SPROUT:

Scalable Query Processing in Probabilistic Databases

Dan Olteanu
Oxford University Computing Laboratory

Joint work with Jiewen Huang (Oxford)

Alice looks for movies

Which movies are really good?
Manos: The Hands of Fate (1966)

IMDB:

- Lots of data
- Well maintained and clean :-)
- But no reviews :-(

On the Web there are lots of reviews..

Alice needs:

- Information extraction
  Is this unstructured text referring to a movie review?
- Similarity joins
  Which movie is the review about?
- Sentiment analysis
  Is the review positive or negative?
  Should I trust the reviewer?
- Social networks
  What do my friends recommend?

A probabilistic database can help Alice store and query her *uncertain* data.

Alice Needs Information Extraction

Possible segmentations of unstructured text [Sarawagi VLDB'06]

`52-A Goregaon West Mumbai 400 076`

ID	HouseNo	Area	City	PinCode	**P**
1	52	Goregaon West	Mumbai	400 062	**0.1**
1	52-A	Goregaon	West Mumbai	400 062	**0.2**
1	52-A	Goregaon West	Mumbai	400 062	**0.4**
1	52	Goregaon	West Mumbai	400 062	**0.2**
...	...	...	...	...	...

- *Sound* confidence values obtained using probabilistic extraction models
- Output a ranked list of possible extractions
  Empty answer to query: Find movies filmed in 'West Mumbai'
- Several segmentations are required to cover most of the probability mass and improve recall

Probabilistic Databases Today

Many active projects

- Mystiq, Lahar (Washington U.)
- Trio (Stanford)
- MCDB (IBM Almaden & Florida)
- BayesStore (Berkeley)
- Orion (Purdue)
- At Maryland, UMass, Waterloo, Hong Kong, Florida State, Wisconsin, etc.

Projects I am involved in

- MayBMS (co-inventor, Cornell joint with Oxford)
  - New uncertainty-aware query language and data representation models
  - Officially released last month, see `maybms.sourceforge.net`
    To be demonstrated at SIGMOD'09!
- SPROUT = Scalable Query PROcessing on Uncertain Tables (PI, Oxford)
  - Query engine that extends PostgreSQL backend
  - Used by MayBMS, but also available standalone
  - **State-of-the-art scalable query processing techniques**

Probabilistic Databases: Syntax

Probabilistic databases are relational databases where

- Tuples are associated with *lineage*, i.e., Boolean expressions over independent random variables.
- Probability distributions over the possible assignments of each variable.

Tuple-independent database: tuples have independent lineage.

Example of a tuple-independent TPC-H database:

Cust

ckey	cname	V	P
1	Joe	x_1	0.1
2	Dan	x_2	0.2
3	Li	x_3	0.3
4	Mo	x_4	0.4

Ord

okey	ckey	odate	V	P
1	1	1995-01-10	y_1	0.1
2	1	1996-01-09	y_2	0.2
3	2	1994-11-11	y_3	0.3
4	2	1993-01-08	y_4	0.4
5	3	1995-08-15	y_5	0.5
6	3	1996-12-25	y_6	0.6

Item

okey	disc	ckey	V	P
1	0.1	1	z_1	0.1
1	0.2	1	z_2	0.2
3	0.4	2	z_3	0.3
3	0.1	2	z_4	0.4
4	0.4	2	z_5	0.5
5	0.1	3	z_6	0.6

Probabilistic Databases: Semantics

One-to-one mapping between *possible worlds* and total valuations over variables.

Consider the total valuation f: x_1, y_1, z_1 are true, all other variables are false.

Cust

ckey	cname	V	P
1	**Joe**	x_1	**0.1**
2	Dan	x_2	0.2
3	Li	x_3	0.3
4	Mo	x_4	0.4

Ord

okey	ckey	odate	V	P
1	**1**	**1995-01-10**	y_1	**0.1**
2	1	1996-01-09	y_2	0.2
3	2	1994-11-11	y_3	0.3
4	2	1993-01-08	y_4	0.4
5	3	1995-08-15	y_5	0.5
6	3	1996-12-25	y_6	0.6

Item

okey	disc	ckey	V	P
1	**0.1**	**1**	z_1	**0.1**
1	0.2	1	z_2	0.2
3	0.4	2	z_3	0.3
3	0.1	2	z_4	0.4
4	0.4	2	z_5	0.5
5	0.1	3	z_6	0.6

Probabilistic Databases: Semantics

One-to-one mapping between *possible worlds* and total valuations over variables.

Consider the total valuation f: x_1, y_1, z_1 are true, all other variables are false.

Cust

ckey	cname
1	Joe

Ord

okey	ckey	odate
1	1	1995-01-10

Item

okey	disc	ckey
1	0.1	1

What about the probability of a world?

- Probability of a world $\mathcal{A}$ is the product of the probabilities of the chosen assignments defining $\mathcal{A}$.

For the above world:

$Pr(f) = \Pi\{Pr(v) \mid v \in \{x_1, y_1, z_1\}\} \cdot \Pi\{Pr(\neg v) \mid v \in \{x_2, \ldots, x_4, y_2, \ldots, y_6, z_2, \ldots, z_6\}\}$

Query Evaluation on Probabilistic Databases

- Follows standard semantics, with the addition that
- Each answer tuple is associated with the lineage of its input tuples.

