Markov Chain Monte Carlo and Databases

christoph.koch@epfl.ch

Overview of this talk

- Introduction to MCMC
- Probabilistic databases
- MCMC databases
- Incremental view maintenance for MCMC databases
- Query languages for transition kernels
- Further remarks, related work, and conclusions

Markov Chain Monte Carlo (MCMC)

- A hot tool for many areas of scientific computing.
- Interesting motivation from and analogies with statistical physics
  - Heat bath algorithm/Gibbs sampling
  - Ising model
  - Simulated annealing
- Applications: many, e.g.
  - Integration – physics, engineering, …
  - Bayesian inference: Biology, …, AI (graphical models)
  - algorithm theory (!) – different intuition, requires provable bounds on mixing time, but the models and distributions may be changed.
- We will discuss the Bayesian scenario, but MCMC is not restricted to it.

Bayesian Inference

- Posterior distribution:

$$P(\theta \mid D) = \frac{P(\theta)P(D \mid \theta)}{\int P(\theta)P(D \mid \theta)d\theta} \qquad \text{(data } D\text{, parameters } \theta\text{)}$$

- Posterior expectation of f:

$$E[f(\theta) \mid D] = \frac{\int f(\theta)P(\theta)P(D \mid \theta)d\theta}{\int P(\theta)P(D \mid \theta)d\theta}$$

(features of $P(\theta \mid D)$ – moments, quantiles, ...)

- Our notation:

$$E[f(X)] = \frac{\int f(X)\pi(X)dX}{\int \pi(X)dX} \qquad (\pi \text{ posterior, } X \text{ random vars})$$

- $\int \pi(X)dX$ may be unknown or hard to compute

Monte Carlo integration

- Samples {Xt | t= 1, …, n} from π

$$E[f(X)] \approx \frac{1}{n}\sum_{t=1}^{n} f(X_t)$$

- Approximation of population mean by sample mean
- Sample size *n* can be chosen freely
- Problem: distributions are often nonstandard – impossible to create independent samples from, or very inefficient (rejection of lots of samples)

Markov Chain Monte Carlo (MCMC)

- Monte Carlo where samples are drawn dependently: through a Markov Chain that has π as its stationary distribution.
- Choose sequence of samples $\{X_0, X_1, \ldots, X_t, X_{t+1}\}$ such that X_{t+1} only depends on X_t.
- Suitable transition kernel $P(X_{t+1}|X_t)$
- (Ergodic) Markov chain gradually forgets X_0: $P^{(t)}(\cdot \mid X_0)$ converges to stationary distribution $\pi(\cdot)$.
- After a long enough burn-in, the samples are dependent samples from $\pi(.)$.
- Ergodic average: $\bar{f} = \frac{1}{n-m} \sum_{t=m+1}^{n} f(X_t)$

Metropolis-Hastings Algorithm

Initialize X_0; $t := 0$;
Forever {
  $Y :=$ sample from $q(\cdot \mid X_t)$;
  $X_{t+1} := \begin{cases} Y & \ldots \text{ with probability } \alpha(X_t, Y) \\ X_t & \ldots \text{ otherwise} \end{cases}$
  $t \mathrel{+}= 1$
}

- $\alpha(X_t, Y) = \min\left(1, \frac{\pi(Y)q(X_t|Y)}{\pi(X_t)q(Y|X_t)}\right)$
- The *proposal distribution* $q(\cdot \mid X_t)$ can be chosen arbitrarily (e.g., Gaussian with mean X_t and fixed σ), but not all will mix equally well.
- If X_t is from π, then X_{t+1} is also.

+ Convergence to pi from arbitrary X0.

Single-component Metropolis-Hastings

- Divide X into blocks of random vars and update them individually.
- An iteration of the single-comp MH algorithm updates each of the components.
  - Update order does not matter (random order of components).
  - Can even omit some randomly, or according to a distribution.
  - Possible to have that distribution depend on Xt, but alpha needs to be adapted.

$$X = \{X_{.1}, \ldots, X_{.k}\}; X_{.-i} = \{X_{.1}, \ldots, X_{.i-1}, X_{.i+1}, \ldots, X_{.k}\}$$

$$\alpha(X_{.-i}, X_{.i}, Y_{.i}) = \min(1, \frac{\pi(Y_{.i} \mid X_{.-i}) q_i(X_{.i} \mid Y_{.i}, X_{.-i})}{\pi(X_{.i} \mid X_{.-i}) q_i(Y_{.i} \mid X_{.i}, X_{.-i})})$$

- Gibbs sampler: $q_i(Y_{.i} \mid X_{.i}, X_{.-i}) = \pi(Y_{.i} \mid X_{.i})$

Markov Chain Monte Carlo (MCMC)

