An Extension of Datalog for Graph Queries

M. Mazuran\textsuperscript{1}  E. Serra\textsuperscript{2}  C. Zaniolo\textsuperscript{3}

Politecnico di Milano DEI\textsuperscript{1}– mazuran@elet.polimi.it
University of Calabria DEIS\textsuperscript{2}– eserra@deis.unical.it
University of California, Los Angeles UCLA\textsuperscript{3}– zaniolo@cs.ucla.edu

June 26, 2012
1 Motivations

2 Datalog$^{FS}$

3 Applications

4 Computation and Optimization

5 Conclusion and Future Work
Motivation
Motivation

A resurgence of interest in Datalog in many research areas—in particular "Big Data".

Parallel fixpoint-based computation of recursive predicates dovetails with MapReduce framework [Afrati et al.]

There are several limitations due to the fact that aggregates cannot be allowed in recursive definitions since they are non-monotone.

Most systems only support aggregates outside recursion, i.e., in programs that are stratified w.r.t. aggregates and negation.

Much previous work has not produced a general solution.
Motivation

- A resurgence of interest in Datalog in many research areas—in particular “Big Data”.

Parallel fixpoint-based computation of recursive predicates dovetails with MapReduce framework [Afrati et al.].

There are several limitations due to the fact that aggregates cannot be allowed in recursive definitions since they are non-monotone.

Most systems only support aggregates outside recursion, i.e., in programs that are stratified w.r.t. aggregates and negation.

Much previous work has not produced a general solution.
Motivation

- A resurgence of interest in Datalog in many research areas—in particular “Big Data”.
- Parallel fixpoint-based computation of recursive predicates dovetails with MapReduce framework [Afrati et al.]
Motivation

- A resurgence of interest in Datalog in many research areas—in particular “Big Data”.
- Parallel fixpoint-based computation of recursive predicates dovetails with MapReduce framework [Afrati et al.]
- There are several limitation due to the fact that aggregates cannot be allowed in recursive definitions since they are non-monotone.
Motivation

- A resurgence of interest in Datalog in many research areas—in particular “Big Data”.
- Parallel fixpoint-based computation of recursive predicates dovetails with MapReduce framework [Afrati et al.]
- There are several limitation due to the fact that aggregates cannot be allowed in recursive definitions since they are non-monotone.
- Most systems only support aggregates outside recursion, i.e., in programs that are stratified w.r.t. aggregates and negation.
Motivation

- A resurgence of interest in Datalog in many research areas—in particular “Big Data”.
- Parallel fixpoint-based computation of recursive predicates dovetails with MapReduce framework [Afrati et al.]
- There are several limitation due to the fact that aggregates cannot be allowed in recursive definitions since they are non-monotone.
- Most systems only support aggregates outside recursion, i.e., in programs that are stratified w.r.t. aggregates and negation.
- Much previous work has not produced a general solution.
A resurgence of interest in Datalog in many research areas—in particular “Big Data”.

Parallel fixpoint-based computation of recursive predicates dovetails with MapReduce framework [Afrati et al.]

There are several limitation due to the fact that aggregates cannot be allowed in recursive definitions since they are non-monotone.

Most systems only support aggregates outside recursion, i.e., in programs that are stratified w.r.t. aggregates and negation.

Much previous work has not produced a general solution.
An Extension of Datalog for Graph Queries

Datalog$^{FS}$
An Extension of Datalog for Graph Queries

Datalog

A program $P$ is a finite set of rules. A rule is of the form $A \leftarrow A_1, \ldots, A_m$ where $A$ is the head and $A_i$ is a body literal (can be negated or not).

Stratified Negation.

Reachability Example

\[
\text{path}(X, Y) \leftarrow \text{arc}(X, Y)
\]

\[
\text{path}(X, Y) \leftarrow \text{path}(X, Z), \text{path}(Z, Y)
\]

Datalog $^{FS}$

An extended Datalog that allows reasoning about the number of distinct occurrences satisfying a conjunction of goals. It adds 2 special constructs that appear as body literals:

1. Frequency Support goal (FS goal).
2. Final-FS goal: derived from the first.

and define a new type of predicates called Multi-Occuring Predicates.

Datalog $^{FS}$
Datalog

- A program $P$ is a finite set of rules.

Datalog$_{FS}$

An extended Datalog that allows reasoning about the number of distinct occurrences satisfying a conjunction of goals.

It adds 2 special constructs that appear as body literals:

1. Frequency Support goal (FS goal).
2. Final-FS goal: derived from the first.

and define a new type of predicates called Multi-Occurring Predicates.
From Datalog to Datalog\textsuperscript{FS}

Datalog

- A program $P$ is a finite set of rules.
- A rule is of the form $A \leftarrow A_1, \ldots, A_m$ where $A$ is the head and $A_i$ is a body literal (can be negated or not).

Datalog\textsuperscript{FS}

An extended Datalog that allows reasoning about the number of distinct occurrences satisfying a conjunction of goals.

It adds 2 special constructs that appear as body literals:

1. Frequency Support goal (FS goal).
2. Final-FS goal: derived from the first.

and define a new type of predicates called Multi-Occuring Predicates.
An Extension of Datalog for Graph Queries

Datalog

A program $P$ is a finite set of rules.

A rule is of the form $A \leftarrow A_1, \ldots, A_m$ where $A$ is the head and $A_i$ is a body literal (can be negated or not).

