ELSEVIER The Journal of Logic Programming 41 (1999) 129–137

THE JOURNAL OF LOGIC PROGRAMMING

www.elsevier.com/locate/jlpr

Technical Note

On the Floyd–Warshall algorithm for logic programs

Christos Papadimitriou [a,*], Martha Sideri [b]

[a] *Division of Computer Science, U.C. Berkeley, Berkeley CA 94720, USA*
[b] *Athens University of Economics and Business, Athens, Greece*

Received 19 December 1997; received in revised form 2 February 1999; accepted 10 February 1999

Abstract

We explore the possibility of evaluating single-rule Datalog programs efficiently and with logarithmic work space by a natural extension of the Floyd–Warshall algorithm for transitive closure. We characterize exactly the single rule chain programs that can be so evaluated – they are rather modest generalizations of the transitive closure. The proof relies on an interesting language-theoretic concept, *total ambiguity*. Extensions to more general classes of programs, and more general algorithms, are discussed. © 1999 Published by Elsevier Science Inc. All rights reserved.

Keywords: Floyd–Warshall algorithm; Logic programs; Datalog programs; Total ambiguity

1. Introduction

A *Datalog program* is a recursive method for defining functions from relations to relations; for example, the program below defines the transitive closure of a relation T_0

$$T(x,y) \leftarrow T(x,z), T(z,y). \tag{1}$$

In this paper we shall be interested in such *single-rule Datalog programs*, *sirups* [1] for short. Sirups are important because they are extremely simple, and still they capture the diversity and complexity of datalog. We are assuming that the defined relation T in the sirup is initialized to a given initial value T_0 (in this case the graph whose transitive closure is being computed); the clause $T(x,y) \leftarrow T_0(x,y)$ will always be implicit. It is well-known that any sirup with a body containing k variables can be evaluated

[*] Corresponding author. Tel. +1-510-642-1559; fax: +1-510-642-5775; e-mail: christos@cs.berkeley.edu.
[1] Sirups were first introduced and studied in the work of our unforgettable friend Paris C. Kanellakis; we dedicate this note to his memory.

in $O(n^k)$ time (in the example above, in $O(n^3)$ time, where n is the number of constants in the given relation T_0) by the so-called *semi-naive algorithm*, a Horn-clause decision procedure applied to the Horn formula implicit in the rule-goal graph of the ground atoms [6,1]. During the past fifteen years, a central problem in Database Theory has been to find more efficient specialized algorithms for large classes of Datalog programs – see for example Refs. [3–5,2,1].

If a Datalog program is *first-order* or *nonrecursive* (that is, if it does not mention in the right-hand side any relation that occurs on the left-hand side), then it can be computed by a simple iterative algorithm using logarithmic space [1]. The semi-naive algorithm for Datalog programs is a more complex algorithm, relying on data structures that require $O(n^k)$ space, in addition to the relational store. In this paper we consider an intriguing extension of first-order programs which can be evaluated by a different, but also very simple, iterative algorithm.

The transitive closure program (1) can be evaluated by the (rather unDatalog-like) *Floyd–Warshall algorithm* [7,8] as follows:

```
The Floyd–Warshall Algorithm
$T := T_0$;
let the universe be $\{1, 2, \ldots, n\}$;
for $z := 1 \ldots n$ do
  for $x := 1 \ldots n$ do
   for $y := 1 \ldots n$ do
   if $T(x,z)$ and $T(z,y)$ then add $(x,y)$ to $T$
```

In fact, it is easy to see that the order in which variables x and y are considered does not matter, *as long as variable z increases monotonically in the outer loop.*

