SPROUT: Scalable Query Processing in Probabilistic Databases

Oxford University Computing Laboratory

http://www.comlab.ox.ac.uk/projects/SPROUT/

February 2010, Dan Olteanu

Key goals and contributions:

- discover tractable query&data (sub)instances: tractable inequality (<,≠) queries, database restrictions (e.g., functional dependencies, tuple independent),
- design scalable techniques for exact and approximate query evaluation: incremental lineage factorization, compilation into read-once functions, OBDDs,
- implement open-source query engine SPROUT as an extension of PostgreSQL backend: secondary-storage confidence computation, lazy/eager query plans.

Incremental Lineage Factorization

- Complete factorization in polynomial time for tractable query & data instances.
- Partial factorization for hard instances gives lower/upper bounds on probability.

- Independent-or $\otimes$: Partition Φ into independent DNFs $\Phi_1, \Phi_2 \subset \Phi$ such that Φ is equivalent to $\Phi_1 \vee \Phi_2$.
- Independent-and $\odot$: Partition Φ into independent DNFs $\Phi_1, \Phi_2 \subset \Phi$ such that Φ is equivalent to $\Phi_1 \wedge \Phi_2$.
- Exclusive-or $\oplus$: Choose a variable x in Φ. Replace Φ by

$$\bigoplus_{a \in \mathrm{Dom}_x, \Phi|_{x=a} \neq \emptyset} (\{\{x = a\}\} \odot \Phi \mid_{x=a})$$

where the DNF $\Phi \mid_{x=a}$ is obtained from Φ by removing all clauses $\phi \in \Phi$ for which $\phi \wedge (x = a)$ is inconsistent and (syntactically) removing the atomic formula $x = a$ from the remaining clauses in which it occurs. Obviously, $(x = a) \wedge \Phi$ is equivalent to $(x = a) \wedge \Phi \mid_{x=a}$. This decomposition is called Shannon expansion.

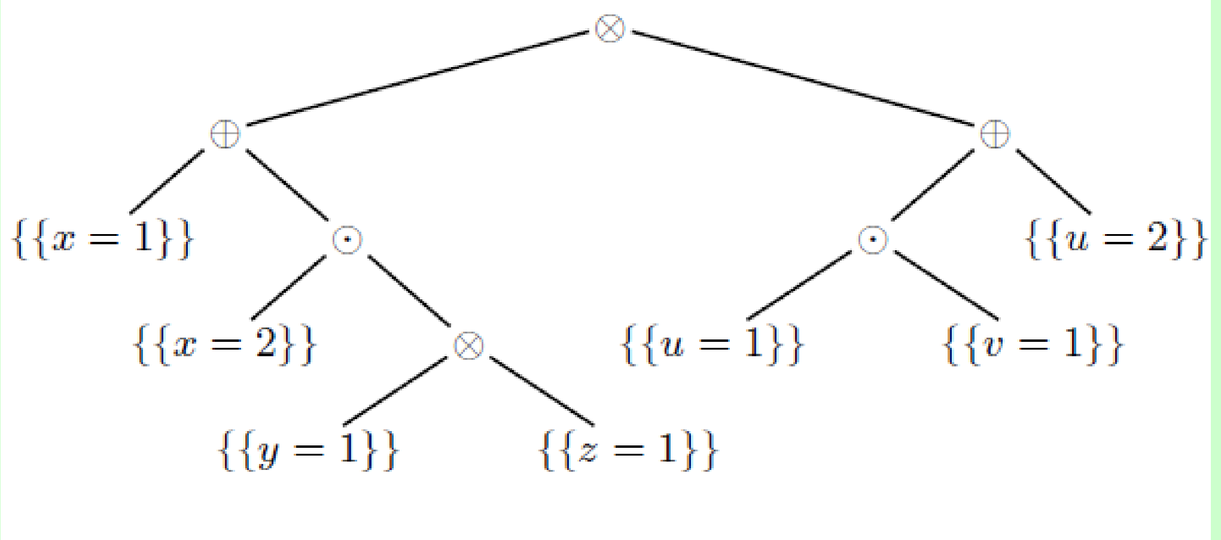


Fig. 2. D-tree of DNF $\Phi = \{\{x = 1\}, \{x = 2, y = 1\}, \{x = 2, z = 1\}, \{u = 1, v = 1\}, \{u = 2\}\}$.