Stratified Negation.

Datalog\(^{FS}\)

An extended Datalog that allows reasoning about the number of distinct occurrences satisfying a conjunction of goals.

It adds 2 special constructs that appear as body literals:

1. Frequency Support goal (FS goal).
2. Final-FS goal: derived from the first.

and define a new type of predicates called Multi-Occuring Predicates.
An Extension of Datalog for Graph Queries

**Datalog**

- A program \( P \) is a finite set of rules.
- A rule is of the form \( A \leftarrow A_1, \ldots, A_m \) where \( A \) is the head and \( A_i \) is a body literal (can be negated or not).
- Stratified Negation.
- Reachability Example
  
  \[
  \begin{align*}
  \text{path}(X, Y) & \leftarrow \text{arc}(X, Y) \\
  \text{path}(X, Y) & \leftarrow \text{path}(X, Z), \text{path}(Z, Y).
  \end{align*}
  \]

**Datalog\(^FS\)**

An extended Datalog that allows reasoning about the number of distinct occurrences satisfying a conjunction of goals. It adds 2 special constructs that appear as body literals:

1. Frequency Support goal (FS goal).
2. Final-FS goal: derived from the first.

And define a new type of predicates called Multi-Occuring Predicates.
An Extension of Datalog for Graph Queries

Datalog

A program $P$ is a finite set of rules.

A rule is of the form $A \leftarrow A_1, \ldots, A_m$ where $A$ is the head and $A_i$ is a body literal (can be negated or not).

Stratified Negation.

Reachability Example

\[
\text{path}(X, Y) \leftarrow \text{arc}(X, Y)
\]

\[
\text{path}(X, Y) \leftarrow \text{path}(X, Z), \text{path}(Z, Y).
\]

Datalog\textsuperscript{FS}

An extended Datalog that allows reasoning about the number of distinct occurrences satisfying a conjunction of goals
An extended Datalog that allows reasoning about the number of distinct occurrences satisfying a conjunction of goals.

It adds 2 special constructs that appear as body literals:

- Frequency Support goal (FS goal).
- Final-FS goal: derived from the first. 

Define a new type of predicates called Multi-Occuring Predicates.
From Datalog to Datalog\(^FS\)

Datalog

- A program P is a finite set of rules.
- A rule is of the form \( A \leftarrow A_1, \ldots, A_m \) where \( A \) is the head and \( A_i \) is a body literal (can be negated or not).
- Stratified Negation.
- Reachability Example

\[
\begin{align*}
\text{path}(X, Y) & \leftarrow \text{arc}(X, Y) \\
\text{path}(X, Y) & \leftarrow \text{path}(X, Z), \text{path}(Z, Y).
\end{align*}
\]

Datalog\(^FS\)

- An extended Datalog that allows reasoning about the number of distinct occurrences satisfying a conjunction of goals
- It adds 2 special constructs that appear as body literals:
  - Frequency Support goal (FS goal).
An Extension of Datalog for Graph Queries

Datalog

A program P is a finite set of rules.
A rule is of the form $A \leftarrow A_1, \ldots, A_m$ where $A$ is the head and $A_i$ is a body literal (can be negated or not).
Stratified Negation.
Reachability Example

\[
\text{path}(X,Y) \leftarrow \text{arc}(X,Y)
\]
\[
\text{path}(X,Y) \leftarrow \text{path}(X,Z), \text{path}(Z,Y).
\]

Datalog$^{FS}$

An extended Datalog that allows reasoning about the number of distinct occurrences satisfying a conjunction of goals.
It adds 2 special constructs that appear as body literals:

1. Frequency Support goal (FS goal).
2. Final-FS goal: derived from the first.
From Datalog to Datalog\textsuperscript{FS}

### Datalog
- A program $P$ is a finite set of rules.
- A rule is of the form $A \leftarrow A_1, \ldots, A_m$ where $A$ is the head and $A_i$ is a body literal (can be negated or not).
- Stratified Negation.
- Reachability Example

\[
\text{path}(X, Y) \leftarrow \text{arc}(X, Y)
\]
\[
\text{path}(X, Y) \leftarrow \text{path}(X, Z), \text{path}(Z, Y).
\]

### Datalog\textsuperscript{FS}
- An extended Datalog that allows reasoning about the number of distinct occurrences satisfying a conjunction of goals.
- It adds 2 special constructs that appear as body literals:
  1. Frequency Support goal (FS goal).
  2. Final-FS goal: derived from the first.
- and define a new type of predicates called Multi-Occurring Predicates.
**Datalog\(^{FS}\) constructs**

- **Frequency Support goal**

- **Final-FS goal**

**Frequency Support goal**

- \( K \): \([ Bexpr(X, Y) \]) where:
  - \( K \) is a positive integer variable and
  - \( Bexpr(X, Y) \) is a conjunction of positive literals with variables \( X \) and \( Y \).

**Example about friends that will come to the party.**

\[
\text{willcome}(X) \leftarrow \text{sure}(X).
\]

\[
\text{willcome}(X) \leftarrow 3: \{ \text{friend}(X, Y), \text{willcome}(Y) \}.
\]

**Semantics:** there exist at least \( K \) assignments of variables \( Y \) that satisfy the conjunction \( Bexpr(X, Y) \) (\( \text{friend}(X, Y), \text{willcome}(Y) \)).

