Datalog

Susan B. Davidson

CIS 700: Advanced Topics in Databases

MW 1:30-3

Towne 309

http://www.cis.upenn.edu/~susan/cis700/homepage.html

Homework for this week

- Sign up to present a paper (the Google doc link was sent on Friday)
- Class schedule is being updated based on this.

First paper summary due 2/5

- First summary is on the following paper:
  - “Big Data Analytics with Datalog Queries on Spark” SIGMOD 2016
- What is a summary (print and bring to class)?
  - Short paragraph describing paper
  - 1-3 “strengths”, 1-3 “weaknesses”
  - At least one question you have about the paper.

Last time: Datalog

- Facts (EDB) and rules (IDB)
- Safe queries
- Negation can be tricky…

The Bachelor problem

Suppose we have an EDB relation married(x,y) and want to calculate the bachelors.

Not correct (and not safe):

```
bachelor(y) :- NOT married(x,y)
```

Also not correct (but safe):

```
bachelor(y) :- person(x), person(Y), NOT married(x,y)
```

Correct (and safe):

```
notBachelor(y):- married(x,y)
notBachelor(x):- married(x,y)
bachelor(y) :- person(y), NOT notBachelor(y)
```

This time: Datalog$^+$ with Recursion

- A simple recursive program and naïve evaluation
- Evaluating Datalog$^+$ programs
- Negation can still be tricky…

Datalog versus SQL

- Non-recursive Datalog with negation is a cleaned-up core of SQL
  - Unions of conjunctive queries
  - Forms the core of query optimization, what we know how to reason over easily
- You can translate easily between non-recursive Datalog with negation and SQL.
  - Take the join of the nonnegated, relational subgoals and select/delete from there.

Why Datalog?

- Recursion

- Rules express things that go on in both FROM and WHERE clauses, and let us state some general principles (e.g., containment of rules) that are almost impossible to state correctly in SQL.

Simple recursive Datalog program

R encodes a graph.

R=

1	2
2	1
2	3
1	4
3	4
4	5

What does T compute?

```
T(x,y):- R(x,y)
T(x,y):- R(x,z), T(z,y)
```

Naïve Evaluation

T= {}

WHILE (changes to T) DO

$\quad T = T \cup (R(x,y) \cup (R(x,y) \bowtie T(y,z)))$

Simple recursive Datalog program

R encodes a graph.

R=

1	2
2	1
2	3
1	4
3	4
4	5

Alternate ways to compute transitive closure:

T(x,y):- R(x,y)
T(x,y):- R(x,z), T(z,y)

Right linear

T(x,y):- R(x,y)
T(x,y):- T(x,z), R(z,y)

Left linear

T(x,y):- R(x,y)
T(x,y):- T(x,z), T(z,y)

Non-linear

Another Interesting Program

R encodes a graph.

R=

1	2
2	1
2	3
1	4
3	4
4	5

Non 2-colorability:

```
ODD(x,y):- R(x,y)
ODD(x,y):- R(x,z), EVEN(z,y)
EVEN(x,y):-R(x,z), ODD(z,y)
Q:- ODD(x,x)
```

Evaluating Datalog^{+} Programs

1. Nonrecursive programs.
2. Naïve evaluation of recursive programs without negation.
3. Semi-naïve evaluation of recursive programs without negation.
   - Eliminates some redundant computation.

Nonrecursive Evaluation

- If (and only if!) a Datalog program is not recursive, then we can order the IDB predicates so that in any rule for *p* (i.e., *p* is the head predicate), the only IDB predicates in the body precede *p*.

Why?

- Consider the *dependency graph* with:
  - Nodes = IDB predicates.
  - Arc $p \rightarrow q$ iff there is a rule for p with q in the body.
- Cycle involving node p means p is recursive.
- No cycles: use topological order to evaluate predicates.

Applying Rules

To evaluate an IDB predicate *p* :

1. *Apply* each rule for *p* to the current relations corresponding to its subgoals.
   - "Apply" = If an assignment of values to variables makes the body true, insert the tuple that the head becomes into the relation for *p* (no duplicates).
   - Also think of the "product" of the relations corresponding to the subgoals with selection/join conditions
2. Take the union of the result for each *p*-rule.

Example

Q={(1,2), (3,4)}
R={(2,5),(4,9),(4,10), (6,7)}

P(x,y) :- Q(x,z), R(z,y), y<10

- Assignments making the body true:

  (x,y,z) = (1,5,2), (3,9,4)

- So P = {(1,5), (3,9)}.

Algorithm for Nonrecursive

FOR (each predicate P in topological order) DO

Apply the rules for P to previously computed relations to compute relation P;

Naïve Evaluation for Recursive

make all IDB relations empty;

WHILE (changes to IDB) DO

FOR (each IDB predicate P) DO

Evaluate P using current values of all relations;

Important Points

- As long as there is no negation of IDB subgoals, then each IDB relation "grows," i.e., on each round it contains at least what it used to contain.
  - monotonicity
- Since relations are finite, the loop must eventually terminate.
- Result is the *least fixedpoint* (*minimal model*) of rules.