Approximate evaluation for positive relational algebra

- Given a partial factorization (**d-tree**) and lower & upper bounds for the probabilities of leaf DNFs, we can efficiently compute bounds for the probability of the d-tree.

Proposition 5.8: Given a DNF Φ, a fixed error ϵ, and a d-tree for Φ with bounds $[L, U]$.

- If $U - \epsilon \leq L + \epsilon$, then any value in $[U - \epsilon, L + \epsilon]$ is an absolute ϵ-approximation of $P(\Phi)$.
- If $(1 - \epsilon) \cdot U \leq (1 + \epsilon) \cdot L$, then any value in $[(1 - \epsilon) \cdot U, (1 + \epsilon) \cdot L]$ is a relative ϵ-approximation of $P(\Phi)$. □

- The factorization is continued at promising leaves until the bounds on the probability of the d-tree get tight enough.
- **Memory-efficient version**: only store the current root-to-leaf path; in depth-first construction of the d-tree, before factorizing the current leaf, we can decide *locally* whether the overall desired approximation can still be met even if that leaf is *closed* (not factorized further).
- Underlying idea: after a certain depth in the d-tree, *the approximation introduced by discarding a leaf may be big locally, but it is insignificant from a global perspective.*

Example: Absolute error = .012.
We cannot stop: Upper – Lower = .644 – .595 = .049 > 2*.012 = .024
We may close the current leaf (and be pessimistic about the remaining leaves): Upper' – Lower = .6173 – .595 = .0223 < .024.

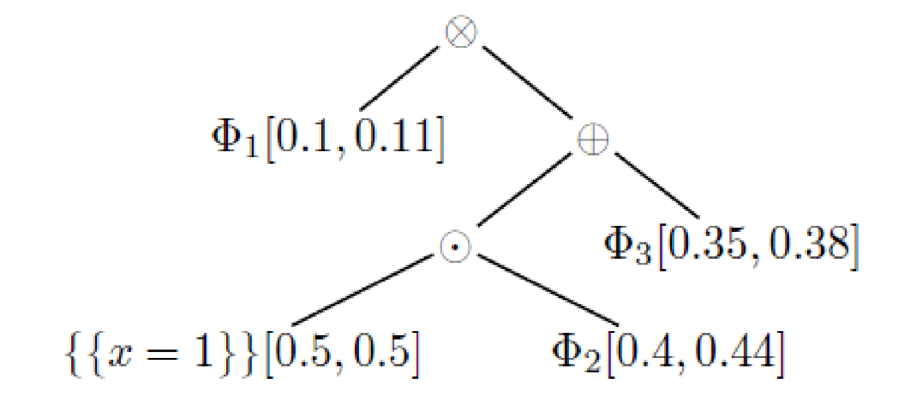


Fig. 4. D-tree. Leaves: Φ_1 is closed, Φ_2 is current, Φ_3 is open.

Tractable conjunctive queries

For the class **TQ** of all tractable conjunctive queries without self-joins (*hierarchical*), query lineage can be factorized into read-once functions for any tuple-independent probabilistic database.