**It is monotone and can be used in recursion.**

**Final-FS goal**

- \( K: \{ Bexpr(X, Y) \} \), \( \neg K + 1: \{ Bexpr(X, Y) \} \).

**Semantics by using FS-goal and Negation:**

**It is not monotone and requires stratified negation.**
**Datalog$^{FS}$ constructs**

### Frequency Support goal

- $K : [\text{Bexpr}(X, Y)]$ where:
  - $K$ is a positive integer variable and
  - $\text{Bexpr}(X, Y)$ is a conjunction of positive literals with variables $X$ and $Y$.

### Final-FS goal

- $K = ![[\text{Bexpr}(X, Y)]$
An Extension of Datalog for Graph Queries

Datalog$^{FS}$

Datalog$^{FS}$ constructs

**Frequency Support goal**

- $K : [\text{Bexpr}(X, Y)]$ where:
  - $K$ is a positive integer variable and
  - $\text{Bexpr}(X, Y)$ is a conjunction of positive literals with variables $X$ and $Y$.

- Example about friends that will come to the party.
  \[
  \text{willcome}(X) \leftarrow \text{sure}(X).
  \]
  \[
  \text{willcome}(X) \leftarrow 3: \text{[friend}(X, Y), \text{willcome}(Y)].
  \]

**Final-FS goal**
**Datalog\(^{FS}\) constructs**

**Frequency Support goal**

- \(K : [\text{Bexpr}(X, Y)]\) where:
  - \(K\) is a positive integer variable and
  - \(\text{Bexpr}(X, Y)\) is a conjunction of positive literals with variables \(X\) and \(Y\).

- Example about friends that will come to the party.
  - \(\text{willcome}(X) \leftarrow \text{sure}(X)\).
  - \(\text{willcome}(X) \leftarrow 3: [\text{friend}(X, Y), \text{willcome}(Y)]\).

- Semantics: there exist at least \(K\) assignments of variables \(Y\) that satisfy the conjunction \(\text{Bexpr}(X, Y)\) (\(\text{friend}(X, Y), \text{willcome}(Y)\)).

**Final-FS goal**
**Datalog\(^{FS}\) constructs**

**Frequency Support goal**
- \( K : [Bexpr(X, Y)] \) where:
  - \( K \) is a positive integer variable and
  - \( Bexpr(X, Y) \) is a conjunction of positive literals with variables \( X \) and \( Y \).
- Example about friends that will come to the party.
  
  \[
  \text{willcome}(X) \leftarrow \text{sure}(X).
  \]

  \[
  \text{willcome}(X) \leftarrow 3: [\text{friend}(X, Y), \text{willcome}(Y)].
  \]

- Semantics: there exist at least \( K \) assignments of variables \( Y \) that satisfy the conjunction \( Bexpr(X, Y) \) \((\text{friend}(X, Y), \text{willcome}(Y))\).
- It is monotone and can be used in recursion.

**Final-FS goal**
An Extension of Datalog for Graph Queries

Datalog$^{FS}$

Datalog$^{FS}$ constructs

**Frequency Support goal**
- $K : \lbrack Bexpr(X, Y) \rbrack$ where:
  - $K$ is a positive integer variable and
  - $Bexpr(X, Y)$ is a conjunction of positive literals with variables $X$ and $Y$.
- Example about friends that will come to the party.
  - `willcome(X) ← sure(X).`
  - `willcome(X) ← 3:[friend(X, Y), willcome(Y)].`
- Semantics: there exist at least $K$ assignments of variables $Y$ that satisfy the conjunction $Bexpr(X, Y)$ ($friend(X, Y), willcome(Y)$).
- It is monotone and can be used in recursion.

**Final-FS goal**
- $K = ![Bexpr(X, Y)]$
Datalog^{FS} constructs

**Frequency Support goal**

- **K : \[Bexpr(X, Y)\]** where:
  - K is a positive integer variable and
  - Bexpr(X, Y) is a conjunction of positive literals with variables X and Y.

- Example about friends that will come to the party.
  - `willcome(X) ← sure(X).`
  - `willcome(X) ← 3 : [friend(X, Y), willcome(Y)].`

- Semantics: there exist at least K assignments of variables Y that satisfy the conjunction `Bexpr(X, Y)` (friend(X, Y), willcome(Y)).

- It is monotone and can be used in recursion.

**Final-FS goal**

- **K =! [Bexpr(X, Y)]**

- Example about person that has exactly 10 friends:
  - `p(X) ← 10 =! [friend(X, Y)].`
Datalog$^{FS}$ constructs

**Frequency Support goal**
- $K : [\text{Bexpr}(X, Y)]$ where:
  - $K$ is a positive integer variable and
  - $\text{Bexpr}(X, Y)$ is a conjunction of positive literals with variables $X$ and $Y$.

Example about friends that will come to the party.

```
willcome(X) ← sure(X).
willcome(X) ← 3 : [friend(X, Y), willcome(Y)].
```

Semantics: there exist at least $K$ assignments of variables $Y$ that satisfy the conjunction $\text{Bexpr}(X, Y)$ ($\text{friend}(X, Y), \text{willcome}(Y)$).

It is monotone and can be used in recursion.

