CMPS 277 – Principles of Database Systems

https://courses.soe.ucsc.edu/courses/cmps277/Fall11/01

Lecture #7
Three Fundamental Algorithmic Problems about Queries

- **The Query Evaluation Problem**: Given a query $q$ and a database instance $I$, find $q(I)$.

- **The Query Equivalence Problem**: Given two queries $q$ and $q'$ of the same arity, is it the case that $q \equiv q'$?
  (i.e., is it the case that, for every database instance $I$, we have that $q(I) = q'(I)$?)

- **The Query Containment Problem**: Given two queries $q$ and $q'$ of the same arity, is it the case that $q \subseteq q'$?
  (i.e., is it the case that, for every database instance $I$, we have that $q(I) \subseteq q'(I)$?)
Undecidability of the Query Containment Problem

- The Query Equivalence Problem for relational calculus is **undecidable**
  - The same holds true for relational algebra

- The Query Containment Problem for relational calculus is **undecidable**.
  - The same holds true for relational algebra.

**Proof Technique:** The Reduction Method
- Finite Validity $\preceq$ Query Equivalence $\preceq$ Query Containment.

- The Query Evaluation Problem for relational calculus is **decidable**, but is of **high** computational complexity.
  - The same holds true for relational algebra.
The Five Basic Computational Complexity Classes

- **LOGSPACE (or, L):** All decision problems solvable by a TM using extra memory bounded by a logarithmic amount in the input size.

- **NLOGSPACE (or, NL):** All decision problems solvable by a NTM using extra memory bounded by a logarithmic amount in the input size.

- **P (or, PTIME):** All decision problems solvable by a TM in time bounded by some polynomial in the input size.

- **NP:** All decision problems solvable by a NTM in time bounded by some polynomial in the input size.

- **PSPACE:** All decision problems solvable by a TM using memory bounded by a polynomial in the input size.
The Five Basic Computational Complexity Classes

Theorem:

- The following inclusions hold:
  \[ \text{LOGSPACE} \subseteq \text{NLOGSPACE} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE}. \]

- Moreover, it is known that \( \text{LOGSPACE} \subset \text{PSPACE} \).

- No other proper inclusion between these classes is known at present. In particular, it is \textbf{not} known whether \( \text{P} = \text{NP} \).

- We can get insights about a complexity class by understanding and analyzing its \textit{complete problems} (provided they exist).
  - Complete problems are the “\textit{hardest}” problems in the class
  - Complete problems are the most “\textit{representative}” problems.
Complete Problems for Complexity Classes

- **Definition:** A decision problem $Q$ is **PSPACE-complete** if
  - $Q$ is in PSPACE.
  - If $Q'$ is in PSPACE, then there is a polynomial-time computable function $f$ such that for every string $x$, we have that
    \[ x \in Q' \iff f(x) \in Q. \]

- **Definition:** A decision problem $Q$ is **NP-complete** if
  - $Q$ is in NP.
  - If $Q'$ is in NP, then there is a polynomial-time computable function $f$ such that for every string $x$, we have that
    \[ x \in Q' \iff f(x) \in Q. \]
  - Such an $f$ is a **polynomial-time reduction** of $L$ to $L^*$ ($L \leq_p L^*$)
Complete Problems for Computational Complexity Classes

- **PSPACE-complete:**
  - **Quantified Boolean Formulas (QBF):** Given a quantified Boolean formula $\forall x_1 \exists x_2 \ldots \forall x_k \varphi$, is it true?

**Examples:**
- $\forall x_1 \exists x_2 ((x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2))$
  - Is true.
  - If $x_1 = 1$, then put $x_2 = 0$.
  - If $x_1 = 0$, then put $x_2 = 1$.

- $\forall x_1 \exists x_2 ((x_1 \lor x_2) \land (x_1 \lor \neg x_2))$
  - Is false.
  - If $x_1 = 1$, then we put $x_2 = 0$.
  - If $x_1 = 0$, then the formula evaluates to false, no matter what $x_2$ is.
Complete Problems for Computational Complexity Classes

- **NP-complete:**
  - **Satisfiability (SAT):** Given a CNF formula $\varphi$, is it satisfiable?

  **Note:** SAT is a special case of QBF (Why?)

  - **3-Colorability:** Given a graph $G=(V,E)$, is it 3-colorable?

  - **Integer Linear Inequalities (ILI):** Given a system of linear inequalities with integer coeffs., does it have an integer solution?

  - ...  

  **Note:** Thousands of NP-complete problems are known by now.
Proposition: Let $C$ and $C'$ be one of the complexity classes NLOGSPACE, P, NP, PSPACE such that $C \subseteq C'$ and let $Q$ be a $C'$-complete problem. Then the following statements are equivalent:

- $C = C'$.
- $Q$ is in $C$.

