To appear in DOOD 97.

# Maintaining Constrained Transitive Closure by Conjunctive Queries

Guozhu Dong and Kotagiri Ramamohanarao

Computer Science Department
University of Melbourne
Parkville, Vic 3052, Australia
Email: {dong,rao}@cs.mu.oz.au

**Abstract.** Recently there has been considerable effort on the maintenance of database views in general, and of deductive database views in particular. Such work considered the incremental maintenance of deductive views defined by expensive database queries, including the transitive closure query. An interesting approach is to use only first-order queries in this maintenance, after small changes (e.g. one tuple insertions and deletions) to base relations. In this paper we consider such incremental maintenance of views defined by *constrained* transitive closure queries, for the insertion case. The constraints may refer to node costs such as height, voltage and temperature at spatial locations of interest, and to edge costs such as distances between spatial locations. When the constraints are based on node costs, we divide the maintenance problem into several cases, depending on the nature of the constraints. For some cases, the constrained transitive closure becomes bounded in the sense that the recursion can be removed and thus the maintenance problem becomes trivial. For the other cases we provide complete solutions for maintaining the constrained transitive closure. The size of the auxiliary relations can be kept bounded by the size of the transitive closure of the graphs. We also illustrate how to maintain the constrained transitive closure in the presence of other kinds of constraints. However, whether such views can be maintained in first-order after the deletion of edges is a major open problem, even for the constraint-free transitive closure of directed graphs.

Key words: Database management systems; database systems, query processing; transitive closure, constraints, incremental query evaluation, view maintenance, conjunctive query, first-order query.

## 1 Introduction

In database applications there is a need to access the answers to recursive queries frequently. In such a situation it is a common practice to *materialise* the answer to the query as a view, i.e., the answer is stored in a relation. However, the database can be updated between the access requests. When an update to the base relations occurs, the view needs to be adjusted so that it remains the answer to the query in the current database state. The adjustment to the view can be done using one of two methods: (i) by recomputing the view without using the old view, (ii) or by using the old view, and repairing it to get the new view. Using the second method one tries to find the difference between the old view and the new view in some way more efficient than the first method under normal situations. This is the so-called *view maintenance* problem. (See Figure 1.)

$$\begin{tikzcd}
\boxed{\mathrm{DB}^{old}} \arrow[r, "\text{update}"] \arrow[d, "\text{Qdfn}"'] & \boxed{\mathrm{DB}^{new}} \arrow[d, "\text{Qdfn}"'] \\
\boxed{\mathrm{V}^{old}} \arrow[r, "\text{Qmnt}"] & \boxed{\mathrm{V}^{new}}
\end{tikzcd}$$

**Fig. 1.** Maintenance of materialized views

The view maintenance problem has attracted considerable attention recently. Some efforts were made for views defined by SQL queries, with or without aggregation, (e.g. [QW91, GMS93, GL95, RRSS94], and the proceedings of a workshop, which attracted nearly 100 participants, on view maintenance held after SIGMOD96), while some others were directed at the maintenance of recursive (or deductive) views [DT92, HD92, DS95a, DS93, DST95, PI94, DS95b, DLW95, DP97, DS97]. (More details in section on related work.)

### 1.1 The FOIES Framework

In the framework of first-order incremental evaluation systems of recursive queries (FOIES for short), or equivalently the maintenance of recursive views using first-order queries, we consider the maintenance of expensive recursive views using first-order queries. We chose this approach for two main reasons: (i) A recursive view can be maintained within every relational database system if the view is maintainable using first-order queries, since such queries are available in every relational database system. (ii) First-order maintenance algorithms have great potential to be adapted for parallel execution, since they have constant parallel complexity [AHV95]. Besides these practical advantages, such maintenance is also of interest from a mathematical logic perspective and from a descriptive complexity perspective.

Like other approaches to incremental (or on-line) computation, in the FOIES framework some auxiliary relations may also be stored and used in maintaining the views of interest, and the auxiliary relations must also be maintained in the same manner, i.e., derived using first-order queries.

### 1.2 Contributions of This Paper

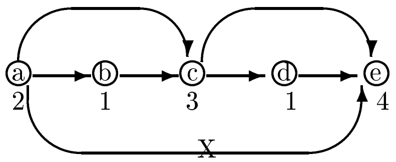
The current paper extends previous work and considers the incremental maintenance of constrained transitive closures of graphs using unions of conjunctive queries. (A conjunctive query is also called a select-project-join query.) The constraints give a useful extension of the transitive closure query. For example, they allow us to refer to node costs such as height and temperature at spatial locations of interest, and to edge costs such as distances between spatial locations. We give complete solutions when constraints are on node costs, and illustrate how to deal with other kinds of constraints. These results are obtained by introducing some new techniques, which were not considered for the unconstrained transitive closure. More specifically, when the constraints are based on node costs, we divide the maintenance problem into several cases, depending on the nature of the constraints. For some cases, the constrained transitive closure becomes bounded in the sense that the recursion can be removed and thus the maintenance problem becomes trivial. For the other cases we provide complete solutions for maintaining the constrained transitive closure. The size of the auxiliary relations can be kept bounded by the size of the transitive closure of the graphs.

It is already a major open problem whether the constraint-free transitive closure of directed graphs can be maintained using first-order queries after the deletion of a single edge [DS95b]. With constrained transitive closure the problem is more complicated; thus in this paper we do not consider the deletion of edges.

Even for the insertion case constrained transitive closures are much more involved than the unconstrained transitive closures. Indeed, unlike transitive closure of the unrestricted case, even very simple constrained transitive closures may not be incremental after insertions. As an example, consider the constrained transitive closure defined by the following:

$$p(x, y) \leftarrow E(x, y)$$
$$p(x, y) \leftarrow p(x, z), E(z, y), c_n(x) < c_n(y)$$

Consider the graph given below, where the horizontal edges are the input graph edges and the numbers below the nodes are their node costs. The two curved edges above the nodes are in the constrained transitive closure but their concatenation (shown by the curved edge below the nodes) is not.




