TLP **17** (3): 244–265, 2017.
doi:10.1017/S1471068417000023

A linear algebraic approach to datalog evaluation

TAISUKE SATO

AI research center AIST/National Institute of Informatics, Tokyo, Japan
(*e-mails*: satou.taisuke@aist.go.jp)

submitted 29 July 2016; revised 22 March 2017; accepted 25 March 2017

Abstract

We propose a fundamentally new approach to Datalog evaluation. Given a linear Datalog program DB written using N constants and binary predicates, we first translate if-and-only-if completions of clauses in DB into a set $\mathbf{E}_q(\mathrm{DB})$ of matrix equations with a non-linear operation, where relations in $\mathbf{M}_{\mathrm{DB}}$, the least Herbrand model of DB, are encoded as adjacency matrices. We then translate $\mathbf{E}_q(\mathrm{DB})$ into another, but purely linear matrix equations $\tilde{\mathbf{E}}_q(\mathrm{DB})$. It is proved that the least solution of $\tilde{\mathbf{E}}_q(\mathrm{DB})$ in the sense of matrix ordering is converted to the least solution of $\mathbf{E}_q(\mathrm{DB})$ and the latter gives $\mathbf{M}_{\mathrm{DB}}$ as a set of adjacency matrices. Hence, computing the least solution of $\tilde{\mathbf{E}}_q(\mathrm{DB})$ is equivalent to computing $\mathbf{M}_{\mathrm{DB}}$ specified by DB. For a class of tail recursive programs and for some other types of programs, our approach achieves $O(N^3)$ time complexity irrespective of the number of variables in a clause since only matrix operations costing $O(N^3)$ or less are used. We conducted two experiments that compute the least Herbrand models of linear Datalog programs. The first experiment computes transitive closure of artificial data and real network data taken from the Koblenz Network Collection. The second one compared the proposed approach with the state-of-the-art symbolic systems including two Prolog systems and two ASP systems, in terms of computation time for a transitive closure program and the same generation program. In the experiment, it is observed that our linear algebraic approach runs $10^1 \sim 10^4$ times faster than the symbolic systems when data is not sparse. Our approach is inspired by the emergence of big knowledge graphs and expected to contribute to the realization of rich and scalable logical inference for knowledge graphs.

KEYWORDS: Datalog, least model, matrix, vector space

1 Introduction

Top-down and bottom-up have been the two major approaches in traditional logic programming. They are of contrasting nature but both compute the least model semantics symbolically. In this paper, we propose a third approach, a fundamentally new one, which evaluates logic programs in vector spaces to exploit the potential of logic programming in emerging areas.

Given a class of Datalog programs DB written using N constants and binary predicates, we first translate if-and-only-if completions of DB into a set $\mathbf{E}_q(\mathrm{DB})$ of matrix equations in the N-dimensional Euclidean space $\mathbb{R}^N$ with a non-linear

operation. We further translate $\mathbf{E}_q(\mathrm{DB})$ into another, but purely linear matrix equations $\tilde{\mathbf{E}}_q(\mathrm{DB})$. It is proved that the least solution of $\tilde{\mathbf{E}}_q(\mathrm{DB})$ in the sense of matrix ordering[1] can be converted to the least solution of $\mathbf{E}_q(\mathrm{DB})$ and the latter gives the least Herbrand model $\mathbf{M}_{\mathrm{DB}}$ of DB as a set of adjacency matrices. We thus can compute $\mathbf{M}_{\mathrm{DB}}$ by way of solving $\tilde{\mathbf{E}}_q(\mathrm{DB})$ algebraically in the vector space $\mathbb{R}^N$. We emphasize that our approach is not only new but time complexity wise compared favorably with or better than conventional Datalog evaluation methods for many important cases as we discuss later.

Our approach is inspired by the emergence of big knowledge graphs (KGs) such as YAGO (Suchanek *et al.* 2007), Freebase (Bollacker *et al.* 2008) and Knowledge Vault (Dong *et al.* 2014). A KG is a graph representing RDF triples of the form (subject : s, predicate : p, object : o) and logically speaking, they are just a set of ground atoms $p(s, o)$ with binary predicates. So one could say that they are simple. However, the point is not their logical simplicity but their size; some contain tens of millions of data, i.e., ground atoms. Researchers working in the field of KGs therefore developed scalable techniques to cope with huge KGs, one of which is a latent feature approach that translates entities and predicates in the domain into vectors, matrices and tensors (Cichocki *et al.* 2009; Kolda and Bader 2009) respectively in vector spaces and apply matrix and tensor decomposition for dimension reduction to realize efficient computation (Nickel *et al.* 2015).

Although KGs are just Datalog programs consisting of ground atoms with binary predicates and as such it should be possible to apply a variety of logical inference, little attention seems paid to logical aspects of KGs. Only simple types of logical inference are investigated so far (Krompaß *et al.* 2014; Rocktäschel *et al.* 2014; Rocktäschel *et al.* 2015; Yang *et al.* 2015). Thus, the objective of this paper is to introduce a linear algebraic approach to logical inference in vector spaces, thereby, bridging KGs and logic programming in general, or KGs and Datalog, in particular. By doing so, we hope to enrich logical inference for KGs on one hand and to realize robust and scalable inference for logic programming on the other hand.

In what follows, after a preliminary section, we describe, using a simple tail recursive Datalog program DB_1 as a running example, how to convert it to a matrix equation $\mathbf{E}_q(\mathrm{DB}_1)$ with a non-linear operation in Section 3. We then prove that $\mathbf{E}_q(\mathrm{DB}_1)$ is solvable by way of solving an isomorphic but purely linear equation $\tilde{\mathbf{E}}_q(\mathrm{DB}_1)$ in Section 4. We generalize our linear algebraic approach to a more general class of Datalog programs than tail-recursive ones in Section 5. In Section 6, we examine subclasses explicitly solvable in closed form by linear algebra. We validate our approach empirically through two experiments in Section 7. In Section 8, we briefly discuss related work and remaining problems. Section 9 is conclusion.

We assume the reader is familiar with basics of logic programming and linear algebra including tensors (Cichocki *et al.* 2009; Kolda and Bader 2009). We also

[1] Matrices $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$ are ordered by $\mathbf{A} \leqslant \mathbf{B}$ such that $\mathbf{A} \leqslant \mathbf{B}$ if-and-only-if $a_{ij} \leqslant b_{ij}$ for all i, j.

assume that throughout this paper, our first-order language $\mathscr{L}$ for Datalog programs, i.e., logic programs without function symbols, contains N constants $\{e_1,\ldots,e_N\}$ and only M binary predicates $\{r_1(\cdot,\cdot),\ldots,r_M(\cdot,\cdot)\}$.

2 Preliminaries

In this paper, vectors are always column vectors and denoted by boldface lower case letters like "**a**". Similarly, matrices are square and written by boldface upper case letters like "**A**". In particular, **I** is an identity matrix. For a matrix $\mathbf{A}=[a_{ij}]$, put $(\mathbf{A})_{ij}=a_{ij}$. We use $\mathbf{A}\otimes\mathbf{B}$ for the Kronecker product of **A** and **B**. $\mathrm{vec}(\mathbf{A})=[\mathbf{a}_1^T,\ldots,\mathbf{a}_M^T]^T$ is the vectorization of a matrix $\mathbf{A}=[\mathbf{a}_1,\ldots,\mathbf{a}_M]$. Note the fact that $\mathbf{Y}=\mathbf{AXB}$ if-and-only-if $\mathrm{vec}(\mathbf{Y})=(\mathbf{B}^T\otimes\mathbf{A})\mathrm{vec}(\mathbf{X})$. We use $(\cdot\bullet\cdot)$ for inner products. So $(\mathbf{a}\bullet\mathbf{b})=\mathbf{a}^T\mathbf{b}$. $\mathbb{1}$ denotes a matrix of all ones. We introduce two operator norms for matrices, $\|\mathbf{A}\|_\infty=\max_i\sum_j|a_{ij}|$ and $\|\mathbf{A}\|_1=\max_j\sum_i|a_{ij}|$. $\|\mathbf{A}^T\|_1=\|\mathbf{A}\|_\infty$ holds by definition.

Next, we review some logic programming terminology and definitions (Lloyd 1993). Let DB be a Datalog program in a given first-order language $\mathscr{L}$ and DB^g, the set of ground instances of clauses in DB. Also, let **HB** be the *Herbrand base*, i.e., the set of all ground atoms in $\mathscr{L}$. Define a mapping $T_{\mathrm{DB}}(\cdot):2^{\mathbf{HB}}\to 2^{\mathbf{HB}}$ by

$$T_{\mathrm{DB}}(I)\stackrel{\mathrm{def}}{=}\{\,a\mid \text{there is some } a\leftarrow b_1\wedge\cdots\wedge b_k\in\mathrm{DB}^g\ (k\geqslant 0)\\ \text{such that } \{b_1,\ldots,b_k\}\subseteq I\,\}$$

and a series $\{I^{(n)}\}_{n=0,1,\ldots}$ by $I^{(0)}=\emptyset, I^{(n+1)}=T_{\mathrm{DB}}(I^{(n)})$. Then, we see $I^{(0)}\subseteq I^{(1)}\subseteq\cdots$ and $I^\infty=\bigcup_n I^{(n)}$ gives the *least Herbrand model* $\mathbf{M}_{\mathrm{DB}}$ of DB, least in the sense of set inclusion ordering, which is defined by $\mathbf{M}_{\mathrm{DB}}\models a$ if-and-only-if $a\in I^\infty$ for any ground atom $a\in\mathbf{HB}$.

Let us *encode* $\mathbf{M}_{\mathrm{DB}}$, i.e., isomorphically map $\mathbf{M}_{\mathrm{DB}}$ while preserving truth values into the N-dimensional Euclidean space $\mathbb{R}^N$. Recall that the domain of $\mathbf{M}_{\mathrm{DB}}$ is a set $\mathscr{D}=\{e_1,\ldots,e_N\}$ of N constants and there are M binary predicates $\{r_1(\cdot,\cdot),\ldots,r_M(\cdot,\cdot)\}$ in $\mathbf{M}_{\mathrm{DB}}$. We translate each $e_i\,(1\leqslant i\leqslant N)$ by one-hot encoding into the N-dimensional column vector $\mathbf{e}_i=(0,\ldots 1\ldots,0)^T$ in $\mathbb{R}^N$ which has 1 as the ith element and 0 for other elements. The set $\mathscr{D}'=\{\mathbf{e}_1,\ldots,\mathbf{e}_N\}$ forms the standard basis of $\mathbb{R}^N$.

