Optimized and Parallel Query Processing in Similarity-based Databases

Petr Krajča

DEPARTMENT OF COMPUTER SCIENCE
PALACKÝ UNIVERSITY, OLOMOUC

Motivation

Natural query

Find a *hatchback* which costs *about $11,500* or *less*.

Database query

$$\begin{aligned}&\texttt{RETRIEVE}\ \mathit{cars}\ \texttt{WHERE}\ \mathit{type} \approx_{\mathit{type}} \texttt{'Hatchback'}\\ &\qquad\qquad \otimes (\mathit{price} \approx_{\mathit{price}} 11500 \vee \mathit{price} < 11500);\end{aligned}$$

Result

	name	price	type	year
1.00	Ford Focus	9811.0	Hatchback	2011
0.80	Hyundai i30	11699.0	Hatchback	2010
0.50	Honda Accord	10600.0	Wagon	2010
0.44	Ford Fiesta	11560.0	Wagon	2011

Theoretical Foundations

- generalized Codd's relational model
- relational algebra
- relational calculus
- functional dependencies

(i) Bělohávek R., Vychodil V.: Relational model of data over domains with similarities: An extension for similarity queries and knowledge extraction. In *IRI (2006)*, IEEE Systems, Man, and Cybernetics Society.

(ii) Bělohávek R., Opichal S., Vychodil V.: Relational algebra for ranked tables with similarities: Properties and implementation. In IDA (2007), M. R. Berthold, et al., Eds., vol. 4723 of *Lecture Notes in Computer Science*, Springer.

(iii) Bělohávek R., Vychodil V.: Data tables with similarity relations: functional dependencies, complete rules and non-redundant bases. In: *DASFAA 2006*, LNCS 3882, pp. 644–658 (2006)

(iv) Bělohlávek R., Vychodil V.: Query systems in similarity-based databases: logical foundations, expressive power, and completeness. In: *ACM SAC 2010*, pp. 1648–1655 (2010)

(v) Bělohlávek R., Vychodil V.: Codd's relational model from the point of view of fuzzy logic. *J. Logic and Computation* 21:851–862 (2011)

(vi) . . .

Towards Practical Implementation

Query Language (RESIQL)

- Krajca P., Vychodil V.: *Basic Concepts of Relational Query Language for Similarity-Based Databases*. (MDAI 2012)

Algorithms for Data Processing

- Krajca P., Vychodil V.: *Query Optimization Strategies in Similarity-Based Databases*. (MDAI 2013)
- covers the most common scenarios (top-k queries)
- further requirements (e.g., order of rows, random access)
- unusual or complex queries difficult to optimize
- . . . a fallback plan

Preliminaries

Scale of Truth Degrees

- complete residuated lattice: $\mathbf{L} = \langle L, \wedge, \vee, \otimes, \rightarrow, 0, 1\rangle$
- $\langle L, \wedge, \vee, 0, 1\rangle$ is a complete lattice with 0 and 1 being the least and greatest element of L
- $\langle L, \otimes, 1\rangle$ is a commutative monoid
- $\otimes$ and $\rightarrow$ satisfy so-called adjointness property: $a \otimes b \leq c$ iff $a \leq b \rightarrow c$ for each $a, b, c \in L$

Example 1

- L $= [0,1]$
- $a \otimes b = \max(0, a + b - 1)$
- $a \rightarrow b = \min(1, 1 - a + b)$

Example 2

- L $= [0,1]$
- $a \otimes b = a \cdot b$
- $a \rightarrow b = \begin{cases} 1, & \text{if } a < b \\ \frac{b}{a}, & \text{otherwise} \end{cases}$

Data Model: Basic Concepts

- nonempty **set** Y **of attributes** – names of columns
- finite subset $R \subseteq Y$ is called a **relation scheme** (heading of the table)
- each attribute $y \in Y$ has its **domain** D_y (set of attribute's values)
- having a scale of truth degrees, L each domain D_y can be equipped with a map $\approx_y \colon D_y \times D_y \to L$, called a **similarity**, satisfying conditions of reflexivity and symmetry:

