Answering Conjunctive Queries under Updates
Christoph Berkholz, Jens Keppeler, Nicole Schweikardt
Humboldt-Universität zu Berlin

Abstract
Enumerating and counting answers to k-ary conjunctive queries under single-tuple updates.

- Upper bounds: CQ q-hierarchical \implies constant update time
- Lower bounds: CQ not q-hierarchical \implies \Omega(n^{-1}) update time, n = size of the active domain (under OV- and OMv-conjectures).

Dichotomy for counting CQs
Theorem Let \( \varphi \) be a CQ.
- If \( \varphi \) is q-hierarchical, then \( |\varphi(D)| \) can be computed with linear preprocessing time and constant update time.
- Otherwise, assuming the OMv-conjecture and the OV-conjecture, there is no algorithm that computes \( |\varphi(D)| \) with arbitrary preprocessing time and \( O(n^{1-\varepsilon}) \) update time.

Dichotomy for enumerating CQs
Theorem Let \( \varphi \) be a self-join free CQ.
- If \( \varphi \) is q-hierarchical, then \( \varphi(D) \) can be enumerated with constant delay and constant update time after linear preprocessing.
- Otherwise, assuming the OMv-conjecture, there is no algorithm with arbitrary preprocessing time and \( O(n^{1-\varepsilon}) \) update time that enumerates \( \varphi(D) \) with \( O(n^{1-\varepsilon}) \) delay.

Algorithmic conjectures

Online matrix-vector multiplication (OMv)

Input Boolean \( n \times n \) matrix \( M \) and stream \( v_1, \ldots, v_t \) of \( n \)-dimensional Boolean vectors.

Task After preprocessing \( M \), compute \( Mv_t \) before \( v_{t+1} \) arrives.

OMv-conjecture (Henzinger et al. 2015) For every \( \varepsilon > 0 \), no algorithm solves OMv in time \( O(n^{1-\varepsilon}) \).

Orthogonal vectors (OV)

Input Two sets \( U \) and \( V \) of \( n \) Boolean vectors of dimension \( d \).

Question Are there \( u \in U \) and \( v \in V \) such that \( u^Tv = 0 \)?

OV-conjecture (Williams 2005) For every \( \varepsilon > 0 \), no algorithm solves OV for \( d = \lceil \log^2 n \rceil \) in time \( O(n^{1-\varepsilon}) \).

Example If non-q-hierarchical \( \varphi(x) = \exists y \ E(x, y) \land T(y) \) can be counted in \( O(n^{1-\varepsilon}) \) update time, then the OV-conjecture fails.

\[
\begin{align*}
T_0^D & : v_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \\
T_2^D & : \varphi(D_3) = 4 = |\{u_j \mid u_j^Tv_1 \neq 0\}|
\end{align*}
\]

- update \( T_0^D \) for every \( v_1 \) and test against all \( u_j \in U \) via \( |\varphi(D_i)| \)

q-hierarchical queries
\[
\varphi(x, y, z, y', z') = (Rxyz \land Rxyz' \land Exy \land Exy' \land Sxyz)
\]

A conjunctive query is q-hierarchical if it has a q-tree in which
- variables of every atom form a path in this tree starting at the root,
- free variables form a connected subtree containing the root.

Data structure for q-hierarchical queries
\[
\varphi(x, y, z, y', z') = (Rxyz \land Rxyz' \land Exy \land Exy' \land Sxyz)
\]

\[
E^D = \{(a, c), (a, f), (b, d), (b, g), (b, h)\},
S^D = \{(a, e, a), (a, e, b), (a, f, c), (b, g, b), (b, p, a)\},
R^D = S^D \cup \{(a, e, c), (b, g, a), (b, g, c), (b, p, b), (b, p, c)\}.
\]

Data structure represents the query result (of size \( |D|^2 \)) in space \( O(|D|) \) and can be updated in constant time on single-tuple updates.

Data structure allows to:
- answer a Boolean query in constant time,
- compute the size \( |\varphi(D)| \) of the query result in constant time,
- enumerate the query result with constant delay between the tuples,
- test for a given \( t \), whether \( t \in \varphi(D) \) in constant time,
- enumerate the change \( \varphi(D_{old}) \triangle \varphi(D_{new}) \) in the result with constant delay and compute its size in constant time.

Here, constant time (wrt. data complexity) means \( O(\text{poly}(\varphi)) \) and there are no large “hidden constants”.
It turns out that evaluation of q-hierarchical queries is also efficient in practice. See: Idris, Ugarte, Vansummeren: “The dynamic Yannakakis Algorithm: Compact and Efficient Query Processing Under Updates” at SIGMOD’17.

Contact
Mail \{berkholz, schweikn, keppelj\}@informatik.hu-berlin.de
Url www.informatik.hu-berlin.de/logik
Full version https://arxiv.org/abs/1702.06370