Query asking for the dates of discounted orders shipped to customer 'Joe':

$$Q(odate) \text{ :- } \mathrm{Cust}(ckey, 'Joe'), \mathrm{Ord}(okey, ckey, odate), \mathrm{Item}(okey, disc, ckey), disc > 0$$

odate	V_c	P_c	V_o	P_o	V_i	P_i	tuple probability
1995-01-10	x_1	0.1	y_1	0.1	z_1	0.1	$0.1 \cdot 0.1 \cdot 0.1$
1995-01-10	x_1	0.1	y_1	0.1	z_2	0.2	$0.1 \cdot 0.1 \cdot 0.2$

Probability of (1995-01-10) = Probability of associated lineage $x_1 y_1 z_1 + x_1 y_1 z_2$.

Probability computation for bipartite positive 2DNF formulas is #P-complete.

Challenge: Scalable probability computation for **distinct** answer tuples.

Query Evaluation using SPROUT

Cast the query evaluation problem as an OBDD construction problem.

- Given a query q and a probabilistic database D, each distinct tuple $t \in q(D)$ is associated with a DNF expression ϕ_t.
- Probability of t is probability of lineage ϕ_t.
- Compile ϕ_t into a propositional theory with efficient model counting. We use **ordered binary decision diagrams (OBDDs)**, for which probability computation can be done in one traversal.
- Probability of ϕ_t is then the probability of its OBDD.

To achieve true scalability, SPROUT employs secondary-storage techniques for OBDD construction and probability computation.

Can we leverage existing results on OBDD construction?

- Generic compilation techniques developed by the AI community construct OBDDs whose sizes are exponential in the treewidth of the lineage.

- Conjunctive queries **do** generate lineage with unbounded treewidth.
  - The product query Q :- $R(X), S(Y)$ generates lineage that has a clause for each pair of random variables of R and $S \Rightarrow$ unbounded treewidth.

We need new compilation techniques that take the query structure into account!

OBDD-based Query Evaluation

OBDDs

- Commonly used to represent compactly large Boolean expressions.
- Idea: Decompose Boolean expressions using variable elimination and avoid redundancy in the representation.
  Variable elimination by Shannon's expansion: $\phi = x \cdot \phi \mid_x + \bar{x} \cdot \phi \mid_{\bar{x}}$.
- Variable order π = order of variable eliminations;
  the same variable order on all root-to-leaf paths.
- An OBDD for ϕ is uniquely identified by the pair (ϕ, π).
- Supports linear-time probability computation.

$$\begin{aligned} Pr(\phi) &= Pr(x \cdot \phi \mid_x + \bar{x} \cdot \phi \mid_{\bar{x}}) \\ &= Pr(x \cdot \phi \mid_x) + Pr(\bar{x} \cdot \phi \mid_{\bar{x}}) \\ &= Pr(x) \cdot Pr(\phi \mid_x) + Pr(\bar{x}) \cdot Pr(\phi \mid_{\bar{x}}) \end{aligned}$$

Compilation example

R	A	B	V_r
	a_1	b_1	x_1
	a_2	b_1	x_2
	a_2	b_2	x_3
	a_3	b_3	x_4

S	A	C	V_s
	a_1	c_1	y_1
	a_1	c_2	y_2
	a_2	c_1	y_3
	a_4	c_2	y_4

q :- $R(A, B), S(A, C)$

	V_r	V_s
	x_1	y_1
	x_1	y_2
	x_2	y_3
	x_3	y_3

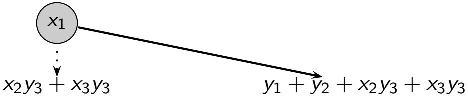
Query q has lineage $\phi = x_1 y_1 + x_1 y_2 + x_2 y_3 + x_3 y_3$.
Assume variable order: $\pi = x_1 y_1 y_2 x_2 x_3 y_3$.
Task: Construct the OBDD (ϕ, π).

Compilation example

Task: Construct OBDD (ϕ, π), where

- $\phi = x_1 y_1 + x_1 y_2 + x_2 y_3 + x_3 y_3$ and
- $\pi = x_1 y_1 y_2 x_2 x_3 y_3$.

Step 1: Eliminate variable x_1 in ϕ.

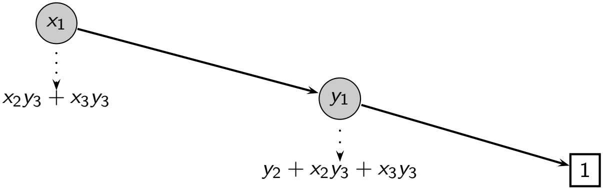


Compilation example

Task: Construct OBDD (ϕ, π), where

- $\phi = x_1 y_1 + x_1 y_2 + x_2 y_3 + x_3 y_3$ and
- $\pi = x_1 y_1 y_2 x_2 x_3 y_3$.

Step 2: Eliminate variable y_1.