Following vector encoding of domain entities, we introduce $N\times N$ adjacency matrices $\mathbf{R}_m\in\{0,1\}^{N\times N}$ to encode relations $r_m(\cdot,\cdot)$ by

$$(\mathbf{R}_m)_{ij}=(\mathbf{e}_i\bullet\mathbf{R}_m\mathbf{e}_j)=\begin{cases}1 & \text{if}\quad \mathbf{M}_{\mathrm{DB}}\models r_m(e_i,e_j)\\ 0 & \text{o.w.}\end{cases}\qquad(1\leqslant i,j\leqslant N, 1\leqslant m\leqslant M)$$

We say $\mathbf{R}_m$ *encodes* $r_m(\cdot,\cdot)$ in $\mathbf{M}_{\mathrm{DB}}$ and call $\mathbf{R}_m$ a matrix encoding $r_m(\cdot,\cdot)$ or representing $r_m(\cdot,\cdot)$.

Now, we introduce the notation $[\![F]\!]$, the truth value of F in $\mathbf{M}_{\mathrm{DB}}$ expressed in terms of vectors and matrices, for a limited class of logical formulas F used as the clause body of Datalog programs. We assume here that *at most two variables are*

existentially quantified in the clause body so that no tensor of order $n > 2$ is required for the encoding. Let x, y, z be variables ranging over $\mathscr{D} = \{e_1, \ldots, e_N\}$ and $\mathbf{x}, \mathbf{y}, \mathbf{z}$ variables over the domain of corresponding one-hot encoding $\mathscr{D}' = \{\mathbf{e}_1, \ldots, \mathbf{e}_N\}$. We use a non-linear function $\min_1(x)$ defined by $\min_1(x) = \begin{cases} 1 & \text{if } x \geqslant 1 \\ x & \text{o.w.} \end{cases}$.

Then, $[\![F]\!]$ is defined for a class of AND/OR formulas which is computed inductively by

$$[\![r(x,y)]\!] = (\mathbf{x} \bullet \mathbf{R}\mathbf{y}) \quad \text{where } \mathbf{R} \text{ encodes } r(\cdot,\cdot) \tag{1}$$

$$[\![F_1 \wedge \cdots \wedge F_K]\!] = [\![F_1]\!] \cdots [\![F_K]\!] \tag{2}$$

$$[\![F_1 \vee \cdots \vee F_K]\!] = \min_1([\![F_1]\!] + \cdots + [\![F_K]\!]) \tag{3}$$

$$\begin{aligned} [\![\exists y\, r_i(x,y) \wedge r_j(y,z)]\!] &= \min_1\Big(\sum_{k=1}^{N} (\mathbf{x} \bullet \mathbf{R}_i\mathbf{e}_k)(\mathbf{e}_k \bullet \mathbf{R}_j\mathbf{z})\Big) \\ &= \min_1\Big(\mathbf{x}^T\mathbf{R}_i\Big(\sum_{k=1}^{N} \mathbf{e}_k\mathbf{e}_k{}^T\Big)\mathbf{R}_j\mathbf{z}\Big) \\ &= \min_1\big((\mathbf{x} \bullet \mathbf{R}_i\mathbf{R}_j\mathbf{z})\big) \quad \Big(\text{as } \sum_{k=1}^{N} \mathbf{e}_k\mathbf{e}_k{}^T = \mathbf{I}\Big). \end{aligned} \tag{4}$$

Note that here the existential quantification $\exists y$ is translated into $\sum_{k=1}^{N} \mathbf{e}_k\mathbf{e}_k{}^T$ though it is an identity matrix. Now, it is easy to see $[\![F]\!] \in \{0, 1\}$ and $\mathbf{M}_{\mathrm{DB}} \models F$ if-and-only-if $[\![F]\!] = 1$ for closed F[2].

3 Datalog programs as non-linear matrix equations

To convey the essential idea quickly, we use the following simple right recursive Datalog program DB_1 as a running example:

$$\begin{aligned} r_2(x,z) &\leftarrow r_1(x,z) \\ r_2(x,z) &\leftarrow r_1(x,y) \wedge r_2(y,z) \end{aligned} \tag{5}$$

DB_1 computes the transitive closure $r_2(x,y)$ of a binary relation $r_1(x,y)$.

We show that we are able to derive a matrix equation whose solution gives $r_2(x,y)$ given $r_1(x,y)$. First recall that the least Herbrand model $\mathbf{M}_{\mathrm{DB}_1}$ of DB_1 satisfies the following logical equivalence (called *if-and-only-if completion* (Lloyd 1993)):

$$\forall x, z\ (r_2(x,z) \Leftrightarrow r_1(x,z) \vee \exists\, y\, (r_1(x,y) \wedge r_2(y,z))) \tag{6}$$

[2] Formally, this is proved by induction on the structure of F.

We translate this equivalence into an equation for matrices $\mathbf{R}_1, \mathbf{R}_2$ encoding $r_1(x,z), r_2(y,z)$ as follows.

$$[\![r_2(x,z)]\!] = [\![r_1(x,z) \vee \exists\, y\, (r_1(x,y) \wedge r_2(y,z))]\!] \text{ for } \forall x,z \in \mathscr{D} = \{e_1,\ldots,e_N\}$$

if-and-only-if

$$\begin{aligned}(\mathbf{x} \bullet \mathbf{R}_2\mathbf{z}) &= \min_1((\mathbf{x} \bullet \mathbf{R}_1\mathbf{z}) + \min_1((\mathbf{x} \bullet \mathbf{R}_1\mathbf{R}_2\mathbf{z})))\\ &= \min_1((\mathbf{x} \bullet \mathbf{R}_1\mathbf{z}) + (\mathbf{x} \bullet \mathbf{R}_1\mathbf{R}_2\mathbf{z}))\\ &= \min_1((\mathbf{x} \bullet (\mathbf{R}_1 + \mathbf{R}_1\mathbf{R}_2)\mathbf{z}))\\ &= (\mathbf{x} \bullet \min_1(\mathbf{R}_1 + \mathbf{R}_1\mathbf{R}_2)\mathbf{z}) \text{ for } \forall \mathbf{x},\mathbf{z} \in \mathscr{D}' = \{\mathbf{e}_1,\ldots,\mathbf{e}_N\}\end{aligned}$$

if-and-only-if

$$\mathbf{R}_2 = \min_1(\mathbf{R}_1 + \mathbf{R}_1\mathbf{R}_2) \text{ for } \mathbf{R}_1, \mathbf{R}_2 \in \{0,1\}^{N\times N}$$

Here, $\min_1(\mathbf{A})$ for a matrix $\mathbf{A}$ means component-wise application of $\min_1(x)$ function. Note that $\min_1((\mathbf{x} \bullet \mathbf{A}\mathbf{y})) = (\mathbf{x} \bullet \min_1(\mathbf{A})\mathbf{y})$ holds for any matrix $\mathbf{A}$ and $\mathbf{x}, \mathbf{y} \in \mathscr{D}' = \{\mathbf{e}_1,\ldots,\mathbf{e}_N\}$. We conclude that $\mathbf{R}_1$, $\mathbf{R}_2 \in \{0,1\}^{N\times N}$, matrices encoding relations $r_1(\cdot,\cdot)$, $r_2(\cdot,\cdot)$ in $\mathbf{M}_{\mathrm{DB}_1}$, respectively, satisfy the following equation:

$$\mathbf{R}_2 = \min_1(\mathbf{R}_1 + \mathbf{R}_1\mathbf{R}_2). \tag{7}$$

We then ask the converse: given $\mathbf{R}_1$ encoding $r_1(\cdot,\cdot)$ in $\mathbf{M}_{\mathrm{DB}_1}$, does a matrix $\mathbf{R}_2$ satisfying (7) encode $r_2(\cdot,\cdot)$ in $\mathbf{M}_{\mathrm{DB}_1}$? The converse is not necessarily true; think of $\mathbf{R}_2 = \mathbb{1}$. However, fortunately and evidently, the least solution $\mathbf{R}_2^*$ of (7) gives $r_2(\cdot,\cdot)$ in the least model $\mathbf{M}_{\mathrm{DB}_1}$. Here, "least" means the ordering among matrices defined by for $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$, $\mathbf{A} \leqslant \mathbf{B}$ if-and-only-if $a_{ij} \leqslant b_{ij}$ for $\forall\, i, j$. To obtain the least solution, we define a series of monotonically increasing matrices ($\in \{0,1\}^{N\times N}$) $\{\mathbf{R}_2^{(k)}\}_{k=0,1\ldots}$.

$$\begin{aligned}\mathbf{R}_2^{(0)} &= \mathbf{0} \text{ (matrix with every element being 0)}\\ \mathbf{R}_2^{(k+1)} &= \min_1(\mathbf{R}_1 + \mathbf{R}_1\mathbf{R}_2^{(k)}).\end{aligned} \tag{8}$$

Note that $\{\mathbf{R}_2^{(n)}\}$ converges at $n \geqslant N$. It is customary to prove that the limit $\mathbf{R}_2^{(\infty)} = \lim_{n\to\infty} \mathbf{R}_2^{(n)} \in \{0,1\}^{N\times N}$ gives the least solution $\mathbf{R}_2^*$ of (7).

4 Evaluation with linear matrix equations

The task of computing the transitive closure of $r_1(\cdot,\cdot)$ is now reduced to computing the least solution of the matrix equation $\mathbf{R}_2 = \min_1(\mathbf{R}_1 + \mathbf{R}_1\mathbf{R}_2)$ (7) which is solvable by constructing a series $\{\mathbf{R}_2^{(k)}\}_{k=0,1\ldots}$. The problem is that constructing $\{\mathbf{R}_2^{(k)}\}_{k=0,1\ldots}$ is essentially nothing but the naive bottom-up evaluation of DB_1. We see no clear computational gain in solving (7) by way of (8) compared to direct bottom-up evaluation. What is sought for here is to develop a better evaluation method than the naive bottom-up evaluation. Consider an alternative equation:

$$\tilde{\mathbf{R}}_2 = \epsilon(\mathbf{R}_1 + \mathbf{R}_1\tilde{\mathbf{R}}_2) \tag{9}$$

where ϵ is a small positive number such that $(\mathbf{I} - \epsilon\mathbf{R}_1)^{-1}$ exists, for example, $\epsilon < \frac{1}{\|\mathbf{R}_1\|_\infty}$. We prove that the least solution of (9) gives the least solution of (7).