- Starting values and proposal distribution
  - In principle don't matter, but bad choices can make convergence slow.
  - Hardly any techniques for choosing smartly available. Over-dispersed starting values in multiple chains [Gelman and Rubin, 1992]
- Determining burn-in and stopping time
  - Single-chain/multi-chain convergence diagnostics [Gelman and Rubin, 1992; Raftery and Lewis, 1992; Cowles and Carlin, 1994].
  - Convergence depends on monitoring fn – misleading.
  - Running several chains in parallel [Roberts, 1992; Raftery and Lewis, 1995.]
  - In practice people observe the run/experiment with multiple runs and decide when to stop.

Applications of MCMC DBMS

- In principle, all applications in which MCMC is used.
- Graphical models for information extraction, data cleaning, and data integration.
  - MCMC for graphical models is well understood.
  - An MCMC DBMS prototype [Wick, McCallum, Miklau, VLDB 2010]
- Query processing in prob. DBs with uncertain, continuously distributed values
  - Needed even if we start with strong independence assumptions.
  - Orion [Prabhakar], PIP [Kennedy and K.]

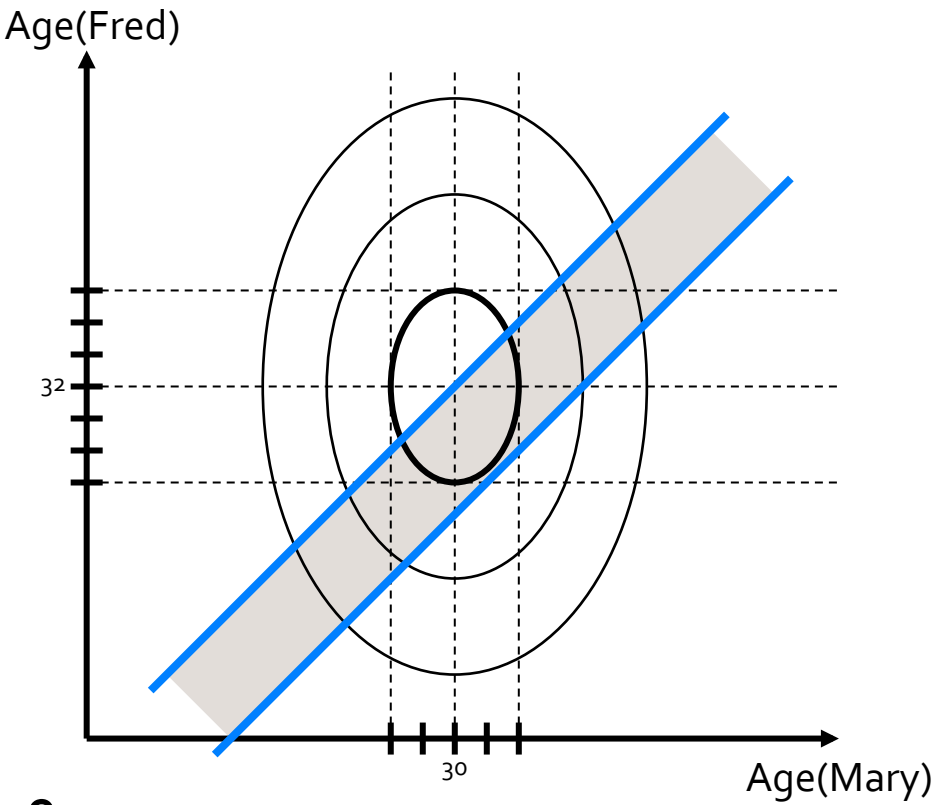
Probabilistic DB Example

Women

Name	Age
Mary	X ~ N(30,2)
Sue	Y ~ N(25,5)

Men

Name	Age
Fred	X ~ N(32,3)
Bob	Y ~ N(30,5)




```
select w.name, m.name
from Women w, Men m
where abs(w.age - m.age) <= 2
```

Orion [Singh, Mayfield, Mittal, Prabhakar, Hambrusch, Shah, SIGMOD 2008],
PIP [Kennedy and K., ICDE 2010]

Advantages of MCMC

- Very general: Applicable when no other technique is known to work
  - For domains that are not yet well studied
- Powerful, elegant abstraction
  - Markov Chain often easy to construct.
  - -> General-purpose probabilistic DBMS
- But: in various well-studied contexts, more efficient techniques are available
  - loopy belief propagation, variational approaches, Karp-Luby, NNF, OBDDs, …