In this paper we take the following more general view of the Floyd–Warshall algorithm, applicable to any sirup. Let m be the number of variables in the sirup. Suppose that we order all m-tuples of constants in the universe in a particular order $\prec$ (independent of the relation T_0, and depending only on the program and the number of constants n), and then run the following algorithm:

```
The Floyd–Warshall Algorithm for Datalog sirups
$T := T_0$;
for each $m$-tuple of database constants $t$ in the order $\prec$ do
  if $t$ satisfies the body of the rule, then add $t'$ to $T$,
```

where by t' we mean t with the free variables (z in our transitive closure example) omitted. Notice that, for sirup (1), and if we take $\prec$ to be *any order that lists the triples in nondecreasing z* (the *second* index), this algorithm correctly evaluates the sirup. Obviously, for other sirups and choices of $\prec$, this algorithm may fail to compute the correct semantics for T. Thus, given any Datalog sirup, it is an interesting question whether an order exists for which the Floyd–Warshall algorithm succeeds in correctly computing T, for all T_0. If such an order exists, we say that the sirup *has the Floyd–Warshall property*. For example, the transitive closure sirup (1) does have the Floyd–Warshall property. In contrast, its slight variant shown below – also computing the transitive closure of T_0 – fails to have the Floyd–Warshall property (and so does the symmetric one, with T first and then T_0):

$$T(x,y) \leftarrow T_0(x,z), T(z,y). \tag{2}$$

To see why, suppose that $\prec$ orders $(0,1,3)$ before $(1,2,3)$ (for example, rename constants so that $(0,1,3)$ is the first triple of distinct constants enumerated by $\prec$), and consider the case in which T_0 is the path $(0,1,2,3)$: The Floyd–Warshall algorithm will fail to insert the edge $(0,3)$.

Although the Floyd–Warshall algorithm has the same asymptotic complexity as the semi-naive one – $O(n^m)$ – it does have certain distinct advantages. It is *extremely* simple, the constant in the O-notation is essentially one, and logarithmic *local* space is required (in addition to the relational store) – to be precise, only the work space needed to enumerate the tuples in the order $\prec$. It would be extremely interesting if this simple idea could be generalized to evaluate large classes of sirups.

In this paper we prove results strongly suggesting that, unfortunately, the applicability of the Floyd–Warshall idea to evaluating Datalog programs is quite limited. That is, our main results are essentially negative. We give a characterization of all *chain sirups* (an interesting special class of sirups first studied in Ref. [2]) that can be solved by the Floyd–Warshall algorithm (Theorem 1). They are precisely those, whose underlying context-free grammar (see Ref. [2] and Section 2 for the relationship between chain sirups and context-free grammars) is of the form $T \to T(AT)^+$, where A contains only database relations – that is to say, they are rather modest generalizations of the transitive closure program above. Our proof relies on an interesting and novel language-theoretic concept, *total ambiguity.*

There are obstacles in extending our characterization to more general sirups. We conjecture that there are no sirups with *binary* head with the Floyd–Warshall property, other than the chain ones characterized in Theorem 1. Our attempts at a proof of this conjecture encounter the usual ‘homomorphism’ complications that haunt many Datalog proofs [2]. It can be shown that no *unary* sirup, other than the first-order ones, can have the Floyd–Warshall property (Proposition 2). However, if we go beyond binary sirups, there are some other families of straightforward extensions of the transitive closure that do have the Floyd–Warshall property; we conjecture that there are no others.

In the next section, after the necessary definitions, we show our main result, the characterization of single-rule chain programs that have the Floyd–Warshall property. In Section 3 we discuss the possible extensions of this result.

2. Chain rules

2.1. Definitions

Let Σ be a vocabulary of relational symbols, with arity $r : \Sigma \mapsto \mathbf{N}$, and let X be a set of variables. An *atom* is an object of the form $T(\bar{x})$, where $T \in \Sigma$ and $\bar{x} \in X^{r(T)}$. A *Datalog rule* is an object of the form $a_0 \leftarrow a_1, \dots, a_k$, where all a_i's are atoms. A *Datalog program* is a nonempty, finite set of Datalog rules. In a Datalog program, the *defined relations* are those that appear in some a_0 atom, while the *database relations* are all other elements of Σ appearing in the program. A *single-rule program*, or *sirup*, is a Datalog program with one rule.

