^G, and store them in Λ^1. The tuples of $SPD_{G_{-e}}$ are formed by concatenating Λ^1 up to ρ times (see Figure 5). Finally, ψ is pulled out from the Λ's. Specially

$$\begin{aligned}
\Phi_e^G(x,y,l,d) &\equiv (\exists d',l')(\exists w_1,w_2)\,\Omega_e^G(x,w_1,l',d') \wedge \widehat{G_{-e}}(w_1,w_2)\\
&\quad \wedge [(l = l'+1) \wedge (w_1 \neq w_2) \vee (l = l') \wedge (w_1 = w_2)]\\
&\quad \wedge (d = d' + dist(w_1,w_2)),\\
MP_e^G(x,y,l,d) &\equiv \Phi_e^G(x,y,l,d) \wedge (\forall d')\left[\Phi_e^G(x,y,l,d') \to ((d \leq d')\right],\\
\Lambda^1(x,y,l,d) &\equiv MP_e^G(x,y,l,d) \wedge (\forall l')\left[MP_e^G(x,y,l',d) \to (l \leq l')\right],\\
&\ldots\\
\Phi_\Lambda^h(x,y,l,d) &\equiv (\exists d_1,d_2)(\exists l_1,l_2)(\exists w)\,\Lambda^{h-1}(x,w,l_1,d_1) \wedge \Lambda^{h-1}(w,y,l_2,d_2)\\
&\quad \wedge (l = l_1 + l_2) \wedge (d = d_1 + d_2),\\
MP_\Lambda^h(x,y,l,d) &\equiv \Phi_\Lambda^h(x,y,l,d) \wedge (\forall d')(\forall l')\left[\Phi_\Lambda^h(x,y,l',d') \to ((d \leq d')\right],\\
\Lambda^h(x,y,l,d) &\equiv MP_\Lambda^h(x,y,l,d) \wedge (\forall l')\left[MP_\Lambda^h(x,y,l',d) \to (l \leq l')\right],\\
\psi(x,y,l,d) &\equiv (\exists h)\,\Lambda^h(x,y,l,d) \wedge (h \leq \rho)\\
&\quad \wedge (\forall l',d',h')\,[\Lambda^{h'}(x,y,l',d') \to (l \leq l') \wedge (d \leq d')].
\end{aligned}$$

Let $p_2 = uw_1 \ldots w_k v$ be a shortest path from u to v ($u = w_0$, $v = w_{k+1}$) in G_{-e} such that $SPD_{G_{-e}}(u,v,k+1,dist(p_2))$ holds and $k+1 = length(p_2)$. If no shortest path from u to v in G goes through arc e then $\Omega_e^G(u,v,k+1,dist(p_2))$. Otherwise, there exists a shortest-path p_1 from node u to node v that goes through arc $e = (a,b)$ such that $SPD_G(u,v,length(p_1),dist(p_1))$ holds. Since G is ρ *sp*-bound, there exist nodes w_{i_j} ($j = 1,2,\ldots,h$) on p_2 where $0 = i_0 < i_1 < \cdots < i_h = k$ and $h \leq \rho + 1$ such that, in G, (i) there exists a shortest path from $w_{i_{j-1}}$ to w_{i_j} that goes through arc e ($j = 1,2,\ldots,h$), and (ii) there exists no shortest path from $w_{i_{j-1}}$ to w_{i_j-1} that goes through arc e ($j = 1,2,\ldots,h$). Condition (ii) implies $\Omega_e^G(w_{i_{j-1}}, w_{i_j-1}, i_j - i_{j-1} - 1, dist(w_{i_{j-1}}, w_{i_j-1}))$ holds for $j = 1,\ldots,h$. Then the tuples of $SPD_{G_{-e}}$ representing the path from $w_{i_{j-1}}$ to w_{i_j} are in Λ^1 ($j = 1,2,\ldots,h$). Similarly, the tuples of $SPD_{G_{-e}}$ representing the path from $w_{i_{j-1}}$ to $w_{i_{j+1}}$ are in Λ^2 ($j = 1,2,\ldots,h-1$). Continuing this process, we could conclude that the tuple of $SPD_{G_{-e}}(u,v,l,d)$ is in $\psi(x,y,l,d)$. □

5. TRANSITIVE CLOSURE AND SHORTEST PATHS

We now extend our earlier results in two directions:

(1) The algorithms given earlier can be used to maintain the transitive closure for the type of graphs considered earlier. This is because the transitive closure can be derived from the answer to the APSD problem.

