Database Queries - Logic and Complexity

Moshe Y. Vardi

Rice University

Logic in Computer Science

During the past fifty years there has been extensive, continuous, and growing interaction between logic and computer science. In many respects, logic provides computer science with both a unifying foundational framework and a tool for modeling computational systems. In fact, logic has been called "the calculus of computer science". The argument is that logic plays a fundamental role in computer science, similar to that played by calculus in the physical sciences and traditional engineering disciplines. Indeed, logic plays an important role in areas of computer science as disparate as machine architecture, computer-aided design, programming languages, databases, artificial intelligence, algorithms, and computability and complexity.

Why on Earth?

Basic Question: What on earth does an obscure, old intellectual discipline have to do with the youngest intellectual discipline?

Cosma R. Shalizi, Santa Fe Institute:

"If, in 1901, a talented and sympathetic outsider had been called upon (say, by a granting-giving agency) to survey the sciences and name the branch that would be least fruitful in century ahead, his choice might well have settled upon mathematical logic, an exceedingly recondite field whose practitioners could all have fit into a small auditorium. It had no practical applications, and not even that much mathematics to show for itself: its crown was an exceedingly obscure definition of cardinal numbers."

Back to The Future

M. Davis (1988): *Influences of Mathematical Logic on Computer Science*:

"When I was a student, even the topologists regarded mathematical logicians as living in outer space. Today the connections between logic and computers are a matter of engineering practice at every level of computer organization."

Question: Why on earth?

Birth of Computer Science: 1930s

Church, Gödel, Kleene, Post, Turing: Mathematical proofs have to be "machine checkable" - *computation* lies at the heart of mathematics!

Fundamental Question: What is "machine checkable"?

Fundamental Concepts:

- *algorithm*: a procedure for solving a problem by carrying out a precisely determined sequence of simpler, unambiguous steps
- distinction between *hardware* and *software*
- a *universal* machine: a machine that can execute arbitrary programs
- a *programming language*–notation to describe algorithms

Leibniz's Dream

An Amazing Dream: a universal mathematical language, *lingua characteristica universalis*, in which all human knowledge can be expressed, and calculational rules, *calculus ratiocinator*, carried out by machines, to derive all logical relationships.

> "If controversies were to arise, there would be no more need of disputation between two philosophers than between two accountants. For it would suffice to take their pencils in their hands, and say to each other: Calculemus–Let us calculate."

Getting Closer to Leibniz's Dream

Friedrich Ludwig Gottlob Frege, Begriffsschrift, 1879: a universal mathematical language – *first-order logic*

- Objects, e.g., 2
- Predicates (relationships), e.g., $2 < 3$
- Operations (functions), e.g., $2 + 3$
- Logical operations (a lá Boole), e.g., "and" ($\wedge$), "or" ($\vee$), "implies" ($\rightarrow$)
- Quantifiers, e.g., "for all" ($\forall$), "exists" ($\exists$)

Example: Aristotle' Syllogisms

- "All men are mortal"
- "For all x, if x is a man, then x is mortal"
- $(\forall x)(Man(x) \rightarrow Mortal(x))$

First-Order Logic

A formalism for specifying properties of mathematical structures, such as *graphs*, *partial orders*, *groups*, *rings*, *fields*, . . .

Mathematical Structure:

$$\mathbf{A} = (D, R_1, \ldots, R_k, f_1, \ldots, f_l),$$

- D is a non-empty set – *universe*, or *domain*
- R_i is an m-ary *relation* on D, for some m (that is, $R_i \subseteq D^m$)
- f_j is an n-ary *function* on D, for some n (that is, $f_i : D^n \rightarrow n$)

Examples

Graph $\mathbf{G} = (V, E)$

- V: nodes
- $E \subseteq V^2$: edges

Groups $\mathbf{G} = (V, \cdot)$

- V: elements
- $\cdot : V^2 \to V$: product

First-Order Logic on Graphs

Syntax:

- First-order variables: x, y, z, . . . (range over nodes)
- Atomic formulas: $E(x, y)$, $x = y$
- Formulas: Atomic Formulas $+$ Boolean Connectives ($\vee$, $\wedge$, $\neg$) $+$ First-Order Quantifiers ($\exists x$, $\forall x$)

Examples:

- "node x has at least two distinct neighbors"

$$(\exists y)(\exists z)(\neg(y = z) \land E(x, y) \land E(x, z))$$

Concept: x is *free* in the above formula, which expresses a property of nodes.