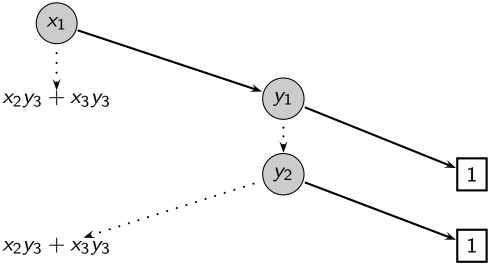


Compilation example

Task: Construct OBDD (ϕ, π), where

- $\phi = x_1 y_1 + x_1 y_2 + x_2 y_3 + x_3 y_3$ and
- $\pi = x_1 y_1 y_2 x_2 x_3 y_3$.

Step 3: Eliminate variable y_2.



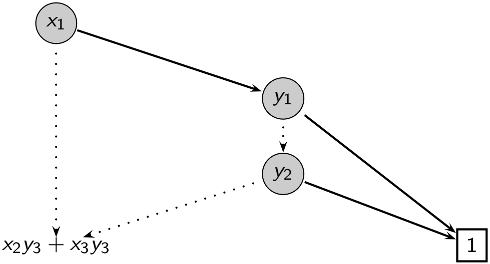
Some leaves have the same expressions $\Rightarrow$ Represent them only once!

Compilation example

Task: Construct OBDD (ϕ, π), where

- $\phi = x_1 y_1 + x_1 y_2 + x_2 y_3 + x_3 y_3$ and
- $\pi = x_1 y_1 y_2 x_2 x_3 y_3$.

Step 4: Merge leaves with the same expressions.

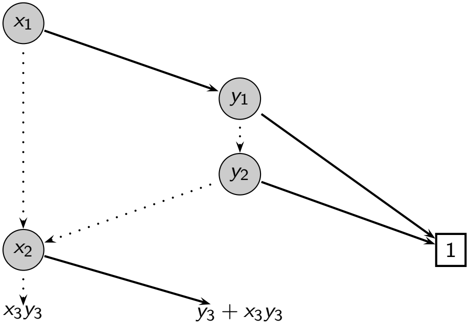


Compilation example

Task: Construct OBDD (ϕ, π), where

- $\phi = x_1 y_1 + x_1 y_2 + x_2 y_3 + x_3 y_3$ and
- $\pi = x_1 y_1 y_2 x_2 x_3 y_3$.

Step 5: Eliminate variable x_2.

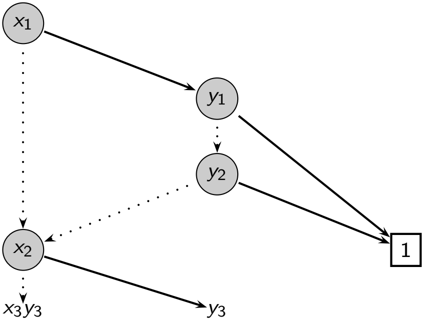


Compilation example

Task: Construct OBDD (ϕ, π), where

- $\phi = x_1 y_1 + x_1 y_2 + x_2 y_3 + x_3 y_3$ and
- $\pi = x_1 y_1 y_2 x_2 x_3 y_3$.

Step 6: Replace $y_3 + x_3 y_3$ by y_3.

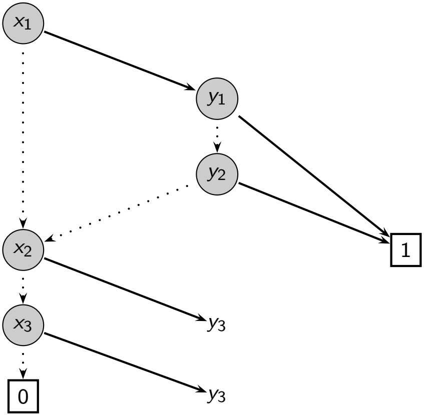


Compilation example

Task: Construct OBDD (ϕ, π), where

- $\phi = x_1 y_1 + x_1 y_2 + x_2 y_3 + x_3 y_3$ and
- $\pi = x_1 y_1 y_2 x_2 x_3 y_3$.

Step 7: Eliminate variable x_3.

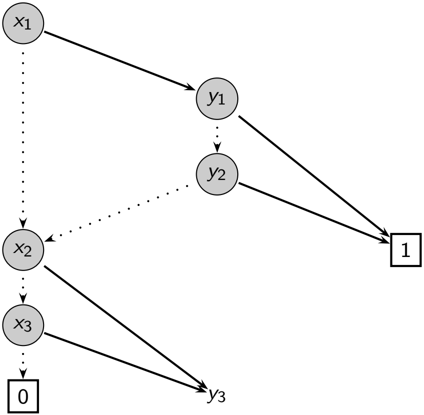


Compilation example

Task: Construct OBDD (ϕ, π), where

- $\phi = x_1 y_1 + x_1 y_2 + x_2 y_3 + x_3 y_3$ and
- $\pi = x_1 y_1 y_2 x_2 x_3 y_3$.

Step 8: Merge leaves with the same expression y_3.