(2) Our algorithms can be extended to maintain the shortest paths themselves in undirected graphs without negative edges or in digraphs without negative cycles. Roughly speaking, we use relation $Spath_G^{(x,y)}$ for the maintenance; $Spath_G^{(x,y)}$ denotes the arc set of a shortest path from node x to node y if there is a path, and it denotes $\emptyset$ otherwise. The interested readers can refer to Pang [1999] for more details.

6. EMPIRICAL EVALUATION AND DISCUSSION ON COMPLEXITY ISSUES

In the previous sections, we have provided algorithms for the incremental maintenance of the $APSD_{>0}$ problem, the $APSD_{\geq 0}$ problem, and the transitive closure on undirected graphs and on *sp*-bound digraphs. Those algorithms are $FO(+, <)$ algorithms, each requiring just a small bounded number of relational join operations as discussed in previous sections. This implies that these algorithms are desirable from a complexity perspective: they can be executed and optimized directly on relational database systems, and they allow efficient parallel execution. Below we first provide some experimental results to evaluate the efficiency of our algorithms, and then give some further comments on complexity issues.

6.1 Empirical Evaluation

We now report results of experiments conducted to evaluate the execution-time performance of our algorithm, for the single edge deletion case of the shortest-distance problem on undirected graphs.[6] Our maintenance algorithm is implemented in SQL, and the static algorithm is an SQL implementation (in Transact-SQL – an extended SQL language) of Floyd's method [Sedgewick 1990].

We did two sets of experiments. The first set involves small graphs with between 100 and 2171 edges, and the second involves a large graph with nearly 1 million edges. In all graphs, the edge weights are randomly generated. The smaller graphs have randomly generated edges, and have varying vertex degrees. The large graph contains 1000 vertices and all possible edges between them, and the weights of the edges are uniformly distributed. The experiments on these graphs show the influence of graph size, node degree and edge weight on run time, and the speedup achieved by our algorithm.

Experimental results reported in Figure 6 indicate that substantial performance gain is achieved using our incremental algorithm over the static algorithm. These experiments were performed on an INGRES database system running on a Sun SPARC machine (with 150MHz CPU and 160MB memory).

[6]Since many papers have studied the algorithm for the edge insertion case, we will not test its performance in this article.

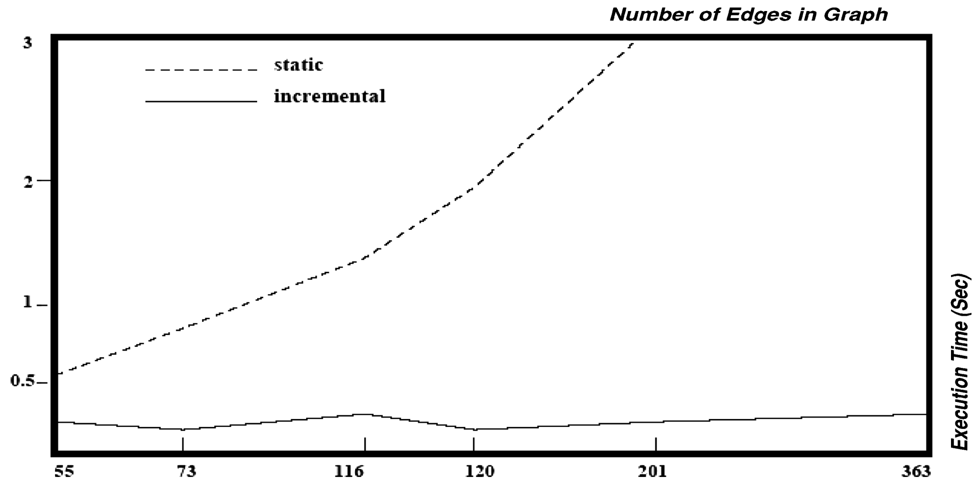


Fig. 6. Comparison: Execution time vs. graph size per edge deletion for shortest distances in undirected graphs.

The *x-axis* is the number of edges in a randomly generated graph, and the *y-axis* is the time spent by the algorithm under consideration. The time shown in the figure is the average time used by the algorithm under consideration for three edge deletions, each deleting a randomly chosen edge. The figure shows that, as the size of the graph increases, the time spent on maintenance (the solid curve) does not vary much, whereas the time used by the static algorithm (which recomputes the shortest distance from scratch) increases significantly (the dashed curve). The reason is that the incremental algorithm avoids a lot of unnecessary computation when maintaining relation *SP*. Moreover, the maintenance algorithm uses much less time than the recomputing from scratch algorithm. In fact, the speed up ranges from 4 for small graphs to hundreds for larger graphs.