- "each node has at least two distinct neighbors"

$$(\forall x)(\exists y)(\exists z)(\neg(y = z) \land E(x, y) \land E(x, z))$$

Concept: The above is a *sentence*, that is, a formula with no free variables; it expresses a property of graphs.

Semantics of First-Order Logic

Semantics:

- First-order variables range over elements of the universes of structures
- To evaluate a formula φ, we need a graph $\mathbf{G}$ and a *binding* α that maps the free variables of φ to nodes of $\mathbf{G}$

Notation: $\mathbf{G} \models_\alpha \varphi(x_1, \ldots, x_k)$

Fundamental Distinction: Syntax vs. semantics (Tarski, 1930)

From Model Theory to Relational Databases

- A sentence ψ is either true or false on a given graph $\mathbf{G}$. In particular, sentences specify classes of graphs: $\text{models}(\psi) = \{\mathbf{G} : \mathbf{G} \models \psi\}$

  Model Theory: Logic provides a metatheory for mathematical modeling.

- E.F. Codd, 1970: Formulas $\varphi(x_1, \ldots, x_k)$ define *queries*:
  $\varphi(\mathbf{G}) = \{\langle \alpha(x_1), \ldots, \alpha(x_k)\rangle : \mathbf{G} \models_\alpha \varphi(x_1, \ldots, x_k)\}$

  Example: $(\exists y)(\exists z)(\neg(y = z) \wedge E(x, y) \wedge E(x, z))$ – "List nodes that have at least two distinct neighbors"

  Relational Databases: \$30B+ industry

Relational Databases

Codd's Two Fundamental Ideas:

- *Tables are relations*: a *row* in a table is just a *tuple* in a relation; order of rows/tuples does not matter!

- *Formulas are queries*: they specified the *what* rather then the *how* – declarative programming!

Algorithmic Problems in First-Order Logic

Truth-Evaluation Problem (Model Checking): Given a first-order formula $\varphi(x_1,\ldots,x_k)$, a graph $\mathbf{G}$, and a binding α, does $\mathbf{G} \models_\alpha \varphi(x_1,\ldots,x_k)$?

Satisfiability Problem: Given a first-order formula ψ, is there a graph $\mathbf{G}$ and binding α, such that $\mathbf{G} \models_\alpha \psi$?

Facts:

- Satisfiability is *undecidable*.
- Truth evaluation, which is query evaluation, is *decidable*.

Beyond First-Order Logic

Fagin, 1976: graph connectivity is *not expressible* in first-order logic!

- There *is no* first-order formula $\varphi(x, y)$ that says there is a path in graph G from node x to node y.

Aho&Ullman, 1980: Augment FO with *fixpoints*.

$$\begin{aligned} Path(X,Y) &\leftarrow E(X,Y) \\ Path(X,Y) &\leftarrow Path(X,Z) \& E(Z,W) \end{aligned}$$

Aho&Ullman, 1980: FO$<$FP.

Complexity Theory

Key CS Question, 1930s: What can be mechanized?

Next Question, 1960s: How hard it is to mechanize it?

Hardness: Usage of computational resources

- *Time*
- *Space*

Complexity Hierarchy:

$$\text{LOGSPACE} \subseteq \text{PTIME} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \ \ldots$$

Nondeterminism

Intuition: "It is easier to *critic* than to *do*."

P vs NP:

PTIME: Can be *solved* in polynomial time

NPTIME: Can be *checked* in polynomial time

Example: *2-colorability* – PTIME, *3-colorability* – NP-complete

The Complexity Hierarchy

Complexity Hierarchy:
LOGSPACE $\subseteq$ NLOGSPACE $\subseteq$ PTIME $\subseteq$ NPTIME $\subseteq$ PSPACE $=$ NPSPACE $\subseteq$ EXPTIME $\subseteq$ NEXPTIME $\subseteq$ EXPSPACE $\subseteq \ldots$

Known: exponential gaps matter, e.g., LOGSPACE$<$PSPACE, PTIME$<$EXPTIME, PSPACE$<$EXPSPACE

Open: Which containment is strict?

Complexity of Relational Queries

Observation: Mismatch in Chandra&Harel, 1979

- Evaluating FO quries is PSPACE-complete.
- Evaluating FP queries is in PTIME.

V., 1981: “Perhaps the theory of relational queries is not fully developed”.

- *Needed*: complexity theory specifically for relational queries

Standard Complexity Theory

Standard Complexity Analysis – *Scaling Behavior*

- Focus on *decision* (yes/no) problems to eliminate dependence on output size.
- Measure how run time/memory usage *grows* as function of input size.