So one part of this investigation is to find incremental auxiliary relations so that we can derive the constrained transitive closures from them. It appears that node-cost constraints are easier than edge-cost constraints.

The technical development of the paper is as follows. We define the notions of node-constraints and edge-constraints and given examples of constrained transitive closures in Section 2. In Section 3 we observe that the conjunction of an arbitrary number of node constraints can be reduced to the conjunction of just one or two such constraints; we then discuss how to maintain the constrained transitive closure of four base cases. In Section 4 we discuss how to deal with other kinds of constraints. In Section 5 we argue that the size of the auxiliary relations can be kept bounded by the size of the transitive closure of the graphs. In Section 6 we discuss related work and open problems on constraint-free transitive closures. Finally in Section 7 we conclude and list some research problems.

## 2 Preliminaries and Examples

For the discussion of constraints, we assume the existence of (a) constants and variables of type *node*, (b) constants and variables of type *real*, (c) two extensional functions, $c_n$ and $c_e$, for specifying node costs and edge costs, and (d) associative built-in functions such as $+$, $\times$, etc.

Examples of edge-cost functions are distance, water-absorption rate, and transport expense. Examples of node-cost functions are height and temperature.

The cost functions are user specified, whereas the built-in functions always have the standard meaning. For simplicity, we will consider the addition function $+$ only; it is easy to see that our results and techniques hold for all associative functions, including multiplication, though they may not apply to non-associative functions such as subtraction $-$.

To define node-cost constraints, we need two kinds of terms:

- *Node-cost terms* are (i) either constants or variables of type real, or (ii) expressions of the form $c_n(v)$, where $v$ is either a constant or variable of type node.
- *Node-cost+ terms* include node-cost terms and are closed under the addition function $+$.

*Primitive node-cost constraints* are expressions of the form $t_1 \theta t_2$, where each $t_i$ is a node-cost term and $\theta$ is one of $<, \leq, =, \geq, >$. *Node-cost constraints* are conjunctions of zero or more primitive node-cost constraints. Node-cost+ constraints are defined similarly.

Edge-cost constraints and edge-cost+ constraints can be defined similarly.

There are two kinds of relation symbols. The extensional relations are used to carry input, whereas the intensional relations are used to carry the computed results when the extensional relations are given. *Atoms* are expressions of the form $q(s_1, \ldots, s_k)$ where $q$ is a $k$-ary relation symbol, and each $s_i$ either a node constant or a node variable. A *Datalog program with constraints* is a finite collection of constrained rules of the form: $A \leftarrow B_1, \ldots, B_m, C$, where $m \geq 1$, each $B_i$ an atom, and $C$ a constraint. We only consider safe programs by requiring that each variable occurring in $A$ or in $C$ must occur in some $B_i$.

We consider several different kinds of constrained transitive closure queries. Suppose the edges of a graph are given in an extensional (or input) relation $E$. Then the constrained transitive closure (TC) of the graph can be specified as the intensional (or output) relation $p$ in Datalog as follows:

$$(*) \qquad \begin{array}{l} p(x,y) \leftarrow E(x,y), C \\ p(x,y) \leftarrow p(x,z), E(z,y), C' \end{array}$$

where $C$ and $C'$ are constraints. If both $C$ and $C'$ are node-cost constraints, then we call the query a *node-cost constrained TC*. We similarly define *node-cost+ constrained TC*. One can also define edge-cost constrained TC and edge-cost+ constrained TC.

### 2.1 Examples of Constrained Transitive Closure

We now give some practical examples of constrained transitive closure.

*Example 1.* Consider the following simplified "trucking" problem: A driver Joe drives a truck from cities to cities and travels on roads. There are bridges on the roads with height clearances. Joe's truck has a height of $h$. When Joe wants to go from one place $u$ to another place $v$, Joe may, at any point along the way, ask whether it is possible to drive the truck to $v$ if a particular freeway entry/exit $x$ is used, i.e., "can Joe drive the truck to $v$ from $x$?" Suppose (i) our nodes are cities, bridges, and entry/exit points of the freeways; (ii) $c_n$ is height of bridges on freeways; and (iii) $E(x, y)$ means there is a segment of roads from $x$ to $y$ without any bridge in between. (For each city and entry/exit points, consider its height as a very large number, say 1000 meters.) Then the query can be expressed as follows:

$$\begin{array}{l} p(x,y) \leftarrow E(x,y), c_n(x) \geq h, c_n(y) \geq h \\ p(x,y) \leftarrow p(x,z), E(z,y), c_n(y) \geq h \end{array}$$

□

*Example 2.* Consider the "rain water flow" problem, where one wishes to determine spatial position pairs $(x, y)$ such that rain water may flow to $y$ from $x$. The node cost function $c_n$ gives the height of spatial positions. Then the query can be expressed as follows:

$$\begin{array}{l} p(x,y) \leftarrow E(x,y), c_n(x) \geq c_n(y) \\ p(x,y) \leftarrow p(x,z), E(z,y), c_n(z) \geq c_n(y) \end{array}$$

□

*Example 3.* Now consider the "budget flier problem." Suppose $c_e$ gives the cost of a leg of a flight. Then the query "find all pairs of $x$ and $y$ where one can fly from $x$ to $y$ with at most 2000 dollars" can be expressed as follows:

$$\begin{array}{l} p_1(x,y,c_e(x,y)) \leftarrow E(x,y), c_e(x,y) \leq 2000 \\ p_1(x,y,u_1+u_2) \leftarrow p_1(x,z,u_1), p_1(z,y,u_2), u_1+u_2 \leq 2000 \\ p(x,y) \leftarrow p_1(x,y,u) \end{array}$$

Note: This transitive closure is still over binary relations. The third attribute only refers to the cost of the edge represented by the other two columns. □

*Example 4.* Continuing from the previous example, one may also wish to consider the maintenance of the "shortest path query," where the cost of each edge is an element of some closed semiring [AHU74]. The query itself can be written using stratified Datalog$^\neg$.