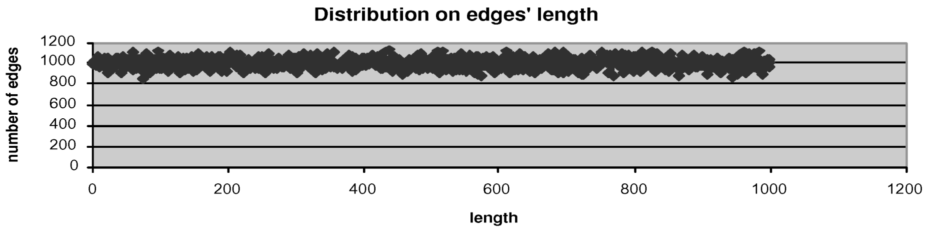
Table III shows that the incremental algorithm achieves more speedup as the size and density of the graph increase. Note that N_v and N_e indicate the size of the graph; N_e/N_v is the average degree of the nodes of the graphs, and hence is an indicator of the density of the graph; and T_S/T_I is the speedup achieved by the incremental algorithm over the static algorithm.

We also tested our deletion algorithm on an SQL Server 2000 (with a Pentium 4 3.20GHz CPU and 1 GB RAM) on a large graph with nearly 1 million edges. Figure 7 shows the distribution of edge-length and small length (*length* $\leq$ 20) for this graph. The execution time used is given in Table IV. For this large graph, our algorithm can maintain the shortest path with between 41 seconds and 1 minute 53 seconds while the static algorithm needs more than five hours. The average speedup is around 233. The computation speed also implies that our deletion algorithm can be used to sequentially delete a small set of edges in practice. Our algorithm uses less time for the deletion of edges with larger weights, than for the deletion of edges with smaller weights. We believe that this is due to the fact that edges with small weights are involved in shortest paths more often than edges with large weights.

Table III. Comparison on Run Time over Graphs with Varying Characteristics

Graphs	NoEdges(N_e)	NoVertices(N_v)	T_S	T_I	N_e/N_v	T_S/T_I
G_1	100	29	0.902	0.221	3.448	4.081
G_2	117	36	1.341	0.311	3.250	4.311
G_3	235	55	4.682	0.275	4.727	17.025
G_4	299	65	7.641	0.285	4.600	26.810
G_5	392	81	13.511	0.245	4.839	55.372
G_6	670	104	27.111	0.247	6.442	109.761
G_7	980	91	20.131	0.237	10.769	84.940
G_8	1292	106	31.332	0.254	12.189	123.354
G_9	1201	146	85.212	0.238	8.226	358.033
G_{10}	1716	171	129.440	0.251	10.035	517.760
G_{11}	1677	139	75.191	0.268	11.992	290.563
G_{12}	2171	181	176.492	0.277	11.994	637.155

T_S: average time used by the static algorithm (seconds).
T_I: average time used by the incremental algorithm (seconds).



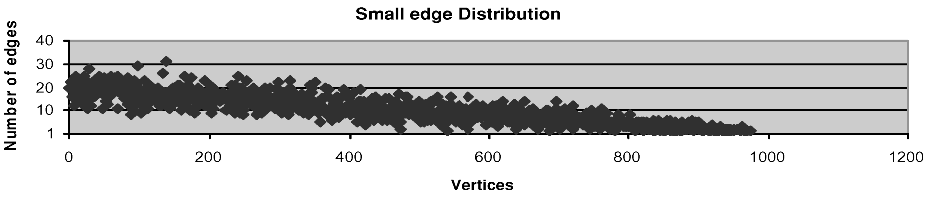


Fig. 7. Distribution on edge-length and on small edge (length $\leq$ 20).

6.2 Discussion

Our algorithms can be optimized by using indexing. Indeed, let $e = (a, b)$ be the edge inserted or deleted. Our algorithms derive the new tuples of *APSD* (or transitive closure) relation for those vertices u that can reach a and those vertices that can be reached from b. Index for these two types of reachability can allow us to execute the algorithms faster.