Database Context:

- Focus on Boolean (yes/no) queries to eliminate dependence on output size.
- Input size: database size *plus* query size.

Failure of Standard Complexity Theory

Difficulty:

- Typical input size is $10^9 + 100$
- Which size is more challenging? $2 \cdot 10^9 + 100$ or $10^9 + 200$?

Intuition: Database size and query size play *very different* roles! This is *not reflected* in standard complexity theory.

Relational Complexity Theory – 1982

Basic Principle: Separate the influences of data and query on complexity

- *Influence of Query*: Fix data
- *Influence of Data*: Fix query

Real-Life Motivation:

- *Census Data Analysis*: data fixed for 10 years, multiple queries
- *Technical Trading*: price-arbitrage fixed query, data changes momentarily

Separate Influence of Data and Query

A Tale of Two Complexities:

- *Query Complexity* of query language L: Fix $\mathbf{B}$, study

$$\{Q \in L : Q(\mathbf{B}) \text{ is nonempty}\}$$

- *Data Complexity* of query language L: Fix $Q \in L$, study

$$\{\mathbf{B} : Q(\mathbf{B}) \text{ is nonempty}\}$$

From Query Complexity to Expession Complexity

Observation:

- Data complexity is insensitive to syntax of queries, as queries are fixed.
- Query complexity is highly sensitive to syntax of queries, e.g.,

– $R \times R \times R \times R \times R \times R \times R \times R \times R \times R$

– R^{10}

Conclusion: Change "*Query Complexity*" to "*Expression Complexity*".

Data vs Expression Complexity

Basic phenomenon: exponential gap!

Query Lang.	Data Comp.	Expression Comp.
FO	LOGSPACE	PSPACE
FP	PTIME	EXPTIME
$\exists$SO	NP	NEXPTIME
PFP	PSPACE	EXPSPACE

Theory justifies intuition: Characteristics of queries matter much more than size of data!

Codd's Relational Algebra

Intuition: Bottom-up evaluation of queries

Relational Algebra (RA):

- $\cup, \cap, -$ correspond to $\vee, \wedge, \neg$.
- $E_1 \bowtie_{2=1} E_2$: $E_1(x, y) \wedge E_2(y, z)$
- $\Pi_1(E)$: $(\exists y)(E(x, y)$

Codd, 1971: FO$\equiv$RA

- FO$\leadsto$SQL
- RA: Foundations for SQL query evaluation

Relational Complexity Theory – 1995

Question: Why is expression complexity so high? How do databases evaluate queries in practice?

Intuitive Answer: Large intermediate results!

- How much is $1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 8 \times 9 \times 0$?

- *Example*: $R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \bowtie R_5$ can be empty, even when $R_1 \bowtie R_2 \bowtie R_3$ is very large.

Question: Can we formalize this intuition?

Answer: *Variable-confined queries*

Projection Pushing

Example: Compare two joins of *ternary* relations

- $\pi_{1,6}(R_1 \bowtie_{3=1} R_2)$ – 6-ary intermediate relation
- $\pi_{1,4}(\pi_{1,3}(R_1) \bowtie \pi_{1,3}(R_2))$ – 4-ary intermediate relation

Observations:

- *Projection pushing* in RA corresponds to *variable re-use* in FO.
- *Bounding width* of intermediate relations corresponds to *bounding number* of variables.

Variable-Confined Queries

Definition: L^k consists of formulas of logic L with at most k variables

Example: "There exists a path of length 2"

- FO^3: $(\exists x)((\exists y)(\exists z)(R(x,y) \wedge R(y,z))$
- FO^2: $(\exists x)((\exists y)(R(x,y) \wedge (\exists x)R(y,x))$

Key Result: Variable-confined queries have lower expression complexity!

Query Lang.	Data Compl.	Expression Comp.	VC Expr. Comp.
FO	LOGSPACE	PSPACE	PTIME
FP	PTIME	EXPTIME	PTIME
∃SO	NP	NEXPTIME	NP
PFP	PSPACE	EXPSPACE	PSPACE

Variable-Confined Queries Are Easier

Conclusion: Exponential gap between data complexity and expression complexity *shrinks or vanishes* for variable-confined queries.

Optimization Problem: Find smallest k such that given FO query Q in is FO^k.

Answer: Undecidable!