In particular, the following statements are equivalent:

- P = NP.
- SAT is in P
- Your favorite NP-complete problem is in P.

Conclusion:

- Complete problems hold the secret of whether or not a higher computational complexity class collapses to a lower one.
- Showing that a decision problem $Q$ is NP-complete provides strong evidence that $Q$ is not in P.
Complexity of the Query Evaluation Problem

- **The Query Evaluation Problem for Relational Calculus:**
  Given a relational calculus formula $\varphi$ and a database instance $I$, find $\varphi^{\text{dom}}(I)$.

- **The Query Evaluation Problem for Relational Algebra:**
  Given a relational algebra expression $E$ and a database instance $I$, find $E(I)$.

- **Theorem:** The Query Evaluation Problem for Relational Calculus is PSPACE-complete.

- **Corollary:** The Query Evaluation Problem for Relational Algebra is PSPACE-complete.
Complexity of the Query Evaluation Problem

- **Theorem:** The Query Evaluation Problem for Relational Calculus is PSPACE-complete.

  **Proof:** We need to show that

  - This problem is in PSPACE (i.e., give a PSPACE-algorithm for it).
  - This problem is PSPACE-hard.

  We start with the second task.
Complexity of the Query Evaluation Problem

- **Theorem:** The Query Evaluation Problem for Relational Calculus is PSPACE-hard.
- **Proof:** Show that

Quantified Boolean Formulas $\preceq_p$ Query Evaluation for Rel. Calc.

Given QBF $\forall x_1 \exists x_2 \ldots \forall x_k \psi$

- Let V and P be two unary relation symbols
- Obtain $\psi^*$ from $\psi$ by replacing $x_i$ by $P(x_i)$, and $\neg x_i$ by $\neg P(x_i)$
- Let I be the database instance with $V = \{0,1\}$, $P = \{1\}$.
- Then the following statements are equivalent:
  - $\forall x_1 \exists x_2 \ldots \forall x_k \psi$ is true
  - $\forall x_1 (V(x_1) \rightarrow \exists x_2 (V(x_2) \wedge (\ldots \forall x_k (V(x_k) \rightarrow \psi^*)) \ldots ))$ is true on I.
Complexity of the Query Evaluation Problem

- **Theorem:** The Query Evaluation Problem for Relational Calculus is in PSPACE.
  
  **Proof (Hint):** Let \( \varphi \) be a relational calculus formula \( \forall x_1 \exists x_2 \ldots \forall x_m \psi \) and let I be a database instance.
  
  - **Exponential Time Algorithm:** We can find \( \varphi^{\text{dom}(I)} \), by exhaustively cycling over all possible interpretations of the \( x_i \)’s. This runs in time \( O(n^m) \), where \( n = |I| \) (size of I).
  
  - A more careful analysis shows that this algorithm can be implemented in \( O(m \cdot \log n) \)-space.
    
    - Use \( m \) blocks of memory, each holding one of the \( n \) elements of \( \text{dom}(I) \) written in binary (so \( O(\log n) \) space is used in each block).
    
    - Maintain also \( m \) counters in binary to keep track of the number of elements examined.

<table>
<thead>
<tr>
<th>( \forall x_1 )</th>
<th>( \exists x_2 )</th>
<th>\ldots</th>
<th>( \forall x_m )</th>
</tr>
</thead>
</table>
| \( a_1 \) in \( \text{dom}(I) \)
  written in binary |
| \( a_2 \) in \( \text{dom}(I) \)
  written in binary |
| \ldots |
| \( a_m \) in \( \text{dom}(I) \)
  written in binary |
Corollary: The Query Evaluation Problem for Relational Algebra is PSPACE-complete.

Proof: The translation of relational calculus to relational algebra yields a polynomial-time reduction of the Query Evaluation Problem for Relational Calculus to the Query Evaluation Problem for Relational Algebra.
Summary

- The Query Evaluation Problem for Relational Calculus is PSPACE-complete.

- The Query Equivalence Problem for Relational Calculus is undecidable.

- The Query Containment Problem for Relational Calculus is undecidable.