**Final-FS goal**
- $K = ! [\text{Bexpr}(X, Y)]$

Example about person that has exactly 10 friends:

```
p(X) ← 10 = ! [friend(X, Y)].
```

Semantics by using FS-goal and Negation:

```
K : [\text{Bexpr}(X, Y)], \neg K + 1 : [\text{Bexpr}(X, Y)].
```
Datalog\textsuperscript{FS} constructs

**Frequency Support goal**

- \( K : [\text{Bexpr}(X, Y)] \) where:
  - \( K \) is a positive integer variable and
  - \( \text{Bexpr}(X, Y) \) is a conjunction of positive literals with variables \( X \) and \( Y \).

  Example about friends that will come to the party.
  
  \[
  \text{willcome}(X) \leftarrow \text{sure}(X).
  \]
  
  \[
  \text{willcome}(X) \leftarrow 3 : [\text{friend}(X, Y), \text{willcome}(Y)].
  \]

  Semantics: there exist at least \( K \) assignments of variables \( Y \) that satisfy the conjunction \( \text{Bexpr}(X, Y) \) (\( \text{friend}(X, Y), \text{willcome}(Y) \)).

  It is monotone and can be used in recursion.

**Final-FS goal**

- \( K =! [\text{Bexpr}(X, Y)] \)

  Example about person that has exactly 10 friends:
  
  \[
  p(X) \leftarrow 10 =! [\text{friend}(X, Y)].
  \]

  Semantics by using FS-goal and Negation:
  
  \[
  K : [\text{Bexpr}(X, Y)], \neg K + 1 : [\text{Bexpr}(X, Y)].
  \]

  It is not monotone and requires stratified negation.
Multi-occurring predicates

Multi-occurring predicates

Multi-occurring predicates

- \( m - \text{predicate}(x): k \)
Multi-occurring predicates

- \( m - \text{predicate}(x):k \)
- Semantics: the \( x \) value has \( k \) occurrences.
Multi-occurring predicates

- $m \cdot \text{predicate}(x): k$
- Semantics: the $x$ value has $k$ occurrences.
- Facts Examples
Multi-occurring predicates

- \( m - \text{predicate}(x): k \)
- Semantics: the \( x \) value has \( k \) occurrences.
- Facts Examples
  - \( \text{ref}("Bob2012") : 6. \)
  - \( \text{ref}("Bob2012", journals) : 6. \)
  - \( \text{ref}("Bob2012", others) : 4. \)
- Rule example
  - \( \text{ref}("Bob2012") : 6. \)
  - \( \text{tref}(\text{Author}) : N \leftarrow N : [\text{author}(\text{Author, Pno}), \text{ref}(\text{Pno})]. \)
Multi-occurring predicates

- $m \_ predicate(x): k$
- Semantics: the $x$ value has $k$ occurrences.
- Facts Examples
  - $\text{ref}("\text{Bob2012}"): 6.$ $\text{ref}("\text{Bob2012}"): 4.$
  - $\text{ref}("\text{Bob2012}, \text{journals}"): 6.$ $\text{ref}("\text{Bob2012}, \text{others}"): 4.$
- Rule example
  - $\text{ref}("\text{Bob2012}"): 6.$
  - $\text{tref}(\text{Author}): N \leftarrow N : \left[ \text{author}(\text{Author}, \text{Pno}), \text{ref}(\text{Pno}) \right].$

$$total_{\_ref}(A) = \sum_{\text{Pno} \in \text{paper}(A)} \text{reference}(\text{Pno})$$
Multi-occurring predicates

- \( m - \text{predicate}(x) : k \)
- Semantics: the \( x \) value has \( k \) occurrences.
- Facts Examples
  - \( \text{ref("Bob2012") : 6.} \)
  - \( \text{ref("Bob2012", journals) : 6.} \)
  - \( \text{ref("Bob2012", others) : 4.} \)
- Rule example
  - \( \text{ref("Bob2012") : 6.} \)
  - \( \text{tref(Author) : N } \leftarrow N : [\text{author(Author, Pno), ref(Pno)}]. \)
- Semantics by using Datalog.
  - \( \text{ref("Bob2012", J) } \leftarrow \text{lessthan(J, 6).} \)
  - \( \text{tref(Author, N) } \leftarrow N : [\text{author(Author, Pno), ref(Pno, J)}]. \)
  - \( \text{lessthan(1, K) } \leftarrow K \geq 1. \)
  - \( \text{lessthan(J1, K) } \leftarrow \text{lessthan(J, K), } K > J, J1 = J + 1. \)
Applications
Some Examples

Example: Number of paths between two nodes in a Direct Acyclic Graph (DAG).

Example: Reachability in a directed hypergraph.

A hyperedge \( \{a, b\}, c \) is represented by the following facts:

\[
\begin{align*}
&\text{hedgeS} (1, a) \\
&\text{hedgeS} (1, b) \\
&\text{hedgeT} (1, c) \\
&\text{node} (a) \\
&\text{node} (b) \\
&\text{node} (c)
\end{align*}
\]

\[
\begin{align*}
&\text{reach} (X, X) \leftarrow \text{node} (X) \\
&\text{reach} (X, Y) \leftarrow K : [\text{hedgeS} (ID, Z), \text{reach} (X, Z)], \text{hedgeT} (ID, Y), K \neq [\text{hedgeS} (ID, Z)]
\end{align*}
\]
Some Examples

Example

Number of paths between two nodes in a Direct Acyclic Graph (DAG).

\[
\text{path}(X, Y) : 1 \leftarrow \text{edge}(X, Y)
\]

\[
\text{path}(X, Y) : K \leftarrow K : [\text{path}(X, Z), \text{path}(Z, Y)].
\]
Some Examples

