Advanced Data Management

Medha Atre

Office: KD-219
atrem@cse.iitk.ac.in

Aug 4, 2016

First Assignment

First assignment is posted on the course webpage. Due date is August 15, 2016 23:59 IST. Submission instructions will be included within a few days in the assignment description. Please check the course webpage regularly for any important announcements, and assignment submission instructions.
www.cse.iitk.ac.in/users/atrem/courses/cs698f2016fall/

Recap

- Relational query optimization techniques.
- IBM System R optimizer concepts.
- Problems unique to the graph shaped data.
- BitMat indexing structure for a directed edge-labeled graph.
- Fold and unfold operations on a BitMat.

Graph data and queries

Data

:Jerry	:hasFriend	:Larry
:Jerry	:hasFriend	:Julia
:Larry	:actedIn	:CurbYourEnthu
:Julia	:actedIn	:Seinfeld
:Julia	:actedIn	:Veep
:Julia	:actedIn	:CurbYourEnthu
:Julia	:actedIn	:NewAdvOldChristine
:Seinfeld	:location	:NewYorkCity
:Veep	:location	:D.C.
:CurbYourEnthu	:location	:LosAngeles
:NewAdvOldChristine	:location	:Jersey

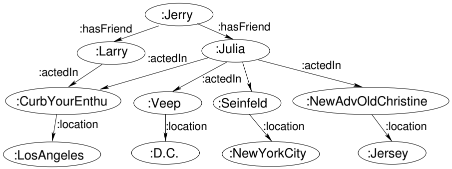
SPARQL

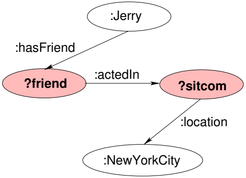
SELECT ?friend ?sitcom **WHERE** {
    :Jerry :hasFriend **?friend** .
    ?friend :actedIn **?sitcom** .
    ?sitcom :location :NewYorkCity .
}

Eqv. SQL query

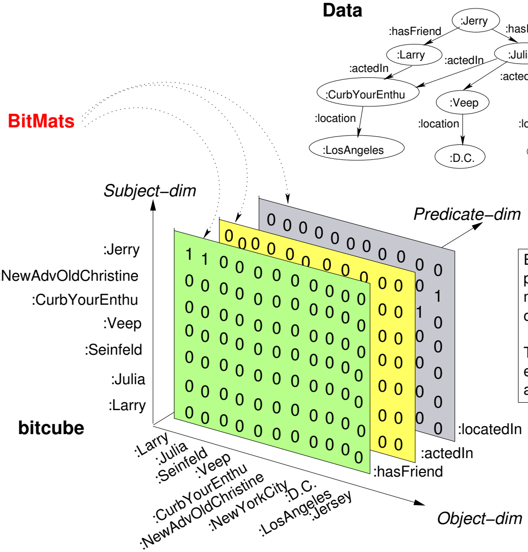
SELECT t1.o, t2.o from rdf as t1, rdf as t2, rdf as t3 **WHERE** t1.s=":Jerry" and t1.p=":hasFriend" and t2.p=":actedIn" and t3.p=":location" and t3.o=":NewYorkCity" and **t1.o=t2.s and t2.o=t3.s**

Graphical Representation





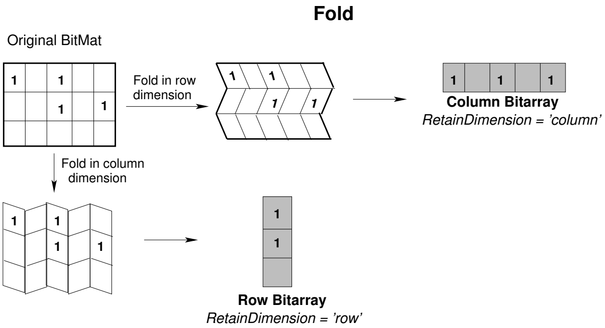
BitMat – brief overview



Each unique value of subjects, predicates, and objects in the data is mapped to the respective dimension of the bitcube.

This bitcube is then sliced along each dimension and the 2D BitMats are stored as the index structure.