$$
\begin{array}{l}
p_1(x, y, c_e(x, y)) \leftarrow E(x, y) \\
p_1(x, y, u_1 + u_2) \leftarrow p_1(x, z, u_1), p_1(z, y, u_2) \\
longerthansome(x, y, u) \leftarrow p_1(x, y, u), p_1(x, y, v), v < u \\
p(x, y, u) \leftarrow p_1(x, y, u), \neg longerthansome(x, y, u)
\end{array}
$$

□

### 2.2 Some Notations

We now introduce some notations needed in the specification of the incremental programs. These will be given as nonrecursive Datalog programs; given a query predicate, a nonrecursive Datalog program is roughly a union of conjunctive queries. (Note that each such program may define several output relations.) For simplicity we assume for the time being that there are no constraints $C$ and $C'$. After inserting one edge, the new TC can be obtained by adding to the old TC the tuples computed by the following nonrecursive program [DT92]:

$$
\begin{array}{l}
p^n(x, y) \leftarrow E^n(x, y) \\
p^n(x, y) \leftarrow E^n(x, z), p^o(z, y) \\
p^n(x, y) \leftarrow p^o(x, z), E^n(z, y) \\
p^n(x, y) \leftarrow p^o(x, z_1), E^n(z_1, z_2), p^o(z_2, y)
\end{array}
$$

Here $E^n$ represents the newly inserted edge, $p^o$ represents the tuples in the old TC, and $p^n$ represents the new tuples in the new TC. The new TC itself is $p = p^o \cup p^n$. Roughly speaking, this program constructs $p^n$ by finding each pair $(x, y)$ such that (i) it is in the old transitive closure, (ii) it is the newly inserted edge, (iii) it is the result of appending the newly inserted edge to the front (or tail) end of an old path, (iv) it is the result of appending two old paths, one to the left of the newly inserted edge and one to the right. (The program as given was first given in [DT92]; however, many authors observed that TC can be incrementally computed this way earlier. The incremental computation of more general queries, namely the regular chain Datalog queries, was considered in [DST95].)

## 3 Node-Cost Constrained Transitive Closure

In this section we provide complete solutions to the incremental evaluation of node-cost constrained transitive closure queries.

We will first solve the problem for four basic cases, and then reduce the general case to these basic cases. In order to allow these reductions to proceed, we need to consider more general queries of the following form:

$$p_1(x,y) \leftarrow E(x,y), C_1$$
$$p_2(x,y) \leftarrow E(x,y), C_2$$
$$p_3(x,y) \leftarrow p_1(x,y)$$
$$p_3(x,y) \leftarrow p_3(x,z), p_2(z,y), C_3$$

$p(x,y)$ defined from $E$, $p_1$, $p_2$, and $p_3$ using nonrecursive rules.

The basic cases are concerned with situations where $C_3$ is the conjunction of at most two primitive node-cost constraints. For later usage, note that we can derive at most one new tuple for each of $p_1$ and $p_2$ after the insertion of one edge into $E$.

### 3.1 Four Basic Cases

We now consider the four basic cases which are about situations where $C_3$ is the conjunction of at most two primitive node-cost constraints.

**Case A**: There is in $C_3$ exactly one primitive constraint, and which is of the form $c_n(x)\theta c_n(y)$ with $\theta \in \{<, \leq, =, \geq, >\}$. We consider the sub-cases where $\theta$ is one of $\leq$ and $<$, the sub-cases where $\theta$ is one of $>$ and $\geq$ being dual, and where $\theta$ is $=$ being similar. Note that $p_3(x,y)$ is derivable iff $p_1(x,y)$ or the following constrained conjunctive query is true for some $k \geq 0$:

$$(**)\quad \begin{array}{l} p_1(x,z_1), p_2(z_1,z_2), \ldots, p_2(z_{k-1},z_k), p_2(z_k,y), \\ \quad c_n(x)\theta c_n(z_2), \ldots, c_n(x)\theta c_n(z_k), c_n(x)\theta c_n(y) \end{array}$$

We use a ternary auxiliary relation $q$, where $q(x,u,v)$ implies that there is a path $p_2(u,z_1), p_2(z_1,z_2), \ldots, p_2(z_{k-1},z_k)$ $(k \geq 1)$ from $u$ to $v = z_k$ with $c_n(x) \leq c_n(z_i)$ for each $i$. We will maintain the relation $q$ so that $x$ must be a node on some path from $u$ to $v$.

In the following, we will use a shorthand to save space: We will use the function min which selects from its arguments the node having the smallest cost. For example, the following rule

$$q^n(\min(v_1,x_1),u,v) \leftarrow p_2^n(u,v_1), q^o(x_1,v_1,v)$$

can be rewritten as

$$q^n(v_1,u,v) \leftarrow p_2^n(u,v_1), q^o(x_1,v_1,v), c_n(v_1) \leq c_n(x_1)$$
$$q^n(x_1,u,v) \leftarrow p_2^n(u,v_1), q^o(x_1,v_1,v), c_n(v_1) > c_n(x_1)$$

Thus min can be eliminated by rewriting.

Now, $q$ can be incrementally maintained using conjunctive queries:

$$q^n(v,u,v) \leftarrow p_2^n(u,v)$$
$$q^n(\min(v_1,x_1),u,v) \leftarrow p_2^n(u,v_1), q^o(x_1,v_1,v)$$
$$q^n(\min(x_1,v),u,v) \leftarrow q^o(x_1,u,v_1), p_2^n(v_1,v)$$
$$q^n(\min(x_1,v_2,x_2),u,v) \leftarrow q^o(x_1,u,v_1), p_2^n(v_1,v_2), q^o(x_2,v_2,v)$$

Intuitively, the above rules simply concatenate, for each of the possible combinations of old paths and the new edges, to form a possible new path belonging to the $q$ relation. For example, the first rule corresponds to the case where the new $q$ path uses only the new $p_2$ edge, whereas the fourth rule corresponds to the case where the new $q$ path is the concatenation of an old $q$ path, the new $p_2$ edge, and another old $q$ path.