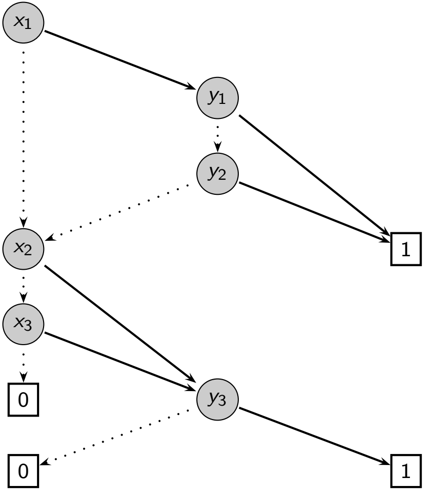


Compilation example

Task: Construct OBDD (ϕ, π), where

- $\phi = x_1 y_1 + x_1 y_2 + x_2 y_3 + x_3 y_3$ and
- $\pi = x_1 y_1 y_2 x_2 x_3 y_3$.

Step 9: Eliminate variable y_3.

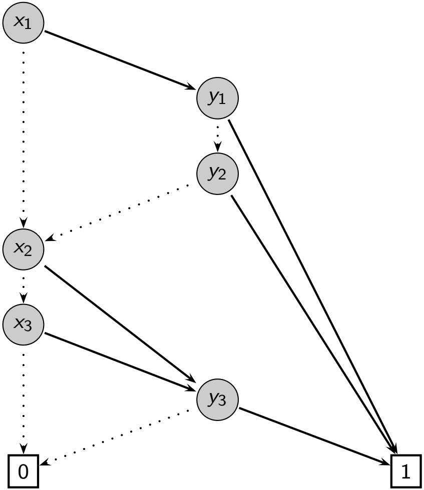


Compilation example

Task: Construct OBDD (ϕ, π), where

- $\phi = x_1 y_1 + x_1 y_2 + x_2 y_3 + x_3 y_3$ and
- $\pi = x_1 y_1 y_2 x_2 x_3 y_3$.

Step 10 (final): Merge leaves with the same expression (0 or 1).



Compilation example: Summing Up

OBDD (ϕ, π)

- has *exactly one node per variable* in ϕ,
- although the size of ϕ can be exponential in the arity of its clauses.

Questions

1. Is this property shared by the OBDDs of many queries?
2. Can we directly and efficiently construct such succinct OBDDs?
3. Can we efficiently find such *good* variable orders?

Compilation example: Summing Up

OBDD (ϕ, π)

- has *exactly one node per variable* in ϕ,
- although the size of ϕ can be exponential in the arity of its clauses.

Questions

1. Is this property shared by the OBDDs of many queries?
2. Can we directly and efficiently construct such succinct OBDDs?
3. Can we efficiently find such *good* variable orders?

The answer is in the affirmative for all of the three questions!

Tractable Queries and Succinct OBDDs

Class of tractable queries TQ on probabilistic structures (wrt data complexity):

- *all* hierarchical queries, i.e., tractable conjunctive queries without self-joins.
- natural classes of conjunctive queries with inequalities.

Theorem: For any TQ query q and database D, $\forall t \in q(D)$, and lineage ϕ_t,

- There is a variable order π computable in time $O(|\phi_t| \cdot \log^2 |\phi_t|)$ such that
- The OBDD (ϕ_t, π) has size and can be computed in time $O(f(|q|) \cdot |Vars(\phi_t)|)$, where $f(\cdot)$ is a function of the query size only.

Classes of such good variable orders can be *statically* derived from queries!

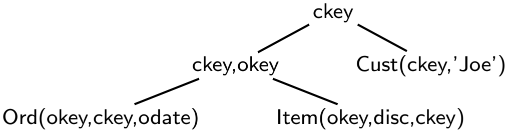
Static Query Analysis

Hierarchical Queries

A query is *hierarchical* if for any two non-head variables, either their sets of subgoals are disjoint, or one set is contained in the other.

$Q(odate)$:- $\text{Cust}(ckey, 'Joe'), \text{Ord}(okey, ckey, odate), \text{Item}(okey, disc, ckey), disc > 0.$
is hierarchical; also without odate as head variable.

subgoals(disc)=$\{\text{Item}\}$, subgoals(okey)=$\{\text{Ord}, \text{Item}\}$, subgoals(ckey)=$\{\text{Cust}, \text{Ord}, \text{Item}\}$.
It holds that subgoals(disc)$\subseteq$ subgoals(okey)$\subseteq$ subgoals(ckey).



Tractability beyond Hierarchical Queries

Tractable queries with inequalities

- At most one query variable v per subgoal can occur in join conditions,
- Variable v may be a head variable of a hierarchical query.
- For $\neq$-joins only: the inequality graph is a tree.

Examples of tractable queries:

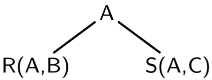
- $Q_1 \,{:}{-}\, R(A, B), S(C), T(D, E), A < C < E.$
- $Q_2 \,{:}{-}\, R(A, B), S(C), T(D, E), A < C, A < E.$
- $Q_3 \,{:}{-}\, R(A, B), S(C), T(D, E), A < C, A < E, C < E.$
- $Q_4 \,{:}{-}\, R(A, B), S(C), T(D, E), A \neq C, A \neq D.$

Results published in SUM'08 and SIGMOD'09.

Query Signatures

Query signatures for TQ queries capture

- the structures of queries and
- the one/many-to-one/many relationships between the query tables;
- variable orders for succinct OBDDs representing compiled lineage!