Define $\{\tilde{\mathbf{R}}_2^{(k)}\}_{k=0,1\ldots}$ by

$$\begin{aligned}\tilde{\mathbf{R}}_2^{(0)} &= \mathbf{0}\\ \tilde{\mathbf{R}}_2^{(k+1)} &= \epsilon(\mathbf{R}_1 + \mathbf{R}_1\tilde{\mathbf{R}}_2^{(k)})\end{aligned} \tag{10}$$

Lemma 1
Suppose $0 < \epsilon \leqslant \frac{1}{1+\|\mathbf{R}_1\|_\infty}$. Then, $\{\tilde{\mathbf{R}}_2^{(k)}\}_{k=0,1\ldots}$ converges and its limit $\tilde{\mathbf{R}}_2^{(\infty)}$ is the least solution of (9).

Proof
We first prove $\tilde{\mathbf{R}}_2^{(k)} \leqslant \mathbb{1}$ for $\forall\, k \in \mathbb{N}$ by mathematical induction. This obviously holds for $k = 0$. Suppose $\tilde{\mathbf{R}}_2^{(k)} \leqslant \mathbb{1}$. Then,

$$\begin{aligned}\tilde{\mathbf{R}}_2^{(k+1)} &= \epsilon(\mathbf{R}_1 + \mathbf{R}_1\tilde{\mathbf{R}}_2^{(k)})\\ &\leqslant \epsilon(\mathbf{R}_1 + \mathbf{R}_1\mathbb{1}) \text{ by the induction hypothesis}\\ &\leqslant \epsilon(1 + \|\mathbf{R}_1\|_\infty)\mathbb{1}\\ &\leqslant \mathbb{1} \quad \text{because } \epsilon(1 + \|\mathbf{R}_1\|_\infty) \leqslant 1\end{aligned}$$

So $\{\tilde{\mathbf{R}}_2^{(k)}\}_{k=0,1\ldots}$ converges, as a series of monotonically increasing matrices with an upper bound, to $\tilde{\mathbf{R}}_2^{(\infty)}$. Furthermore,

$$\begin{aligned}\tilde{\mathbf{R}}_2^{(\infty)} &= \lim_{k\to\infty} \tilde{\mathbf{R}}_2^{(k+1)}\\ &= \lim_{k\to\infty} \epsilon(\mathbf{R}_1 + \mathbf{R}_1\tilde{\mathbf{R}}_2^{(k)})\\ &= \epsilon(\mathbf{R}_1 + \mathbf{R}_1 \lim_{k\to\infty} \tilde{\mathbf{R}}_2^{(k)})\\ &= \epsilon(\mathbf{R}_1 + \mathbf{R}_1\tilde{\mathbf{R}}_2^{(\infty)})\end{aligned}$$

Also, let $\mathbf{R}'_2$ be an arbitrary solution of (9). It can be proved that $\tilde{\mathbf{R}}_2^{(k)} \leqslant \mathbf{R}'_2$ holds for $\forall k \in \mathbb{N}$ by mathematical induction. So, $\tilde{\mathbf{R}}_2^{(\infty)} = \lim_k \tilde{\mathbf{R}}_2^{(k)} \leqslant \mathbf{R}'_2$ is the least solution of (9). □

Lemma 2
$\left(\mathbf{R}_2^{(k)}\right)_{ij} = 1$ if-and-only-if $\left(\tilde{\mathbf{R}}_2^{(k)}\right)_{ij} > 0$ for $\forall\, k \in \mathbb{N}, 1 \leqslant i, j \leqslant N$.[3]

Proof
We prove by mathematical induction. When $k = 0$, $\mathbf{R}_2^{(0)} = \tilde{\mathbf{R}}_2^{(0)} = \mathbf{0}$ and both sides are false. Suppose $\left(\mathbf{R}_2^{(k+1)}\right)_{ij} = 1$. Then $\left(\min_1(\mathbf{R}_1 + \mathbf{R}_1\mathbf{R}_2^{(k)})\right)_{ij} = 1$ holds, which implies $(\mathbf{R}_1)_{ij} = 1$ or $\left(\mathbf{R}_1\mathbf{R}_2^{(k)}\right)_{ij} \geqslant 1$. The latter implies that for some $m\,(1 \leqslant m \leqslant N)$, $(\mathbf{R}_1)_{im}(\mathbf{R}_2^{(k)})_{mj} = 1$, and hence $(\mathbf{R}_1)_{im} = 1$ and $(\tilde{\mathbf{R}}_2^{(k)})_{mj} > 0$ by the induction hypothesis. So $\left(\mathbf{R}_1\tilde{\mathbf{R}}_2^{(k)}\right)_{ij} > 0$. By combining this and $(\mathbf{R}_1)_{ij} = 1$ disjunctively, we conclude $(\tilde{\mathbf{R}}_2^{(k+1)})_{ij} = \epsilon(\mathbf{R}_1 + \mathbf{R}_1\tilde{\mathbf{R}}_2^{(k)})_{ij} = \epsilon((\mathbf{R}_1)_{ij} + (\mathbf{R}_1\tilde{\mathbf{R}}^{(k)})_{ij}) > 0$. The argument goes the other way around, so we are done. □

[3] It is also proved that $\left(\mathbf{R}_2^{(k)}\right)_{ij} = 1$ implies $\left(\tilde{\mathbf{R}}_2^{(k)}\right)_{ij} \geqslant \epsilon^k$ for $\forall\, k \in \mathbb{N}, 1 \leqslant i, j \leqslant N$.

Theorem 1
Let $\tilde{\mathbf{R}}_2^*$ be the least solution of the matrix equation $\tilde{\mathbf{R}}_2 = \epsilon(\mathbf{R}_1 + \mathbf{R}_1\tilde{\mathbf{R}}_2)$ (9) where ϵ is a positive number satisfying $0 < \epsilon \leqslant \frac{1}{1+\|\mathbf{R}_1\|_\infty}$. Define $\mathbf{R}_2^* \in \{0,1\}^{N\times N}$ by

$$(\mathbf{R}_2^*)_{ij} = \begin{cases} 1 & \text{if } (\tilde{\mathbf{R}}_2^*)_{ij} > 0 \\ 0 & \text{o.w.} \end{cases} \qquad (1 \leqslant i, j \leqslant N)$$

Then, $\mathbf{R}_2^*$ is the least solution of the matrix equation $\mathbf{R}_2 = \min_1(\mathbf{R}_1 + \mathbf{R}_1\mathbf{R}_2)$ (7). In other words, $\mathbf{R}_2^*$ encodes the transitive closure of $r_1(\cdot,\cdot)$ in $\mathbf{M}_{\mathrm{DB}_1}$.

Proof
By Lemma 1, $\tilde{\mathbf{R}}_2^* = \tilde{\mathbf{R}}_2^{(\infty)} = \lim_k \tilde{\mathbf{R}}_2^{(k)}$. So for any i, j $(1 \leqslant i, j \leqslant N)$,

$$\begin{aligned}(\mathbf{R}_2^*)_{ij} = 1 &\text{ if-and-only-if } (\tilde{\mathbf{R}}_2^{(\infty)})_{ij} > 0 \\ &\text{if-and-only-if } (\tilde{\mathbf{R}}_2^{(k)})_{ij} > 0 \text{ for some } k \\ &\text{if-and-only-if } (\mathbf{R}_2^{(k)})_{ij} = 1 \text{ by Lemma 2} \\ &\text{if-and-only-if } (\mathbf{R}_2^{(\infty)})_{ij} = 1.\end{aligned}$$

Therefore, we have $\mathbf{R}_2^* = \mathbf{R}_2^{(\infty)}$. Since $\mathbf{R}_2^{(\infty)}$ is the least solution of (7), so is $\mathbf{R}_2^*$. □

Proposition 1
Suppose $0 < \epsilon \leqslant \frac{1}{1+\|\mathbf{R}_1\|_\infty}$. Compute $(\mathbf{I} - \epsilon\mathbf{R}_1)^{-1}\epsilon\mathbf{R}_1$. It coincides with the least solution $\tilde{\mathbf{R}}_2^*$ of (9) and hence its conversion to a 0-1 matrix $\mathbf{R}_2^*$ described in Theorem 1 gives the transitive closure of $r_1(\cdot,\cdot)$ in $\mathbf{M}_{\mathrm{DB}_1}$.

Proof
Since $0 < \epsilon \leqslant \frac{1}{1+\|\mathbf{R}_1\|_\infty}$, $\rho(\epsilon\mathbf{R}_1)$, the spectral radius of $\epsilon\mathbf{R}_1$, satisfies $\rho(\epsilon\mathbf{R}_1) \leqslant \epsilon\|\mathbf{R}_1\|_\infty \leqslant 1 - \epsilon < 1$. Consequently, $(\mathbf{I} - \epsilon\mathbf{R}_1)^{-1}$ exists and the matrix equation (9) has a unique solution $(\mathbf{I} - \epsilon\mathbf{R}_1)^{-1}\epsilon\mathbf{R}_1$ which must coincide with another solution $\tilde{\mathbf{R}}_2^*$. □

The choice of ϵ is arbitrary but the largest value, $\frac{1}{1+\|\mathbf{R}_1\|_\infty}$, would be preferable from the viewpoint of the conversion of $\tilde{\mathbf{R}}_2^{(\infty)}$ to $\mathbf{R}_2^*$. Note that $\|\mathbf{R}_1\|_\infty$ is the maximum out-degree of nodes in $\mathbf{R}_1$ as a graph and is possibly independent of the graph size.

The time complexity of computing $(\mathbf{I} - \epsilon\mathbf{R}_1)^{-1}\epsilon\mathbf{R}_1$ is $O(N^3)$, or less, theoretically, if we use the Coppersmith–Winograd algorithm (Coppersmith and Winograd 1990) which gives $O(N^{2.376})$. Hence, we may say that in the case of transitive closure computation, our matrix approach which can be $O(N^{2.376})$, is comparable with or slightly better than, say, tabled top-down evaluation of DB_1 which requires $O(N^3)$[4].

[4] Generally, tabled top-down evaluation requires $O(N^v)$, where v is the maximum number of variables of the body clause in a program (Warren 1999). A deeper analysis of the time complexity of Datalog execution for transitive closure programs is given in Tekle and Liu (2010). Unfortunately, it concentrates on the case of the query of the form ?-$r_2(x, y)$, where either x or y is ground, and is not directly applicable to our case where both x and y are variables.