Let U be a nonempty, finite universe of constants (U may be assumed to always be $\{1,2\ldots,|U|\}$), and for each $T \in \Sigma$ let T_0 be a subset of $U^{r(T)}$. If $f : X \mapsto U$, $T \in \Sigma$, $\hat{T} \subseteq U^{r(T)}$, and $a = T(x_1,\ldots,x_k)$ is an atom, then we say that $\hat{T}$ *satisfies a with f* if $(f(x_1),\ldots,f(x_k)) \in \hat{T}$. The *semantics* of a Datalog program is defined in terms of a relation $\hat{T} \subseteq U^{r(T)}$ for each $T \in \Sigma$, such that (a) $T_0 \subseteq \hat{T}$ for each $T \in \Sigma$; (b) for each $f : X \mapsto U$, if $a_0 \leftarrow a_1,\ldots,a_k$ is a rule of the program, and the $\hat{T}$'s satisfy all atoms $a_1,\ldots,a_k$ with f, then they also satisfy a_0 with f; and (c) each of the $\hat{T}$'s is minimal among all relations satisfying (a) and (b). For example, the semantics of the program displayed in the introduction is precisely the transitive closure of the directed graph T_0.

We next define an important syntactic subclass of Datalog programs, first studied in Ref. [2]. A Datalog rule $a_0 \leftarrow a_1,\ldots,a_k$ is a *chain rule* if for some sequence of *distinct* variables $x_0,\ldots,x_k \in X^{k+1}$, a_0 is of the form $T(x_0,x_k)$, while for $i \geqslant 1$ a_i is of the form $T_i(x_{i-1},x_i)$. A *chain program* is a Datalog program with only chain rules. The transitive closure program in the introduction is a single-rule chain program (a chain sirup).

Finally, consider the following attempt at evaluating sirups: For a fixed sirup π with k variables, we have an algorithm $\prec$ that enumerates k-tuples of $U = \{1,2,\ldots,n\}$ – presumably, very efficiently in time and space. The *Floyd–Warshall algorithm* for π and $\prec$ is then the one displayed in the introduction. We say that sirup π has the *Floyd–Warshall property* if there is a $\prec$ such that for every T_0 the Floyd–Warshall algorithm correctly computes its semantics $\hat{T}$. For example, all nonrecursive programs have the Floyd–Warshall property, with any $\prec$; and it is a classical observation due to Floyd and Warshall independently [7,8], that the transitive closure program (1) has the Floyd–Warshall property, with $\prec$ being any enumeration of the triples of nodes in nondecreasing *second* index. Thus, the Floyd–Warshall property is an intriguing extension of first order.

If $\pi = T(x_0,x_k) \leftarrow T_1(x_0,x_1),\ldots,T_k(x_{k-1},x_k)$ is a chain sirup, then its *underlying context-free grammar* denoted $G(\pi)$ consists of the two rules $T \rightarrow T_1 \cdots T_k$ and $T \rightarrow T_0$, where T_0 is the initializing relation in the definition of the semantics of π. For example, the transitive closure program in the introduction has this underlying grammar: $T \rightarrow TT, T \rightarrow T_0$.

2.2. The main result

Chain sirups are a benign class of Datalog programs, which exhibit very limited homomorphisms, and are thus easier to understand and characterize. The context-free connection is very important in such investigations. For example in Ref. [2] the parallel complexity of chain sirups was explored and characterized exactly, by using a novel language-theoretic concept called *the polynomial stack property*. In this section we shall characterize exactly the class of chain sirups that have the Floyd–Warshall property by relying on another language-theoretic concept called *total ambiguity*.

Let G be a context-free grammar. A *sentential form* of G is a string derived, following the rules of the grammar, from the start symbol. Call a sentential form of G *nontrivial* if it has a parse tree with at least two nodes of outdegree greater than one (that is, if it contains at least two expansions of nonterminals). We say that a context-free grammar is *totally ambiguous* if each nontrivial sentential form has at least two distinct parse trees. For example, the grammar $T \rightarrow TT|T_0$ which corresponds to the transitive closure program is totally ambiguous. Any nontrivial sentential form has at least two parse trees – for example TTT can be parsed by expanding either the first or the second T of TT. In contrast, the equivalent grammars $T \rightarrow T_0T|T_0$ and $T \rightarrow TT_0|T_0$ are not totally ambiguous. Similarly, the grammars $T \rightarrow TTT|T_0$ and $T \rightarrow TT_0T|T_0$ are totally ambiguous, whereas $T \rightarrow T_0TT|T_0$ is not – the nontrivial sentential form T_0T_0TTT has only one parse tree.