Problem with Naïve Evaluation

- The same facts are discovered over and over again.
- The semi-naïve algorithm tries to reduce the number of facts discovered multiple times.
  - There is a similarity to incremental view maintenance

Background: Incremental View Maintenance

V(x,y):- R(x,z),S(z,y)

If R← R U ΔR then what is ΔV(X,Y)?

ΔV(x,y):- ΔR(x,z),S(z,y)

If R← R U ΔR and S← S U ΔS then what is ΔV(X,Y)?

ΔV(x,y):- ΔR(x,z),S(z,y)
ΔV(x,y):- R(x,z), ΔS(z,y)
ΔV(x,y):- ΔR(x,z), ΔS(z,y)

Background: Incremental View Maintenance

V(x,y):- T(x,z),T(z,y)

If T← T U ΔT then what is ΔV(X,Y)?

ΔV(x,y):- ΔT(x,z),T(z,y)
ΔV(x,y):- T(x,z), ΔT(z,y)
ΔV(x,y):- ΔT(x,z), ΔT(z,y)

Semi-naïve Evaluation

- Key idea: to get a new tuple for relation P on one round, the evaluation must use some tuple for some relation of the body that was obtained on the previous round.
- Maintain ΔP = new tuples added to P on previous round.
- "Differentiate" rule bodies to be union of bodies with **one IDB subgoal made "Δ."**

Semi-naïve Evaluation

- Separate the Datalog program into the non-recursive, and the recursive part.
- Each P_i defined by non-recursive-$SPJU_i$ and (recursive-)$SPJU_i$.

$P1 = \Delta P1 =$ non-recursive-SPJU1,

$P2 = \Delta P2 =$ non-recursive-SPJU2,

…

Loop

$\Delta P1 = \Delta$ SPJU1 – P1; $\Delta P2 = \Delta$SPJU2 – P2; …

if ($\Delta P1 = \varnothing$ and $\Delta P2 = \varnothing$ and …) then break

$P1 = P1 \cup \Delta P1$; $P2 = P2 \cup \Delta P2$; …

Endloop

Semi-naïve Evaluation

$P_1 = \Delta P_1$ = non-recursive-$SPJU_1$,

$P_2 = \Delta P_2$ = non-recursive-$SPJU_2$,

…

Loop

$\Delta P1 = \Delta SPJU1 - P1$; $\Delta P2 = \Delta SPJU2 - P2$; …

if ($\Delta P_1 = \varnothing$ and $\Delta P_2 = \varnothing$ and …) then break

$P1 = P1 \cup \Delta P1$; $P2 = P2 \cup \Delta P2$; …

Endloop

T(x,y):- R(x,y)

T(x,y):- R(x,z), T(z,y)

T(x,y) = ΔT(x,y) = ? (nonrecursive rule)

Loop

ΔT(x,y) = ? (recursive Δ-rule)

if ($\Delta T = \varnothing$) then break

$T = T \cup \Delta T$

Endloop

Semi-naïve Evaluation

$P_1 = \Delta P_1$ = non-recursive-$SPJU_1$,

$P_2 = \Delta P_2$ = non-recursive-$SPJU_2$,

…

Loop

$\Delta P1 = \Delta\, SPJU1 - P1$; $\Delta P2 = \Delta SPJU2 - P2$; …

if ($\Delta P_1 = \varnothing$ and $\Delta P_2 = \varnothing$ and …) then break

$P1 = P1 \cup \Delta P1$; $P2 = P2 \cup \Delta P2$; …

Endloop

T(x,y):- R(x,y)

T(x,y):- R(x,z), T(z,y)

T(x,y) = ΔT(x,y) = R(x,y)

Loop

ΔT(x,y) = (R(x,z), ΔT(z,y))- R(x,y)

if ($\Delta T = \varnothing$) then break

$T = T \cup \Delta T$

Endloop

Discussion of Semi-Naïve Algorithm

- Avoids recomputing some (but not all) tuples
- Easy to implement, no disadvantage over naïve

- A rule is called *linear* if its body contains only one recursive IDB predicate:
  - A linear rule always results in a single incremental rule
  - A non-linear rule may result in multiple incremental rules

Recursion and Negation Don't Like Each Other

- When rules have negated IDB subgoals, there can be several minimal models.
- Recall: *model* = set of IDB facts, plus the given EDB facts, that make the rules true for every assignment of values to variables.
  - Rule is true unless body is true <u>and</u> head is false.

Suppose R(a).
What are S and T?

S(x):- R(x), not T(x)
T(x):- R(x), not S(x)

Next time: Datalog^{-}