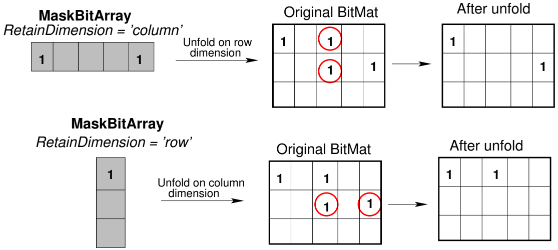
Fold and Unfold



fold(BM_{tp}, RetainDimension) procedure is nothing but projection of distinct values from the given dimension of BitMat, e.g., in the triple pattern (?friend :actedIn ?sitcom) if BM_{tp} is an O-S BitMat, then *?sitcom* is in the "row" dimension of the BitMat.

$$fold(BM_{tp}, dim_{?j}) \equiv \pi_{?j}(BM_{tp})$$

Unfold



For every *unset* bit in the *MaskBitArray*, *unfold(BM_{tp}, MaskBitArray, RetainDimension)* clears all the bits corresponding to that position of the *RetainDimension*.

$$unfold(BM_{tp}, \beta_{?j}, dim_{?j}) \equiv \{t \mid t \in BM_{tp}, t.?j \in \beta_{?j}\}$$

t is a triple in BM_{tp} that matches tp. $\beta_{?j}$ is the *MaskBitArray* containing bindings of $?j$ to be retained. $dim_{?j}$ is the dimension of BM_{tp} that represents $?j$, and $t.?j$ is a binding of $?j$ in triple t. In short, `unfold` keeps only those triples whose respective bindings of $?j$ are set to 1 in $\beta_{?j}$, and removes all other.

Semi-join and clustered-semi-join

- $tp_2 \ltimes_{?j} tp_1 = \pi_{attr(tp_2)}(tp_2 \bowtie_{?j} tp_1)$ is a *semi-join* [Bernstein1981, Ullman1989].
- A *clustered-semi-join* between $(tp_1, tp_2, \ldots tp_n)$ over $?j$ is similar to n-way semi-join.
- Semi-joins are achieved through the *fold* and *unfold* primitives of BitMat.

tp1 :Jerry :hasFriend ?friend

:Jerry	:hasFriend	:Larry
:Jerry	:hasFriend	:Julia

tp2 ?friend :actedIn ?sitcom .

:Larry	:actedIn	:CurbYourEnthu
:Julia	:actedIn	:Seinfeld
:Julia	:actedIn	:Veep
:Julia	:actedIn	:NewAdvOldChristine
:Julia	:actedIn	:CurbYourEnthu

tp3 ?sitcom :location :NewYorkCity .

:Seinfeld	:location	:NewYorkCity

$tp2 \ltimes_{(?friend)} tp1$

:Larry	:actedIn	:CurbYourEnthu
:Julia	:actedIn	:Seinfeld
:Julia	:actedIn	:Veep
:Julia	:actedIn	:NewAdvOldChristine
:Julia	:actedIn	:CurbYourEnthu

tp2 left with all the triples

$$clustered\text{-}semi\text{-}join(?sitcom, (tp2, tp3)) = \begin{pmatrix} tp2 \ltimes_{(?sitcom)} tp3 \\ tp3 \ltimes_{(?sitcom)} tp2 \end{pmatrix}$$

$tp2 \ltimes_{(?sitcom)} tp3$

:Julia	:actedIn	:Seinfeld

Now *tp2* left with only one triple

$tp3 \ltimes_{(?sitcom)} tp2$

:Seinfeld	:location	:NewYorkCity

tp3 left with the original one triple

Revisit the example of a low selectivity query

T1

N	M
1	a
2	a
3	a
1	b
2	b
3	b
4	b
1	c
2	c
3	c
1	d
2	d
3	d

$\bowtie$ T1.M=T2.M

T2

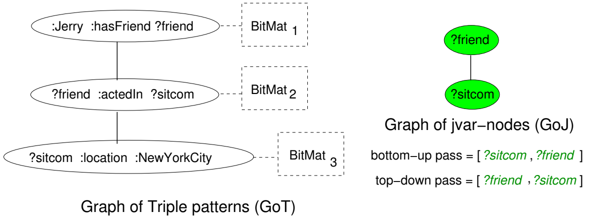
M	S
a	1
a	2
a	3
a	4
b	1
b	2
b	3
c	1
c	2
c	3
c	4
c	5
c	6
c	7
c	8

13 tuples in T1
15 tuples in T2
28 tuples in all

If we do a standard join of these two tables, we get **48** results (tuples) – a polynomial increase in the size of the results.