We close this section with a concrete example of transitive closure computation. Suppose our Herbrand model $\mathbf{M}_{\mathrm{DB}_1}$ of DB_1 has a domain $\mathscr{D} = \{e_1, \ldots, e_4\}$ of four constants and assume $\{r_1(e_1, e_2), r_1(e_2, e_3), r_1(e_3, e_1), r_1(e_4, e_1)\}$ are true w.r.t. the relation $r_1(\cdot, \cdot)$. Then, the adjacency matrix $\mathbf{R}_1$ encoding $r_1(\cdot, \cdot)$ is given by

$$\mathbf{R}_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

and we have $\|\mathbf{R}_1\|_\infty = \max\{1, 1, 1, 1\} = 1$. Put $\epsilon = (1 + \|\mathbf{R}_1\|_\infty)^{-1} = 1/2$.

$$\begin{aligned} \tilde{\mathbf{R}}_2^* &= (\mathbf{I} - \epsilon\mathbf{R}_1)^{-1}\epsilon\mathbf{R}_1 \\ &= \left(\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 0 \\ 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 \end{pmatrix} \right)^{-1} \begin{pmatrix} 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 0 \\ 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0.1428 & 0.5714 & 0.2857 & 0.0000 \\ 0.2857 & 0.1428 & 0.5714 & 0.0000 \\ 0.5714 & 0.2857 & 0.1428 & 0.0000 \\ 0.5714 & 0.2857 & 0.1428 & 0.0000 \end{pmatrix} \end{aligned}$$

Hence, by thresholding $\tilde{\mathbf{R}}_2^*$ at 0, we reach

$$\mathbf{R}_2^* = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

which is the adjacency matrix encoding the transitive closure of $r_1(\cdot, \cdot)$.

5 Generalization

DB_1 is just one example of Datalog program. We here discuss how far we can generalize our linear algebraic approach to Datalog evaluation. We first generalize Lemma 1, Lemma 2 and Theorem 1 for a class $\mathscr{C}_{lin}$ of linear Datalog programs. A program DB is in $\mathscr{C}_{lin}$ if

- DB contains only binary predicates,
- non-unit clauses do not contain constants and take the following form: $r_0(\{x_0, x_n\}) \leftarrow r_1(\{x_0, x_1\}) \wedge \cdots \wedge r_n(\{x_{n-1}, x_n\})$ such that $x_0, \ldots, x_n$ are all different and $r_i(\{x_{i-1}, x_i\})$ represents either $r_i(x_{i-1}, x_i)$ or $r_i(x_i, x_{i-1})$ $(0 \leqslant i \leqslant n)$ and
- DB is *linear* in the following sense.

Let r and r' be predicates appearing in a program DB. We say r depends on r' if there is a clause $H \leftarrow W$ in DB such that H contains r and W contains r' respectively, and write $r \geq_{\mathrm{DB}} r'$. Extend $r \geq r'_{\mathrm{DB}}$ to its transitive closure (but we use the same

symbol $\succeq_{\mathrm{DB}}$ for the closure). Put $[r] \stackrel{\mathrm{def}}{=} \{r' \mid r \succeq_{\mathrm{DB}} r' \text{ and } r' \succeq_{\mathrm{DB}} r\}$. $[r]$ is a set of mutually dependent predicates and called a *strongly connected component (SCC)* of DB. Predicates in DB are partitioned into a set of SCCs and SCCs themselves are partially ordered by a partial ordering, *SCC ordering of* DB, $[r] >_{\mathrm{DB}} [r']$ defined as $[r] >_{\mathrm{DB}} [r']$ if-and-only-if $[r] \succeq_{\mathrm{DB}} [r']$ and $[r'] \not\succeq_{\mathrm{DB}} [r]$. We call DB *linear* if no clause body contains two predicates in the same SCC.

Linear programs are, intuitively, programs consisting of clauses such that there is at most one recursive goal in the clause body. Note that checking if DB is linear can be done mechanically without difficulty as the main part, the construction of SCCs, is carried out by Tarjan's algorithm (Tarjan 1972) efficiently in time linear in the number of atoms in DB. In what follows, we only deal with linear Datalog programs in $\mathscr{C}_{lin}$.

Let DB be a linear program in $\mathscr{C}_{lin}$. Clauses in DB are partitioned into disjoint sets $\{\mathrm{DB}'_i\}_{i=1,\ldots,L}$ called *layer*s such that $\mathrm{DB} = \mathrm{DB}'_1 \cup \cdots \cup \mathrm{DB}'_L$ and the head predicates of $\mathrm{DB}'_i\,(1 \leqslant i \leqslant L)$ coincide with an SCC which we denote by $\mathrm{SCC}_{\mathrm{DB}'_i}$. Furthermore, we assume no predicate in DB'_i depends on predicates in higher layers $\mathrm{DB}'_j\,(i < j)$. In other words $\{\mathrm{DB}'_1, \ldots, \mathrm{DB}'_L\}$ is a list of layers topologically sorted in the ascending order by the SCC ordering. So, the bottom layer program DB'_1 contains only predicates minimal in $\succeq_{\mathrm{DB}}$ and is a union of ground unit clauses and clauses of the form $r(x, y) \leftarrow s(x, y)$ or $r(x, y) \leftarrow s(y, x)$.

Now, fix DB'_i. DB'_i consists of clauses of the form:

$$r(x, y) \leftarrow A$$
$$r(x, y) \leftarrow B \wedge s(u, v) \wedge C$$

where $r, s \in \mathrm{SCC}_{\mathrm{DB}'_i}$. A, B and C are conjunctions, possibly empty, of atoms whose predicates are defined in lower layers $\bigcup_{j<i} \mathrm{DB}'_j$. Put $\mathrm{DB}'_{\leqslant i} \stackrel{\mathrm{def}}{=} \bigcup_{j \leqslant i} \mathrm{DB}'_j$. We translate if-and-only completions of clauses in DB'_i, which always hold in the least Herbrand model $\mathbf{M}_{\mathrm{DB}'_{\leqslant i}}$ of $\mathrm{DB}'_{\leqslant i}$ (Lloyd 1993), into matrix equations just like the case of the transitive closure program (7).

Let $\mathrm{SCC}_{\mathrm{DB}'_i} = \{r_1, \ldots, r_M\}$ be the head predicates of DB'_i and $\{\mathbf{R}_1, \ldots, \mathbf{R}_M\}$ matrices encoding $\{r_1, \ldots, r_M\}$ in $\mathbf{M}_{\mathrm{DB}'_{\leqslant i}}$. Since a conjunction of atoms from lower layers below DB'_i is translated into a single matrix by multiplying matrices, an if-and-only completion of clauses in DB'_i is translated into a matrix equation of the form below:

$$\mathbf{R}_h = \min_1(F_h[\mathbf{R}_1, \ldots, \mathbf{R}_M])$$
$$F_h[\mathbf{R}_1, \ldots, \mathbf{R}_M] = \mathbf{A}_1 + \mathbf{B}_1\mathbf{R}^{\circ}_{j_1}\mathbf{C}_1 + \cdots + \mathbf{B}_q\mathbf{R}^{\circ}_{j_q}\mathbf{C}_q \tag{11}$$

Here, $\{\mathbf{R}_h, \mathbf{R}_{j_1}, \ldots, \mathbf{R}_{j_q}\} \subseteq \{\mathbf{R}_1, \ldots, \mathbf{R}_M\}$ and $\mathbf{R}^{\circ}$ is either $\mathbf{R}$ or $\mathbf{R}^T$. $\mathbf{A}_1$ is an $N \times N$ adjacency matrix encoding a disjunction of conjunctions, while $\mathbf{B}_1, \ldots, \mathbf{B}_q, \mathbf{C}_1, \ldots, \mathbf{C}_q$ are $N \times N$ adjacency matrices encoding purely conjunctions, and these conjunctions are made out of predicates in layers below DB'_i. In summary, $\{\mathbf{R}_1, \ldots, \mathbf{R}_M\}$ satisfy a

system $\mathbf{E}_q(\mathrm{DB}'_i)$ of non-linear matrix equations below:

$$\begin{aligned}\mathbf{R}_1 &= \min_1(F_1[\mathbf{R}_1,\ldots,\mathbf{R}_M])\\ &\ldots\\ \mathbf{R}_M &= \min_1(F_M[\mathbf{R}_1,\ldots,\mathbf{R}_M])\end{aligned} \tag{12}$$

where each $F_h[\mathbf{R}_1,\ldots,\mathbf{R}_M]\,(1 \leqslant h \leqslant M)$ takes the form shown in (11).

Then, conversely, consider $\mathbf{E}_q(\mathrm{DB}'_i)$ (12) as a set of non-linear matrix equations for unknown $\{\mathbf{R}_1,\ldots,\mathbf{R}_M\}$ and try to solve it. We define sequences of matrices $\{\mathbf{R}_1^{(k)},\ldots,\mathbf{R}_M^{(k)}\}_{k=0,1\ldots}$ corresponding to (8) by

$$\begin{aligned}\mathbf{R}_h^{(0)} &= \mathbf{0}\\ \mathbf{R}_h^{(k+1)} &= \min_1(F_h[\mathbf{R}_1^{(k)},\ldots,\mathbf{R}_M^{(k)}])\end{aligned} \tag{13}$$

for $h\,(1 \leqslant h \leqslant M)$. We state Lemma 3 without proof.

Lemma 3
$\{\mathbf{R}_1^{(k)},\ldots,\mathbf{R}_M^{(k)}\}_{k=0,1\ldots}$ are monotonically increasing sequences of matrices and converge to the least solution $\{\mathbf{R}_1^{(\infty)},\ldots,\mathbf{R}_M^{(\infty)}\}$ of $\mathbf{E}_q(\mathrm{DB}'_i)$ (12). $\{\mathbf{R}_1^{(\infty)},\ldots,\mathbf{R}_M^{(\infty)}\}$ encode $\{r_1,\ldots,r_M\}$ in the least Herbrand model $\mathbf{M}_{\mathrm{DB}'_{\leqslant i}}$ of $\mathrm{DB}'_{\leqslant i}$.