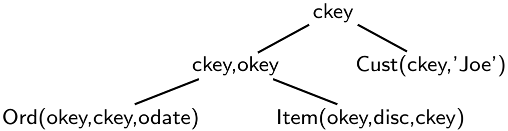
Query q :- $R(A,B), S(A,C)$ has signature $(R^*S^*)^*$.

- There may be several R-tuples with the same A-value, hence R^*
- There may be several S-tuples with the same A-value, hence S^*
- R and S join on A, hence R^*S^*
- There may be several A-values in R and S, hence $(R^*S^*)^*$

Variable orders captured by $(R^*S^*)^*$ (x_i's are from R, y_j's are from S):
$\{[x_1(y_1y_2)][(x_2x_3)y_3]\}$, $\{[(x_2x_3)y_3][x_1(y_1y_2)]\}$, $\{[y_3(x_3x_2)][x_1(y_2y_1)]\}$, etc.

Deriving Better Query Signatures

Q :- $\text{Cust}(ckey, 'Joe'), \text{Ord}(okey, ckey, odate), \text{Item}(okey, disc, ckey), disc > 0$



Query Q has signature $(\text{Cust}^*(\text{Ord}^*\text{Item}^*)^*)^*$.

Database constraints can make the signature more precise

- If ckey is key in Cust, we obtain the signature $(\text{Cust}(\text{Ord}^*\text{Item}^*)^*)^*$.
  The many-to-many relationship between Cust and Ord is now one-to-many
- If in addition okey is key in Ord, we obtain the signature $(\text{Cust}(\text{Ord Item}^*)^*)^*$.

Secondary-storage Query Evaluation

Secondary-storage Query Evaluation

Query evaluation in two logically-independent steps

1. Compute query answer using a good *relational* query plan of your choice
2. Compute probabilities of each distinct answer (or temporary) tuple

Probability computation supported by a new aggregation operator that can

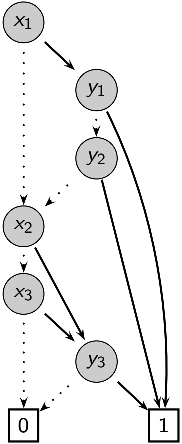
- blend itself in any relational query plan
- be placed on top of the query plan, or *partially* pushed down past joins
- compute in parallel different fragments of the OBDD for the lineage *without* materializing the OBDD.

Our aggregation operator is a sequence of

- **aggregation** steps. Effect on query signature: $\alpha^* \rightarrow \alpha$
- **propagation** steps. Effect on query signature: $\alpha\beta \rightarrow \alpha$

Results published in ICDE'09 and SIGMOD'09.

Example of Probability Computation



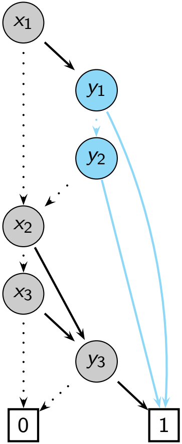
q :- $R(A, B), S(A, C)$

	V_r	V_s
	x_1	y_1
	x_1	y_2
	x_2	y_3
	x_3	y_3

How to proceed?

1. **Sort query answer by** (V_r, V_s)**.**
   Initial signature: $(R^*S^*)^*$
2. Apply aggregation step $S^* \rightarrow S$.
   New signature: $(R^*S)^*$
3. Apply aggregation step $R^* \rightarrow R$.
   New signature: $(RS)^*$
4. Apply propagation step $RS \rightarrow R$.
   New signature: R^*
5. Apply aggregation step $R^* \rightarrow R$.
   New signature: R

Example of Probability Computation



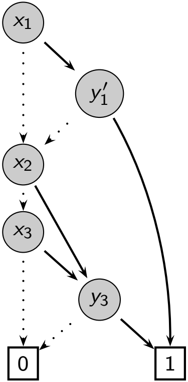
q :- $R(A, B), S(A, C)$

	V_r	V_s
	x_1	$y_1 + y_2$
	x_2	y_3
	x_3	y_3

How to proceed?

1. Sort query answer by (V_r, V_s).
   Initial signature: $(R^* S^*)^*$
2. **Apply aggregation step $S^* \longrightarrow S$.**
   New signature: $(R^* S)^*$
3. Apply aggregation step $R^* \longrightarrow R$.
   New signature: $(RS)^*$
4. Apply propagation step $RS \longrightarrow R$.
   New signature: R^*
5. Apply aggregation step $R^* \longrightarrow R$.
   New signature: R

Example of Probability Computation



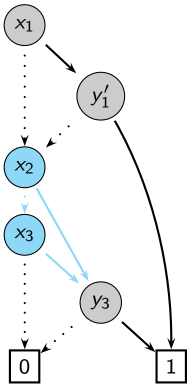
q :- $R(A, B), S(A, C)$

	V_r	V_s
	x_1	y'_1
	x_2	y_3
	x_3	y_3

How to proceed?