The Probabilistic DBMS Design Space

- Discrete vs. continuous distributions
- Uncertainty modeling:
  - Uncertain fields/tuples: ProbView [Maryland], Orion [Purdue], MystiQ [Washington]
  - Initially certain data; uncertain rules/queries [Dortmund]
  - C-tables: Trio [Stanford], MayBMS/PIP [Cornell/EPFL]
    - = Uncertain fields/tuples + views.
  - Graphical models: PrDB [Maryland], BayesStore [Berkeley]
  - VG-functions: MCDB [Florida/Rice], MayBMS/PIP [Cornell]
    - Sampling

The Probabilistic DBMS Design Space

- Query languages: Representation-system in/dependent
- Query evaluation: Sampling vs. transformation of representations
- Early sampling (MCDB) vs. late sampling (MystiQ, Orion?, MayBMS/PIP)
  - Bottom-up (MCDB, tuple bundles) vs. top-down (MayBMS/PIP) optimization, avoidance of redundant computations.
  - Top-down allows for stronger optimization than bottom-up (PIP [Kennedy and K., ICDE 2010]).
    - Declaration of regularities rather than inference.

Probabilistic c-tables

- Conditional tables [Imielinski, Lipski]
- Relational tables with variables (labeled nulls) in which each tuple has a local (Boolean) condition.
- Possible worlds semantics: world given by variable assignment; fill in variables, drop tuples that do not satisfy condition.

Name	Salary	Phi
Bob	X~N(50k, 10k)	Y=0
Joe	65k	Y=1 and Z=2

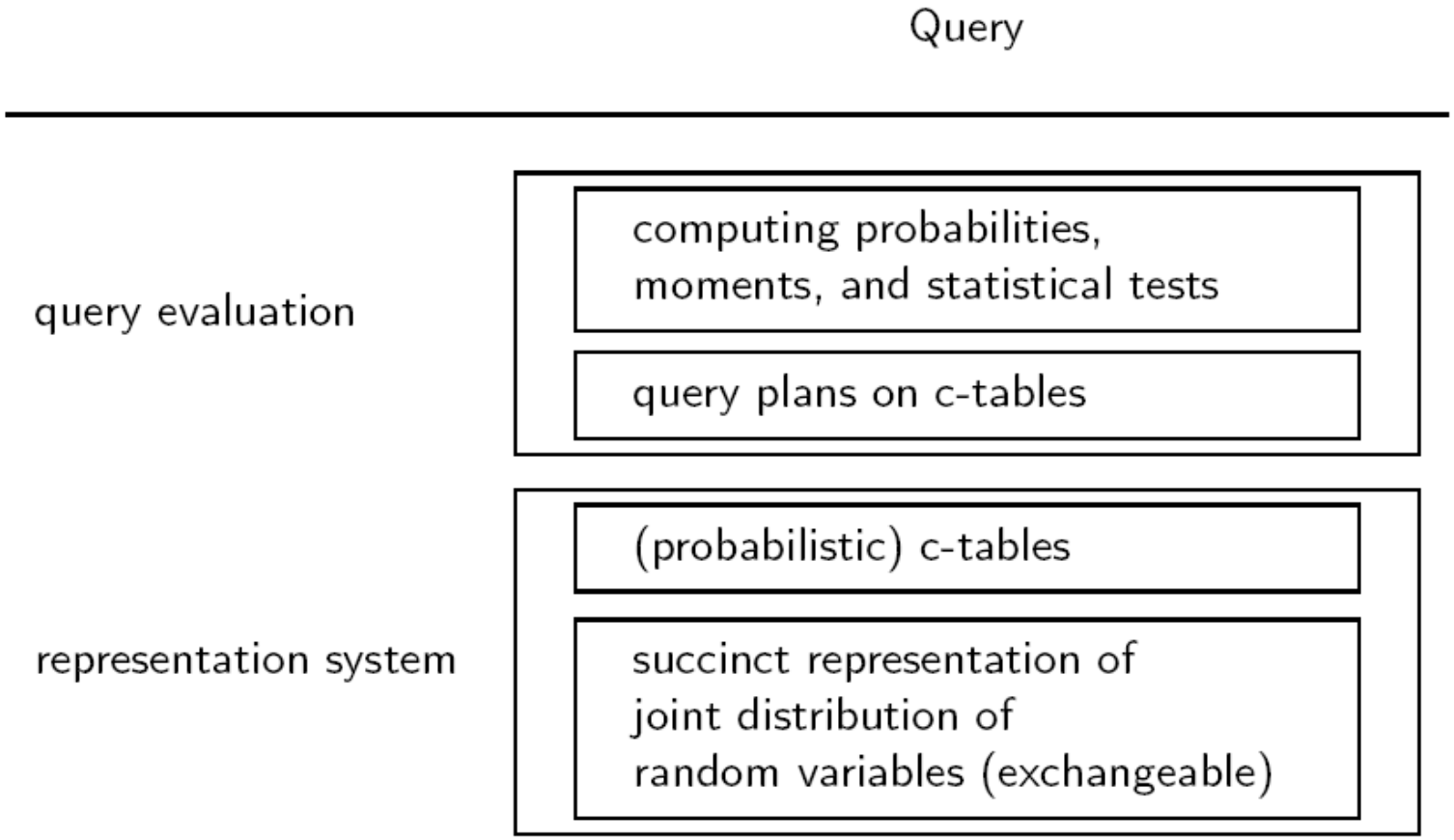
- Evaluation of relational algebra on such tables easy.