Next, we introduce, isomorphically to (12), a system $\tilde{\mathbf{E}}_q(\mathrm{DB}'_i)$ of linear matrix equations:

$$\begin{aligned}\tilde{\mathbf{R}}_1 &= \epsilon_1 F_1[\tilde{\mathbf{R}}_1,\ldots,\tilde{\mathbf{R}}_M]\\ &\ldots\\ \tilde{\mathbf{R}}_M &= \epsilon_M F_M[\tilde{\mathbf{R}}_1,\ldots,\tilde{\mathbf{R}}_M]\end{aligned} \tag{14}$$

where ϵ_h is a small positive number satisfying $\epsilon_h F_h[\mathbb{1},\ldots,\mathbb{1}] \leqslant \mathbb{1}\,(1 \leqslant h \leqslant M)$.

Define $\{\tilde{\mathbf{R}}_1^{(k)},\ldots,\tilde{\mathbf{R}}_M^{(k)}\}_{k=0,1\ldots}$, correspondingly to (10), by

$$\begin{aligned}\tilde{\mathbf{R}}_h^{(0)} &= \mathbf{0}\\ \tilde{\mathbf{R}}_h^{(k+1)} &= \epsilon_h F_h[\tilde{\mathbf{R}}_1^{(k)},\ldots,\tilde{\mathbf{R}}_M^{(k)}]\end{aligned} \tag{15}$$

for $h\,(1 \leqslant h \leqslant M)$. Proving Lemma 4 is straightforward:

Lemma 4
$\{\tilde{\mathbf{R}}_1^{(k)},\ldots,\tilde{\mathbf{R}}_M^{(k)}\}_{k=0,1\ldots}$ are monotonically increasing sequences of matrices with upper bound $\mathbb{1}$ and converge to $\{\tilde{\mathbf{R}}_1^{(\infty)},\ldots,\tilde{\mathbf{R}}_M^{(\infty)}\}$ that give the least solution of $\tilde{\mathbf{E}}_q(\mathrm{DB}'_i)$ (14).

We can also prove Lemma 5 by analyzing the form of the right-hand side of equation shown in (11) (proof omitted).

Lemma 5
$\left(\mathbf{R}_h^{(k)}\right)_{ij} = 1$ if-and-only-if $\left(\tilde{\mathbf{R}}_h^{(k)}\right)_{ij} > 0$ for $\forall\, k \in \mathbb{N}, 1 \leqslant h \leqslant M, 1 \leqslant i,j \leqslant N$.

Finally, from Lemma 5, we conclude Theorem 2 that generalizes Theorem 1 (proof omitted):

Theorem 2

Let DB be a linear program in $\mathscr{C}_{lin}$ partitioned and topologically sorted in the ascending order as $\mathrm{DB} = \mathrm{DB}'_1 \cup \cdots \cup \mathrm{DB}'_L$ by the SCC ordering of DB. Also, let $\mathbf{E}_q(\mathrm{DB}'_i)$ be a system of non-linear matrix equations (12) for matrices encoding head predicates $\{r_1, \ldots, r_M\}$ of DB'_i. We suppose predicates in layers below DB'_i are already computed.

Choose a positive number $\epsilon_h\,(1 \leqslant h \leqslant M)$ so that $0 < \epsilon_h F_h[\mathbb{1}, \ldots, \mathbb{1}] \leqslant \mathbb{1}$ holds in $\mathbf{E}_q(\mathrm{DB}'_i)$. Let $\tilde{\mathbf{E}}_q(\mathrm{DB}'_i)$ be a system of linear matrix equations (14) and $\{\tilde{\mathbf{R}}^*_h\}_{h=1,\ldots,M}$ be the least solution of (14). Define, for $h\,(1 \leqslant h \leqslant M)$, $\mathbf{R}^*_h \in \{0,1\}^{N\times N}$ by

$$(\mathbf{R}^*_h)_{ij} = \begin{cases} 1 & \text{if } (\tilde{\mathbf{R}}^*_h)_{ij} > 0 \\ 0 & \text{o.w.} \end{cases} \qquad (1 \leqslant i, j \leqslant N)$$

Then, $\{\mathbf{R}^*_h\}_{h=1,\ldots,M}$ are the least solution of $\mathbf{E}_q(\mathrm{DB}'_i)$ encoding $\{r_1, \ldots, r_M\}$ in the least Herbrand model $\mathbf{M}_{\mathrm{DB}'_{\leqslant i}}$.

What Theorem 2 tells us is that we can evaluate a Datalog program $\mathrm{DB} = \mathrm{DB}'_1 \cup \cdots \cup \mathrm{DB}'_L$ in $\mathscr{C}_{lin}$ by computing the least solution of $\tilde{\mathbf{E}}_q(\mathrm{DB}'_i)$ in turn for $i = 1, \ldots, L$ and by converting resulting solution matrices to 0-1 matrices by thresholding.

6 Solving a system of linear matrix equations

Let DB be a Datalog program in $\mathscr{C}_{lin}$ and write $\mathrm{DB} = \mathrm{DB}'_1 \cup \cdots \cup \mathrm{DB}'_L$ as before. Put $\tilde{\mathbf{E}}_q(\mathrm{DB}) \stackrel{\mathrm{def}}{=} \bigcup_{i=1}^{L} \tilde{\mathbf{E}}_q(\mathrm{DB}'_i)$, where $\tilde{\mathbf{E}}_q(\mathrm{DB}'_i)$ is a system of linear matrix equations for the ith layer program DB'_i. $\tilde{\mathbf{E}}_q(\mathrm{DB})$ is called a system of linear matrix equations for DB.

We here discuss how to compute the least solution of $\tilde{\mathbf{E}}_q(\mathrm{DB})$, or equivalently, the least solution of each $\tilde{\mathbf{E}}_q(\mathrm{DB}'_i)$ (14):

$$\begin{aligned} \tilde{\mathbf{R}}_1 &= \epsilon_1 F_1[\tilde{\mathbf{R}}_1, \ldots, \tilde{\mathbf{R}}_M] \\ &\cdots \\ \tilde{\mathbf{R}}_M &= \epsilon_M F_M[\tilde{\mathbf{R}}_1, \ldots, \tilde{\mathbf{R}}_M]. \end{aligned}$$

Here, $F_h\,(1 \leqslant h \leqslant M)$ is written as (11):

$$F_h[\tilde{\mathbf{R}}_1, \ldots, \tilde{\mathbf{R}}_M] = \mathbf{A}_1 + \mathbf{B}_1\tilde{\mathbf{R}}^{\circ}_{j_1}\mathbf{C}_1 + \cdots + \mathbf{B}_q\tilde{\mathbf{R}}^{\circ}_{j_q}\mathbf{C}_q.$$

Solving $\tilde{\mathbf{E}}_q(\mathrm{DB}'_i)$ is not a simple task and the difficulty varies with the form of $\tilde{\mathbf{E}}_q(\mathrm{DB}'_i)$. So, we discuss three program classes, i.e., *tail recursive class*, *transposed class* and *two-sided class*, each generating different types of $\tilde{\mathbf{E}}_q(\mathrm{DB})$. We explain them subsequently using examples.

6.1 Tail recursive class

This class is a direct generalization of the transitive closure program. A program $\mathrm{DB} = \mathrm{DB}'_1 \cup \cdots \cup \mathrm{DB}'_L \in \mathscr{C}_{lin}$ is tail recursive if each layer program DB'_i consists of

clauses of the form

$$r(x,z) \leftarrow s_1(x,y_1) \wedge \cdots \wedge s_n(y_{n-1},z)$$
$$r(x,z) \leftarrow s_1(x,y_1) \wedge \cdots \wedge s_n(y_{n-1},y) \wedge t(y,z) \tag{16}$$

where r and t are mutually dependent predicates in DB'_i and the $s_i(\cdot,\cdot)$'s are defined in layers below DB'_i. The translation of if-and-only-if completions of these clauses into matrix equations yields a system of matrix equations of the following form:

$$\tilde{\mathbf{R}}_h = \epsilon_h(\mathbf{A}_1 + \mathbf{B}_0\tilde{\mathbf{R}}_h + \mathbf{B}_1\tilde{\mathbf{R}}_{j_1} + \cdots + \mathbf{B}_q\tilde{\mathbf{R}}_{j_q}). \tag{17}$$

This is uniquely solvable if $\epsilon_h < \frac{1}{\|\mathbf{B}_0\|_\infty}$ and the solution

$$\tilde{\mathbf{R}}_h = (\mathbf{I} - \epsilon_h\mathbf{B}_0)^{-1}\epsilon_h(\mathbf{A}_1 + \mathbf{B}_1\tilde{\mathbf{R}}_{j_1} + \cdots + \mathbf{B}_q\tilde{\mathbf{R}}_{j_q})$$

is computed in $O(N^3)$. By substituting the solution for $\tilde{\mathbf{R}}_h$ in other matrix equations, we can eliminate $\tilde{\mathbf{R}}_h$ and eventually, by repeatedly solving matrix equations M times for $\tilde{\mathbf{R}}_1,\ldots,\tilde{\mathbf{R}}_M$, reach a unique solution, i.e., the least solution of $\tilde{\mathbf{E}}_q(\mathrm{DB})$ (details omitted).

6.2 *Transposed class*

Programs $\mathrm{DB} \in \mathscr{C}_{lin}$ in this class generate $\tilde{\mathbf{E}}_q(\mathrm{DB})$ comprised of matrix equations of the following form:

$$\tilde{\mathbf{R}}_h = \epsilon_h(\mathbf{A}_1 + \mathbf{B}_1\tilde{\mathbf{R}}_{j_1}^T + \cdots + \mathbf{B}_q\tilde{\mathbf{R}}_{j_q}^T). \tag{18}$$

An example of this class, DB_2, is shown below.

$$r_2(x,z) \leftarrow r_1(x,z)$$
$$r_2(x,z) \leftarrow r_1(x,y) \wedge r_2(z,y) \tag{19}$$

The difference from the transitive closure program (5) is that $r_2(y,z)$ in (5) is replaced by $r_2(z,y)$. So the arguments of $r_2(y,z)$ are interchanged. In such case, $\tilde{\mathbf{R}}_2^T$, the transpose of $\tilde{\mathbf{R}}_2$ for $r_2(y,z)$, gives a matrix encoding $r_2(z,y)$. Assuming that $\mathbf{R}_1 \in \{0,1\}^{N\times N}$ for $r_1(\cdot,\cdot)$ is already computed, the linear matrix equation for $r_2(y,z)$ becomes

$$\tilde{\mathbf{R}}_2 = \epsilon(\mathbf{R}_1 + \mathbf{R}_1\tilde{\mathbf{R}}_2^T). \tag{20}$$

To ensure (20) has a least solution, we also assume ϵ satisfies $\epsilon \leqslant \frac{1}{1+\|\mathbf{R}_1\|_\infty}$ so that $\epsilon(\mathbf{R}_1 + \mathbf{R}_1\mathbf{1}^T) \leqslant \mathbf{1}$ holds.