Conjunctive Queries

Conjunctive Query: First-order logic without $\forall, \vee, \neg$; written as a rule
$Q(X_1, \ldots, X_n) \ :- \ R(X_3, Y_2, X_4), \ldots, S(X_2, Y_3)$

Significance: most *common* SQL queries (*Select-Project-Join*)

Example: $GrandParent(X, Y) \ :- \ Parent(X, Z), Parent(Z, Y)$

Equivalently: $(\exists Z)(Parent(X, Z) \wedge Parent(Z, Y))$

Complexity of Conjunctive Queries

Chandra&Merlin, 1977: Expression complexity of CQ is NP-complete.

Precise Complexity Analysis: $||B|^{||Q||}$, for evaluating query Q, over database B.

Yannakakis, 1995: $||B||^{||Q||}$ is much worse than $c^{||Q||} \cdot ||B|^d$ for fixed c, d, which is *fixed-parameter tractable* (FPT) – **parameterized complexity analysis**

Papadimitriou&Yannakakis, 1997: CQ evaluation is *W[1]-complete* – unlikely to be FPT.

Variable-Confined CQ

V., 1995: CQ^k – CQ using at most k variables.

- If Q is in CQ^k, then query can be evaluated over database B in time $||Q|| \cdot ||B||^k$ - FPT!

Example: Contrast

$$(\exists x, y, z)(R(x, y) \wedge R(y, z))$$

and

$$(\exists x)((\exists y)(R(x, y) \wedge (\exists x)R(y, x))$$

Hardness of CQs

Observation: The critical parameter is *number of variables*, *not size of query*!

Question: Characterize smallest k such that a given conjunctive query Q is in CQ^k.

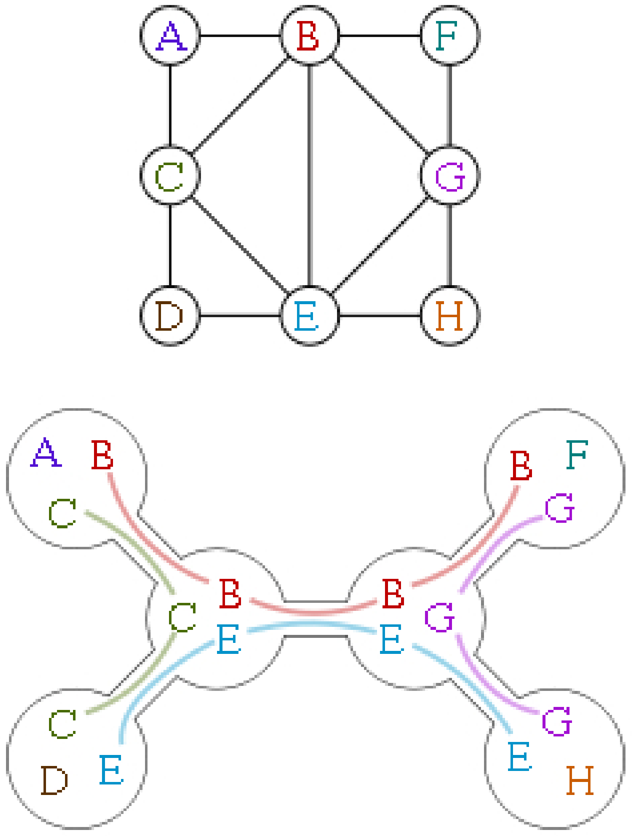

Figure 1: Tree Decomposition of Width 2

Treewidth

Treewidth: "width" of best tree decomposition – measures "tree-likeness" of graphs

- A tree has treewidth 1.
- A cycle has treewith 2.
- An $m \times m$ grid has treewidth m.

CQs Treewidth

Query Graph: graph of a conjunctive query

- *Nodes*: variables
- *Edges*: connect nodes that co-occur in an atom

Definition: $treewidth(Q)$ is $treewidth(graph(Q))$.

Kolaitis&V., 1998: Q is in CQ^k iff $treewidth(Q) < k$.

Corollary: Bounded treewidth CQs are fixed-parameter tractable.

Theory and Practice

Question: Can theory be used to optimize CQs?

Partial Answer: Not easily!

- Finding treewidth of a graph is NP-hard!

CQ Evaluation I

Hard problem for databases: evaluation of large conjunctive queries

- Corresponds to evaluating a long sequence of joins and projections.
- Many possible evaluation orders possible.
- Query optimizer has to search a very large space.