Instead, if we do a *semi-join* ($T1 \ltimes_M T2$) of $T1$ and $T2$, we are left with 10 tuples in $T1$ (3 tuples with M=‘d’ get eliminated from $T1$) and 15 in $T2$, **25** in all, *down* from 28 original tuples.

Inner-joins background



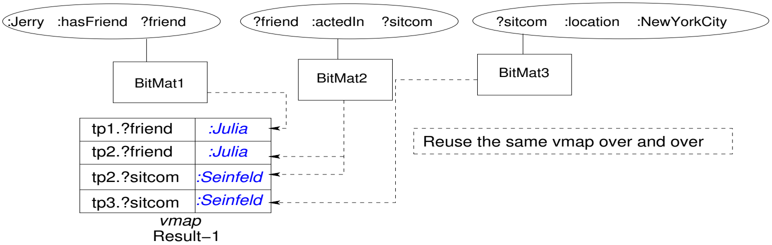
- If the Graph of Tables (GoT) is *acyclic (tree)*, then the tuples in each table can be reduced to a *minimal* by traversing the GoT in a *bottom-up* followed by *top-down* fashion, performing a *semi-join* at each table node [Bernstein1981, Ullman1989].
  - A table has *minimal* tuples for a query, if every tuple contributes to at least one final result, none of the tuples gets eliminated in the final result generation.
- If the *Graph of Triple Patterns* (GoT) is *acyclic*, the *Graph of Join-variables (GoJ)* is acyclic too, and vice versa (Lemma 3.2 in [Atre2015]).

Pattern Query Processing

- Choose the *least selective* join variable (jvar) as the root of the GoJ tree, so that more selective jvars are leaves[1], and do a bottom-up and top-down pass on GoJ with *clustered-semi-joins* at each jvar.
  - This leaves a *minimal* set of triples in the BitMat associated with each triple pattern.
- Do *n*-way *multi-join* to join all the triple patterns to produce the final results.

[1] Any jvar can be chosen as the root, but this *anti-greedy* selection favors query performance.

N-way multi-joins



How to evaluate?

Development environment: Lenovo T540p laptop with Intel Core i3-4000M 2.40GHz CPU, 8 GB memory, 12 GB swap space, and 1 TB Western Digital 5400RPM SATA hard disk, C/C++ language, compiler g++ v4.8.2, -O3 -m64 flags, 64 bit Linux 3.13.0-34-generic SMP kernel (Ubuntu 14.04 LTS distribution).

Competitive RDF stores: Virtuoso and MonetDB (contemporary research systems such as RDF-3X or TripleBit cannot process left-outer-join queries).

Datasets:

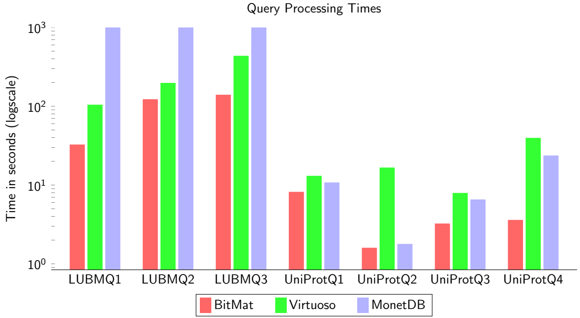
- LUBM: Synthetic university dataset with ≈ 1.33 billion triples.
- UniProt: A real life protein network with ≈ 845 million triples.
- DBPedia: RDF version of the Wikipedia network with ≈ 565 million triples.

Metrics

1. End-to-end query processing times for all three systems.
2. Time required to load initial data through BitMats.
3. Time required for pruning via semi-joins.
4. Time required for multi-way pipelined joins.
5. Initial number of triples required to be accessed by a query.
6. Triples left after pruning (semi-joins).
7. Total number of final results.

Metrics 5–7 help in understanding the *selectivity* of the queries.

How to present



Next Class

In the next class we will go over a some main research and general purpose graph storage and querying systems – RDF-3X, TripleBit, gStore, Neo4j, Openlink Virtuoso, MonetDB (database), Titan.
A much more comprehensive list on Wikipedia: https://en.wikipedia.org/wiki/Graph_database#List_of_graph_databases

Questions?