Example

Number of paths between two nodes in a Direct Acyclic Graph (DAG):

\[ n_{-path}(X, Y) = \sum_{Z \neq X, Y} n_{-path}(X, Z) \times n_{-path}(Z, Y) \]
Some Examples

Example

Number of paths between two nodes in a Direct Acyclic Graph (DAG).

\[
\text{path}(X, Y) : 1 \leftarrow \text{edge}(X, Y) \\
\text{path}(X, Y) : K \leftarrow K : [\text{path}(X, Z), \text{path}(Z, Y)].
\]

Example

Reachability in a directed hyper graph.
A hyperedge \(\{a, b\}, c\) is represented by the following facts:

- hedgeS(1, a).  hedgeS(1, b).  hedgeT(1, c).
- node(a).  node(b).  node(c).

\[
\text{reach}(X, X) \leftarrow \text{node}(X).
\]

\[
\text{reach}(X, Y) \leftarrow K : [\text{hedgeS}(\text{ID}, Z), \text{reach}(X, Z)], \\
\text{hedgeT}(\text{ID}, Y), K =! [\text{hedgeS}(\text{ID}, \_)].
\]
Some Examples

Example

Number of paths between two nodes in a Direct Acyclic Graph (DAG).

\[
\begin{align*}
\text{path}(X, Y) & : 1 \leftarrow \text{edge}(X, Y) \\
\text{path}(X, Y) & : K \leftarrow K : [\text{path}(X, Z), \text{path}(Z, Y)].
\end{align*}
\]

Example

Reachability in a directed hyper graph.

A hyperedge \( \{a, b\}, c \) is represented by the following facts:

\[
\begin{align*}
\text{hedgeS}(1, a). & \quad \text{hedgeS}(1, b). \quad \text{hedgeT}(1, c). \\
\text{node}(a). & \quad \text{node}(b). \quad \text{node}(c).
\end{align*}
\]

\[
\begin{align*}
\text{reach}(X, X) & \leftarrow \text{node}(X). \\
\text{reach}(X, Y) & \leftarrow K : [\text{hedgeS}(\text{ID}, Z), \text{reach}(X, Z)], \\
& \quad \text{hedgeT}(\text{ID}, Y), K = ![\text{hedgeS}(\text{ID}, \_)].
\end{align*}
\]

Bill of Material
Some Examples

**Example**

Number of paths between two nodes in a Direct Acyclic Graph (DAG).

\[
\text{path}(X, Y) : 1 \leftarrow \text{edge}(X, Y) \\
\text{path}(X, Y) : K \leftarrow K: [\text{path}(X, Z), \text{path}(Z, Y)].
\]

**Example**

Reachability in a directed hyper graph.
A hyperedge \( \{a, b\}, c \) is represented by the following facts:

\[
\text{hedgeS}(1, a). \quad \text{hedgeS}(1, b). \quad \text{hedgeT}(1, c). \\
\quad \text{node}(a). \quad \text{node}(b). \quad \text{node}(c).
\]

\[
\text{reach}(X, X) \leftarrow \text{node}(X). \\
\text{reach}(X, Y) \leftarrow K : [\text{hedgeS}(ID, Z), \text{reach}(X, Z)], \\
\quad \text{hedgeT}(ID, Y), K =! [\text{hedgeS}(ID, _)].
\]

- **Bill of Material**
- **Path of minimum and maximum cost (more probable path)**
Some Examples

**Example**

Number of paths between two nodes in a Direct Acyclic Graph (DAG).

\[
\text{path}(X, Y) : 1 \leftarrow \text{edge}(X, Y)
\]

\[
\text{path}(X, Y) : K \leftarrow K : [\text{path}(X, Z), \text{path}(Z, Y)].
\]

**Example**

Reachability in a directed hyper graph.

A hyperedge \(\{a, b\}, c\) is represented by the following facts:

\[
\begin{align*}
\text{hedge}_S(1, a). & \quad \text{hedge}_S(1, b). & \quad \text{hedge}_T(1, c). \\
\text{node}(a). & \quad \text{node}(b). & \quad \text{node}(c).
\end{align*}
\]

\[
\begin{align*}
\text{reach}(X, X) & \leftarrow \text{node}(X). \\
\text{reach}(X, Y) & \leftarrow K : [\text{hedge}_S(ID, Z), \text{reach}(X, Z)], \\
& \quad \text{hedge}_T(ID, Y), K \neq [\text{hedge}_S(ID, \_)].
\end{align*}
\]

- **Bill of Material**
- **Path of minimum and maximum cost (more probable path)**
- **Dynamic Programming (Knapsak, Viterbi, ...)**
- ........
Jackson-Yariv Diffusion Model

Models how social structures influence the spread of behavior and trends in social networks (diffusion of innovations, behaviors, information and political movements).

