CMPS 277 – Principles of Database Systems

https://courses.soe.ucsc.edu/courses/cmgs277/Fall11/01

Lecture #6
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 $q(I) \subseteq q'(I)$?)
Three Fundamental Algorithmic Problems about Queries

- **The Query Evaluation Problem** is the main problem in query processing.

- **The Query Equivalence Problem** underlies query processing and optimization, as we often need to transform a given query to an equivalent one.

- **The Query Containment Problem** and the **Query Equivalence Problem** are *closely related* to each other:
  1. $q \equiv q'$ if and only if $q \subseteq q'$ and $q' \subseteq q$.
  2. $q \subseteq q'$ if and only if $(q \land q') \equiv q$.  

  (Why?)
Theorem: The following problems are undecidable:

- **The Halting Problem (A. Turing – 1936):** Given a Turing machine M and an input x, does M halt on x?

- **The Finite Validity Problem (B. Trakhtenbrot – 1949):** Given a first-order sentence $\varphi$ on graphs, is $\varphi$ true on every finite graph?

- **Hilbert’s 10th Problem (Y. Matijacevic – 1971):** Given a multivariate polynomial $p(x_1,\ldots,x_n)$ with integer coefficients, does $p(x_1,\ldots,x_n)$ have an all-integers solution?
The Reduction Method

By now there is a vast library of undecidable problems.

The Reduction Method is the main technique for establishing undecidability.

**Reduction Method:** To show that a language $L^*$ is not recursive, it suffices to find a non-recursive language $L$ and a total Turing computable function $f$ such that for every string $x$, we have that

$$x \in L \iff f(x) \in L^*.$$ 

- Such a function $f$ is called a reduction of $L$ to $L^*$
- $L \preceq L^*$ means that there is a reduction of $L$ to $L^*$. 

The Reduction Method

- The Halting Problem was the first fundamental decision problem shown to be undecidable.

- The Finite Validity Problem was shown to be undecidable by showing that Halting Problem $\preceq$ Finite Validity Problem.

- Many database problems have been shown to be undecidable via reductions from one of the following problems:
  - The Halting Problem
  - The Finite Validity Problem
  - Hilbert’s 10th Problem.

- In particular, Di Paola proved that the Domain Independence Problem for relational calculus formulas is undecidable by showing that Finite Validity Problem $\preceq$ Domain Independence.
Undecidability of The Query Equivalence Problem

- **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 $q(I) = q'(I)$ on every database instance I?)

- **Theorem**: The Query Equivalence Problem for relational calculus queries is undecidable.
  
  **Proof**: Finite Validity Problem $\preceq$ Query Equivalence Problem
  
  - Then, for every relational calculus sentence $\varphi$, we have that $\varphi$ is finitely valid $\iff \varphi \equiv \psi^*$.

  - To see, this let $\psi^*$ be a fixed finitely valid relational calculus sentence (say, $\forall x (E(x,x) \rightarrow \exists y E(x,y))$).

- **Corollary**: The Query Equivalence Problem for relational algebra queries is undecidable.
Undecidability of the Query Containment Problem

- **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 \( q(I) \subseteq q'(I) \) on every database instance \( I \)?)

- **Corollary:** The Query Containment Problem for relational calculus queries is undecidable.
  **Proof:** Query Equivalence \( \preceq \) Query Containment, since
  \[ q \equiv q' \iff q \subseteq q' \text{ and } q' \subseteq q. \]

- **Notice the chain of reductions:**
  Halting Problem \( \preceq \) Finite Validity \( \preceq \) Query Equiv. \( \preceq \) Query Cont.
The Query Evaluation Problem

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

- The Query Evaluation Problem for relational calculus queries is **decidable**, but, as we will see, it has **high computational complexity**.

- To understand the precise algorithmic difficulty of the Query Evaluation Problem, we need some basic notions and results from computational complexity.
Decidable Problems and Computational Complexity

- **Computational Complexity** is the quantitative study of decidable problems.

- “From these and other considerations grew our deep conviction that there must be quantitative laws that govern the behavior of information and computing. The results of this research effort were summarized in our first paper on this topic, which also named this new research area, "On the computational complexity of algorithms".”

  J. Hartmanis, Turing Award Lecture, 1993
Decidable problems are grouped together in computational complexity classes.

Each computational complexity class consists of all problems that can be solved in a computational model under certain restrictions on the resources used to solve the problem.

Examples of computational models:
- Turing Machine TM (deterministic Turing machine)
- Non-deterministic Turing machine NTM
- ...

Examples of resources:
- Amount of time needed to solve the problem
- Amount of space (memory) needed to solve the problem.
- ...
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 \( \text{LOGSPACE} \subset \text{PSPACE} \).
- \textbf{No} other proper inclusion between these classes is known at present. In particular, it is \textbf{not} known whether \( \text{P} = \text{NP} \).