CQ Evaluation II

An Alternative Approach: (McMahan&V., 2004)

- Consider the problem as a constraint-satisfaction problem (CSP).
- Apply CSP heuristics for constraint propagation.
- Minize the size of intermediate relations via treewdith minimization
- Essentialy, minimize number of variables, *heuristically*.

Question: Does it work?

Answer: Exponential improvement for large CQs.

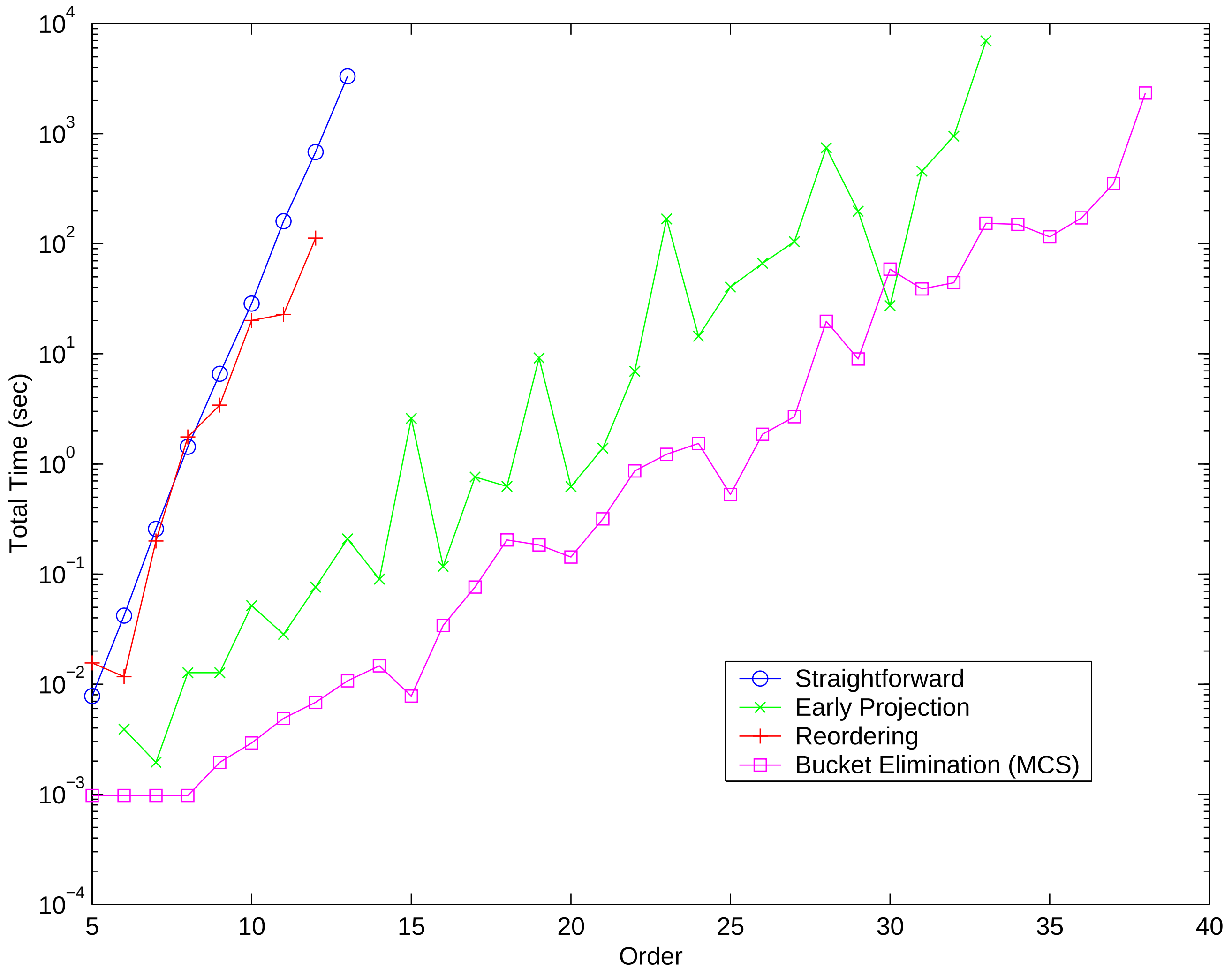

Figure 2: Experimental Results

In Conclusion

Role of Theory:

- Clarify conceptual framework
- Suggest experimental possibilties

Paradigmatic Example: Codd's Relational model

This Talk:

- *Conceptual Framework*: data and expression complexity
- *Optimization Heuristics*: treewidth minimization