Now we argue that the rules for maintaining $q$ are correct. The first rule is correct since, when $p_2^n(u,v)$ is true, there is a path from $u$ to $v$, namely the path $p_2(u,v)$, such that $c_n(v) \leq c_n(w)$ for each node $w$ on the path except the first node. To see that the second rule is correct, assume $p_2^n(u,v_1)$ and $q^o(x_1,v_1,v)$ are true. Then there is a path from $v_1$ to $v$ such that every node $w$ on this path except $v_1$ satisfies the condition that $c_n(x_1) \leq c_n(w)$. Hence there is a path from $u$ to $v$ such that every node $w$ on this path except $u$ satisfies the condition that $c_n(min(v_1,x_1)) \leq c_n(w)$. The correctness of the other rules can be argued similarly.

Now, $p_3^n$ can be defined from $q$ and $p_1$ as follows:

$$\begin{array}{l} p_3^n(x,y) \leftarrow p_1^n(x,y) \\ p_3^n(x,y) \leftarrow p_1^o(x,u), q^n(x_1,u,y), c_n(x)\theta c_n(x_1) \\ p_3^n(x,y) \leftarrow p_1^n(x,u), q^o(x_1,u,y), c_n(x)\theta c_n(x_1) \\ p_3^n(x,y) \leftarrow p_1^n(x,u), q^n(x_1,u,y), c_n(x)\theta c_n(x_1) \end{array}$$

This is correct because, $q$ contains all triples $(x_1,u,v)$ such that there is a $p_2$-path from $u$ to $v$ where every node $w$ except $u$ satisfies $c_n(x_1) \leq c_n(w)$, and the rule bodies of the above four rules imply that $c_n(x)\theta c_n(x_1)$; combining the above, i.e. $c_n(x_1) \leq c_n(w)$ and $c_n(x)\theta c_n(x_1)$, we see that the condition given in (**) is satisfied.

**Case B**: There is in $C_3$ exactly one primitive constraint, and which is of the form $c_n(x)\theta c_n(z)$ with $\theta \in \{<,\leq,=,\geq,>\}$. This case is similar to Case A. Indeed, again we consider the sub-cases where $\theta$ is one of $\leq$ and $<$. Then $p_3(x,y)$ is derivable iff $p_1(x,y)$ or the following conjunction is true for some $k \geq 0$:

$$\begin{array}{l} p_1(x,z_1), p_2(z_1,z_2), \ldots, p_2(z_{k-1},z_k), p_2(z_k,y) \\ \quad c_n(x)\theta c_n(z_1), c_n(x)\theta c_n(z_2), \ldots, c_n(x)\theta c_n(z_k) \end{array}$$

We use a ternary auxiliary relation $q$, where $q(x,u,v)$ means that there is a path

$$p_2(u,z_1), p_2(z_1,z_2), \ldots, p_2(z_{k-1},z_k)$$

$(k \geq 1)$ from $u$ to $v = z_k$ where $c_n(x) \leq c_n(z)$ for each node $z \in \{u,z_1,\ldots,z_k\}$. Observe that this $q$ is different from the $q$ for case A by requiring the first node to satisfy the inequality a swell. Then $q$ can be incrementally computed using conjunctive queries:

$$\begin{array}{l} q^n(\min(u,v),u,v) \leftarrow p_2^n(u,v) \\ q^n(\min(u,x_1),u,v) \leftarrow p_2^n(u,v_1), q^o(x_1,v_1,v) \\ q^n(\min(x_1,v),u,v) \leftarrow q^o(x_1,u,v_1), p_2^n(v_1,v) \\ q^n(\min(x_1,x_2),u,v) \leftarrow q^o(x_1,u,v_1), p_2^n(v_1,v_2), q^o(x_2,v_2,v) \end{array}$$

Now, $p_3^n$ can be defined from $q$ and $p_1$ using conjunctive queries as follows. We give two ways of doing this. The first way is more concise while the second way is more efficient. Intuitively, they simply concatenate the old paths and the new edges to form the possible new paths belonging to the $p_3$ relation. The second way is more efficient because it avoids the unnecessary ways of using old paths in the concatenations.

For the first way, let $p'$ denote $p^o \cup p^n$ where $p$ is an arbitrary relation name.

The First Way for Maintaining $p_3$

$$p_3^n(x,y) \leftarrow p_1^n(x,y)$$
$$p_3^n(x,y) \leftarrow p_1'(x,z), p_2'(z,y), c_n(x)\theta c_n(z)$$
$$p_3^n(x,y) \leftarrow p_1'(x,u), q'(x_1,u,v), p_2'(v,y), c_n(x)\theta c_n(x_1)$$

The Second Way for Maintaining $p_3$

$$p_3^n(x,y) \leftarrow p_1^n(x,y)$$
$$p_3^n(x,y) \leftarrow p_1^o(x,z), p_2^n(z,y), c_n(x)\theta c_n(z)$$
$$p_3^n(x,y) \leftarrow p_1^n(x,z), p_2^o(z,y), c_n(x)\theta c_n(z)$$
$$p_3^n(x,y) \leftarrow p_1^n(x,z), p_2^n(z,y), c_n(x)\theta c_n(z)$$
$$p_3^n(x,y) \leftarrow p_1^o(x,u), q^o(x_1,u,v), p_2^n(v,y), c_n(x)\theta c_n(x_1)$$
$$p_3^n(x,y) \leftarrow p_1^o(x,u), q^n(x_1,u,v), p_2^o(v,y), c_n(x)\theta c_n(x_1)$$
$$p_3^n(x,y) \leftarrow p_1^o(x,u), q^n(x_1,u,v), p_2^n(v,y), c_n(x)\theta c_n(x_1)$$
$$p_3^n(x,y) \leftarrow p_1^n(x,u), q^o(x_1,u,v), p_2^o(v,y), c_n(x)\theta c_n(x_1)$$
$$p_3^n(x,y) \leftarrow p_1^n(x,u), q^o(x_1,u,v), p_2^n(v,y), c_n(x)\theta c_n(x_1)$$
$$p_3^n(x,y) \leftarrow p_1^n(x,u), q^n(x_1,u,v), p_2^o(v,y), c_n(x)\theta c_n(x_1)$$
$$p_3^n(x,y) \leftarrow p_1^n(x,u), q^n(x_1,u,v), p_2^n(v,y), c_n(x)\theta c_n(x_1)$$