For the shortest distance problem in undirected graphs, our algorithm on single edge deletion is the first to utilize the property that each shortest distance of G_{-e} can be formed through concatenating at most two shortest paths of G via an edge of G_{-e}. This property helps us avoid at least one additional join operation and one termination check at the path-regeneration stage adopted in many sequential/parallel incremental algorithms.[7] Most previous approaches to this problem treated undirected graphs and directed graphs in the same

[7]For example, in DRed Algorithm [Gupta et al. 1993b], this stage is to put back those deleted tuples that have alternative derivations.

Table IV. Performance Evaluation on the 1 Million Edge Graph

Edge Length	No of Susp	No of Trust	No of Affected Paths	Time (min:sec)
1	4328	993673	2378	01:16
1	8460	989541	3213	01:26
1	10148	987853	5044	01:27
1	8532	989469	4538	01:17
1	10800	987201	5510	01:53
2	2780	995221	1353	01:21
13	8	997993	4	00:46
13	0	998001	0	00:41
21	0	998001	0	00:40
11	10	997991	10	00:46
16	2	997999	0	00:43
20	2	997999	2	01:14

manner and, therefore, they did not take advantage of the special properties of undirected graphs. Moreover, we extended this property to the digraphs (ρ *sp*-bound) in the article and found a new class of digraphs, which could be maintained in a complexity similar to that for undirected graphs.

For the transitive closure in digraphs with cycles, we use the auxiliary relations SP_G, which keeps the shortest length (of derivations) between each pair of nodes for the maintenance of transitive closure. In this way, deleting an arc will only lead to the deletion of those derived tuples (u, v, l) in SP_G such that there is a path p from u to v going through the deleted arc and $length(p) = l$. This leads to a small set of deleted (and recomputed) tuples (u, v, l). The number of tuples deleted by this algorithm is no more, and it should typically be much smaller, than the number of tuples deleted by the algorithms in Harrison and Dietrich [1992]; Kuchenhoff [1991]; Urpi and Olive [1992]; Gupta et al. [1992, 1993b] and Ceri and Widom [1994]. Therefore, it could save the time for recomputing new derived tuples. For instance, consider the graph in Figure 3 and suppose the deleted arc is (y_1, y_2). The algorithm first deletes the shaded tuples in Figure 3(2). Then using join operations upon the unshaded data derive those tuples that either should not have been deleted in the first step or have a new shortest length. All the shaded tuples (except tuple $(y_1, y_2, 1)$) are rederived with two joins in this step.

7. RELATED WORK

This article extends the results of Pang et al. [1999].

For a digraph with n nodes and m arcs, Even and Gazit [1985] gives a full dynamic algorithm that requires $O(n^2)$ sequential time for an arc insertion and/or an arc cost decrease, and $O(mn + n^2 \log n)$ time for an arc deletion and/or an arc cost increase; Roditty [2003] gives an algorithm to maintain the transitive closure matrix in a total running time of $O(mn + n^2)$. More recently, Roditty and Zwick [2004] gives an almost linear time fully dynamic algorithm for the transitive closure problem for directed graphs. Demetrescu and Italiano [2003] presents a fully dynamic algorithm that requires *O(mnpolylog(n))* space with $O(n^2 polylog(n))$ amortized time per update. The algorithm in Ausiello et al. [1992] is for graphs with nice topologies such as trees and outerplanar graphs. Djidjev et al. [2000] achieves logarithmic query and update times for planar digraphs through graph decomposition. Feuerstein and Marchetti-Spaccamela [1993] and Klein et al. [1994] also consider planar graphs. All these previous dynamic algorithms and batch algorithms for the *APSD* problem use elaborate data structures and need a recursive mechanism. There is not any previously known more efficient algorithm specifically for undirected graphs or *sp*-bound digraphs. Since most commercial database systems are relational database systems, which do not directly support recursion, a more powerful host language is needed for those algorithms to maintain the *APSD* views.

Given a recursive query, the nonrecursive incremental evaluation approach uses nonrecursive programs to compute the difference of the answers to the query against successive databases updates. The mechanism used in this approach is called a "First-Order Incremental Evaluation System" in Dong and Topor [1992] and "Dyn-FO" in Patnaik and Immerman [1994]. For undirected graphs, reachability, connectivity, bipartiteness and the minimum spanning forests problems have first-order incremental algorithms for edge insertion and edge deletion [Dong and Su 1995; Patnaik and Immerman 1994]. Dong and Kotagiri [1997] considers maintaining views defined by constrained transitive closure queries in directed graphs with weights, but for insertion only.