$$
\begin{aligned}
[\![R \times S]\!] &= \{\langle r, s, \phi \wedge \psi\rangle \mid \langle r, \phi\rangle \in R, \langle s, \psi\rangle \in S\} \\
[\![\sigma_\psi(R)]\!] &= \{\langle r, \phi \wedge \psi\rangle \mid \langle r, \phi\rangle \in R\} \\
[\![R - R']\!] &= \{\langle r, \phi \wedge \neg\psi\rangle \mid \langle r, \phi\rangle \in R, \langle r, \psi\rangle \in R'\} \\
\pi, \cup &\quad \ldots \quad \text{similar.}
\end{aligned}
$$

(Difference operation – here, simplifying assumption that tuples do not contain variables.)

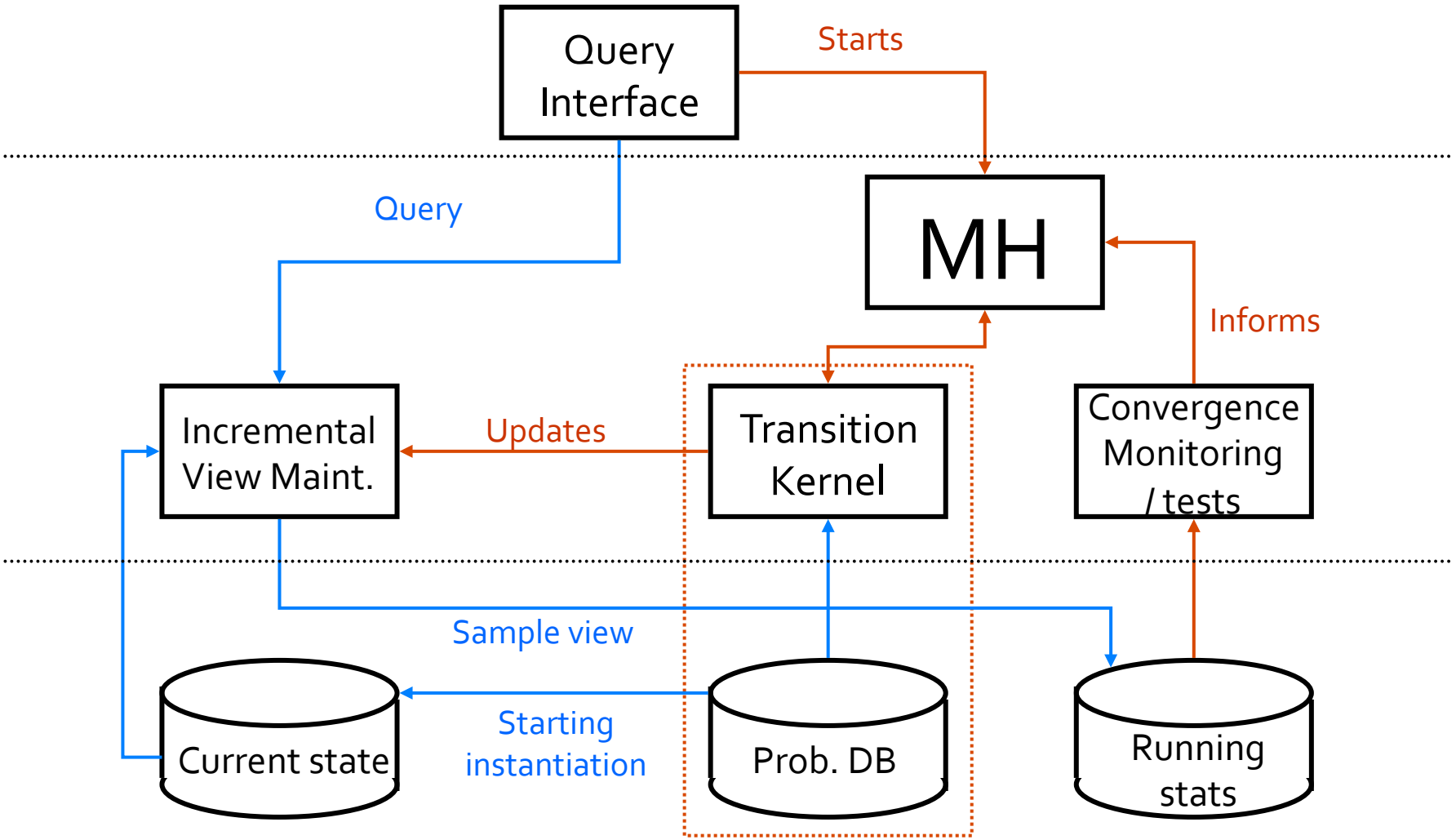
- Probabilistic c-tables: variables are random variables with some joint distribution.
- Finite case: independent random variables no loss of generality!