1. Sort query answer by (V_r, V_s).
   Initial signature: $(R^* S^*)^*$
2. **Apply aggregation step $S^* \longrightarrow S$.**
   New signature: $(R^* S)^*$
3. Apply aggregation step $R^* \longrightarrow R$.
   New signature: $(RS)^*$
4. Apply propagation step $RS \longrightarrow R$.
   New signature: R^*
5. Apply aggregation step $R^* \longrightarrow R$.
   New signature: R

Example of Probability Computation



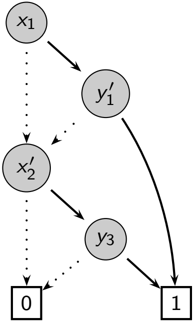
q :- $R(A, B), S(A, C)$

	V_r	V_s
	x_1	y'_1
	$x_2 + x_3$	y_3

How to proceed?

1. Sort query answer by (V_r, V_s).
   Initial signature: $(R^* S^*)^*$
2. Apply aggregation step $S^* \rightarrow S$.
   New signature: $(R^* S)^*$
3. **Apply aggregation step $R^* \rightarrow R$.**
   New signature: $(RS)^*$
4. Apply propagation step $RS \rightarrow R$.
   New signature: R^*
5. Apply aggregation step $R^* \rightarrow R$.
   New signature: R

Example of Probability Computation



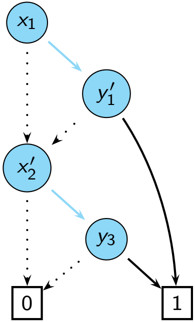
q :- $R(A, B), S(A, C)$

	V_r	V_s
	x_1	y'_1
	x'_2	y_3

How to proceed?

1. Sort query answer by (V_r, V_s).
   Initial signature: $(R^* S^*)^*$
2. Apply aggregation step $S^* \longrightarrow S$.
   New signature: $(R^* S)^*$
3. **Apply aggregation step $R^* \longrightarrow R$.**
   New signature: $(RS)^*$
4. Apply propagation step $RS \longrightarrow R$.
   New signature: R^*
5. Apply aggregation step $R^* \longrightarrow R$.
   New signature: R

Example of Probability Computation



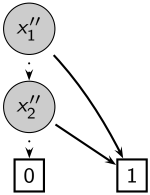
q :- $R(A, B), S(A, C)$

	V_r
	$x_1 y_1'$
	$x_2' y_3$

How to proceed?

1. Sort query answer by (V_r, V_s).
   Initial signature: $(R^* S^*)^*$
2. Apply aggregation step $S^* \longrightarrow S$.
   New signature: $(R^* S)^*$
3. Apply aggregation step $R^* \longrightarrow R$.
   New signature: $(RS)^*$
4. **Apply propagation step $RS \longrightarrow R$.**
   New signature: R^*
5. Apply aggregation step $R^* \longrightarrow R$.
   New signature: R

Example of Probability Computation



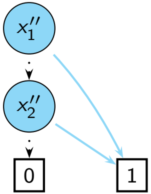
q :- $R(A, B), S(A, C)$

	V_r
	x_1''
	x_2''

How to proceed?

1. Sort query answer by (V_r, V_s).
   Initial signature: $(R^* S^*)^*$
2. Apply aggregation step $S^* \longrightarrow S$.
   New signature: $(R^* S)^*$
3. Apply aggregation step $R^* \longrightarrow R$.
   New signature: $(RS)^*$
4. **Apply propagation step $RS \longrightarrow R$.**
   New signature: R^*
5. Apply aggregation step $R^* \longrightarrow R$.
   New signature: R

Example of Probability Computation



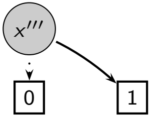
q :- $R(A, B), S(A, C)$

	V_r
	$x_1'' + x_2''$

How to proceed?

1. Sort query answer by (V_r, V_s).
   Initial signature: $(R^* S^*)^*$
2. Apply aggregation step $S^* \longrightarrow S$.
   New signature: $(R^* S)^*$
3. Apply aggregation step $R^* \longrightarrow R$.
   New signature: $(RS)^*$
4. Apply propagation step $RS \longrightarrow R$.
   New signature: R^*
5. **Apply aggregation step $R^* \longrightarrow R$.**
   New signature: R

Example of Probability Computation



	q :- $R(A, B), S(A, C)$
	V_r
	x'''

Return the probability of x'''.

How to proceed?

1. Sort query answer by (V_r, V_s).
   Initial signature: $(R^* S^*)^*$
2. Apply aggregation step $S^* \longrightarrow S$.
   New signature: $(R^* S)^*$
3. Apply aggregation step $R^* \longrightarrow R$.
   New signature: $(RS)^*$
4. Apply propagation step $RS \longrightarrow R$.
   New signature: R^*
5. **Apply aggregation step $R^* \longrightarrow R$.**
   New signature: R

Grouping Aggregations and Propagations

Groups of aggregations/propagations can be computed in **one scan**.

Definition: A signature has the *1scan* property if each of its composite expressions is made up by concatenating signatures with the 1scan property and at least one table without (*).

Examples of 1scan signatures:

- $(RS^*)^*$ (last 3 steps in the previous example)
- R^*S^* (relational product)
- $\text{Nation}_1\text{Supp}(\text{Nation}_2(\text{Cust}(\text{Ord Item}^*)^*)^*)^*$ (conj. part of TPC-H query 7)

For signature α: #scans(α) = one plus the number of its starred (*) subexpressions, including itself, without the 1scan property.