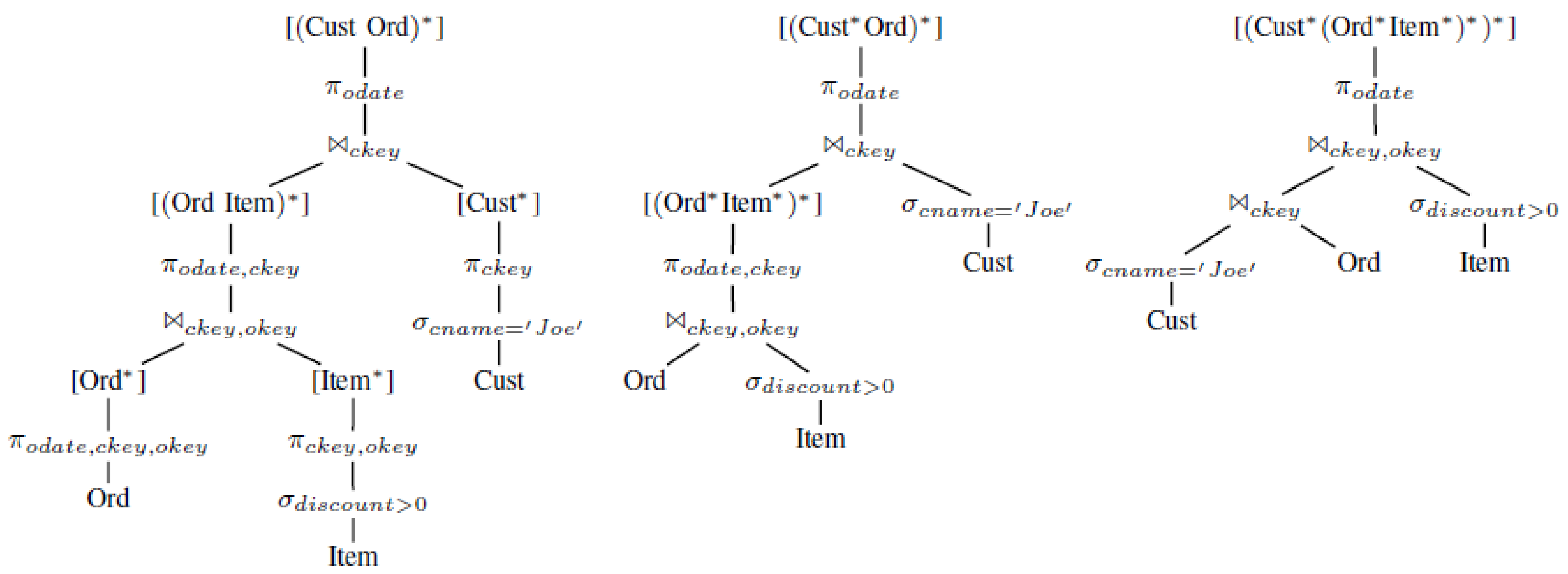
Theorem: For any TQ query q and database D, $\forall t \in q(D)$, and lineage ϕ_t,

- 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.