MayBMS Query Engine Architecture



[Antova, K., Olteanu, 2007, Antova, Jansen, K., Olteanu, 2007 2008; Kennedy and K., 2010]

Architecture of an MCMC DBMS



MCMC DBMS Remarks #1

- If the probabilistic DB makes initial independence assumptions (uncertain independent values), single-component MH is particularly natural.
  - Defining the transition kernel is easy, by component.
  - Dependencies constructed through views are no problem.
- If not, how to specify the transition kernel?
  - Outside of standard domains such as graphical models/Markov Random Fields?
  - -> A "query language" for defining transition kernels: MORE IN THIS TALK.

MCMC DBMS Remarks #2

Runtime efficiency:

- Collect as samples as fast as possible.
  - Performance bottleneck: evaluating the query on each sample/ MCMC state [Wick et al, 2010].
  - Solution: Incremental view maintenance. → MORE IN THIS TALK.
- Parallelization – use multiple cores.
  - A good reason for running multiple chains.
  - Running multiple chains to avoid being wrong about convergence is not accepted to be a good idea. (Ongoing dispute.)

MCMC DBMS Design Space

- What is data? Prob. DB. / graphical model. What else?
- What is query / declaratively specified?
  - Query: A function, to be computed over each sample [MCDB; Wick et. al, 2001].
  - The transition kernel [Deutch, K., Milo, 2010].
- Transparent vs. visible sampling process
  - Sampling hidden in the DBMS would be desirable/easier to use.
    - In general we don't have the tests to automatize MCMC completely.
  - For important special cases, encapsulation is feasible.
    - e.g., independent uncertain values plus views; certain classes of graphical models.
- Algorithms: MH refinements, statistical tests.

Incremental View Maintenance

- Given a DB update, do not recompute the view from scratch.
- Perform only the work needed to update the view.
- Compute delta query:
  - Delta queries work on less data; also slightly simpler.
  - But: we still need a classical query engine for processing the delta queries.

[Roussopoulos, 1991; Yan and Larson, 1995; Colby et al, 1996; Kotidis and Roussopoulos, 2001; Zhou et al, 2007]

```
CREATE MATERIALIZED VIEW empdep
REFRESH FAST ON COMMIT AS
SELECT empno, ename, dname
FROM emp e, dept d
WHERE e.deptno = d.deptno;
```

(Example in Oracle)

Recursive incremental processing

Given a function f, let

$$\Delta f(x) := f(x+1) - f(x) .$$

On increment $x \mathrel{+}= 1$: $f(x) \mathrel{+}= \Delta f(x)$.

If f is a polynomial, then $\deg(\Delta f(x)) = \max(0, \deg(f(x)) - 1)$.
So there is a k such that $\Delta^k f = 0$.

x	$g(x) = 3x^2$	$\Delta g(x) = 6x + 3$	$\Delta^2 g(x) = 6$	$\Delta^3 g(x) = 0$
0	0	3	6	0
1	3	9	6	0
2	12	15	6	0
3	27	21	6	0
4	48	27	6	0

Compiling incremental view maintenance

Aggressive recursive incremental view maintenance: maintain $Q, \Delta Q, \Delta^2 Q, \Delta^3 Q, \ldots$.

Compile(query Q):
To incrementally maintain a materialized view of Q,

1. Compute ΔQ for tuple insertion/deletion.
2. The incremental view maintenance code is $Q \pm= \Delta Q$.
3. Recursively compile ΔQ.

Requirements on the query language L:

- L must be *closed under taking deltas*: if $Q \in L$, then $\Delta Q \in L$.
- For all $Q \in L$, there is a k such that $\Delta^k Q = 0$.