Intuitively, the above rules simply concatenate, for each of the possible combinations of old paths and the new edges, to form a possible new path belonging to the $p_3$ relation. For example, the first rule corresponds to the case where the new $p_3$ path uses only the new $p_1$ edge, the second rule corresponds to the case where the new $p_3$ path uses an old $p_1$ edge and a new $p_2$ edge, and the third rule the fifth rule corresponds to the case where the new $p_3$ path uses an old $p_1$ edge, an old $q$ path and a new $p_2$ edge, etc.

**Case C**: $C_3$ is the conjunction of two primitive constraints, and which are of the form $c_n(x)\theta_1 c_n(y)$ and $c_n(x)\theta_2 c_n(z)$ with $\theta_1, \theta_2 \in \{\leq, <\}$. We use the same ternary auxiliary predicate $q$ as in Case B. Now, $p_3$ is incrementally maintained by:

$$p_3^n(x,y) \leftarrow p_1^n(x,y)$$
$$p_3^n(x,y) \leftarrow p_1^o(x,z), q^n(x_1,z,y), c_n(x)\theta_1 c_n(x_1), c_n(x)\theta_2 c_n(x_1)$$
$$p_3^n(x,y) \leftarrow p_1^n(x,z), q^o(x_1,z,y), c_n(x)\theta_1 c_n(x_1), c_n(x)\theta_2 c_n(x_1)$$
$$p_3^n(x,y) \leftarrow p_1^n(x,z), q^n(x_1,z,y), c_n(x)\theta_1 c_n(x_1), c_n(x)\theta_2 c_n(x_1)$$

Again, intuitively, the above rules simply concatenate, for each of the possible combinations of old paths and the new edges, to form a possible new path belonging to the $p_3$ relation. Their correctness can be argued in a manner similar to case A.

**Case D**: $C_3$ is the conjunction of two primitive constraints, and which are of the form $c_n(x)\theta_1 c_n(y)$ and $c_n(x)\theta_2 c_n(z)$ with $\theta_1, \theta_2 \in \{\geq, >\}$. This is similar to Case C, and can be solved by replacing min (in the construction for Case C) by max, $<$ by $>$, and $\leq$ by $\geq$; thus the details are omitted.

### 3.2 The General Case

Suppose now we are given the following constrained transitive closure query:

$$\begin{array}{l} p(x,y) \leftarrow E(x,y), C_1 \\ p(x,y) \leftarrow p(x,z), E(z,y), C_2 \end{array}$$

where $C_2$ is the conjunction of one or more primitive node-cost constraints. We first transform the above program into the following:

$$\begin{array}{l} p_1(x,y) \leftarrow E(x,y), C_1 \\ p(x,y) \leftarrow p_1(x,y) \\ p(x,y) \leftarrow p(x,z), E(z,y), C_2 \end{array}$$

We can assume that there are no contradictions in the conditions. Indeed, with contradictions in $C_1$ the query always returns the empty answer; and with contradictions in $C_2$, the recursive rule is redundant.

We divide the primitive constraints in $C_2$ into four groups: Let $C_2|_x$ consist of the primitive constraints with $x$ as the only variable; $C_2|_{yz}$ those not involving $x$; $C_2|_{xz}$ those involving both $x$ and $z$; and $C_2|_{xy}$ those involving both $x$ and $y$.

The first two groups can be eliminated by pushing into the nonrecursive predicates. Indeed, we can push $C_2|_x$ into the second rule above, and then push it into the first rule above, resulting the first rule below. We push $C_2|_{yz}$ into $E$, resulting in the second rule below.

$$\begin{array}{l} p_1(x,y) \leftarrow E(x,y), C_1, C_2|_x \\ p_2(z,y) \leftarrow E(z,y), C_2|_{yz} \\ p_3(x,y) \leftarrow p_1(x,y) \\ p_3(x,y) \leftarrow p_3(x,z), p_2(z,y), C_2|_{xy}, C_2|_{xz} \\ p(x,y) \leftarrow E(x,y), C_1 \\ p(x,y) \leftarrow p_3(x,y) \end{array}$$

By removing primitive constraints that can be inferred from the others if necessary, we can assume that $C_2|_{xz}$ and $C_2|_{xy}$ each consists of at most one primitive node-cost constraint. If at least one of them is empty, then the problem is reduced to either the case with no constraint in the recursive rule, or Case A or Case B. We are now left with the case when $C_2|_{xz}$ and $C_2|_{xy}$ are both singletons; assume $C_2|_{xy} = \{c_n(x)\theta_1 c_n(y)\}$ and $C_2|_{xz} = \{c_n(x)\theta_2 c_n(z)\}$. This case is divided into sub-cases covered by the following three tables.

| $\theta_1$ | $\theta_2$ |
|---|---|
| $<, \leq$ | $<, \leq$ |
| $\geq, >$ | $\geq, >$ |

Table I

| $\theta_1$ | $\theta_2$ |
|---|---|
| $<$ | $>, \geq, =$ |
| $\leq$ | $>$ |
| $=$ | $>, <$ |
| $\geq$ | $<$ |
| $>$ | $<, \leq, =$ |

Table II

| $\theta_1$ | $\theta_2$ |
|---|---|
| $\leq$ | $=, \geq$ |
| $=$ | $\leq, =, \geq$ |
| $\geq$ | $\leq, =$ |

Table III

The cases in Table I are covered by Cases C and D.