- Similar results hold for the Relational Algebra version of these problems.
The Query Evaluation Problem Revisited

- Since the Query Evaluation Problem for Relational Calculus is PSPACE-hard, there are no polynomial-time algorithms for this problem, unless PSPACE = P (which is considered highly unlikely).
- Let’s take another look at the exponential-time algorithm for this problem:
  - Let $\varphi$ be a relational calculus formula $\forall x_1 \exists x_2 \ldots \forall x_m \psi$ and let $I$ be a database instance.
  - **Exponential Time Algorithm:** We can find $\varphi^{\text{adom}}(I)$, by exhaustively cycling over all possible interpretations of the $x_i$’s. This runs in time $O(n^m)$, where $n = |I|)$.
  - So, the running time is $O(|I|^{|\varphi|})$, where $|I|$ is the size of $I$ and $|\varphi|$ is the size of the relational calculus formula $\varphi$.
  - This tells that the source of exponentiality is the formula size.
The Query Evaluation Problem Revisited

- **Theorem:** Let $\varphi$ be a fixed relational calculus formula. Then the following problem is solvable in polynomial time: given a database instance $I$, find $\varphi^{\text{adom}}(I)$. In fact, this problem is in LOGSPACE.

- **Proof:** Let $\varphi$ be a fixed relational calculus formula $\forall x_1 \exists x_2 \ldots \forall x_m \psi$
  - The previous algorithm has running time $O(|I|^{|\varphi|})$, which is a polynomial, since now $|\varphi|$ is a constant.
  - Moreover, the algorithm can now be implemented using logarithmic-space only, since we need only maintain a constant number of memory blocks, each of logarithmic size.

<table>
<thead>
<tr>
<th>$\forall x_1$</th>
<th>$\exists x_2$</th>
<th>$\ldots$</th>
<th>$\forall x_m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$ in $\text{adom}(I)$ written in binary</td>
<td>$a_2$ in $\text{adom}(I)$ written in binary</td>
<td>$\ldots$</td>
<td>$a_m$ in $\text{adom}(I)$ written in binary</td>
</tr>
</tbody>
</table>
Vardi’s Taxonomy of the Query Evaluation Problem

M.Y Vardi, “The Complexity of Relational Query Languages”, 1982

- **Definition:** Let L be a database query language.
  - The **combined complexity of L** is the decision problem:
    given an L-sentence \( \varphi \) and a database instance I, is \( \varphi \) true on I? (does I satisfy \( \varphi \)?) (in symbols, does I \( \models \varphi \)?)
  
  - The **data complexity of L** is the family of the following decision problems \( Q_{\varphi} \), where \( \varphi \) is an L-sentence:
    given a database instance I, does I \( \models \varphi \)?
  
  - The **query complexity of L** is the family of the following decision problems \( Q_{I} \), where I is a database instance:
    given an L-sentence \( \varphi \), does I \( \models \varphi \)?
Vardi’s Taxonomy of the Query Evaluation Problem

**Note:** Let L be a database query language

- The input to the combined complexity problem consists of two parts: an L-sentence and a database instance.

- The input to a member of the data complexity of L consists of a database instance only (the L-sentence is fixed).
  - Hence, the data complexity of L is a special case of the combined complexity of L.

- The input to a member of the query complexity of L consists of an L-sentence only (the database instance is fixed).
  - Hence, the query complexity of L is a special case of the combined complexity of L.
**Vardi’s Taxonomy of the Query Evaluation Problem**

- **Definition:** Let $L$ be a database query language and let $C$ be a computational complexity class.
  - The **data complexity of $L$ is in $C$** if for each $L$-sentence $\varphi$, the decision problem $Q_\varphi$ is in $C$.
  - The **query complexity of $L$ is in $C$** if for every database instance, the decision problem $Q_I$ is in $C$.

- **Vardi’s discovery:**
  For most query languages $L$:
  - The data complexity of $L$ is of lower complexity than both the combined complexity of $L$ and the query complexity of $L$.
  - The query complexity of $L$ can be as hard as the combined complexity of $L$. 
The Query Evaluation Problem for Relational Calculus

<table>
<thead>
<tr>
<th>Problem</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combined Complexity</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>Query Complexity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▪ Is in PSPACE</td>
</tr>
<tr>
<td></td>
<td>▪ It can be PSPACE-complete</td>
</tr>
<tr>
<td>Data Complexity</td>
<td>In LOGSPACE</td>
</tr>
</tbody>
</table>

Computational Complexity Classes:

- LOGSPACE
- NLOGSPACE
- \( P \)
- NP
- \( \text{PSPACE} \)
The Query Evaluation Problem for Relational Calculus

- **Paradox:**
  - The Query Evaluation Problem for Relational Calculus has very high combined complexity (PSPACE-complete, so “harder” than NP-complete).
  - Yet, database systems evaluate SQL queries “efficiently”.

- **Resolution of the Paradox:**
  - In practice, we deal with the data complexity of the Query Evaluation Problem for Relational Calculus, because we typically have a small fixed collection of queries to answer (while of course the database instances vary).
  - The data complexity of the Query Evaluation Problem for Relational Calculus is in LOGSPACE (hence, in PTIME); so, in principle, it is a tractable problem.
Question: Are there interesting sublanguages of relational calculus for which the Query Containment Problem and the Query Evaluation Problem are “easier” than the full relational calculus?