  (i) $\approx_y(u, u) = 1$ for all $u \in D_y$;

  (ii) $\approx_y(u, v) = \approx_y(v, u)$
  for all $u, v \in D_y$
- $u \approx_y v$ is interpreted as a degree to which $u \in D_y$ is similar to $v \in D_y$

Data Model: Ranked Data Tables (Definition)

- **cartesian product** of domains D_y $(y \in R)$, denoted by $\prod_{y \in R} D_y$, is a set of all maps $r\colon R \to \bigcup_{y \in R} D_y$ such that $r(y) \in D_y$ for all $y \in R$
- each $r \in \bigcup_{y \in R} D_y$ shall be called a **tuple** on R over domains D_y $(y \in R)$
- a **ranked data table** on R (shortly, an RDT) over domains D_y with similarities $\approx_y$ $(y \in R)$ is any map

$$\mathcal{D}\colon \prod_{y \in R} D_y \to L$$

  such that there are at most finitely many tuples r such that $\mathcal{D}(r) > 0$.
- the degree $\mathcal{D}(r)$ assigned to tuple r by $\mathcal{D}$ shall be called a **rank** of tuple r in $\mathcal{D}$

Data Model: Ranked Data Tables (Remarks)

- attributes from R denote table columns
- values from D_y are table entries
- order of tuples and columns does not matter
- RDTs are counterparts to the ordinary data tables in the original Codd's model
- RDTs represent stored data
- RDTs are results of similarity-based queries where tuples are allowed to match conditions to degrees
- rank indicates the degree to which tuple satisfies the given query

Query Processing

Common Strategy

1. query is transformed from a query language (SQL, RESIQL) into a relational algebra expression
2. rules of rel. algebra are applied
3. execution plan is constructed (physical operators working with data)
4. data are retrieved

Remarks

- for Codd's original RM set of known physical operators exists (MergeJoin, HashJoin, etc.)
- for generalized RM limited number of physical operators
    - variants of Fagin's algorithm
    - top-k queries
    - further conditions have to be fulfilled (sorted access, random access)

Our Alternative Strategy

- inspiration in compilers of general purpose programming languages

1. query is transformed from a query language (SQL, RESIQL) into a relational algebra expression
2. rules of rel. algebra are applied
3. rel. algebra operators are decomposed to elementary operations
4. elementary operations are subject of optimizations
5. query is processed

Operation: Restriction

- for an RDT $\mathcal{D}$ on $R = \{r_1, \ldots, r_n\}$, attribute $r_i \in R$, and $c \in D_{r_i}$ similarity based restriction $\sigma_{r_i \approx c}(\mathcal{D})$ is defined by

$$(\sigma_{r_i \approx c}(\mathcal{D}))(u) = \mathcal{D}(u) \otimes (u(r_i) \approx c)$$

- if $\mathcal{D}$ is result of query Q, the rank given by restriction is a degree to which "u matches Q and its r_i-value is similar to c.

Decomposition

```
for $u$ in $\mathcal{D}$
  emit ($rank : \mathcal{D}(u)$ $\otimes$ ($u(r_i)$ $\approx$ $d$), $r_1 : u(r_1)$, ..., $r_n : u(r_n)$)
```

New operators

- `for` u `in` $\mathcal{D}$ – loops over all tuples u in $\mathcal{D}$; collects all emitted tuples
- `emit` $f(u)$ – emits new tuple (applies transformation function f on each tuple u)
- relationship to *map* function from Lisp