The maintenance of the constrained transitive closure becomes trivial for cases covered by Table II, because of the following interesting result:

**Proposition 1.** *For the cases in Table II, the original query is bounded.*

*Proof.* We illustrate by considering the second case of Table II. So $C_2|_{xy} = \{c_n(x) \leq c_n(y)\}$ and $C_2|_{xz} = \{c_n(x) > c_n(z)\}$. We now show that there are no paths of length three or longer satisfying the conditions, by unfolding $p_3(x, y)$ twice, with respect to the rules above, to get the following:

$$p_3(x, z), p_2(z, y), c_n(x) \leq c_n(y), c_n(x) > c_n(z)$$
$$p_3(x, z_1), p_2(z_1, z), p_2(z, y), c_n(x) \leq c_n(y), c_n(x) > c_n(z), c_n(x) \leq c_n(z), c_n(x) > c_n(z_1)$$

Observe that both $c_n(x) > c_n(z)$ and $c_n(x) \leq c_n(z)$ are present in the last conjunctive query; so this conjunctive query is unsatisfiable. This implies that the recursive rule defining $p_3$ can be replaced by the following rule

$$p_1(x, z), p_2(z, y), c_n(x) \leq c_n(y), c_n(x) > c_n(z).$$

□

For each case in Table III, we can reduce it to the case covered by Case B. For example, consider the case when $\theta_1$ is $\leq$ and $\theta_2$ is $\geq$. The expansions of $p_3(x, y)$ are exactly the following conjunctive queries:

$$p_1(x, y)$$

and for each positive integer $k$:

$$\begin{aligned} &p_1(x, z_1), p_2(z_1, z_2), \ldots, p_2(z_{k-1}, z_k), p_2(z_k, y), \\ &\qquad c_n(x) \leq c_n(z_2), \ldots, c_n(x) \leq c_n(z_k), c_n(x) \leq c_n(y), \\ &c_n(x) \geq c_n(z_1), \ldots, c_n(x) \geq c_n(z_k) \end{aligned}$$

Obviously the above conjunctive query is equivalent to

$$\begin{aligned} &p_1(x, z_1), p_2(z_1, z_2), \ldots, p_2(z_{k-1}, z_k), p_2(z_k, y), \\ &c_n(x) = c_n(z_2), \ldots, c_n(x) = c_n(z_k), c_n(x) \leq c_n(y), c_n(x) \geq c_n(z_1) \end{aligned}$$

Thus the old definition of $p_3$ is equivalent to the following:

$$
\begin{aligned}
&p_4(x,y) \leftarrow p_2(x,y)\\
&p_4(x,y) \leftarrow p_4(x,z), p_2(z,y), c_n(x) = c_n(z)\\
&p_3(x,y) \leftarrow p_1(x,y)\\
&p_3(x,y) \leftarrow p_1(x,z), p_2(z,y), c_n(x) \geq c_n(z), c_n(x) \leq c_n(y)\\
&p_3(x,y) \leftarrow p_1(x,z_1), p_2(z_1,z_2), p_4(z_2,y), c_n(x) \geq c_n(z_1),\\
&\qquad\qquad c_n(x) \leq c_n(y), c_n(x) = c_n(z_2)
\end{aligned}
$$

This is now covered by Case B. (Observe that $p_4$ is the new recursive predicate.)

## 4 Other Constraints

In this section we consider transitive closures with other kinds of constraints. Unlike node-cost constraints, we do not have complete solutions; so we illustrate with examples. We also give examples which do not currently have solutions.

### 4.1 Constraints with Arithmetics

*Example 5.* We consider the situation when there is exactly one primitive constraint in the recursive rule, and which is of the form $c_n(x) + c_n(y) < c_n(z)$:

$$
\begin{aligned}
&p(x,y) \leftarrow E(x,y)\\
&p(x,y) \leftarrow p(x,z), E(z,y), c_n(x) + c_n(y) < c_n(z)
\end{aligned}
$$

We will use $q(u,v,y)$ to mean the following: there is a path $E(z_1,z_2),\ldots,E(z_{k-1},z_k)$ from $v = z_1$ to $y = z_k$ such that $u + c_n(z_{i+1}) \leq c_n(z_i)$ for each $1 \leq i < k$.

Rules for maintaining $q$ are:

$$
\begin{aligned}
&p_0^n(c_n(v) - c_n(y), v, y) \leftarrow E^n(v,y), c_n(y) \leq c_n(v)\\
&q^n(u,v,y) \leftarrow p_0^n(u,v,y)\\
&q^n(\min(u_1,u_2), v, y) \leftarrow q^o(u_1,v,y_1), p_0^n(u_2,y_1,y)\\
&q^n(\min(u_1,u_2), v, y) \leftarrow p_0^n(u_1,v,y_1), q^o(u_2,y_1,y)\\
&q^n(\min(u_1,u_2,u_3), v, y) \leftarrow q^o(u_1,v,y_1), p_0^n(u_2,y_1,y_2), q^o(u_3,y_2,y)
\end{aligned}
$$

The min function used here is different from the min function used in the previous section; it returns the minimal value among its arguments. As before, this use of min can also be eliminated by rewriting.

Rules for maintaining $p$ are:

$$
p^n(x,y) \leftarrow \tau_1(x,v), \tau_2(u,v,y), c_n(x) < u
$$

where $\tau_1 \in \{E^n, E^o, p^o\}$ and $\tau_2 \in \{q^n, q^o\}$ with at least one $\tau_i$ superscripted $n$. $\square$

### 4.2 Edge-Cost Constraints

*Example 6.* We now illustrate how to deal with edge-cost constraints using the budget flier example, which is "Find all city pairs $x$ and $y$ where one can fly from $x$ to $y$ with up to 2000 dollars."

$$
\begin{aligned}
&p_1(x,y,c_e(x,y)) \leftarrow E(x,y), c_e(x,y) \leq 2000 \\
&p_1(x,y,u_1+u_2) \leftarrow p_1(x,z,u_1), p_1(z,y,u_2), u_1+u_2 \leq 2000 \\
&p(x,y) \leftarrow p_1(x,y,u)
\end{aligned}
$$

