CS 838: Foundations of Data Management **Spring 2016**

Lecture 3: The Complexity of Relational Queries

Instructor: Paris Koutris 2016

When we study a query language $\mathcal{L}$ from a theoretical perspective, we typically focus on the following two questions:

1. How expressive is $\mathcal{L}$, i.e. what properties can we express using the language?
2. What is the complexity of evaluating a query from $\mathcal{L}$?

In this lecture we examine the complexity of evaluating queries, and in particular conjunctive queries, and relational algebra queries. Before that, we will start with a brief revision of basic complexity concepts.

3.1 A Short Introduction to Complexity

When we refer to the complexity of a problem, we typically talk about a *decision problem*: given some input $\mathbf{x}$ of size n in bits, we have to decide whether the output is yes or no.

3.1.1 Basic Complexity Classes

We will be mainly interested in the following complexity classes: $\mathsf{AC}^0, \mathsf{L}, \mathsf{NC}, \mathsf{NL}, \mathsf{P}, \mathsf{NP}, \mathsf{PSPACE}$, which we discuss in more detail next.

Definition 3.1 (NC)**.** *NC^i is the class of languages solved using circuits of depth $O(\log^i n)$ and a polynomial number of AND, OR, NOT gates of fan-in ≤ 2.*

NC stands for Nick's Class. Some problems that are in NC: integer multiplication and addition, matrix multiplication.

Exercise 3.2. *Consider the problem* PARITY*: given a string with n bits as input, decide whether the string contains an even number of 1's or not. Show that* PARITY *is in NC^1.*

A similar class to NC is the class AC.

Definition 3.3 (AC)**.** *The class AC^i is the same as NC^i, but the AND, OR gates are allowed to have unbounded fan-in (i.e. as many inputs as we want).*

One can easily see that $\mathsf{NC}^i \subseteq \mathsf{AC}^i \subseteq \mathsf{NC}^{i+1}$ (prove as an exercise why this is true). It thus holds that $\mathsf{NC} = \mathsf{AC}$. The smallest class of AC is the class AC^0, which consists of circuits of constant depth, unbounded fan-in and polynomially many gates. The corresponding class NC^0 is not very interesting, since it captures only constant functions.

Example 3.4. *Construct an* AC^0 *circuit that, given a string of n bits as input, decides whether it contains at least two 1's or not.*

Why are we interested in the classes NC, AC^0? I turns out that we can equivalently describe NC^i as the class of decision problems solvable in time $O(\log^i n)$ on a parallel computer with a polynomial number of processors. The idea is that the gates at each level of the circuit can in parallel evaluate their outputs, and thus the running time will be equal to the depth of the circuit. Hence, NC can be thought as the class of problems that can be solved efficiently on a parallel computer. It is easy to see than that AC^0 is the "easiest" complexity class that we can parallelize, since we need only a circuit of constant depth! In other words, AC^0 is a very weak complexity class, which means that many problems can not be expressed in it. For example, it can be shown that PARITY is not expressible in AC^0. Another problem that is not expressible in AC^0 is the graph reachability problem STCON: given a directed graph, and two nodes s, t, is there a directed path from s to t?

Definition 3.5 (Logarithmic Space)**.** *A problem is in* L *if it can be computed by a deterministic Turing machine using* $O(\log n)$ *space. A problem is in* NL *if it can be computed by a non-deterministic Turing machine using* $O(\log n)$ *space.*

It is easy to see that STCON is in NL. In particular, it is NL-complete. Here, we have to be very careful on how we define a reduction to prove completeness for L or NL, since the reduction must be a logarithmic-space reduction (and not a polynomial time). Another NL-complete problem is 2-satisfiability (satisfiability for a SAT formula where each clause has two variables). A complete problem for L is the undirected connectivity problem USTCON: given an undirected graph, is there a path from node s to node t? What is the relation between L, NL and the logarithmic space classes?

$$\mathsf{AC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{NC}^2$$

Definition 3.6 (PSPACE)**.** *A problem is in* PSPACE *if it can be computed by a Turing machine using a polynomial amount of space.*

Including deterministic polynomial (P) and non-deterministic polynomial (NP), the hierarchy of the aforementioned complexity classes is as follows:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE}.$$

3.1.2 Complexity for Queries

We now turn our attention to how complexity is defined for evaluating queries in the context of databases. Recall that we so far talked about decision problems, where the answer is yes or no. In

the case of a boolean query, the correspondence to a decision problem is straightforward. In the case of non-boolean queries, we will distinguish between two different problems: (a) computing the full query result (all tuples in the output), and (b) computing the decision problem of whether a tuple t belongs in the query output.

Exercise 3.7. *Suppose that we have an algorithm that answers in polynomial time whether a tuple t belongs in $q(I)$, where q is a CQ. How can we construct a polynomial time algorithm that computes all answers?*

The other important distinction for complexity in the database context is what is considered as an input. Typically, we would consider both the query and the database instance as input, but as we will see their roles are very different in defining the complexity of evaluating a query q. We thus distinguish between 3 different types of complexity of evaluating a query, so that we can separate the influence of the query and the data on the complexity:

Data Complexity : the query is fixed, and the complexity is expressed in terms of the size of the database. This is useful in the context of databases, since the query is typically much smaller in size than the database.

Query Complexity : the database remains fixed, and the cost is expressed in terms of the size of the query. This complexity is not commonly considered in a database context.

Combined Complexity : the complexity is measured in the size of both the query and the database.

Observe that the data complexity is always lower than the combined complexity. We will mainly focus on data and combined complexity.

3.2 The Complexity of Conjunctive Queries

We start by examining the data complexity of conjunctive queries. The naive way of evaluating a CQ is the following. Suppose that the arity of the result is k. Then, consider every possible valuation, and for each valuation check whether it leads to an output tuple. The number of valuations is $|\mathbf{adom}(q)|^k$. The quantity $|\mathbf{adom}(q)|$ is connected polynomially to the input size. Thus, the running time is polynomial in the size of the database, but exponential in the size of the query! Hence, we obtain directly that the data complexity for CQs is P. Can we do better?

Theorem 3.8. *The data complexity of evaluating a boolean CQ is* AC^0.

Proof. Here goes a proof sketch of the construction. As an example, we will construct the circuit for the query $Q() : -R(a, z), S(z, b)$. □

In fact, the data complexity of every relational query (so evaluating FO formulas in general) is AC^0 (prove this more general result as an exercise!). This means that the data complexity for

evaluating any SQL query is AC^0, which is the class easiest to parallelize! This is why we say that SQL is *embarrassingly parallel*.

We have already shown that the combined complexity of evaluating a general CQ is NP-complete. This follows directly from the homomorphism theorem. In the next lecture, we will consider CQs for which we can have faster evaluation.

References

[Alice] S. ABITEBOUL, R. HULL and V. VIANU, "Foundations of Databases."