Comparing the algorithms of Dong and Su [1995] and Patnaik and Immerman [1994] with ours for the transitive closure problem in undirected graphs, their algorithms are based on the maintenance of spanning forests of the given undirected graph while ours are not. Our algorithms are structurally simple and do not need to maintain the order of edges, a successor relation on all vertices. The algorithms based on maintaining spanning forests are hard to convert to solve the *APSD* problem since the set of all edges on the shortest paths does not necessarily have a "tree-like" structure and, therefore, single modification to the undirected graph may cause the reconstruction of $O(|V|)$ number of spanning trees (where V is the set of vertices of the graph). Our algorithms also have a unifying scheme of computation, for transitive closure in undirected graphs, in acyclic digraphs [Dong and Pang 1997], and for the $APSD > 0$ (or $APSD_{\geq 0}$) problem in undirected graphs.

Our algorithms are in $FO(+, <)$, using "+" and "<" to add and select the minimum length of paths. The storage demands of our algorithms stay within the same order of magnitude as the algorithm of Dong and Su [1995].

Gupta, Mumick and Subrahmanian [Gupta et al. 1993b] give two general algorithms for view maintenance: *Counting* algorithm and *DRed* (Delete and Rederive) algorithm. The counting algorithm stores the number of alternative derivations for each derived tuple in a view and works on nonrecursive views defined by SQL or Datalog. The DRed algorithm works on recursive views (negation and aggregation are allowed). When a tuple (or an edge or an arc) is deleted from the base relations, the DRed algorithm works in two steps: (1) Delete those derived tuples that depend upon the deleted base tuple (i.e. if a derived tuple has a derivation tree that contains the deleted base tuple). This step normally deletes more than necessary; (2) Then the DRed algorithm refines this overestimation by considering alternative derivations of the deleted tuples. When the

DRed algorithm is used for the transitive closure problem, deleting a tuple can require regenerating a large portion of the data in the view from scratch.

For maintaining transitive closure, our algorithm has a similar space complexity to that of the DRed algorithm of Gupta et al. [1993a]. When deleting a tuple, the time complexity of DRed Algorithm is related to the size of the cluster including that tuple. The time complexity of our algorithm is related to the number of cycles including the tuple, it is no more expensive than the DRed algorithm and in many situations it is much cheaper. Our method is different from the *counting method*.

This article provided first-order maintenance results to more general classes of graphs than earlier studies. Indeed, the class of *sp*-bound graphs is more general than the classes of graphs that were known earlier to be first-order maintainable.

8. CONCLUDING REMARKS

We have given $FO(+, <)$ algorithms for the incremental maintenance of the $APSD_{>0}$ problem, the $APSD_{\geq 0}$ problem, and the transitive closure. Since our algorithms belong to $FO(+, <)$, they have a low parallel complexity. They also make additional optimization techniques such as those designed for relational database systems possible. These results extend earlier ones on the maintenance of transitive closure of weighted undirected graphs and on the maintenance of shortest distance of weighted undirected graphs. The $FO(+, <)$ incremental algorithms for arc insertion and deletion in ρ sp-bounded digraphs also extend the boundary of previous results for digraphs.

REFERENCES

AUSIELLO, G., ITALIANO, G. F., SPACCAMELA, A. M., AND NANNI, U. 1992. On-line computation of minimal and maximal length path. *Theor. Comput. Sci. 95*, 2, Elsevier Science 245-261.

BLAKELEY, J. A., LARSON, P.-A., AND TOMPA, F. W. 1986. Efficiently updating materialized views. In *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*. ACM Press, 61–71.

BUCHSBAUM, A. L., KANELLAKIS, P. C., AND VITTER, J. S. 1990. A data structure for arc insertion and regular path finding. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 22–31.

CERI, S. AND WIDOM, J. 1994. Deriving incremental production rules for deductive data. *Infor. Syst. 19(6)*, 467–490.

DEMETRESCU, C. AND ITALIANO, G. F. 2003. A new approach to dynamic all pairs shortest paths. In *Proceedings of the 35th ACM Symposium on Theory of computing*. ACM Press, 159–166.

DJIDJEV, H., PANTZIOU, G. E., AND ZAROLIAGIS, C. D. 2000. Improved algorithms for dynamic shortest paths. *Algorithmica 28,* 4, 367–389.

DONG, G. AND KOTAGIRI, R. 1997. Maintaining constrained transitive closure by conjunctive queries. In *Proceedings of International Conference on Deductive and Object-Oriented Databases (DOOD), LNCS 1341*. 35–51.