It suffices to see that $p_1$ can be incrementally maintained using conjunctive queries. (Note that there may exist, for some $x$ and $y$, $p_1(x,y,u_1)$ and $p_1(x,y,u_2)$ where $u_1 \neq u_2$. However, to ensure correct maintenance of $p$, we only need the minimum such $u$.) Indeed, this can be done using the following:

$$
\begin{aligned}
&p_1^n(x,y,c_e(x,y)) \leftarrow E^n(x,y), c_e(x,y) \leq 2000 \\
&p_1^n(x,y,u_1+c_e(z,y)) \leftarrow p_1^o(x,z,u_1), E^n(z,y), u_1+c_e(z,y) \leq 2000 \\
&p_1^n(x,y,c_e(x,z)+u_2) \leftarrow E^n(x,z), p_1^o(z,y,u_2), c_e(x,z)+u_2 \leq 2000 \\
&p_1^n(x,y,u_1+c_e(z_1,z_2)+u_2) \leftarrow p_1^o(x,z_1,u_1), E^n(z_1,z_2), \\
&\qquad\qquad p_1^o(z_2,y,u_2), u_1+c_e(z_1,z_2)+u_2 \leq 2000
\end{aligned}
$$

□

When edge-cost constraints are only tested on edges then the corresponding transitive closures are easy to maintain. When such constraints are applied to constructed "edges" (which in reality are just pairs of nodes), the corresponding transitive closures may be hard to maintain. Indeed, we do not currently have a solution to the incremental maintenance of edge-cost constrained transitive closure of the following kind:

$$
\begin{aligned}
&p(x,y) \leftarrow E(x,y) \\
&p(x,y) \leftarrow p(x,z), E(z,y), c_e(x,z) < c_e(z,y)
\end{aligned}
$$

### 4.3 Shortest Path and Other Possibilities

The "shortest path" transitive closure can be maintained in *first order* (instead of just conjunctive queries): For each pair of nodes, we maintain the minimum cost among all paths between this pair of nodes.

Transitive closure with constraints involving both edge costs and node costs are also possible in practical situations and thus deserve further attention. For example, is the voltage at one position high enough (node cost) so that the electricity can reach another position at a voltage high enough, assuming different segments of the power line decrease the voltage at different rates (edge cost)?

## 5 Bound on the Size of the Auxiliary Relations

In the discussion so far no effort was made to minimize the size of the auxiliary relations. It is easy to see that, for the cases where we give solutions to the maintenance of the constrained transitive closure, the size of the auxiliary relations can be kept bounded by the size of the transitive closure, and that this can be achieved with very little effort.

Indeed, fix a constrained transitive closure program. As argued in Section 3.2, the maintenance of the query predicate can be reduced to the maintenance of several auxiliary predicates corresponding to one or more of the four cases in Section 3.1. Clearly, for those binary auxiliary relations the size of the auxiliary relations can be kept bounded by the size of the transitive closure. Let $q$ be any of the ternary auxiliary relations. It can be easily observed that, for each pair of nodes $x$ and $y$, one only needs to keep one tuple $(z, x, y)$ in each auxiliary relation. That is, the second and third columns form a relation key for each auxiliary relation. For example, consider case A and assume $\theta$ is $<$: If $q(u_1, x, y)$ and $q(u_2, x, y)$ are both true, then we can simply remove the $q(u_i, x, y)$ where $c_n(u_i) > c_n(u_j)$ and $j \in \{1, 2\} - \{i\}$. (If $c_n(u_i) = c_n(u_j)$ then we may need to maintain a total order [DS95b] on the set of nodes and delete the node which is smaller in the total order.)

Hence the size of each auxiliary relation can indeed be kept bounded by the size of the transitive closure. It should be pointed out that to achieve this goal we need to use full first-order logic (because we need to use negation), instead of unions of conjunctive queries.

## 6 Related Work and Open Problems

We now review related work, including previous results and current open problems for maintaining views in first-order.

Major positive results on the maintenance of recursive database views using first-order queries include the following: (A) Transitive closure of graphs can be maintained in first-order (i) after the insertions of edges [DT92], (ii) after the deletion of edges if the graphs are acyclic [DS95a], and (iii) after inserting or deleting edges if the graphs are undirected [PI94] using binary auxiliary relations [DS95b]. (B) All views defined by regular chain Datalog programs can be thus maintained after inserting edges [DT92], and this result was generalized to allow certain kind of initializations to the nonrecursive predicates of the regular chain Datalog programs [DS93, DST95]. In [DLW95] the maintenance of recursive views using SQL and other nonrecursive query languages is considered.

Major negative results in this framework are concerned with the amount of auxiliary relations one needs to maintain to help maintain the views of interest. In [DS95b] it was shown that transitive closure of undirected graphs cannot be maintained in first-order using only unary auxiliary relations.

Major open problems include the following: Can one maintain in first-order the transitive closure of directed graphs, using auxiliary relations, after the deletion of edges? Can one maintain the same-generation view, using auxiliary relations, after the insertion of tuples? For more open problems one can see [PI94, DS95b, DS97]

Other work on the maintenance of recursive views (e.g. [AP87, Kuc91, HD92, GMS93]) considered using recursive queries, counting, or while-loops in maintenance algorithms. Our framework only uses relational queries as such algorithms.

Besides related work on view maintenance, there are other related work which considers other kinds of on-line algorithms for graph problems. Concerned with the design of efficient general algorithms instead of with databases, these algorithms typically use non-relational data structures such as queues or linked lists, use invented values such as integers, or use the iteration constructs such as while-loops and for-loops (see [DS95a] for references to such work). A more general complexity-theoretical framework for on-line or incremental computation is reported in [MSVT94]. Motivated from the view maintenance problem of databases, our approach uses only relations and has none of the above non-logical features. A comparison of our approach with other methods for the incremental maintenance of database views is given in [DST95].

The constrained transitive closures considered in this paper are special cases of the "generalized transitive closure" studied in [Agr87, CN89, DA93]. Although only treated in passing, [DA93] recognized the usefulness of incrementally computing costs associated with paths from the components of these paths.