It is easy to see that this strong variant of ambiguity is decidable:

Proposition 1. *For any sirup π, grammar $G(\pi)$ is totally ambiguous if and only if each parse tree with exactly two nodes of outdegree greater than one has an alternative parse tree.*

Proof. The *only if* direction follows trivially from the definition of total ambiguity. For the *if* direction, an alternative tree of any parse tree containing more than two nodes of outdegree greater than one can be obtained by varying its top-level subtree. □

We can now state and prove our main result.

Theorem 1. *Let π be a single-rule chain program that is not first-order (that is, T appears on the rhs of the rule). Then the following are equivalent:*

(a) *π has the Floyd–Warshall property.*

(b) *$G(\pi)$ is totally ambiguous.*

(c) *$G(\pi)$ is of the form $T \rightarrow T(DT)^k$, where $k \geqslant 1$, and D is a string (possibly empty) of database predicates (that is, with no occurrences of T).*

Proof. (a) *implies* (b): Suppose for the sake of contradiction that π has the Floyd–Warshall property, say with order $\prec$, and yet $G(\pi)$ is not totally ambiguous. Then by the proposition there is a parse tree with a single expansion that has no alternative parse tree, say corresponding to the derivation $T \rightarrow ATB \rightarrow AATBB$, where $G(\pi)$ is $T \rightarrow ATB$ for some $A, B \in \Sigma^*$. We shall create, by an adversary argument, initial values for the database relations and for T_0 that falsify the Floyd–Warshall algorithm. The construction is a generalization of the argument given after Eq. (2). Let the universe U be the set $\{0, 1, \ldots, |AATBB|\}$, and let $k = |ATB|$. Appropriately rename the elements of U so that the $k+1$-tuple $t = (|A|, |A|+1, \ldots, |A|+k)$ is enumerated by $\prec$ *after* the $k+1$-tuple $t' = (0, 1, \ldots, |A|, |A|+k, |A|+k+1, \ldots, |AATBB|)$ (this can be achieved by renaming $(0, 1, \ldots, |A|, |A|+k, |A|+k+1, \ldots, |AATBB|)$ the first $k+1$-tuple of $\prec$ that has distinct elements). For each relation $R \in \Sigma$, the initial relation R_0 contains precisely the pairs $(i-1, i) \in U^2$ such that the ith symbol of $AATBB$ is an R. Then the correct value of $\hat{T}$ contains the tuples in $T_0 \cup \{(0, |AATBB|), (|A|, |A|+k)\}$, while the Floyd–Warshall algorithm will compute $T_0 \cup \{(|A|, |A|+k)\}$. That is, the Floyd–Warshall algorithm will fail to notice that $(0, |AATBB|)$ should also be included in T, because the tuple t' above is enumerated by $\prec$ *before* the pair $(|A|, |A|+k)$ is inserted. Hence the Floyd–Warshall algorithm fails to correctly compute $\hat{T}$ in this case, and the proof of this direction is complete.

(b) *implies* (c): This part is by string calculus. Suppose that we are given a sirup with rule $T \rightarrow R$. Since the grammar is totally ambiguous, there are at least two occurrences of T in R. It is easy to see that, without loss of generality, we can write the rule as $T \rightarrow ATBTC$, where we assume that A contains only database predicates, and the alternative parse of $ARBTC$ (which must exist, since $G(\pi)$ is totally ambiguous) is $ATBRC$. Then we must have the following string equality:

$$AATBTCBTC = ATBATBTCC.$$

For this equality to hold, A must be empty (since A has been assumed to contain no occurrences of T), and also $CBT = BTC$. The latter equation is of the form $XY = YX$, and it is well-known (see, for example, Lemma 6(i) in Ref. [2]) that this equation has a solution only if X and Y are both powers of the same string. If C is empty, then the rule is $T \to TBT$. If not, define DT to be the shortest string such that $C = (DT)^i$ for some i; that is, DT is not a nontrivial power of any string. Then BT will also be a power of the same string. We conclude that, in both cases, the rule is of the form $T \to T(DT)^k$, for some $k > 0$, where DT is not a proper power of any string.