Note:
- The question: “is \( \text{P} = \text{NP} \)” is the central open problem in computational complexity.
- It is one of the Millennium Prize Problems – see \url{http://www.claymath.org/millennium/}
There are many other complexity classes. For a comprehensive catalog, visit the Complexity Zoo at qwiki.stanford.edu/wiki/Complexity_Zoo
Complete Problems

- A key property of most complexity classes is that they possess complete problems.

- Intuitively, complete problems are the “hardest” problems in the class in the sense that every other problem can be reduced to it.

- **Definition:** Let $C$ be a complexity class. A decision problem $Q$ is $C$-complete if
  - $Q$ is in $C$.
  - If $Q'$ is in $C$, then there is a “suitable” total Turing computable function $f$ such that for every string $x$, we have that
    \[ x \in Q' \iff f(x) \in Q. \]

  - “Suitable” means that $f$ can be computed with fewer resources than those used to define $C$.
  - So, $f$ is a reduction of a restricted nature.
Complete Problems and Reductions

<table>
<thead>
<tr>
<th>Complexity Class</th>
<th>Reductions for Complete Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSPACE</td>
<td>Polynomial-time computable</td>
</tr>
<tr>
<td>NP</td>
<td>Polynomial-time computable</td>
</tr>
<tr>
<td>P</td>
<td>Logspace-computable</td>
</tr>
<tr>
<td>NL</td>
<td>Logspace-computable</td>
</tr>
</tbody>
</table>

**Definition:** Let $C$ be a complexity class. A decision problem $Q$ is $C$-complete if

- $Q$ is in $C$.
- If $Q'$ is in $C$, then there is a “suitable” total Turing computable function $f$ such that for every string $x$, we have that
  \[ x \in Q' \iff f(x) \in Q. \]

“Suitable” means that $f$ can be computed with fewer resources than those used to define $C$. 
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.
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?

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

- **P-complete:**
  - **Horn SAT:** Given a Horn CNF formula $\varphi$, is it satisfiable?
  - **Linear Inequalities (LI):** Given a system of linear inequalities with integer coefficients, does it have a rational solution?

- **NL-complete:**
  - **Directed Graph Reachability:** Given a directed graph $G=(V,E)$ and two nodes $s$ and $t$, is there a path from $s$ to $t$?
Polynomial-Time Reductions

- **3-Satisfiability (3SAT):** Given a 3CNF formula \( \varphi \), is it satisfiable? (each clause has at most 3 literals)

- **Theorem:** 3SAT is NP-complete
  
  **Proof:** Show that SAT \( \leq_p \) 3SAT

  - Let \( \varphi \) be a CNF formula \( c_1 \land c_2 \land \ldots \land c_m \)
  - If a clause \( c_i \) has more than three literals, then we replace it with a set of clauses each with three literals and certain new variables.
  - For example, if \( c_i \) is \((x_1 \lor \neg x_2 \lor x_3 \lor x_4 \lor x_5)\), then we replace \( c_i \) by \((x_1 \lor \neg x_2 \lor y_1), (\neg y_1 \lor x_3 \lor y_2), (\neg y_2 \lor x_4 \lor x_5)\).
  - Let \( \varphi^* \) be the resulting 3CNF formula. Then
    \[
    \varphi \text{ is satisfiable } \iff \varphi^* \text{ is satisfiable (check this).}
    \]
Another Perspective on the classes P and NP

- Intuitively, P is the class of all decision problems for which we can find a “solution” efficiently (where “efficiently” means in time bounded by a polynomial in the size of the input).

- Intuitively, NP is the class of all decision problems for which we can check efficiently whether a “candidate solution” is indeed a “solution”.
  (“We can guess a “solution” and verify that it is indeed a solution in polynomial time”)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Candidate Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT</td>
<td>An assignment of Boolean values to the variables</td>
</tr>
<tr>
<td>3-Colorability</td>
<td>An assignment of colors B, R, G to the nodes.</td>
</tr>
</tbody>
</table>
The “P = NP?” question is equivalent to the following:

**Question:**
Is it true that every decision problem for which a “candidate solution” can be verified efficiently has the property that a “solution” can also be found efficiently?

- In particular, this means that the “candidate solution” must be “small” (bounded by a polynomial in the size of the input).

**Note:**
The prevailing belief is that the answer to the above question is “No”, which means that P ≠ NP.
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.
The Query Evaluation Problem for Relational Calculus:
Given a relational calculus formula $\varphi$ and a database instance $I$, find $\varphi^{\text{adom}}(I)$.

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.