Given a graph $G = \langle V, E \rangle$ where $V$ are agents and $E$ are edges denoting their relationships, each agent has a default behavior $A$ and decides on whether to adopt a new behavior $B$ based on:

$$B_i = bc_i \ast g(\Gamma_i) \ast 1 \Gamma_i \ast \sum_{(j, i) \in E} B_j, \forall i \in V$$

- A constant $bc$ to denote how much an agent is susceptible to make a change.
- A function $g$ denoting how much the number of neighbors influence the change.
- The percentage of neighbors that changed behavior.
Jackson-Yariv Diffusion Model

- Models how social structures influence the spread of behavior and trends in social networks (diffusion of innovations, behaviors, information and political movements).
Jackson-Yariv Diffusion Model

- Models how social structures influence the spread of behavior and trends in social networks (diffusion of innovations, behaviors, information and political movements).
- Given a graph $G = \langle V, E \rangle$ where $V$ are agents and $E$ are edges denoting their relationships, each agent has a default behavior $A$ and decides on whether to adopt a new behavior $B$ based on:
Jackson-Yariv Diffusion Model

- Models how social structures influence the spread of behavior and trends in social networks (diffusion of innovations, behaviors, information and political movements).
- Given a graph \( G = \langle V, E \rangle \) where \( V \) are agents and \( E \) are edges denoting their relationships, each agent has a default behavior \( A \) and decides on whether to adopt a new behavior \( B \) based on:

\[
B_i = bc_i \ast g(\Gamma_i) \ast \frac{1}{\Gamma_i} \ast \sum_{(j,i) \in E} B_j, \ \forall i \in V
\]
Jackson-Yariv Diffusion Model

- Models how social structures influence the spread of behavior and trends in social networks (diffusion of innovations, behaviors, information and political movements).
- Given a graph $G = \langle V, E \rangle$ where $V$ are agents and $E$ are edges denoting their relationships, each agent has a default behavior $A$ and decides on whether to adopt a new behavior $B$ based on:

$$B_i = bc_i \times g(\Gamma_i) \times \frac{1}{\Gamma_i} \times \sum_{(j,i) \in E} B_j, \ \forall i \in V$$

- a constant $bc$ to denote how much an agent is susceptible to make a change.
Models how social structures influence the spread of behavior and trends in social networks (diffusion of innovations, behaviors, information and political movements).

Given a graph $G = \langle V, E \rangle$ where $V$ are agents and $E$ are edges denoting their relationships, each agent has a default behavior $A$ and decides on whether to adopt a new behavior $B$ based on:

$$B_i = bc_i \times g(\Gamma_i) \times \frac{1}{\Gamma_i} \times \sum_{(j,i) \in E} B_j, \ \forall i \in V$$

- a constant $bc$ to denote how much an agent is susceptible to make a change.
- a function $g$ denoting how much the number of neighbors influence the change.
Jackson-Yariv Diffusion Model

- Models how social structures influence the spread of behavior and trends in social networks (diffusion of innovations, behaviors, information and political movements).
- Given a graph $G = \langle V, E \rangle$ where $V$ are agents and $E$ are edges denoting their relationships, each agent has a default behavior $A$ and decides on whether to adopt a new behavior $B$ based on:

$$B_i = bc_i \times g(\Gamma_i) \times \frac{1}{\Gamma_i} \times \sum_{(j,i) \in E} B_j, \forall i \in V$$

- a constant $bc$ to denote how much an agent is susceptible to make a change.
- a function $g$ denoting how much the number of neighbors influence the change.
- the percentage of neighbors that changed behavior.
Jackson-Yariv Diffusion Model
Jackson-Yariv Diffusion Model

\[
\text{coeff}(X, C) \leftarrow \begin{cases} 
K2 = ![\text{followd}(Y, X)], \text{bc}(X, V1), \\
g(K2, V3), C = V1 \times V3/K2.
\end{cases}
\]

\[
b(X) \leftarrow \begin{cases} 
\text{source}(X), \\
\text{coeff}(X, C), K \geq 1/C, K : [\text{followd}(Y, X), \text{b}(Y)].
\end{cases}
\]

```
follwd(u_1, u_2).  bc(u_1, 1).  g(1, 1.2).
follwd(u_1, u_3).  bc(u_2, 0.9).  g(2, 2.3).
follwd(u_2, u_4).  bc(u_3, 0.5).
follwd(u_3, u_4).  bc(u_4, 1).
source(u_1).
```
Jackson-Yariv Diffusion Model

\[
\text{coeff}(X, C) \leftarrow \quad K2 = ![\text{followd}(Y, X)], \text{bc}(X, V1), \\
g(K2, V3), C = V1 \ast V3/K2.
\]

\[
b(X) \leftarrow \quad \text{source}(X).
\]

\[
b(X) \leftarrow \quad \text{coeff}(X, C), K \geq 1/C, K : [\text{followd}(Y, X), b(Y)].
\]

\[
\begin{align*}
\text{follwd}(u_1, u_2). & \quad \text{bc}(u_1, 1). & \quad g(1, 1.2). \\
\text{follwd}(u_1, u_3). & \quad \text{bc}(u_2, 0.9). & \quad g(2, 2.3). \\
\text{follwd}(u_2, u_4). & \quad \text{bc}(u_3, 0.5). \\
\text{follwd}(u_3, u_4). & \quad \text{bc}(u_4, 1). \\
\text{source}(u_1). & \quad & \\
\end{align*}
\]

The program derives the following atoms:

\[
\text{coeff}(u_2, 1.08), \quad \text{coeff}(u_3, 0.6), \quad \text{coeff}(u_4, 1.15), \\
b(u_1), \quad b(u_2), \quad b(u_4).
\]
Markov chains

A process that consists of a finite number of states. The process starts in one state and then moves from one state to another represented by the transition matrix $W$ of $s \times s$ components where $w_{ij}$ is the probability to go from state $i$ to state $j$ in one step. Given $P$ a vector of stabilized probabilities of cardinality $s$, then for each component we have:

$$p_i = \sum_{j=1}^{s} w_{ji} \cdot p_j$$

This is the equilibrium condition expressed by the fixpoint equation:

$$P = W \cdot P$$

Used for Page Rank.
Markov chains

- A process that consists of a finite number of states
Markov chains

- A process that consists of a finite number of states
- The process starts in one state and then moves from one state to another

Represented by the transition matrix $W$ of $s \times s$ components where $w_{ij}$ is the probability to go from state $i$ to state $j$ in one step.