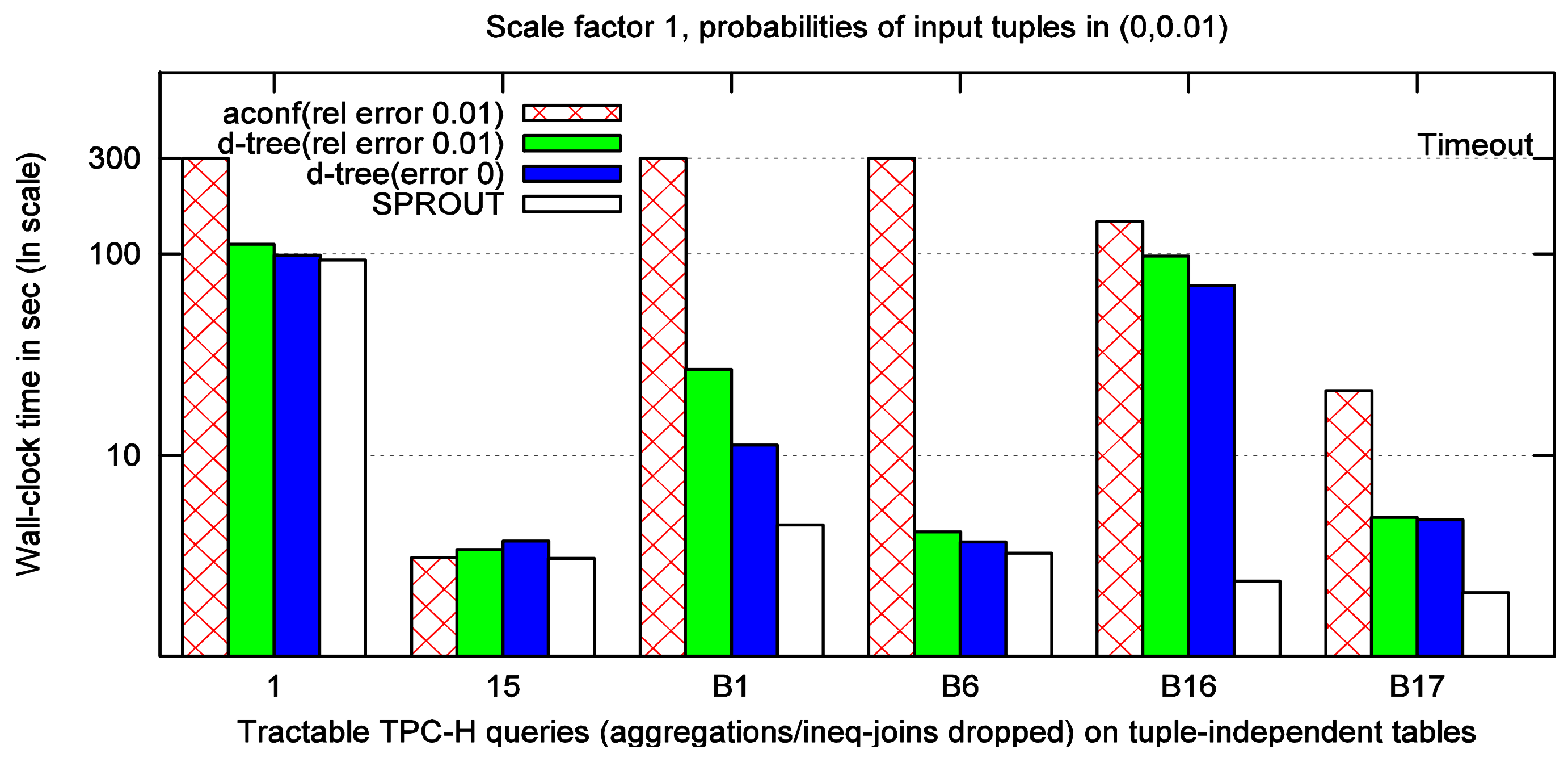
Convex conjunctive queries with inequalities (<) admit OBDDs quadratic in the size of the query lineage. This tractability result carries over to counting vertex covers in convex bipartite graphs.