Answer:
- Yes, the language of conjunctive queries is such a sublanguage.
- Moreover, conjunctive queries are the most frequently asked queries against relational databases.
Conjunctive Queries

**Definition:** A conjunctive query is a query expressible by a relational calculus formula in prenex normal form built from atomic formulas $R(y_1,\ldots,y_n)$, and $\land$ and $\exists$ only.

$$\{ (x_1,\ldots,x_k) : \exists z_1 \ldots \exists z_m \chi(x_1, \ldots, x_k, z_1,\ldots, z_k) \}$$

where $\chi(x_1, \ldots, x_k, z_1,\ldots, z_k)$ is a conjunction of atomic formulas of the form $R(y_1,\ldots,y_m)$.

- Equivalently, a conjunctive query is a query expressible by a relational algebra expression of the form
  $$\pi_X(\sigma_{\Theta}(R_1 \times \ldots \times R_n)),$$
  where
  $\Theta$ is a conjunction of equality atomic formulas (equijoin).

- Equivalently, a conjunctive query is a query expressible by an SQL expression of the form
  
  SELECT <list of attributes>
  FROM <list of relation names>
  WHERE <conjunction of equalities>
**Conjunctive Queries**

- **Definition:** A conjunctive query is a query expressible by a relational calculus formula in prenex normal form built from atomic formulas $R(y_1,\ldots,y_n)$, and $\land$ and $\exists$ only.
  \[
  \{ (x_1,\ldots,x_k) : \exists z_1 \ldots \exists z_m \chi(x_1, \ldots, x_k, z_1, \ldots, z_k) \} 
  \]

- A conjunctive query can be written as a **logic-programming rule**:

  \[
  Q(x_1,\ldots,x_k) :- R_1(u_1), \ldots, R_n(u_n), \text{ where} 
  \]

  - Each variable $x_i$ occurs in the right-hand side of the rule.
  - Each $u_i$ is a tuple of variables (not necessarily distinct)
  - The variables occurring in the right-hand side (the body), but not in the left-hand side (the head) of the rule are existentially quantified (but the quantifiers are not displayed).
  - “,$,” stands for conjunction.
Conjunctive Queries

Examples:

- **Path of Length 2**: (Binary query)
  \[
  \{ (x,y) : \exists z \ (E(x,z) \land E(z,y)) \}
  \]
  - As a relational algebra expression,
    \[
    \pi_{1,4}(\sigma_{2 = 3} (E \times E))
    \]
  - As a rule:
    \[
    q(x,y) : \leftarrow \ E(x,z), E(z,y)
    \]

- **Cycle of Length 3**: (Boolean query)
  \[
  \exists x \exists y \exists z (E(x,y) \land E(y,z) \land E(z,x))
  \]
  - As a rule (the head has no variables)
    - Q : \leftarrow E(x,z), E(z,y), E(z,x)
Every relational join is a conjunctive query:
P(A,B,C), R(B,C,D) two relation symbols

- \( P \bowtie R = \{ (x,y,z,w) : P(x,y,z) \land R(y,z,w) \} \)

- \( q(x,y,z,w) :-- P(x,y,z), R(y,z,w) \)
  (no variables are existentially quantified)

- SELECT P.A, P.B, P.C, R.D
  FROM P, R
  WHERE P.B = R.B AND P.C = R.C

Conjunctive queries are also known as **SPJ-queries**
(SELECT-PROJECT-JOIN queries)
Conjunctive Query Evaluation and Containment

- **Definition:** Two fundamental problems about CQs
  - **Conjunctive Query Evaluation** (CQE):
    Given a conjunctive query $q$ and an instance $I$, find $q(I)$.
  - **Conjunctive Query Containment** (CQC):
    - Given two $k$-ary conjunctive queries $q_1$ and $q_2$, is it true that $q_1 \subseteq q_2$? (i.e., for every instance $I$, we have that $q_1(I) \subseteq q_2(I)$)
    - Given two Boolean conjunctive queries $q_1$ and $q_2$, is it true that $q_1 \models q_2$? (that is, for all $I$, if $I \models q_1$, then $I \models q_2$)?
    CQC is **logical implication**.
CQE vs. CQC

- Recall that for relational calculus queries:
  - The Query Evaluation Problem is PSPACE-complete (combined complexity).
  - The Query Containment Problem is undecidable.

- **Theorem**: Chandra & Merlin, 1977
  - CQE and CQC are the “same” problem.
  - Moreover, each is an NP-complete problem.

- **Question**: What is the common link?
- **Answer**: The Homomorphism Problem