It remains to show that D contains no occurrence of T. Suppose that it does contain such occurrences, consider the leftmost occurrence of T in D, and write $D = XTY$. Then there must be an alternative parse of $TXRYT(DT)^{k-1}$. There are two cases: (i) This alternative parse involves expanding an occurrence of T not within Y, that is, it starts with $TXTYT$. Then TXR has $TXTYT$ as a prefix, and thus TYT is a prefix of R. But this can happen only if YT is a prefix of $XTYT$, or $DT = YTXT = XTYT$. As we mentioned above, such equations imply that YT and XT are both powers of the same string, contradicting our assumption that DT is the shortest string such that $C = (DT)^i$. Or (ii) we can write $Y = WTZ$, and $TXRYT(DT)^{k-1} = TXTWRZT(DT)^{k-1}$, and thus $(DT)^k WT = WT(DT)^k$. Since WT was supposed to be a proper substring of DT, we conclude again that DT is a nontrivial power of WT, a contradiction.

(c) *implies* (a): We shall show this for grammars of the form $T \to T^{k+1}$ for $k \geqslant 1$; the generalization to $T \to T(DT)^k$ with $|D| = m \geqslant 1$ is straightforward, by replacing "constants" by "$m+1$-tuples of constants related by D" in the argument below.

Consider then the sirup

$$T(x,y) \leftarrow T(x,z_1), T(z_1,z_2), \ldots, T(z_{k-1},z_k), T(z_k,y).$$

It computes all paths in T_0 of length 1 mod k. We assume that the constants are $\{1,2,\ldots,n\}$. Consider all k-tuples of constants in increasing order of the smallest constant appearing in them. That is, first come all k-tuples containing a 1, then all k-tuples containing a 2 but no 1, and so on. Among all k-tuples containing constants i and above, we list those that start with an i first. Call this order $\prec$. We claim that, with this $\prec$, the Floyd–Warshall algorithm correctly evaluates this sirup.

Specifically, we shall show the following inductive statement. If there is a path of length 1 mod k between two nodes in T_0, then, after the i first stages of the Floyd–Warshall algorithm – that is, once all k-tuples that contain any constant in $\{1,2,\ldots,i\}$ have been generated – there is such a path in T between the same nodes that contains no constants in $\{1,2,\ldots,i\}$. Notice that this inductive statement proves the correctness of the Floyd–Warshall algorithm. After n stages, there is a direct edge between any nodes connected, in T_0, by a path of length 1 mod k.

Consider a path of length 1 mod k just before the ith stage, say

$$a_0 a_1 a_2 \cdots a_{p-1} i \cdots i a_{p+1} a_{p+2} \cdots a_n,$$

where $0 < p < n$, $a_j > i$ for all $0 < j < n$ (that is, the endpoints can be i), and there are $c+1$ appearances of i between a_{p-1} and a_{p+1}. Notice that if $c < 1$ – that is, if the path has either no appearance of i, or just one, as internal nodes – then we are done with either no shortcuts or one shortcut. Otherwise, the above path can be considered as a path of length n $a_0a_1a_2\cdots a_{p-1}ia_{p+1}a_{p+2}\cdots a_n$, with c cycles attached to it at the point i. Suppose that the lengths of these cycles are $\ell_1,\dots,\ell_c$.

Suppose first that there is a cycle of length ℓ that is prime to k. If $\ell = 1$ (that is, if the cycle is a self-loop), then with a single shortcut we can obtain a path of length 1 mod k, so assume that $\ell > 1$. Consider the path of the a_j's and this cycle:

$$a_0a_1a_2\cdots a_{p-1}ib_1b_2\cdots b_{\ell-1}ia_{p+1}a_{p+2}\cdots a_n.$$

First, consider the case in which $k+1$ is a multiple of ℓ. When the k-tuples starting with i are considered at the beginning of the ith stage [2], edges $(a_{p-1}, b_{\ell-1})$ and $(b_{\ell-1}, b_{\ell-1})$ are added. Also, subsequently in the ith stage, the edge $(b_{\ell-1}, a_{p+1})$ will be added (while considering the k-tuple $(b_{\ell-1},\dots,b_{\ell-1},i)$, with $b_{\ell-1}$ repeated $k-1$ times). Thus, the path from a_0 to a_{p-1} to $b_{\ell-1}$, looping back to $b_{\ell-1}$ $(1-n \bmod k)$ times, then to a_{p+1} and on to a_n, is a path of length 1 mod k involving only nodes larger than i.