Operators: Natural Join

- for two RDTs $\mathcal{D}_1$ and $\mathcal{D}_2$ with relation schemes $\{r_1 \ldots, r_n, t_1, \ldots, t_n\}$ and $\{s_1, \ldots, s_n, t_1, \ldots, t_n\}$, respectively, with common attributes $t_1, \ldots, t_n$
- natural join is a relation on $\{r_1 \ldots, r_n, t_1, \ldots, t_n, s_1, \ldots, s_n\}$ consisting of (set-theoretic) concatenation of all joinable tuples uw and vw from $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively, such that

$$(\mathcal{D}_1 \bowtie \mathcal{D}_2)(uvw) = \mathcal{D}_1(uw) \otimes \mathcal{D}_2(vw)$$

Decomposition

```
for $u$ in $\mathcal{D}_1$
  for $v$ in $\mathcal{D}_2$
    emit ($rank : \mathcal{D}_1(u) \otimes \mathcal{D}_2(v) \otimes u(t_1) = v(t_1) \otimes \cdots \otimes u(t_n) = v(t_n)$,
          $r_1 : u(r_1)$, ..., $r_n : u(r_n)$,
          $t_1 : u(t_1)$, ..., $t_n : u(t_n)$,
          $s_1 : v(s_1)$, ..., $s_n : v(s_n)$)
```

Operators: Projection

- for an RDT $\mathcal{D}$ on T is the *projection* $\pi_R(\mathcal{D})$ of $\mathcal{D}$ onto $R \subseteq T$ defined as follows

$$(\pi_R(\mathcal{D}))(u) = \bigvee\{\mathcal{D}(uv) \mid v \in \textstyle\prod_{y \in T \setminus R} D_y\}$$

  for each tuple $u \in \prod_{y \in R} D_y$.

Decomposition

```
reduce $\bigvee$
  for $u$ in $\mathfrak{R}$
    emit ($rank : \mathcal{D}(u)$, $r_1 : u(r_1)$, ..., $r_m : u(r_m)$)
```

New Operator

- `reduce` g – aggregates ranks of the same tuples with the aggregation function g

Operators: more

- `reduce` allows to implement set-theoretic operations by choosing proper aggregation function
- for instance, for union $(\mathcal{D}_1 \cup \mathcal{D}_2)$ we use $\bigvee$

Decomposition

```
reduce $\bigvee$
  for $u$ in $\mathcal{D}_1$
    emit $u$
  for $v$ in $\mathcal{D}_2$
    emit $v$
```

- intersection more complicated (see proceedings)

Optimizations

Optimization: Composition

- joins two `for` operators

Expressions

```
$\mathfrak{R}_1$: for $u$ in $\mathfrak{R}$
        $body$
           emit $f_1(u)$
$\mathfrak{R}_2$: for $v$ in $\mathfrak{R}_1$
        emit $f_2(v)$
```

is transformed to

```
$\mathfrak{R}_2$: for $u$ in $\mathfrak{R}$
        $body$
           emit $f_2(f_1(u))$
```

Optimization: Composition (Example)

- two restrictions $\sigma_{r_i \approx c_i}(\sigma_{r_j \approx c_j}(\mathcal{D}))$ on RDT $\mathcal{D}$ with relation scheme $\{r_1, \ldots r_n\}$.
- two `for`-loops

```
$\mathfrak{R}_1$: for $u$ in $\mathcal{D}$
        emit ($rank : \mathcal{D}(u) \otimes (u(r_i) \approx c_i)$, $r_1 : u(r_1)$, ..., $r_n : u(r_n)$)
$\mathfrak{R}_2$: for $v$ in $\mathfrak{R}_1$
        emit ($rank : \mathcal{D}(v) \otimes (v(r_j) \approx c_j)$, $r_1 : v(r_1)$, ..., $r_n : v(r_n)$)
```

after transformation

```
$\mathfrak{R}_2$: for $u$ in $\mathcal{D}$
        emit ($rank : \mathcal{D}(u) \otimes (u(r_i) \approx c_i) \otimes (u(r_j) \approx c_j)$,
                $r_1 : u(r_1)$, ..., $r_n : u(r_n)$)
```

Remark

- new representation corresponds to $\sigma_{(r_i \approx c_i) \otimes (r_j \approx c_j)}(\mathcal{D})$ (consistent with rel. algebra)