Proposition: An operator with signature α needs #scans(α) scans.

Examples:

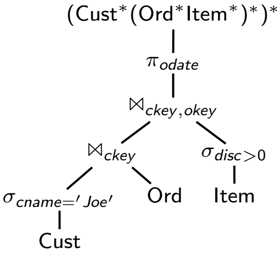
- $\#\text{scans}((R^*S^*)^*) = 2$
- $\#\text{scans}((\text{Cust}^*(\text{Ord}^*\text{Item}^*)^*)^*) = 3$, BUT $\#\text{scans}((\text{Cust}(\text{Ord Item}^*)^*)^*) = 1$

Query Optimization

Types of Query Plans

Our previous examples considered *lazy* plans

- probability computation done *after* the computation of answer tuples
- unrestricted search space for good query plans
- especially desirable when join conditions are selective (eg, TPC-H)!

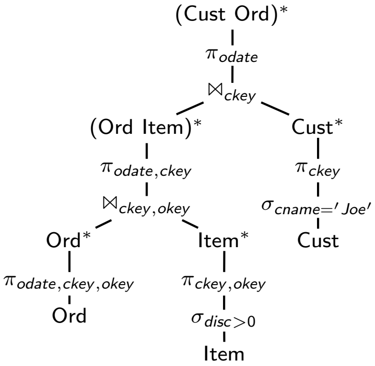


BUT, we can push down probability computation!

Proposition: Any subquery of a hierarchical query is hierarchical.

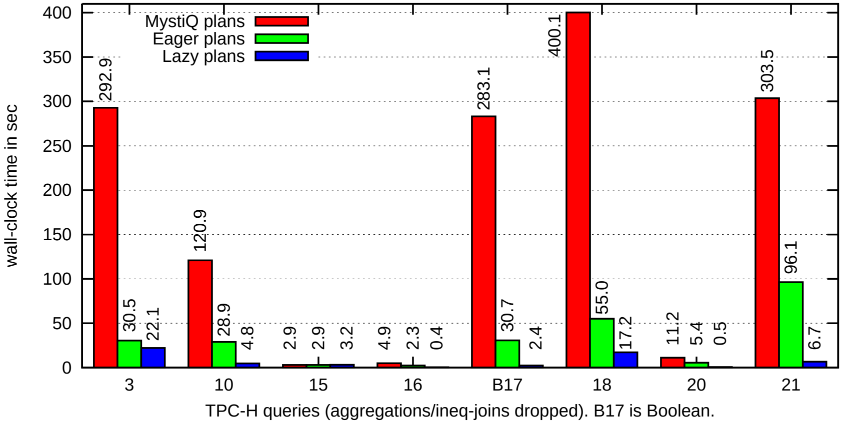
Types of Query Plans

Eager plans discard duplicates and compute probabilities on each temporary table.



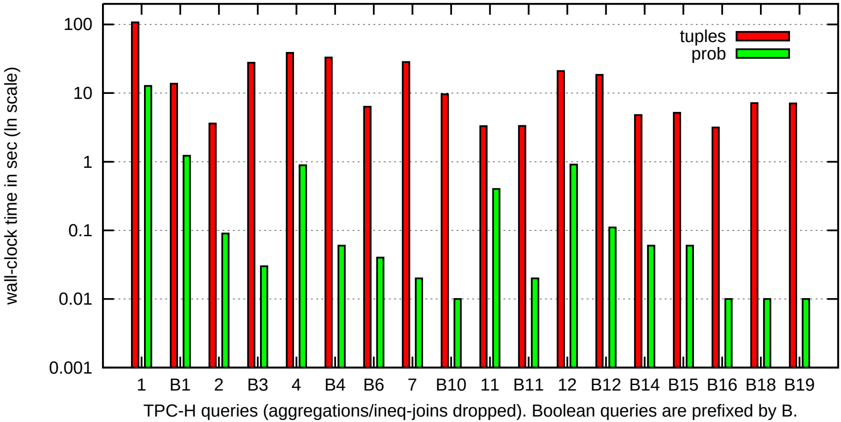
Experiments

Experiments: SPROUT vs. MystiQ



SPROUT query engine extends PosgreSQL backend. MystiQ is a middleware. TPC-H conj. queries accepted by MystiQ on 1GB tuple-independent TPC-H.

Experiments: Probability Computation with SPROUT



Computing the answer tuples vs duplicate removal and probability computation.
TPC-H conj. queries on 1GB tuple-independent TPC-H.

Thanks!

Why are Non-hierarchical Queries Hard?

Key ingredients:

- The query pattern $R(\ldots, X, \ldots), S(\ldots, X, \ldots, Y, \ldots), T(\ldots, Y, \ldots)$ can produce *any* bipartite positive 2DNF lineage ϕ, given suitable R, S, and T.
- #SAT for bipartite positive 2DNF formulas is #P-complete.

Proof idea:

- Find tuple-independent tables R and T and a certain table S such that the query answer is associated with lineage ϕ.
- S has precisely one tuple pairing the variables in each clause. of ϕ.