One way to solve (20) is to substitute (20) into itself, resulting in

$$\begin{aligned}\tilde{\mathbf{R}}_2 &= \epsilon(\mathbf{R}_1 + \mathbf{R}_1\tilde{\mathbf{R}}_2^T)\\ &= \epsilon(\mathbf{R}_1 + \mathbf{R}_1(\epsilon(\mathbf{R}_1^T + \tilde{\mathbf{R}}_2\mathbf{R}_1^T))\\ &= \epsilon(\mathbf{R}_1 + \epsilon\mathbf{R}_1\mathbf{R}_1^T) + \epsilon^2\mathbf{R}_1\tilde{\mathbf{R}}_2\mathbf{R}_1^T.\end{aligned} \tag{21}$$

(21) is a case of two-sided class treated in the next subsection.

Another, more general, way is to transform (20) to a system of matrix equations about $\{\tilde{\mathbf{R}}_2, \tilde{\mathbf{R}}_3\}$ without transposition, by introducing a new matrix $\tilde{\mathbf{R}}_3 \stackrel{\text{def}}{=} \tilde{\mathbf{R}}_2^T$. We obtain

$$\begin{aligned}\tilde{\mathbf{R}}_2 &= \epsilon(\mathbf{R}_1 + \mathbf{R}_1\tilde{\mathbf{R}}_3)\\ \tilde{\mathbf{R}}_3 &= \epsilon(\mathbf{R}_1^T + \tilde{\mathbf{R}}_2\mathbf{R}_1^T).\end{aligned} \tag{22}$$

Again (22) is a case of two-sided class discussed in the next subsection.

6.3 Two-sided class

This is a more general class and much more difficult to evaluate than previous classes. Programs in this class have a recursive goal in the clause body which is sandwiched between two or more non-recursive goals. For simplicity, we assume they generate matrix equations of the following form:

$$\tilde{\mathbf{R}}_h = \epsilon_h(\mathbf{A}_1 + \mathbf{B}_0\tilde{\mathbf{R}}_h\mathbf{C}_0 + \mathbf{B}_1\tilde{\mathbf{R}}_{j_1}\mathbf{C}_1 + \cdots + \mathbf{B}_q\tilde{\mathbf{R}}_{j_q}\mathbf{C}_q). \tag{23}$$

A typical example is DB_3 below.

$$\begin{aligned}r_2(x,z) &\leftarrow r_1(x,z)\\ r_2(x,z) &\leftarrow r_1(x,y) \wedge r_2(y,w) \wedge r_3(w,z)\end{aligned} \tag{24}$$

The linear matrix equation computing $r_2(y,z)$ becomes

$$\tilde{\mathbf{R}}_2 = \epsilon_2(\mathbf{R}_1 + \mathbf{R}_1\tilde{\mathbf{R}}_2\mathbf{R}_3). \tag{25}$$

We assume $\mathbf{R}_1, \mathbf{R}_3 \in \{0,1\}^{N\times N}$ are already computed.

Equation (25) is an example of class of matrix equation called *discrete Sylvester equation*, which has been extensively studied in the field of control theory (Bartels and Stewart 1972; Golub *et al.* 1979; Jonsson and Kågström 2002; Saberi *et al.* 2007; Simoncini 2013).

A condition on ϵ for (25) to have a unique solution is stated in the literature using eigen values of $\mathbf{R}_1$ and $\mathbf{R}_3$, but we need a concrete criterion to decide ϵ. So, we rewrite (25) to an equivalent vector equation (26), using the fact that $\mathrm{vec}(\mathbf{AXB}) = (\mathbf{B}^T \otimes \mathbf{A})\mathrm{vec}(\mathbf{X})$ holds for any matrices $\mathbf{A}$, $\mathbf{X}$ and $\mathbf{B}$.

$$\mathrm{vec}(\tilde{\mathbf{R}}_2) = \epsilon_2(\mathrm{vec}(\mathbf{R}_1) + (\mathbf{R}_3^T \otimes \mathbf{R}_1)\mathrm{vec}(\tilde{\mathbf{R}}_2)) \tag{26}$$

It is now apparent that (26), hence (25), is uniquely solvable if $\epsilon_2\|\mathbf{R}_3^T \otimes \mathbf{R}_1\|_\infty < 1$, for example, $\epsilon_2 \leqslant \frac{1}{1+\|\mathbf{R}_3\|_1\|\mathbf{R}_1\|_\infty}$[5] and the solution is $\mathrm{vec}(\tilde{\mathbf{R}}_2) = (\mathbf{I}\otimes\mathbf{I} - \epsilon_2(\mathbf{R}_3^T \otimes \mathbf{R}_1))^{-1}\epsilon_2\mathrm{vec}(\mathbf{R}_1)$.

However, be warned that computing the solution this way requires $O(N^6)$ time because $\mathbf{R}_3^T \otimes \mathbf{R}_1$ is a $\{0,1\}^{N^2\times N^2}$ matrix. Fortunately, we can solve (25) directly in $O(N^3)$ as a discrete Sylvester equation (Granat *et al.* 2009), and hence we can obtain the least model of (24) in $O(N^3)$, an order of magnitude faster than $O(N^4)$ required by the tabled top-down evaluation method (Warren 1999).

[5] Recall that $\epsilon_2\|\mathbf{R}_3^T \otimes \mathbf{R}_1\|_\infty \leqslant \epsilon_2\|\mathbf{R}_3^T\|_\infty\|\mathbf{R}_1\|_\infty = \epsilon_2\|\mathbf{R}_3\|_1\|\mathbf{R}_1\|_\infty$.

In general, we can always convert matrix equations (23) to vector equations like (26) and solve them to obtain the least model of the original program, but this process requires $O(N^6)$ time, prohibitively large in practice. So a more desirable and viable approach is to solve (23) as a set of discrete Sylvester equations, which can be done in $O(N^3)$ for some programs as we have seen. However, when (23) forms a system of mutually recursive discrete Sylvester equations, solving (23) remains a challenging task, and regrettably, is left for future work.

7 Experiments

To empirically validate our matrix-based method for Datalog program evaluation (we hereafter refer to our matrix-based method as *the Matrix method* or just Matrix), we conduct two experiments[6]. The first experiment measures the computation time of Matrix for transitive closure computation to see if it is usable in practice. We use artificial data and real data. The second one compares Matrix and the state-of-the-art symbolic systems including two Prolog systems (B-Prolog (Zhou *et al.* 2010) and XSB (Swift and Warren 2012)) and two ASP systems (DLV (Alviano *et al.* 2010) and Clingo (Gebser *et al.* 2014)) in terms of the computation time required for computing the transitive closure relation and the same-generation relation which is explained later. We use artificial data. This experiment revealed an advantage of Matrix in speed over the compared systems in the case of non-sparse data.

7.1 Computation time for transitive closure: Matrix versus Iteration

Suppose $\mathbf{R}_1$ is an $N \times N$ adjacency matrix encoding a binary relation $r_1(x, y)$. We denote by trcl($\mathbf{R}_1$) the adjacency matrix that encodes the transitive closure of $r_1(x, y)$ and call it *the transitive closure matrix of* $\mathbf{R}_1$. We consider here two linear algebraic methods of computing trcl($\mathbf{R}_1$)[7].

The first one, termed the Iteration method or just Iteration, is a base-line method which is a faithful implementation of (8). It computes the least solution of $\mathbf{R}_2 = \min_1(\mathbf{R}_1 + \mathbf{R}_1\mathbf{R}_2)$ by iterating

$$\mathbf{R}_2^{(0)} = \mathbf{0}$$
$$\mathbf{R}_2^{(k+1)} = \min_1(\mathbf{R}_1 + \mathbf{R}_1\mathbf{R}_2^{(k)})$$

until convergence and returns the converged result as trcl($\mathbf{R}_1$).

The second one is Matrix which computes $(\mathbf{I}-\epsilon\mathbf{R}_1)^{-1}\epsilon\mathbf{R}_1$ ($\epsilon = \frac{1}{1+\|\mathbf{R}_1\|_\infty}$), thresholds matrix entries at 0 as described in Theorem 1 and returns the resulting matrix as trcl($\mathbf{R}_1$). $O(N^3)$ time is the theoretically expected time complexity but may deviate due to implementation details. We apply these two methods to compute trcl($\mathbf{R}_1$) and measure their computation time.

[6] All experiments are carried out on a PC with Intel(R) Core(TM) i7-3770@3.40 GHz CPU, 28 GB memory.

[7] All matrix computation here is done with GNU Octave4.0.0 (`https://www.gnu.org/software/octave/`).

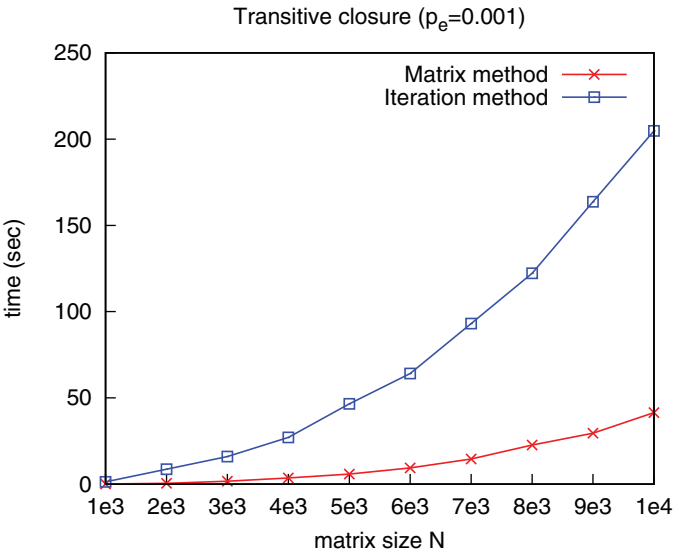

Fig. 1. Average computation time for trcl($\mathbf{R}_1$).