Compilation, Example 1

```
q[] = select sum(LI.P * O.XCH)
      from Order O, LineItem LI
      where O.OK = LI.OK;
```

Compilation, Example 1

```
q[] = select sum(LI.P * O.XCH)
      from Order O, LineItem LI
      where O.OK = LI.OK;

+O(xOK, xCK, xD, xXCH) q[] +=

      select sum(LI.P * O.XCH)
      from {<xOK, xCK, xD, xXCH>} O, LineItem LI
      where O.OK = LI.OK;

+LI(yOK, yPK, yP)       q[] += ...
```

Compilation, Example 1

```
q[] = select sum(LI.P * O.XCH)
      from Order O, LineItem LI
      where O.OK = LI.OK;

+O(xOK, xCK, xD, xXCH) q[] +=

      select sum(LI.P *  xXCH)
      from LineItem LI
      where  xOK = LI.OK;

+LI(yOK, yPK, yP)       q[] += ...
```

Compilation, Example 1

```
q[] = select sum(LI.P * O.XCH)
      from Order O, LineItem LI
      where O.OK = LI.OK;

+O(xOK, xCK, xD, xXCH) q[] += xXCH *

      select sum(LI.P)
      from LineItem LI
      where  xOK = LI.OK;
```

} qO[xOK]

```
+LI(yOK, yPK, yP)       q[] += ...
```

Compilation, Example 1

```
q[] = select sum(LI.P * O.XCH)
      from Order O, LineItem LI
      where O.OK = LI.OK;

+O(xOK, xCK, xD, xXCH) q[] += xXCH * qO[xOK];
                       foreach xOK: qO[xOK] =
      select sum(LI.P)
      from LineItem LI
      where  xOK = LI.OK;

+LI(yOK, yPK, yP)      q[] += ...
```

Compilation, Example 1

```
q[] = select sum(LI.P * O.XCH)
      from Order O, LineItem LI
      where O.OK = LI.OK;

+O(xOK, xCK, xD, xXCH) q[] += xXCH * qO[xOK];
+LI(yOK, yPK, yP)      foreach xOK: qO[xOK] +=
      select sum(LI.P)
      from {<yOK, yPK, yP>} LI
      where  xOK = LI.OK;

+LI(yOK, yPK, yP)      q[] += ...
```

Compilation, Example 1

```
q[] = select sum(LI.P * O.XCH)
      from Order O, LineItem LI
      where O.OK = LI.OK;

+O(xOK, xCK, xD, xXCH) q[] += xXCH * qO[xOK];
+LI(yOK, yPK, yP)      foreach xOK: qO[xOK] +=
      select yP

      where  xOK = yOK;

+LI(yOK, yPK, yP)      q[] += ...
```

Compilation, Example 1

```
q[] = select sum(LI.P * O.XCH)
      from Order O, LineItem LI
      where O.OK = LI.OK;
```

```
+O(xOK, xCK, xD, xXCH) q[] += xXCH * qO[xOK];
+LI(yOK, yPK, yP)      qO[yOK] += yP;
```

```
+LI(yOK, yPK, yP)      q[] += ...
```

Compilation, Example 1

```
q[] = select sum(LI.P * O.XCH)
      from Order O, LineItem LI
      where O.OK = LI.OK;

+O(xOK, xCK, xD, xXCH) q[] += xXCH * qO[xOK];
+LI(yOK, yPK, yP)      qO[yOK] += yP;
+LI(yOK, yPK, yP)      q[] +=

      select sum(LI.P * O.XCH)
      from Order O, {<yOK, yPK, yP>} LI
      where O.OK = LI.OK;
```

Compilation, Example 1

```
q[] = select sum(LI.P * O.XCH)
      from Order O, LineItem LI
      where O.OK = LI.OK;

+O(xOK, xCK, xD, xXCH) q[] += xXCH * qO[xOK];
+LI(yOK, yPK, yP)      qO[yOK] += yP;
+LI(yOK, yPK, yP)      q[] +=

      select sum(  yP * O.XCH)
      from Order O
      where O.OK =   yOK;
```

Compilation, Example 1

```
q[] = select sum(LI.P * O.XCH)
      from Order O, LineItem LI
      where O.OK = LI.OK;

+O(xOK, xCK, xD, xXCH) q[] += xXCH * qO[xOK];
+LI(yOK, yPK, yP)      qO[yOK] += yP;
+LI(yOK, yPK, yP)      q[] += yP *

      select sum(       O.XCH)
      from Order O
      where O.OK =   yOK;
```

Compilation, Example 1

```
q[] = select sum(LI.P * O.XCH)
      from Order O, LineItem LI
      where O.OK = LI.OK;
```

```
+O(xOK, xCK, xD, xXCH) q[] += xXCH * qO[xOK];
+LI(yOK, yPK, yP)      qO[yOK] += yP;
+LI(yOK, yPK, yP)      q[] += yP * qLI[yOK];
```

```
      select sum(       O.XCH)
      from Order O
      where O.OK =   yOK;
```