## 7 Concluding Remarks and Future Work

We have presented unions of conjunctive queries for the maintenance of constrained transitive closure where the constraints are on the node cost, and we have illustrated how the constrained transitive closure can be maintained when the constraints involve arithmetics or are on edge costs.

This work extends our earlier work on the maintenance of database views, including transitive closure of directed and undirected graphs, using relational queries after the insertion or deletion of one tuple. Our work is related (a) to the maintenance of relational views such as in [QW91, GL95] and of recursive views in [GMS93], and (b) to the maintenance of transitive closure of graphs using general purpose algorithms (see [DST95] for references on research in this direction), although their algorithms may not be readily usable in relational database systems.

There are several possible directions for future work. (i) Consider the deletion of edges. As mentioned earlier, it is open if one can maintain in first order the constraint-free transitive closure of directed graphs after the deletion of a single edge. So it is perhaps hard to get a generic maintenance result for the constrained transitive closure. However, it is possible to get such maintenance results for some combinations of special types of constraints and special types of graphs. (ii) Consider the maintenance after the cost functions are changed. (iii) Consider obtaining negative results. It is possible that some negative results can be proven in this framework which cannot be proven for the unconstrained transitive closure. So this could be a useful avenue for further theoretical work. (iv) Compare the cost of computing views from scratch and the cost of view maintenance algorithms as proposed in this paper. We believe that each of the two approaches may be better than the other under certain situations, depending on the size of the update vs the size of the base relations (and the views) and on the data distribution.

## Acknowledgement

The authors wish to thank James Harland for interesting discussions on constraints, and the anonymous referees for their very helpful comments. This work was partially supported by the Centre for Intelligent Decision Systems and CRC, and by the Australian Research Council through research grants.

## References


[Agr87] Rakesh Agrawal. Alpha: An extension of relational algebra to express a class of recursive queires. In *Proc. IEEE 3rd International Conference on Data Engineering*, pages 580–590, 1987.

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[AP87] K. R. Apt and J.-M. Pugin. Maintenance of stratified databases viewed as a belief revision system. In *Proc. Sixth ACM Symp. on Principles of Database Systems*, pages 136–145, San Diego, Ca., 1987.

[CN89] I.F. Cruz and T.S. Norvell. Aggregative transitive closure: An extension of transitive closure. In *Proc. IEEE 5th International Conference on Data Engineering*, pages 384–393, 1989.

[DA93] Shaul Dar and Rakesh Agrawal. Extending sql with generalized transitive closure. *IEEE Transactions on Knowledge and Data Engineering*, 5(5):799–812, October 1993.

[DLW95] G. Dong, L. Libkin, and L. Wong. On impossibility of decremental recomputation of recursive queries in relational calculus and SQL. In Paolo Atzeni and Val Tannen, editors, *Proceedings of the Fifth International Workshop on Database Programming Languages – http://www.springer.co.uk/eWiC/Workshops/DBPL5.html*, pages 1–15, 1995.

[DP97] G. Dong and C. Pang. Maintaining transitive closure in first-order after node-set and edge-set deletions. *Information Processing Letters*, 62(3):193–199, 1997.

[DS93] G. Dong and J. Su. First-order incremental evaluation of datalog queries. In *Proc. of 4th International Workshop on Database Programming Languages, C. Beeri, A. Ohori and D. Shasha (Eds)*, pages 295–308. Workshop in Computing Series, Springer-Verlag, 1993.

[DS95a] G. Dong and J. Su. Incremental and decremental evaluation of transitive closure by first-order queries. *Information and Computation*, 120(1):101–106, July 1995.

[DS95b] G. Dong and J. Su. Space bounded FOIES. In *Proc. of the ACM Symposium on Principles of Database Systems*, pages 139–150, May 1995.

[DS97] G. Dong and J. Su. Arity Bounds in First-Order Incremental Evaluation and Definition of Polynomial Time database Queries. *Journal of Computer and System Sciences*, to appear. Special issue for PODS 95.

[DST95] G. Dong, J. Su, and R. Topor. Nonrecursive incremental evaluation of datalog queries. *Annals of Mathematics and Artificial Intelligence*, 14(2-4):187–223, 1995. Festschrift in honor of Jack Minker.

[DT92] G. Dong and R. Topor. Incremental evaluation of datalog queries. In *Proc. Int'l Conference on Database Theory*, pages 282–296, Berlin, Germany, October 1992.

[GL95] T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1995.

[GMS93] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 157–166, 1993.

[HD92] J. V. Harrison and S. W. Dietrich. Maintenance of materialized views in a deductive database: An update propagation approach. In K. Ramamohanarao, J. Harland, and G. Dong, editors, *Proceedings of the JICSLP Workshop on Deductive Databases*, pages 56–65, November 1992.

[Kuc91] V. Kuchenhoff. On the efficient computation of the difference between consecutive database states. In C. Delobel, M. Kifer, and Y. Masunaga, editors, *Proc. Second Int. Conf. on Deductive Object-Oriented Databases*, volume 566 of *Lecture Notes in Computer Science, Springer-Verlag*, pages 478–502. Springer-Verlag, 1991.

[MSVT94] P. B. Miltersen, S. Subramanian, J. S. Vitter, and R. Tamassia. Complexity models for incremental computation. *Theoretical Computer Science*, 130(1):203–36, August 1994.

[PI94] S. Patnaik and N. Immerman. Dyn-FO: A parallel dynamic complexity class. In *Proc. ACM Symp. on Principles of Database Systems*, pages 210–221, 1994.

[QW91] Xiaolei Qian and Gio Wiederhold. Incremental recomputation of active relational expressions. *IEEE Trans. on Knowledge and Data Engineering*, 3(3):337–341, 1991.

[RRSS94] R. Ramakrishnan, K. A. Ross, D. Srivastava, and S. Sudarshan. Efficient incremental evaluation of queries with aggregation. In *Int. Logic Programming Symp.*, 1994.