So, suppose that ℓ does not divide $k+1$. Consider again the graph G consisting of just the path and this cycle:

$$a_0a_1a_2\cdots a_{p-1}ib_1b_2\cdots b_{\ell-1}ia_{p+1}a_{p+2}\cdots a_n.$$

After the ith stage (during which all paths of length $k+1$ containing an i are shortcut by a single edge), node i is deleted, but these edges are added (among others):
1. $(a_{p-1}, b_{k \bmod \ell})$,
2. $(b_{\ell-k \bmod \ell}, a_{p+1})$,
3. $(b_j, b_{j+k+1 \bmod \ell}), j \in \{1,\dots,\ell-1\} - \{-k-1 \bmod \ell\}$.

Consider G with all a_j's and i deleted, but the edges in (3) added. This graph, call it G', is a cycle with one node deleted, and all chords in (3) added. It is easy to see that G' (1) has a cycle of length $q = -k \bmod \ell$ (in fact, several such cycles), and (2) it is strongly connected (by induction, there is a path from each b_j to b_1, and we know there is a path the other way; here we are using our assumption that ℓ does not divide $k+1$).

We shall show that there exists an $L > 0$ such that there are paths from a_0 to a_n, not going through i, of length $L + m\cdot q$ for all $m \geqslant 0$. This is obtained by first finding a path of the form

$$a_0a_1a_2\cdots a_{p-1}b_{k \bmod \ell}Bb_{\ell-k \bmod \ell}a_{p+1}a_{p+2}\cdots a_n,$$

where B is a path in G' that goes through one of the cycles of length q. Let L be the length of this path. The desired paths are then obtained by adding m copies of the cycle of length q. Since $q = -k \bmod \ell$ is prime to k, this means that there is an m such that $L + m\cdot q = 1 \bmod k$.

If there is no cycle whose length is prime to k, then consider the greatest common divisor d of $\ell_1,\dots,\ell_c$. Assume first that $d = 1$. The argument now is similar to the previous case, but it involves all c cycles. The addition of the edges in the ith stage creates strongly connected components with cycles of length $q_j = \ell_j - k \bmod k$ for $j = 1,\dots,c$, and thus paths of length $L + \sum_{j=1}^{c} m_j\cdot q_j$ for any integer combination

[2] This is the only place in which this restriction in $\prec$ is needed.

of m_j's. Since the greatest common divisor of k and the l_j's is one, there is a combination of the m_j's such that this length is 1 mod k.

Suppose then that $d > 1$. If $d = k$, then the existence of a shortcut is immediate, since the path $a_0 a_1 a_2 \cdots a_{p-1} i a_{p+1} a_{p+2} \cdots a_n$ has length $n = 1 \bmod k$, and will be shortcut in the ith stage. Finally, if $1 < d < k$, the argument of the previous paragraph applies, and the existence of the desired path follows from the facts that (a) d divides all q_j's, and (b) $n = 1 \bmod d$.

This completes the proof of the inductive statement, of the direction from (c) to (a), and therefore of the theorem. □

3. Extensions

It would be interesting to extend the characterization in Theorem 1 to more general Datalog programs. We shall consider only sirups – it is not obvious how one should define the Floyd–Warshall algorithm for multi-rule programs. One important observation in this regard is that the implication from (a) to (b) in Theorem 1 is of very general validity: A similar adversary argument shows that all shallow proof trees of programs with the Floyd–Warshall property must have alternative proof trees. This observation immediately leads to the following result regarding unary sirups – that is, sirups in which the left-hand predicate T is unary, although database predicates may have higher arity.