Example

- Bipartite positive 2DNF $\phi = x_1y_1 + x_1y_2 + x_2y_1 + x_2y_3 + x_3y_2$
- Boolean query Q :- $R(X), S(X, Y), T(Y)$ on the database given below.

R	A	V_r
	1	x_1
	2	x_2
	3	x_3

S	B	C
	1	1
	1	2
	2	1
	2	3
	3	2

T	D	V_t
	1	y_1
	2	y_2
	3	y_3

Q		V_r	V_t
		x_1	y_1
		x_1	y_2
		x_2	y_1
		x_2	y_3
		x_3	y_2

Query Rewriting under Functional Dependencies (FDs)

FDs on tuple-independent databases can help deriving better query signatures.

Definition: Given a set of FDs Σ and a conjunctive query of the form

$$Q = \pi_{\overline{A_0}}(\sigma_\phi(R_1(\overline{A_1}) \bowtie \ldots \bowtie R_n(\overline{A_n}))$$

where ϕ is a conjunction of unary predicates. Let $\Sigma_0 = CLOSURE_\Sigma(\overline{A_0})$.
Then, the Boolean query

$$\pi_\emptyset(\sigma_\phi(R_1(CLOSURE_\Sigma(\overline{A_1}) - \Sigma_0) \bowtie \ldots \bowtie R_n(CLOSURE_\Sigma(\overline{A_n}) - \Sigma_0)))$$

is called the **FD-reduct** of Q under Σ.

Proposition: If there is a sequence of chase steps under Σ that turns Q into a hierarchical query, then the fixpoint of the chase (the FD-reduct) is hierarchical.

Importance of FD-reducts

The signature of Q's FD-reduct captures the structure of Q's lineage.

Two relevant cases

1. Intractable queries may admit tractable FD-reducts.

   Under $X \rightarrow Y$, the hard query Q :- $R(X), S(X, Y), T(Y)$ admits the hierarchical FD-reduct Q' :- $R(X, Y), S(X, Y), T(Y)$ with signature $((RS)^* T)^*$.

2. FD-reducts have more precise query signatures.

   In the presence of keys ckey and okey, the query $Q(odate)$:- $\text{Cust}(ckey, cname), \text{Ord}(okey, ckey, odate), \text{Item}(okey, disc, ckey)$ with signature $(\text{Cust}^*(\text{Ord}^*\text{Item}^*)^*)^*$ rewrites into

   Q' :- $\text{Cust}(ckey, cname), \text{Ord}(okey, ckey, cname), \text{Item}(okey, disc, ckey, cname)$ with signature $(\text{Cust}(\text{Ord Item}^*)^*)^*$.

Case Study: TPC-H Queries

Considered the conjunctive part of each of the 22 TPC-H queries

- Boolean versions (B)
- with original selection attributes, but without aggregates (O)

Hierarchical in the absence of key constraints

- 8 queries (B)
- 13 queries (O)

Hierarchical in the presence of key constraints

- 8+4 queries (B)
- 13+4 queries (O)

In-depth study at

`http://www.comlab.ox.ac.uk/people/dan.olteanu/papers/icde09queries.html`

Grouping Aggregations and Propagations

Groups of aggregations and propagations can be computed in **one sequential scan**.

Definition: A signature has the *1scan* property if each of its composite expressions is made up by concatenating signatures with the 1scan property and at least one table without (*).

Examples of 1scan signatures:

- $(RS^*)^*$ (last 3 steps in the previous example)
- R^*S^* (relational product)
- $\text{Nation}_1\text{Supp}(\text{Nation}_2(\text{Cust}(\text{Ord Item}^*)^*)^*)^*$ (conj. part of TPC-H query 7)

For signature α: $\#\text{scans}(\alpha)$ = one plus the number of its starred (*) subexpressions, including itself, without the 1scan property.

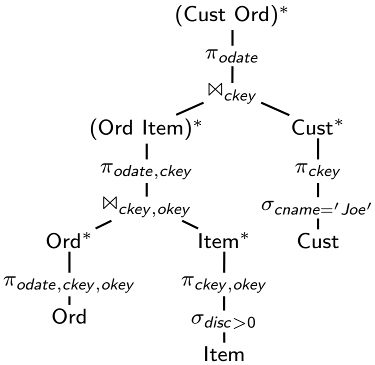
Proposition: An operator with signature α needs $\#\text{scans}(\alpha)$ scans.

Examples:

- $\#\text{scans}((R^*S^*)^*) = 2$
- $\#\text{scans}((\text{Cust}^*(\text{Ord}^*\text{Item}^*)^*)^*) = 3$, BUT $\#\text{scans}((\text{Cust}(\text{Ord Item}^*)^*)^*) = 1$

Types of Query Plans

Eager plans discard duplicates and compute probabilities on each temporary table.



MystiQ's safe plans are special cases of eager plans!

- mirror the hierarchical structure of the query signature
- probability computation restricts join ordering!
- suboptimal join ordering, which is more costly than probability computation

Types of Query Plans

Hybrid plans

- are useful when selectivities of different joins differ significantly
- push down probability computation below unselective joins
- keep probability computation on top of selective joins