Prior to the experiment, we conducted a preliminary experiment to verify the correctness of the Matrix method. We implemented Warshall's algorithm as a third method which is a well-known algorithm for computing transitive closure in $O(N^3)$. We applied all three methods, i.e., Matrix, Iteration and Warshall's algorithm, to various $N \times N$ matrices $\mathbf{R}_1$ to see if they generate the same trcl($\mathbf{R}_1$). $\mathbf{R}_1$s were randomly generated in such a way that for $\forall i, j (1 \leqslant i, j \leqslant N), \mathbf{R}_1(i, j) = 1$ with a probability p_e (*edge probability*)[8]. We tested various p_e and N up to $N = 10^3$ and confirmed that all three methods agree and yield the same trcl($\mathbf{R}_1$).

After having checked the correctness of Matrix, we compare Matrix and Iteration. For each of various Ns ranging from 10^3 to 10^4, we generate $\mathbf{R}_1$ randomly with a fixed edge probability $p_e = 0.001$ and record the computation time for trcl($\mathbf{R}_1$) by Matrix and Iteration, respectively. We repeat this process five times and plot the average computation time (sec) w.r.t. N. The result is shown in Figure 1.

In the graph, the two methods behave similarly w.r.t. N though it is not clear whether their behavior is $O(N^3)$ or not. We also observe that Matrix constantly outperforms Iteration. For example, at $N = 10^4$(1e4), where $\mathbf{R}_1$ has 10^5 non-zero entries and trcl($\mathbf{R}_1$) has 10^8 non-zero entries on average, Matrix finishes its computation in 40 seconds and runs five times faster than Iteration. This graph shows that Matrix can deal with $10^4 \times 10^4$ sized or larger matrices[9].

[8] $\mathbf{R}_1$ encodes an Erdős–Rényi random graph, a well-known type of random graphs and it has $p_e N^2$ edges on average.

[9] Computation time may possibly change depending on the edge probability p_e which determines the density of $\mathbf{R}_1$. Our observation, however, suggests that Matrix's computation time is not much affected by p_e.

Table 1. *Transitive closure computation for real datasets*

Dataset (Koblenz Network Collection)	N	$\lvert\mathbf{R}_1\rvert$	$\lvert\text{trcl}(\mathbf{R}_1)\rvert$	Time (sec)	
				Iteration	Matrix
moreno-blogs	1,224	19,025	982,061	0.7	0.1
reactome	6,327	147,547	4,744,333	106.0	11.3
dblp-cite	12,591	49,743	7,583,575	1,238.1	86.4
subelj-cora	23,166	91,500	93,584,386	17,893.7	497.2
ego-twitter	23,370	33,101	353,882	5,481.9	654.9

```
r2(X,Z):- r1(X,Z).                    r2(X,W):- diag(X,W).
r2(X,Z):- r1(X,Y),r2(Y,Z).            r2(X,W):- r1(X,Y),r2(Y,Z),r1(W,Z).
```

Fig. 2. Tested programs: transitive closure program (left) and same_generation program (right).

We also conduct a similar experiment of computing trcl($\mathbf{R}_1$) with real data. Datasets are taken from the Koblenz Network Collection (`http://konect.uni-koblenz.de/`) (Kunegis 2013). We choose five network graphs with different characters and convert them to adjacency matrices $\mathbf{R}_1$. We then compute their transitive closure matrices trcl($\mathbf{R}_1$) by Iteration and Matrix. Table 1 summarizes the result. There N is the number of entities. $|\mathbf{R}_1|$ is the number of non-zero entries of $N \times N$ matrix $\mathbf{R}_1$ and similarly for $|\text{trcl}(\mathbf{R}_1)|$. Matrix and Iteration indicate their respective computation time. As with the case of artificial data, Matrix outperforms Iteration in speed for all datasets, roughly by an order of magnitude.

We emphasize that although this experiment is a proof-of-concept experiment, the result is encouraging and suggests the potential of our linear algebraic approach.

7.2 *Comparing with the state-of-the-art systems*

We next compare our linear algebraic approach with current major symbolic approaches, i.e., logic programming and ASP. We select two state-of-the-art tabled Prolog systems, B-Prolog8.1 and XSB3.6 and two state-of-the-art ASP systems, DLV(DEC-17-2012 version) and Clingo4.5.4. We let them compute the least Herbrand models of Datalog programs and compare computation time with computation time by the proposed linear algebraic approach of computing matrices encoding the models.

We pick up two linear Datalog programs in $\mathscr{C}_{lin}$ shown in Figure 2. They are a transitive closure program (left) and a program for computing the same generation relation (right). We assume that `r1(X,Z)` and `diag(X,Z)` are extensional predicates defined by a set of ground atoms. In particular, we assume `diag(X,Z)` represents equality `X=Z` and the corresponding ground atoms are of the form `diag(a,a)`, `diag(b,b)`,...

Table 2. *Average computation time for transitive closure computation (sec)*

p_e	Matrix	B-Prolog	XSB	DLV	Clingo
0.0001	0.096	0.000	0.000	0.000	0.000
0.001	0.094	0.004	0.003	0.293	0.038
0.01	0.117	2.520	1.746	10.657	14.618
0.1	0.105	18.382	16.296	75.544	125.993
1.0	0.100	188.280	137.903	483.380	1,073.301

Both programs in Figure 2 define `r2(X,Z)` in the least Herbrand model for a given `r1(X,Z)`. They look syntactically similar but are substantially different; the left one is tail-recursive and hence tail-recursive optimization is possible from the viewpoint of Prolog but the right one is not. Also, the left one defines as `r2(X,Z)` an ancestor relation when `r1(X,Z)` is interpreted as a parent relation (`X` is a parent of `Z`) but the right one defines the same generation relation as `r2(X,Z)` (`X` and `Z` belong to the same generation).

Given these programs, DLV and Clingo automatically compute their least Herbrand models by grounding followed by search for stable models. However, B-Prolog and XSB are designed to answer a query by SLD refutation with tabling. So to let them compute the least models for `r2(X,Z)`, or to compute all solutions for the query `?-r2(X,Y)`, we drive them by a failure loop below and ask `?-top` to measure computation time[10].

```
top:- r2(X,Y),fail.
top.
```

In the experiment, we first use the left program in Figure 2 and measure computation time for transitive closure computation. We set N, the number of entities, to 1,000. Then, we choose p_e and randomly generate an $N \times N$ random adjacency matrix $\mathbf{R}_1$ with edge probability p_e. Finally, we convert $\mathbf{R}_1$ to a set of ground atoms $\mathrm{EDB}(\mathtt{r1}) = \{\mathtt{r1}(i,j) \mid \mathbf{R}_1(i,j) = 1, 1 \leqslant i,j \leqslant N = 1,000\}$.

Next, we run four systems, B-Prolog, XSB, DLV and Clingo, to measure their computation time for the transitive closure relation `r2(X,Y)` of EDB(`r1`). We also compute a transitive closure matrix $\mathrm{trcl}(\mathbf{R}_1)$ encoding `r2(X,Y)` by the Matrix method and measure computation time. We repeat this process five times and compute average computation time for each system. The average computation time for various p_e is listed in Table 2 (column names like Matrix, B-Prolog, XSB, DLV and Clingo indicate the used system).

As seen from Table 2, Matrix finishes transitive closure computation in almost constant time (0.1 second) irrespective of p_e but symbolic systems heavily depend on p_e. This is primarily because the average number of ground atoms in EDB(`r1`) is proportional to p_e. Note that $p_e = 0$ means all entries in $\mathbf{R}_1$ are 0, whereas $p_e = 1.0$

[10] B-Prolog, XSB and Clingo display computation time when the computation terminates. We used `-stats` option to obtain computation time by DLV.

Table 3. *Average computation time for the same generation computation (sec)*

p_e	Matrix	B-Prolog	XSB	DLV	Clingo
0.0001	8.475	0.001	0.001	0.003	0.002
0.001	8.545	0.234	0.805	0.187	0.068
0.01	10.160	1379.090	timeout	139.879	195.970
0.1	10.546	timeout	timeout	timeout	timeout

means all entries in $\mathbf{R}_1$ are 1. For $p_e \leqslant 0.001$, Matrix runs slower than the symbolic systems but for $p_e \geqslant 0.01$, it overwhelms them, runs $15 \sim 10^4$ times faster.

We conduct a similar experiment with the same generation program with $N = 1,000$ while varying p_e. Using the right program in Figure 2 and systematically changing `r1(X,Y)` defined by EDB(`r1`) just as the case of the transitive closure program, we measure average computation time over five runs to compute `r2(X,Y)` for each of Matrix, B-Prolog, XSB, DLV and Clingo.

For the symbolic systems, all we need to compute `r2(X,Y)` for the same generation is to replace the left program with the right program in Figure 2. However, Matrix (now we use it as a term referring to our linear algebraic approach) needs to compute the least fixed point of the matrix equation below:

$$\mathbf{R}_2 = \min_1(\mathbf{I} + \mathbf{R}_1\mathbf{R}_2\mathbf{R}_1^T) \tag{27}$$

($\mathbf{I}$ is an identity matrix)

We therefore first solve $\tilde{\mathbf{R}}_2 = \epsilon(\mathbf{I}+\mathbf{R}_1\tilde{\mathbf{R}}_2\mathbf{R}_1^T)$ with $\epsilon = \frac{1}{1+\|\mathbf{R}_1\|_\infty^2}$, then threshold $\tilde{\mathbf{R}}_2$ at 0 to obtain $\mathbf{R}_2$. Although this equation is not simply solvable by the inverse matrix operation, it is still solvable as a discrete Sylvester equation. Average computation time for each system is summarized in Table 3. Here, timeout signifies computation required more than one hour and was aborted.

Looking at Table 3, one notices the same tendency as Table 2, i.e., Matrix takes almost constant time w.r.t. p_e while the symbolic systems drastically change their computation time depending on p_e. Also, it is observed that when p_e is small ($\leqslant 0.001$), Matrix's performance is relatively poor but for $p_e \geqslant 0.01$, it overwhelms them, ten times or hundreds times faster, just as the case of transitive closure computation.

8 Related work and discussion

Applying linear algebra to logical computation is not new. For example, the SAT problem is formulated using matrices and vectors in Lin (2013). Concerning Datalog, Ceri *et al.* (1989) describes a bottom-up evaluation method which is essentially identical to the one referred to as "Iteration" in Section 7.1. Our approach is neither bottom-up nor iterative. It abolishes iteration and replaces it with inverse matrix application. Also, there are a couple of papers concerning KGs that evaluate ground atoms in a vector space. Grefenstette (2013), for example, successfully embeds Herbrand models in tensor spaces but the embedding excludes quantified

formulas; they need to be treated separately by another framework which does not accept nested quantification. So his formalism is not applicable to our case that embeds Datalog programs into vector spaces, as Datalog programs can have nested existential quantifiers in their clause bodies.