Proposition 2. *A unary single-rule program has the Floyd–Warshall property if and only if it is first-order (that is, nonrecursive).*

Proof (*sketch*)**.** Consider any single expansion of the body of a unary sirup. The occurrence of T that was expanded can be uniquely determined, and the adversary argument of the proof of the direction (a) to (b) of the main theorem repeated. □

Returning to binary sirups, we strongly suspect the following:

Conjecture 1. *The programs identified in Theorem* 1 *are the only binary sirups that have the Floyd–Warshall property.*

The obstacles to proving this conjecture are the usual *homomorphism* problems (unexpected ways of mapping variables in expanded rules, see, for example, citeAP) present in Datalog arguments.

Looking beyond binary sirups, we immediately notice that the following ones do have the Floyd–Warshall property:

1. $T(x,y,w) \leftarrow T(x,z,w), T(z,y,w)$; this is the computation of the transitive closures of a *collection* of directed graphs, indexed by w. Similarly for multiply indexed collections.
2. $T(x,y,w) \leftarrow T(x,z,w'), T(z,y,w'')$; this is the transitive closures of the *union* of a collection of directed graphs.
3. $T(x,x',y,y') \leftarrow T(x,x',z,z'), T(z,z',y,y')$; this is the transitive closure of a directed graph whose nodes are *pairs* of constants. Similarly for triples etc.

4. $T(x,y) \leftarrow T(x,z), T(z,y), A(z)$; this is an *adorned* version of the transitive closure, only seeking paths through acceptable points. It can be generalized to various kinds of adornments.
5. The generalizations of these programs to more than two T's, as in the programs in Theorem 1(c).
6. Combinations of these generalizations of transitive closure. Example: adorned versions of collections of graphs defined on triples of nodes, with odd paths sought – three copies of T, or $k = 2$ in Theorem 1. (We omit the cumbersome formal definition of this class of sirups.)

It is easy to see that all sirups described in (1)–(6) above have the Floyd–Warshall property. A very plausible strengthening of Conjecture 1 is, that *these are the only sirups that have the Floyd–Warshall property.*

Finally, even in programs without the Floyd–Warshall property, we may want to apply a Floyd–Warshall-like algorithm, only with $\prec$ being an enumeration of k-tuples *with repetitions*. All sirups can be evaluated this way, with a sequence of k-tuples of length n^{r+k}, where k is the number of variables in the rhs rule, and r the arity of the defined relation T. But certain programs require less. For example, the variant $T \rightarrow T_0T$ of the transitive closure can be carried out with (*and seems to require*) n^4 triples, instead of n^5. Thus, the Floyd–Warshall algorithm seems to suggest an intriguing *quantitative* hierarchy of Datalog programs, which it would be very interesting to investigate. For example, is $\Omega(n^4)$ necessary for $T \rightarrow T_0T$? And are there simple sirups for which $\Omega(n^{r+k})$ is required?

Acknowledgements

Many thanks to Foto Afrati for discussions on this topic, and to three referees for their criticism, and for valuable suggestions for improving the presentation. This research was supported by the National Science Foundation under grant CCR-9626361, and by a Greek Government grant (ΠΕΝΕΔ) to the Computer Technology Institute in Patras, Greece.

References

[1] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, Reading, MA, 1995.
[2] F. Afrati, C.H. Papadimitriou, The parallel complexity of simple logic programs, J. ACM 40 (4) (1993) 891–916.
[3] F. Bancilhon, D. Maier, Y. Sagiv, J.D. Ullman, Magic sets and other strange ways to implement logic programs, Proceedings of the 1986 PODS, pp. 1–15.
[4] F. Bancilhon, D. Maier, Y. Sagiv, J.D. Ullman, Magic sets and other strange ways to implement logic programs, J. ACM 40 (4) (1993) 891–916.
[5] F. Bancilhon, R. Ramakrishnan An amateur's introduction to recursive query processing strategies, Proceedings 1986 SIGMOD, pp. 16–52.
[6] A. Chang, On the evaluation of queries containing derived relations in relational databases, in: H. Gallaire, J. Minker, J.-M. Nicolas (Eds.), Advances in Database Theory, Plenum Press, New York, 1981, pp. 235–260.
[7] R.W. Floyd, Algorithm 97: Shortest path, C. ACM 5 (6) (1963) 345.
[8] S. Warshall, A theorem on Boolean matrices, J. ACM 9 (1) (1963) 11–12.