} qLI[yOK]

Compilation, Example 1

```
q[] = select sum(LI.P * O.XCH)
      from Order O, LineItem LI
      where O.OK = LI.OK;
```

```
+O(xOK, xCK, xD, xXCH) q[] += xXCH * qO[xOK];
+LI(yOK, yPK, yP)      qO[yOK] += yP;
+LI(yOK, yPK, yP)      q[] += yP * qLI[yOK];
+O(xOK, xCK, xD, xXCH) foreach yOK: qLI[yOK] +=
      select sum(       O.XCH)
      from {<xOK, xCK, xD, xXCH>} O
      where O.OK =   yOK;
```

Compilation, Example 1

```
q[] = select sum(LI.P * O.XCH)
      from Order O, LineItem LI
      where O.OK = LI.OK;
```

```
+O(xOK, xCK, xD, xXCH) q[] += xXCH * qO[xOK];
+LI(yOK, yPK, yP)      qO[yOK] += yP;
+LI(yOK, yPK, yP)      q[] += yP * qLI[yOK];
+O(xOK, xCK, xD, xXCH) foreach yOK: qLI[yOK] +=
      select             xXCH

      where  xOK =   yOK;
```

Compilation, Example 1

```
q[] = select sum(LI.P * O.XCH)
      from Order O, LineItem LI
      where O.OK = LI.OK;

+O(xOK, xCK, xD, xXCH) q[] += xXCH * qO[xOK];
+LI(yOK, yPK, yP)      qO[yOK] += yP;
+LI(yOK, yPK, yP)      q[] += yP * qLI[yOK];
+O(xOK, xCK, xD, xXCH) qLI[xOK] += xXCH;
```

- The triggers for incrementally maintaining all the maps run in constant time!
- No nonincremental algorithm can do that!

Query factorization

```
select   sum(L.revenue), P.partcat, D.year
from     Date D, Part P, LineOrder L
where    D.datekey = L.datekey
and      P.partkey = L.partkey
group by P.partcat, D.year;
```

Query factorization

```
                foreach pc, y: q[pc, y]  =
select    sum(L.revenue)
from      Date D, Part P, LineOrder L
where     D.datekey = L.datekey
and       P.partkey = L.partkey
and       P.partcat = pc
and       D.year = y;
```

Query factorization

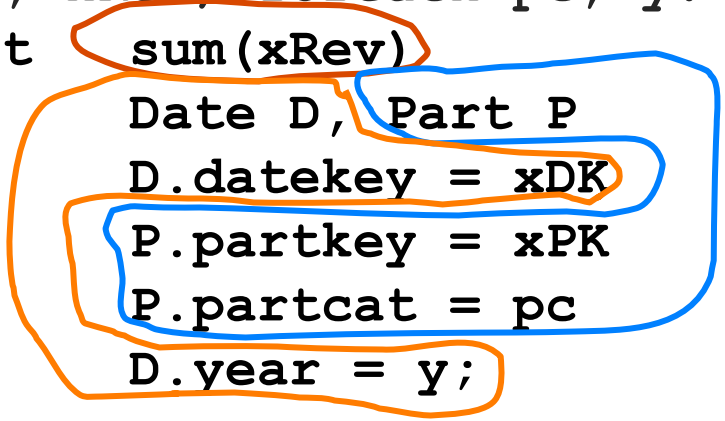
```
+L(xDK, xPK, xRev) foreach pc, y: q[pc, y] +=
      select   sum(L.revenue)
      from     Date D, Part P, {<xDK, xPK, xRev>} L
      where    D.datekey = L.datekey
      and      P.partkey = L.partkey
      and      P.partcat = pc
      and      D.year = y;
```

Query factorization

```
+L(xDK, xPK, xRev) foreach pc, y: q[pc, y] +=
      select   sum(xRev)
      from     Date D, Part P
      where    D.datekey = xDK
      and      P.partkey = xPK
      and      P.partcat = pc
      and      D.year = y;
```

Query factorization

```
+L(xDK, xPK, xRev) foreach pc, y: q[pc, y] +=
      select   sum(xRev)
      from     Date D, Part P
      where    D.datekey = xDK
      and      P.partkey = xPK
      and      P.partcat = pc
      and      D.year = y;
```



Factorization

select sum(t*t') from (Q x Q') =
(select sum(t) from Q) * (select sum(t') from Q')
if no overlap in variables.