Given $P$, a vector of stabilized probabilities of cardinality $s$, then for each component we have:

$$p_i = \sum_{j=1}^{s} w_{ji} \cdot p_j$$

This is the equilibrium condition expressed by the fixpoint equation:

$$P = W \cdot P$$

Used for Page Rank.
Markov chains

- A process that consists of a finite number of states
- The process starts in one state and then moves from one state to another
- Represented by the transition matrix $W$ of $s \times s$ components where $w_{ij}$ is the probability to go from state $i$ to state $j$ in one step

Given $P$ a vector of stabilized probabilities of cardinality $s$, then for each component we have:

$$ p_i = \sum_{j=1}^{s} w_{ji} \cdot p_j $$

This is the equilibrium condition expressed by the fixpoint equation:

$$ P = W \cdot P $$

Used for Page Rank.
Markov chains

- A process that consists of a finite number of states
- The process starts in one state and then moves from one state to another
- Represented by the transition matrix $W$ of $s \times s$ components where $w_{ij}$ is the probability to go from state $i$ to state $j$ in one step
- Given $P$, a vector of stabilized probabilities of cardinality $s$, then for each component we have:

$$p_i = \sum_{j=1}^{s} w_{ji} \cdot p_j$$

this is the equilibrium condition expressed by the fixpoint equation:

$$P = W \cdot P$$
Markov chains

- A process that consists of a finite number of states
- The process starts in one state and then moves from one state to another
- Represented by the transition matrix $W$ of $s \times s$ components where $w_{ij}$ is the probability to go from state $i$ to state $j$ in one step
- Given $P$ a vector of stabilized probabilities of cardinality $s$ then for each component we have:

$$p_i = \sum_{j=1}^{s} w_{ji} \cdot p_j$$

this is the equilibrium condition expressed by the fixpoint equation:

$$P = W \cdot P$$

- Used for Page Rank.
Markov chains
An Extension of Datalog for Graph Queries

Applications

Markov chains

\[
p_{st}(X) : K \leftarrow K : [p_{st}(Y), w_{mat}(Y, X)].
\]

\[
\begin{align*}
    w_{mat}(a, b) & : 1.0. \\
    w_{mat}(b, a) & : 0.5. \\
    w_{mat}(b, c) & : 0.5. \\
    w_{mat}(c, b) & : 1.0. \\
    p_{st}(a). & p_{st}(b). p_{st}(c).
\end{align*}
\]

where:
Markov chains

\[ p_{st}(X) : K \leftarrow K : [p_{st}(Y), w_{mat}(Y, X)]. \]

\begin{align*}
    w_{mat}(a, b) & : 1.0. \\
    w_{mat}(b, a) & : 0.5. \\
    w_{mat}(b, c) & : 0.5. \\
    w_{mat}(c, b) & : 1.0. \\
    p_{st}(a), & p_{st}(b), p_{st}(c).
\end{align*}

where:

- \( p_{st}(X) : K \) means the probability of staying in state \( X \) is \( K \)
Markov chains

\[
p_{\text{st}}(X) : K \leftarrow K : [p_{\text{st}}(Y), w_{\text{mat}}(Y, X)].
\]

\[
w_{\text{mat}}(a, b) : 1.0.
w_{\text{mat}}(b, a) : 0.5.
w_{\text{mat}}(b, c) : 0.5.
w_{\text{mat}}(c, b) : 1.0.
p_{\text{st}}(a). \ p_{\text{st}}(b). \ p_{\text{st}}(c).
\]

where:

- \( p_{\text{st}}(X) : K \) means the probability of staying in state \( X \) is \( K \)
- \( w_{\text{mat}}(Y, X) : W \) means the arc from \( Y \) to \( X \) has weight \( W \)
Markov chains

\[
p_{\text{st}}(X) : K \leftarrow K : [p_{\text{st}}(Y), w_{\text{mat}}(Y, X)].
\]

\[
w_{\text{mat}}(a, b) : 1.0.
\]
\[
w_{\text{mat}}(b, a) : 0.5.
\]
\[
w_{\text{mat}}(b, c) : 0.5.
\]
\[
w_{\text{mat}}(c, b) : 1.0.
\]
\[
p_{\text{st}}(a). \quad p_{\text{st}}(b). \quad p_{\text{st}}(c).
\]

\textbf{where:}

- \textit{p_{\text{st}}(X) : K} means the probability of staying in state \( X \) is \( K \)
- \textit{w_{\text{mat}}(Y, X) : W} means the arc from \( Y \) to \( X \) has weight \( W \)

\[
p_{\text{st}}(a) : 0.25. \quad p_{\text{st}}(b) : 0.5. \quad p_{\text{st}}(c) : 0.25.
\]
Computation and Optimization

Computation and Optimization

Computation and Optimization
An Extension of Datalog for Graph Queries

Computation and Optimization

Frequency Support Goal is monotone

Fixpoint Algorithm for positive programs.

