CS1200: Intro. to Algorithms and their Limitations Anshu & Vadhan

Lecture 18: Computational Complexity

Harvard SEAS - Fall 2024 *2024-11-05*

1 Announcements

Recommended Reading:

- MacCormick §5.3–5.5, Ch. 10, 11
- Reminder: ps7 is *not* due tomorrow.
- Penalties for using excess late days.

2 Loose Ends from Lec17

2.1 The Extended (or Strong) Church–Turing Thesis

The Church–Turing thesis only concerns problems solvable at all by these models of computation (Word-RAM programs, etc.). We haven't even seen any problems that are *not* solvable by Word-RAM programs—that will be a topic for the end of the course. There is, however, a stronger version of the Church–Turing hypothesis that also covers the efficiency with which we can solve problems.

Extended Church–Turing Thesis v1:

The Strong Church–Turing Thesis is not a precise mathematical claim, and thus cannot be formally proven. In fact, randomized algorithms, massively parallel computers, and quantum computers all could potentially provide an exponential savings in runtime. (For randomized algorithms, however, it is conjectured that they provide only a polynomial savings, as discussed in Lecture 8.)

If we modify the statement with some qualifiers, then these challenges no longer apply:

Extended Church–Turing Thesis v2:

"Deterministic" rules out both randomized and quantum computation, as both are inherently probabilistic. "Sequential" rules out parallel computation. This form of the Extended Church–Turing Thesis has stood the test of time for the approximately fifty years since it was formulated, even as computing technology has changed tremendously in that time.

2.2 Turing Machines (informal)

The Extended Church–Turing Thesis is usually stated in terms of *Turing Machines*, rather than Word-RAM programs. Turing Machines can be seen as a variant of the RAM model where:

1. *Finite Alphabet:*

2. *Memory Pointer:*

3. *Read/write:*

4. *Moving Pointer:*

With these changes, there is a mathematically very elegant description of Turing Machines (see optional reading in the detailed lecture notes and the recommended textbooks), with no arbitrary set of operations being chosen (allowing any "constant-sized" computation to happen in one step). For this reason, Turing machines are the main model covered in classes like CS1210 on the Theory of Computation. We work with the Word-RAM Model because it is better-suited for measuring the efficiency of algorithms in practice.

On the surface, Turing Machines seem to be weaker than Word-RAM programs. Nevertheless they are polynomially equivalent:

Theorem 2.1. *A problem can be solved in polynomial time by a Word-RAM program if and only if it can be solved in polynomial time by a Turing Machine.*

A proof sketch an be found in the optional reading. Because of the above, the Word-RAM Model and Turing Machines are referred to *strongly Turing-equivalent* models of computation.

3 Computational Complexity

Computational complexity aims to classify problems according to the amount of resources (e.g. time) that they require to solve.

For example, we've seen problems whose fastest algorithms run in:

- Linear time:
- Nearly linear time:
- Polynomial time:
- Exponential time:

Recall that the same problem can have many different algorithms for item, each with a different runtime. An ultimate goal of Computational Complexity would be to identify the *fastest* possible runtime for a problem, so that once we have achieved it, we can stop trying to improve our algorithms further. As we will see, the field is not quite there yet, but we do have a very rich understanding of the complexity of problems.

To develop a robust and clean theory for classifying problems according to computational complexity, we make two choices:

- A problem-independent size measure. Recall that we allowed ourselves to use different size parameters for different problems (array length n and universe size U for sorting; number n of vertices and number m of edges for graphs, number n of variable and number m of clauses for Satisfiability). To classify problems, it is convenient to simply measure the size of the input by its *length N in bits* (which we call the *bitlength*, or sometimes just *length* of the input). For example:
  - Array of n numbers from universe size U:
  - Graphs on n vertices and m edges in adjacency list notation:
  - 3-SAT formulas with n variables and m clauses:
- Polynomial slackness in running time: We will only try to make coarse distinctions in running time, e.g. polynomial time vs. super-polynomial time. If the Extended Church-Turing Thesis is correct, the theory we develop will be independent of changes in computing technology. It is possible to make finer distinctions, like linear vs. nearly linear vs. quadratic, if we fix a model (like the Word-RAM), and a newer subfield called *Fine-Grained Complexity* does this.

To this end, we define the following *complexity classes*.

Definition 3.1 (complexity classes)**.** For a function $T : \mathbb{N} \to \mathbb{R}^+$,

- $\mathsf{TIME}_{\mathsf{search}}(T(N))$ is
- $\mathsf{TIME}(T(N))$ is
- (Polynomial time)

  $\mathsf{P}_{\mathsf{search}} =$ $\mathsf{P} =$.
- (Exponential time)

  $\mathsf{EXP}_{\mathsf{search}} =$ $\mathsf{EXP} =$.

Note that $\mathsf{P}_{\mathsf{search}}$ and P would be the same if we replace Word-RAM with any strongly Turing-equivalent model, like Turing Machines. (Remark on terminology: what we call $\mathsf{P}_{\mathsf{search}}$ is called Poly in the MacCormick text, and is often called FP elsewhere in the literature.)

By this definition, ShortestPaths, 2-Coloring, Sorting, IntervalScheduling, BipartiteMatching, and 2-SAT are all in $\mathsf{P}_{\mathsf{search}}$ (as well as P for decision versions of the problems). However, all we know to say about 3-Coloring, 3-SAT, IndependentSet, or LongestPath is that they are in $\mathsf{EXP}_{\mathsf{search}}$. Can we prove that they are not in $\mathsf{P}_{\mathsf{search}}$?

The following seems to give some hope:

Theorem 3.2.

Unfortunately, we do not know how to prove that many of the problems we care about (like 3-SAT, 3-Coloring, LongestPath, IndependentSet) are in $\mathsf{EXP}_{\mathsf{search}} \setminus \mathsf{P}_{\mathsf{search}}$. However, what we *can* do is relate the seeming hardness of problems to each other (and many others!) via *reductions*.

4 Polynomial-Time Reductions

Definition 4.1. For computational problems Π and Γ, we write $\Pi \leq_p \Gamma$ if

Q: Why does the equivalence stated at the end of the definition hold?

A:

Some examples of polynomial-time reduction that we've seen include:

- 3-Coloring $\leq_p$ SAT
- LongPath $\leq_p$ SAT

Using polynomial-time reductions to compare problems fits nicely with the study of the classes $\mathsf{P}_{\mathsf{search}}$ and P, since they are "closed" under such reductions:

Lemma 4.2. *Let* Π *and* Γ *be computational problems such that* $\Pi \leq_p \Gamma$. *Then:*

1.

2.

Proof. 1.

2. Contrapositive of Item 1

□

This lemma means that we can use polynomial-time reductions both positively—to show that problems are in $\mathsf{P}_{\mathsf{search}}$— and negatively—to give evidence that problems are not in $\mathsf{P}_{\mathsf{search}}$. For example, under the *assumption* that 3-Coloring is not in $\mathsf{P}_{\mathsf{search}}$, it follows that SAT is not in $\mathsf{P}_{\mathsf{search}}$, by the above lemma and the fact that 3-Coloring $\leq_p$ SAT (SRE5). As always, *the direction of the reduction is crucial!*

Another very useful feature of polynomial-time reductions is that they compose with each other:

Lemma 4.3. *If* $\Pi \leq_p \Gamma$ *and* $\Gamma \leq_p \Theta$ *then* $\Pi \leq_p \Theta$.

This is proven similarly to Lemma 4.2, but substituting Problem 2 from Problem Set 2 in place of Lemma 3.2 from Lecture 4.