DONG, G. AND PANG, C. 1997. Maintaining transitive closure in first-order after node-set and edge-set deletions. *Information Processing Letters 62,* 3, 193–199.

DONG, G. AND SU, J. 1995. Space-bounded foies. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 22–25, 1995, San Jose, California*. ACM Press, 139–150.

DONG, G. AND SU, J. 2000. Incremental maintenance of recursive views using relational calculus/ sql. *SIGMOD Record 29,* 1, 44–51.

DONG, G. AND TOPOR, R. 1992. Incremental evaluation of datalog queries. In *Proceedings of the International Conference on Database Theory*. Berlin, Germany, 282–296.

EVEN, S. AND GAZIT, H. 1985. Updating distances in dynamic graphs. *Methods of Operations Research 49*, 371–387.

FEUERSTEIN, E. AND MARCHETTI-SPACCAMELA, A. 1993. Dynamic algorithms for shortest paths in planar graphs. *Theoret. Comput. Sci. 116,* 2, 359–371.

GRUMBACH, S. AND SU, J. 1995. First-order definability over constraint databases. In *Proceedings of Conference on Constraint Programming*.

GUPTA, A., KATIYAR, D., AND MUMICK, I. S. 1992. Counting solutions to the view maintenance problem. In *Proceedings of the JICSLP Workshop on Deductive Databases, Washington DC, November 1992*, K. Ramamohanarao, J. Harland, and G. Dong, Eds. 185–194.

GUPTA, A., MUMICK, I., AND SUBRAHMANIAN, V. 1993a. Maintaining views incrementally. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 157–166.

GUPTA, A., MUMICK, I. S., AND SUBRAHMANIAN, V. S. 1993b. Maintaining views incrementally. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 157–166.

HARRISON, J. V. AND DIETRICH, S. W. 1992. Maintenance of materialized views in a deductive database: An update propagation approach. In *Proceedings of the JICSLP Workshop on Deductive Databases*, K. Ramamohanarao, J. Harland, and G. Dong, Eds. 56–65.

KLEIN, P. N., RAO, S., RAUCH, M. H., AND SUBRAMANIAN, S. 1994. Faster shortest-path algorithms for planar graphs. In *Proceedings of the 26th Symposium Theory of Computing*. ACM, 27–37.

KUCHENHOFF, V. 1991. On the efficient computation of the difference between consecutive database states. In *Proceedings of the Second International Conference on Deductive Object-Oriented Databases*, C. Delobel, M. Kifer, and Y. Masunaga, Eds. Lecture Notes in Computer Science, Springer-Verlag, vol. 566. Springer-Verlag, 478–502.

LA POUTRE, J. A. AND VAN LEEUWEN, J. 1987. Maintenance of transitive closures and transitive reductions of graphs. Tech. Rep. RUU-CS-87-25, Department of Computer Science, University of Utrecht, The Netherlands.

LIN, C. C. AND ZHANG, R. C. 1990. On the dynamic shortest path problem. *J. Inf. Proc. 13(4)*.

PANG, C. 1999. Incremental maintenance reachability of graph in first-order and its extension. Ph.D. thesis, The University of Melbourne.

PANG, C., RAMAMOHANARAO, K., AND DONG, G. 1999. Incremental FO(+, <) maintenance of all-pairs shortest paths for undirected graphs after insertions and deletions. In *Proceedings of the International Conference on Database Theory*. Lecture Notes in Computer Science, Springer-Verlag.

PATNAIK, S. AND IMMERMAN, N. 1994. Dyn-FO: A parallel dynamic complexity class. In *Proceedings of the ACM Symposium on Principles of Database Systems*. 210–221.

RAMALINGAM, G. 1996. *Bounded Incremental Computation*. LNCS 1089.

RODITTY, L. 2003. A faster and simpler fully dynamic transitive closure. In *Proceedings of the 14th annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 404–412.

RODITTY, L. AND ZWICK, U. 2004. A fully dynamic reachability algorithm for directed graphs with an almost linear update time. In *STOC '04: Proceedings of the 36th Annual ACM Symposium on Theory of Computing*. ACM Press, 184–191.

SEDGEWICK, R. 1990. *Algorithms in C*. Addison-Wesley Publishing Company.

SHMUELI, O. AND ITAI, A. 1984. Maintenance of views. In *Sigmod Record*. Vol. 14(2). 240–255.

URPI, T. AND OLIVE, A. 1992. A method for change computation in deductive databases. In *VLDB*.

Received February 2004; revised September and December 2004 and March 2005; accepted April 2005