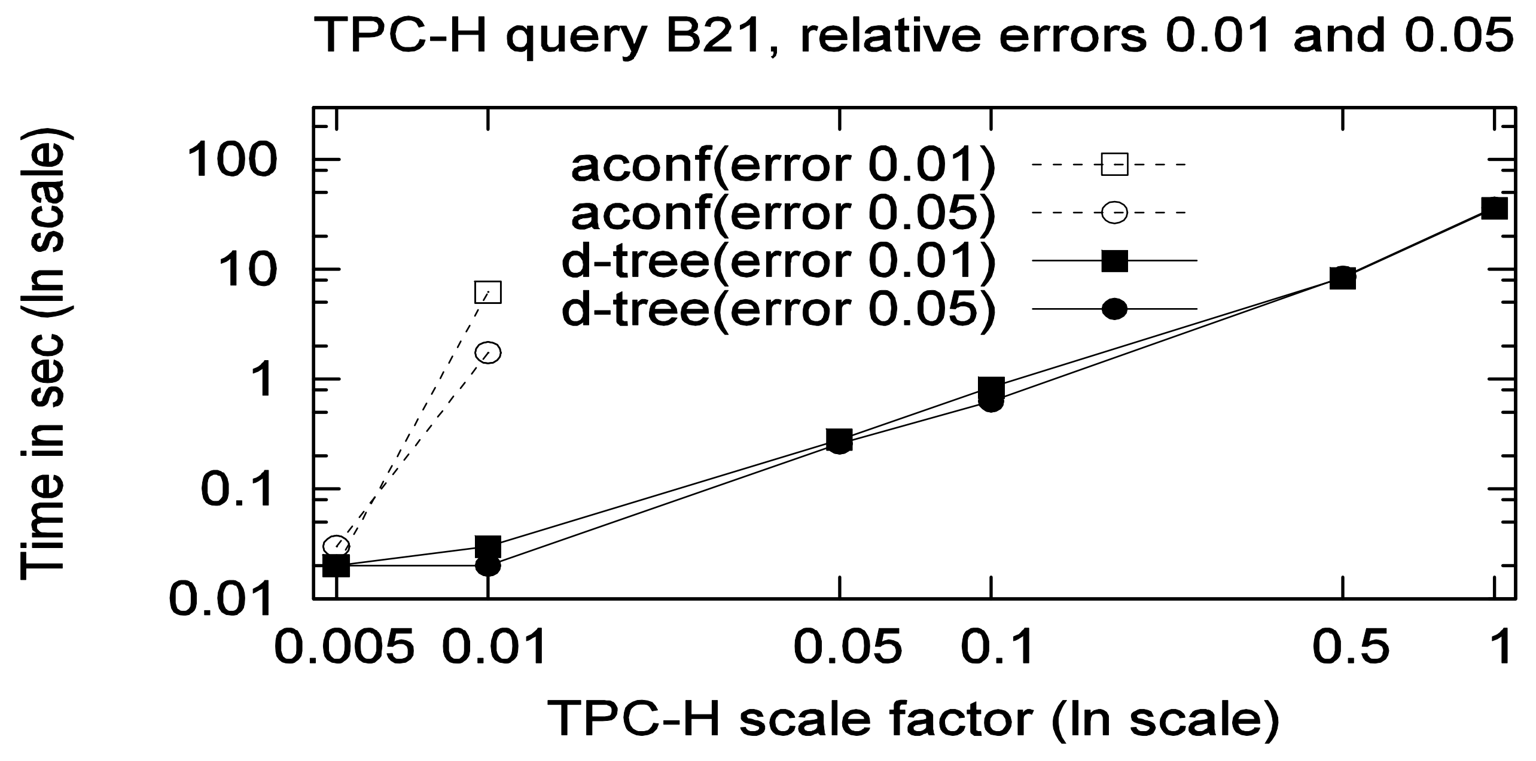
Lazy vs. eager query plans for exact confidence computation of TQ queries

- Confidence computation done by an **aggregation operator fully integrated into relational plans**.
- Uses the **query signature** (in the brackets, e.g., [Cust Ord*]) to understand whether joins are one/many-one/many and derive the number of passes over the lineage needed for computation.
- Left: Eager plan
  (operator pushed down)
- Middle: Hybrid plan
- Right: Lazy plan
  (operator done at the end)



aconf = optimized Karp-Luby FPRAS**. d-tree** = incremental lineage factorization.
SPROUT (here) = secondary-storage lineage factorization for hierarchical queries only.





Selected publications on SPROUT:

Approximate Confidence Computation in Probabilistic Databases.
ICDE'10. D. Olteanu, J. Huang, and C. Koch.

Secondary-Storage Confidence Computation for Conjunctive Queries with Inequalities.
SIGMOD'09. D. Olteanu, J. Huang

SPROUT: Lazy vs. Eager Query Plans for Tuple-Independent Probabilistic Databases.
ICDE'09. D. Olteanu, J. Huang, C. Koch.

Using OBDDs for Efficient Query Evaluation on Probabilistic Databases.
SUM'08. D. Olteanu, J. Huang.