The most technically relevant work is RESCAL (Nickel 2013; Nickel *et al.* 2015), which represents binary relations $r(x, y)$ by bilinear form $(\mathbf{x} \bullet \mathbf{R}\mathbf{y})$. RESCAL is designed to perform approximate inference for truth values of ground atoms in low-dimensional vector spaces and exact inference like ours is not treated. The work done by Rocktäschel *et al.* (2015) intersects our approach. They encode using one-hot encoding pairs of entities to vectors $\mathbf{e}$, binary relations to vectors $\mathbf{r}$, and represent the truth value as the inner product $(\mathbf{r} \bullet \mathbf{e})$. However, unlike us, their encoding is intended solely for approximate inference. No recursion is considered either.

There remain numerous problems to be tackled for further development of our linear algebraic approach. For example, extending binary predicates to arbitrary predicates is one of them. Also, extending the class of Datalog programs beyond linear ones is a big problem. Theoretically, there is no difficulty in dealing with such non-linear programs in a vector space. The hitch is the difficulty in solving derived matrix equations. Consider a non-linear Datalog program for transitive closure:

```
r2(X,Z):- r1(X,Z).
r2(X,Z):- r2(X,Y),r2(Y,Z).
```

The least Herbrand model of the above program is straightforwardly obtained by computing the least solution of $\mathbf{R}_2 = \min_1(\mathbf{R}_1 + \mathbf{R}_2\mathbf{R}_2)$ using the Iteration method we described before. However, if one hopes for efficient computation along the line of the Matrix method, we need to compute a non-negative matrix solution $\tilde{\mathbf{R}}_2$ of the following matrix equation:

$$\tilde{\mathbf{R}}_2 = \epsilon(\mathbf{R}_1 + \tilde{\mathbf{R}}_2\tilde{\mathbf{R}}_2) \tag{28}$$

using an appropriate ϵ[11]. Unfortunately, (28) is a system of multivariate polynomial equations such that the number of variables easily goes up to 10^4 or larger. Solving such equations exactly is a highly technical problem in general and no off-the-shelf answer seems currently available.

Another concern is negation. Our approach is obviously applicable to Datalog programs with stratified negation, as negated atoms $\neg r(x, y)$ in lower layers are expressed by $\mathbb{1} - \mathbf{R}$, where $\mathbf{R}$ is an adjacency matrix encoding $r(x, y)$. If programs are non-stratified, however, the Matrix method, originally designed for definite clause programs, needs to be extended in a fundamental way, which is an interesting but challenging future work.

Finally, although our approach has been successfully applied to domains with tens of thousands of entities where programs are in $\mathscr{C}_{lin}$ and self-recursive as evidenced by the experiments in Section 7, other cases require specific consideration and

[11] To ensures the existence of the least non-negative solution of (28) for $N \times N$ matrix $\mathbf{R}_1$, $\epsilon = \frac{1}{\|\mathbf{R}_1\|_\infty + N}$ is enough.

implementation. In particular, mutual recursive programs that have large SCCs seem difficult to deal with when the size of their matrix equations get large.

9 Conclusion

We introduced an innovative linear algebraic approach to Datalog evaluation for a class $\mathscr{C}_{lin}$ of Datalog programs with binary predicates and linear recursion. We showed how to translate a program DB $\in \mathscr{C}_{lin}$ to a system of linear matrix equations $\tilde{\mathbf{E}}_q(\mathrm{DB})$ and proved that thresholding the solution matrices of $\tilde{\mathbf{E}}_q(\mathrm{DB})$ gives adjacency matrices encoding the relations in the least Herbrand model $\mathbf{M}_{\mathrm{DB}}$ computed by DB.

The validity of our approach is empirically verified through two experiments. The first experiment computed the least Herbrand model of a transitive closure program for artificial data and real data. It is confirmed that our approach can efficiently deal with real network graphs containing more than 2×10^4 nodes. The second experiment compared our approach with the state-of-the-art symbolic systems including two tabled Prolog systems (B-Prolog8.1 and XSB3.6), and two modern ASP systems (DLV(DEC-17-2012 version) and Clingo4.5.4). We measured average time for computing the least Herbrand models of two Datalog programs respectively in the domain of 10^3 constants using $10^3 \times 10^3$ matrices. It is observed that our linear algebraic approach runs $10^1 \sim 10^4$ times faster than the symbolic systems when data is not sparse.

Acknowledgement

This research is supported in part by a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

References

Alviano, M., Faber, W., Leone, N., Perri, S., Pfeifer, G. and Terracina, G. 2010. The disjunctive datalog system DLV. In *Datalog Reloaded, LNCS 6702*, O. de Moor, G. Gottlob, T. Furche, and A. Sellers, Eds. Springer, Berlin, 282–301.

Bartels, R. and Stewart, G. 1972. Solution of the matrix equation AX + XB = C. *Communication of the ACM 15,* 9.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T. and Taylor, J. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proc. of the 2008 ACM SIGMOD International Conference on Management of data*, ACM, New York, NY, USA, 1247–1250.

Ceri, S., Gottlob, G. and Tanca, L. 1989. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering 1,* 1, 146–166.

Cichocki, A., Zdunek, R., Phan, A.-H. and Amari, S. 2009. *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*. Chichester, West Sussex, UK, John Wiley & Sons, Ltd.

Coppersmith, D. and Winograd, S. 1990. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation 9,* 3, 251–280.

Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S. and Zhang, W. 2014. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proc. of 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD2014*, ACM, New York, NY, USA, 601–610.

Gebser, M., Kaminski, R., Kaufmann, B. and Schaub, T. 2014. Clingo = ASP + Control: Preliminary Report. In M. Leuschel and T. Schrijvers, Eds. Technical Communications of the Thirtieth International Conference on Logic Programming (ICLP'14), Vol. 14(4–5). 1–9.

Golub, G., Nash, S. and Van Loan, C. 1979. A Hessenberg-Schur method for the problem AX + XB = C. *IEEE Transion Automated Control AC-24*, Vol. 24(6). 909–913.

Granat, R., Jonsson, I. and Kågström, B. 2009. RECSY and SCASY library software: Recursive blocked and parallel algorithms for Sylvester-type matrix equations with some applications. *Parallel Scientific Computing and Optimization*, Vol. 27, 3–24.

Grefenstette, E. 2013. Towards a formal distributional semantics: Simulating logical calculi with tensors. In *Proc. of the 2nd Joint Conference on Lexical and Computational Semantics*, Association for Computational Linguistics (ACL), Stroudsburg, PA 18360 USA, 1–10.

Jonsson, I. and Kågström, B. 2002. Recursive blocked algorithms for solving triangular systems – Part II: Two-sided and generalized Sylvester and Lyapunov matrix equations. *ACM Transactions on Mathematical Software 28,* 4, 392–415.

Kolda, T. G. and Bader, B. W. 2009. Tensor decompositions and applications. *SIAM Review 51,* 3, 455–500.

Krompass, D., Nickel, M. and Tresp, V. 2014. Querying factorized probabilistic triple databases. In *Proc. of the 13th International Semantic Web Conference(ISWC'14)*, P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. Knoblock, D. Vrandeac, P. Groth, N. Noy, K. Janowicz, C. Goble, Eds. Springer-Verlag New York, Inc., New York, NY, USA, 114–129.

Kunegis, J. 2013. KONECT – The Koblenz network collection. In *Proc. of the International Conference on World Wide Web Companion*, ACM, New York, NY, USA, 1343–1350.

Lin, F. 2013. *From Satisfiability to Linear Algebra.* Technical report, Hong Kong University of Science and Technology.

Lloyd, J. 1993. *Foundations of Logic Programming*, 2nd ed. Springer-Verlag, New York, Inc.

Nickel, M. 2013. Tensor factorization for relational learning. PhD Thesis, Ludwig-Maximilians-Universitat Munchen.

Nickel, M., Murphy, K., Tresp, V. and Gabrilovich, E. 2015. A review of relational machine learning for knowledge graphs: From multi-relational link prediction to automated knowledge graph construction. *CoRR abs/1503.00759*, Proceedings of the IEEE, 104(1), pp. 11–33.

Rocktäschel, T., Bosnjak, M., Singh, S. and Riedel, S. 2014. Low-dimensional embeddings of logic. In Proceedings of the ACL 2014 *ACL Workshop on Semantic Parsing (SP'14)*, Association for Computational Linguistics, pp. 45–49.

Rocktäschel, T., Singh, S. and Riedel, S. 2015. Injecting logical background knowledge into embeddings for relation extraction. Association for Computational Linguistics Eds. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 1119–1129.

Saberi, A., Stoorvogel, A. and Sannuti, P. 2007. Filtering Theory. With Applications to Fault Detection, Isolation, and Estimation, Birkhauser, Boston, Mass, USA, 2007. Birkhäuser, Boston.

Simoncini, V. 2013. Computational Methods for Linear Matrix Equations. Technical report, SIAM REVIEW, 58(3), pp. 377–441.

Suchanek, F. M., Kasneci, G. and Weikum, G. 2007. YAGO: A core of semantic knowledge unifying WordNet and Wikipedia. In *Proc. of the 16th International World Wide Web Conference(WWW'07)*, ACM, New York, NY, USA, 697–706.

Swift, T. and Warren, D. 2012. XSB: Extending prolog with tabled logic programming. *Theory and Practice of Logic Programming (TPLP) 12,* 1–2, 157–187.

Tarjan, R. E. 1972. Depth-first search and linear graph algorithms. *SIAM Journal on Computing 1,* 2, 146–160.

Tekle, K. T. and Liu, Y. A. 2010. Precise complexity analysis for efficient datalog queries. In *Proc. of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*, ACM, New York, NY, USA, 35–44.

Warren, D. S. 1999. Programming in Tabled Prolog (very) DRAFT 1. Technical Report, Stony Brook University.

Yang, B., Yih, W., He, X., Gao, J. and Deng, L. 2015. Embedding entities and relations for learning and inference in knowledge bases. In *Proc. of the International Conference on Learning Representations (ICLR) 2015.*

Zhou, N.-F., Kameya, Y. and Sato, T. 2010. Mode-directed tabling for dynamic programming, machine learning, and constraint solving. In *Proc. of the 22nd International Conference on Tools with Artificial Intelligence (ICTAI-2010)*, IEEE Computer Society, Washington DC, USA, 213–218.