Optimization: Filtering

- queries contains often multiple nested loops
- decision if the tuple will be emitted (has non-zero rank) is always in the inner-most loop
- desirable to skip useless computations
- `emit` is the only place where rank is assigned
- if the final rank is given by subexpressions aggregated by a monotone function (e.g., $\otimes$, $\wedge$), one can determine whether the final rank will be zero, or not

New operator

- `filter` $cond$ – perform expressions in the body if $cond \neq 0$
- `filter` is placed always to the outter most `for`-loop
- corresponds to *loop-invariant code motion* optimization

Optimization: Filtering (Example)

- $\sigma_{r_i=c}(\mathcal{D}_1 \bowtie \mathcal{D}_2)$
- where $\mathcal{D}_1$ and $\mathcal{D}_2$ are RDTs with disjoint schemes may be compiled to:

```
for $u$ in $\mathcal{D}_1$
  filter ($u(r_i)$ $\approx$ $c$)
    for $v$ in $\mathcal{D}_2$
      emit ($rank : \mathcal{D}_1(u)$ $\otimes$ $\mathcal{D}_2(v)$ $\otimes$ $(u(r_i)$ $\approx$ $c)$,
            $r_1 : u(r_1)$, ..., $r_n : u(r_n)$,
            $s_1 : v(s_1)$, ..., $s_n : v(s_n)$)
```

Optimization: Index Selection

- RDBMS uses indexes to efficiently retrieve data from the physical storage
- exists methods for similarity-based databases

New operator

- `index` $\mathcal{D}, cond$ – from the physical storage of the RDT $\mathcal{D}$ (must be relational variable) retrieves tuples satisfying condition $cond$
- if the `for` has a relational variable as its argument and is followed by a `filter` operation, it is an opportunity for optimization

Optimization: Index Selection (Example)

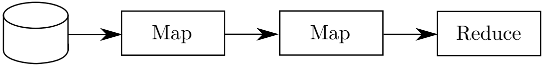
- similarity-based join $\sigma_{r_i \approx s_i}(\mathcal{D}_1 \bowtie \mathcal{D}_2)$
- two RDTs $\mathcal{D}_1$ and $\mathcal{D}_2$ with relation schemes $\{r_1 \ldots, r_n\}$ and $\{s_1, \ldots, s_n\}$, respectively
- an index on attribute r_i

```
for $u$ in $\mathcal{D}_1$
  for $v$ in index($\mathcal{D}_2$, $u(r_i) \approx v(s_i)$)
    filter ($u(r_i) \approx v(s_i)$)
      emit ($rank : \mathcal{D}_1(u) \otimes \mathcal{D}_2(v) \otimes (u(r_i) \approx v(s_i))$,
            $r_1 : u(r_1)$, ..., $r_n : u(r_n)$,
            $s_1 : v(s_1)$, ..., $s_n : v(s_n)$)
```

- a nested-loop join algorithm
- efficiently uses indexes

Implications

- operators of the generalized model can be transformed into 3 (or 5) elementary operations
- three simple rules allows to infer algorithms for otherwise unconsidered combinations of operators
- all operators are compatible with the map/reduce framework for data processing (implicit parallelization)
  - `for`, `emit`, and `filter` – map jobs
  - `reduce` – reduce job
- two nested loops and projection



Evaluation

	unoptimized		optimized		
	tuples	time	tuples	time	time (8 procs.)
adult	829,821	5.6 s	292,938	3.9 s	1.9 s
bank	2,044,034,521	151 min	65,705	2.3 s	1.2 s
cars	65,898	458 ms	43,964	323 ms	178 ms
wine quality	28,302,400	143.6 s	352,841	7.1 s	2.9 s

Number of tuples fetched and processing time

Conclusions

- novel method of optimizations in similarity based database
- complementary to existing algorithms
- allows for implicitly parallel or distributed computing (Apache Hadoop, Apache Spark)