SPROUT: Scalable Query Processing in Probabilistic Databases
Oxford University Computing Laboratory

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

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.

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

\[
\text{Proposition 5.5: Given a DNF } \Phi, \text{ a fixed error } \epsilon, \text{ and a database } L, U, \text{ then any value in } [U - \epsilon, L + \epsilon] \text{ is an } \epsilon \text{-approximation of } P(\Phi).
\]

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.
- 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.

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

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.
- 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.

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 \(\mathcal{TQ}\) queries
- Confidence computation done by an aggregation operator fully integrated into relational plans.
- Uses the query signature (in the brackets, e.g., \([\text{Cust Ord}^*]\)) 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)

Selected publications on SPROUT:
Approximate Confidence Computation in Probabilistic Databases.

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

ICDE’09. D. Olteanu, J. Huang, C. Koch.

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