Differential Fixpoint.

Magic Set.

Final-FS goal: its semantics is defined by using the negation.

The stratified fixpoint for programs with stratified negation is used.

Multiplicity predicate: we only store the maximum value of multiplicity.
Frequency Support Goal is monotone
- Fixpoint Algorithm for positive programs.
- Differential Fixpoint.
- Magic Set.
An Extension of Datalog for Graph Queries

Computation and Optimization

**Computation and Optimization**

- Frequency Support Goal is monotone
  - Fixpoint Algorithm for positive programs.
  - Differential Fixpoint.
  - Magic Set.
- Final-FS goal: its semantics is defined by using the negation. The stratified fixpoint for programs with stratified negation is used.
Frequency Support Goal is monotone
- Fixpoint Algorithm for positive programs.
- Differential Fixpoint.
- Magic Set.

Final-FS goal: its semantics is defined by using the negation. The stratified fixpoint for programs with stratified negation is used.

Multiplicity predicate: we only store the maximum value of multiplicity.
Conclusion

- We proposed a simple extension of Datalog, called Datalog_FS, with two new constructs (FS goal and Final-FS goal) and a new type of predicate (Multiplicity predicate).
Conclusion

- We proposed a simple extension of Datalog, called Datalog$^{FS}$, with two new constructs (FS goal and Final-FS goal) and a new type of predicate (Multiplicity predicate).
- Datalog$^{FS}$ is more expressive than stratified Datalog and stratified Aggregates.
  It allows the recursion with aggregate (FS goal).
Conclusion

- We proposed a simple extension of Datalog, called Datalog$^{FS}$, with two new constructs (FS goal and Final-FS goal) and a new type of predicate (Multiplicity predicate).
- Datalog$^{FS}$ is more expressive than stratified Datalog and stratified Aggregates. It allows the recursion with aggregate (FS goal).
- There are a lot of problems that can be solved by Datalog$^{FS}$ with a simple and compact definition: Reachability Hyper Graph, Path of Minimum Cost, Diffusion Model, Page Rank ...
Conclusion

- We proposed a simple extension of Datalog, called Datalog$^{FS}$, with two new constructs (FS goal and Final-FS goal) and a new type of predicate (Multiplicity predicate).
- Datalog$^{FS}$ is more expressive than of stratified Datalog and stratified Aggregates. It allows the recursion with aggregate (FS goal).
- There are a lot of problems that can be solved by Datalog$^{FS}$ with a simple and compact definition: Reachability Hyper Graph, Path of Minimum Cost, Diffusion Model, Page Rank ...
- Datalog$^{FS}$ uses the same efficient optimization used for Datalog.
Conclusion

- We proposed a simple extension of Datalog, called Datalog\(^{FS}\), with two new constructs (FS goal and Final-FS goal) and a new type of predicate (Multiplicity predicate).
- Datalog\(^{FS}\) is more expressive than stratified Datalog and stratified Aggregates. It allows the recursion with aggregate (FS goal).
- There are a lot of problems that can be solved by Datalog\(^{FS}\) with a simple and compact definition: Reachability Hyper Graph, Path of Minimum Cost, Diffusion Model, Page Rank ...
- Datalog\(^{FS}\) uses the same efficient optimization used for Datalog.
- Recent works show how it is possible to use parallel paradigms (Map Reduce) to improve Datalog’s performances.
Conclusion

- We proposed a simple extension of Datalog, called Datalog$^{FS}$, with two new constructs (FS goal and Final-FS goal) and a new type of predicate (Multiplicity predicate).
- Datalog$^{FS}$ is more expressive than stratified Datalog and stratified Aggregates. It allows the recursion with aggregate (FS goal).
- There are a lot of problems that can be solved by Datalog$^{FS}$ with a simple and compact definition: Reachability Hyper Graph, Path of Minimum Cost, Diffusion Model, Page Rank ...
- Datalog$^{FS}$ uses the same efficient optimization used for Datalog.
- Recent works show how it is possible to use parallel paradigms (Map Reduce) to improve Datalog’s performances.
  - Datalog$^{FS}$ and in particular the multiplicity predicates can be also combined with such technologies.
Thank you!
Any question?
Compact way

Compact way of expressing counting goals:

- in Datalog$^{FS}$
  \[
  \text{sixsubs}(X) \leftarrow \text{detective}(X), 6 : [\text{superior}(X, Y)].
  \]

- in Datalog
  \[
  \text{sixsubs}(X) \leftarrow \text{detective}(X), \text{superior}(X, Y_1), \\
  \text{superior}(X, Y_2), \text{superior}(X, Y_3), \\
  \text{superior}(X, Y_4), \text{superior}(X, Y_5), \\
  \text{superior}(X, Y_6), Y_1 \neq Y_2, Y_1 \neq Y_3, \\
  Y_1 \neq Y_4, Y_1 \neq Y_5, Y_1 \neq Y_6, Y_2 \neq Y_3, \\
  Y_2 \neq Y_4, Y_2 \neq Y_5, Y_2 \neq Y_6, Y_3 \neq Y_4, \\
  Y_3 \neq Y_5, Y_3 \neq Y_6, Y_4 \neq Y_5, Y_4 \neq Y_6, \\
  Y_5 \neq Y_6.
  \]