Query factorization

```
+L(xDK, xPK, xRev) foreach pc, y: q[pc, y] +=
      xRev *
     (select   sum(1)                    } m1[datekey,year]
      from     Date D                    }
      where    D.datekey = xDK           }
      and      D.year = y) *             }
     (select   sum(1)                    } m2[partkey,partcat]
      from     Part P                    }
      where    P.partkey = xPK           }
      and      P.partcat = pc);          }
```

Query languages for MCMC

- Goal: Develop query languages for declaratively defining Markov Chains
  - in a relational or probabilistic database.
- Language semantics: generalization of first-order interpretations.
  - Current database state Q . Define next state Q' as by sampling from a distribution P(Q' | Q).
  - Thus the query directly defines the transition kernel of the Markov Chain.
  - To define, P(Q' | Q), need a language construct for introducing uncertainty: Repair-key construct.
- Language syntax: Relational algebra + repair-key

[Deutch, K., Milo, PODS 2010]

The repair-key operation

Example: Tossing a biased coin twice.

R	Toss	Face	FProb
	1	H	.4
	1	T	.6
	2	H	.4
	2	T	.6

Pr = 1

$S := \text{repair-key}_{\text{Toss@FProb}}(R)$ results in four worlds:

S^1	Toss	Face	FProb
	1	H	.4
	2	H	.4

S^2	Toss	Face	FProb
	1	H	.4
	2	T	.6

S^3	Toss	Face	FProb
	1	T	.6
	2	H	.4

S^4	Toss	Face	FProb
	1	T	.6
	2	T	.6

$$p_1 = 1 \cdot \frac{.4}{.4+.6} \cdot \frac{.4}{.4+.6} = .16, \quad p_2 = p_3 = .24, \quad p_4 = .36$$

Example

- Database schema:
  - E(I,J,P) – edge relation with edge probabilities (stochastic matrix)
  - C(I) – starting node
- Random walk:

$$\begin{aligned} C &:= \rho_I \pi_J \big(\text{repair-key}_{I@P}(C \bowtie E)\big) \\ E &:= E \quad \% \text{ unchanged.} \end{aligned}$$

- PageRank:

$$\begin{aligned} C := \text{repair-key}_{@P} & \\ & \big(\rho_I(\pi_J(\text{repair-key}_{I@P}(C \bowtie E))) \times \rho_P(\{1-\alpha\}) \\ & \quad \cup \pi_I(\text{repair-key} \quad (V)) \times \rho_P(\{\alpha\})\big) \end{aligned}$$

Complexity results

- Data complexity of approximating expectations or probabilities of tuples in query results on the samples/states.
- Also studied: Restriction - Inflationary semantics using datalog with probabilistic rules.
  - Add data of new state to current state. Convergence of the database.
- Generally intractable, but this is not a bug of the language design
  - Usually, for MCMC, we have no complexity results at all.
  - These are basic complexity results for fixed transition kernels/queries.

Language	exact computation	relative approximation	absolute approximation
(linear) datalog without probabilistic rules	$\sharp P$-hard, in PSPACE	NP-hard (Thm. 4.1)	in PTIME
inflationary fixpoint with probabilistic rules	$\sharp P$-hard, in PSPACE	NP-hard	in PTIME (Thm. 4.3)
non-inflationary fixpoint with probabilistic rules	$\sharp P$-hard, in (2-)EXPTIME	NP-hard	NP-hard; PTIME in input size and mixing time (Thm. 5.1,5.6)

Table 1: Overview of complexity results (data complexity).

Further work on MC in Databases

- [Soliman, Ilyas, David: VLDB J., 2010]
  - Using MCMC for ranking in uncertain databases
- [Re, Letchner, Balazinska, Suciu, SIGMOD 2008]
  - Markovian streams/particle filters

Research challenges

- Integrate more sophisticated stats techniques into systems
  - for output analysis
  - prediction of burn-in and stopping times
  - have a database of best practices for determining starting values, mixing time? A library of building blocks for transition kernels?
- How to improve efficiency/scalability beyond incremental view maintenance, parallelization?
- Build an MCMC DBMS!

Conclusions

- MCMC databases are an interesting concept and need to be explored further.
- We have to decide what abstractions to use and what interfaces to provide.
  - If the interfaces are too low-level, the utility is limited.
  - But MCMC in practice requires much experimentation and tweaking: there is no general machinery for
    - deciding burn-in and stopping times
    - automatizing the choice of starting point and proposal distribution.
- Technically the most satisfying contribution from DB is incremental view maintenance.
- Do MCMC DBMS add sufficient utility compared to statistics packages such as BUGS? Our main contribution seem to be *queries*.

Questions/Remarks?

Thank you!