1 Announcements

- Salil's in-person OH this week moved to Thursday 1:30-2:15pm. Next Zoom OH Mon 5:30-6:15pm.

- Next SRE Thursday 11/13.

- PS6 feedback and reflection

  - Median and 75th percentile time spent: College: 8hrs/10hrs, DCE: 9hrs/13.5hrs.
  - Some frustration with experimental question. We'd like to understand better why (beyond the Windows issues, which are hopefully now fixed) - please share.
  - Ways of supporting classmates
    * Better SRE preparation for senders, more active participation for receivers
    * Creating a discord group chat
    * Attending OH more frequently, and helping fellow students there.
    * Active engagement on Ed
  - Remember: Good reflection responses are usually about a paragraph, with something like 7 or 8 sentences. Most importantly, please make sure your answer is specific to this class and your experiences in it. If your answer could have been edited lightly to apply to another class at Harvard, points will be taken off.

Recommended Reading:

- Hesterberg–Vadhan 19

2 Motivation

In the remainder of the course, our focus is understanding why some computational problems seem resistant to satisfactory solutions—that is, what are the inherent *limits* on algorithms?

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 *known* algorithms run in:

- Linear time:

- Nearly linear time:

- Polynomial time:

- Exponential time:

Recall that the same problem can have many different algorithms 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. To have a clean classification of problems, it is preferable to have one, single-parameter size measure that we apply across different problems. With Word-RAM as our main model of computation, it is natural to encode inputs x as arrays of N natural numbers, such that every array entry has $O(\log N)$ bits. The length N of the array x will be our size measure, and we will denote it by $|x|$.

Examples:

- SORTINGONFINITEUNIVERSE:

- Graph Problems:

- k-SAT:

Polynomial slackness in running time. We will only try to make coarse distinctions in running time, such as *polynomial time* vs. super-polynomial time. This choice is motivated in part by the Extended Church–Turing Thesis, which said the following (in our second formulation of it):

It is possible to make finer distinctions, like linear vs. nearly linear vs. quadratic, if we fix a model (like Word-RAM), and a newer area of research called *Fine-Grained Complexity* does this.

3   Complexity Classes

Following up on the above ideas, we define the following *complexity classes*.

Definition 3.1 (complexity classes)**.** For a function $T : \mathbb{N} \to \mathbb{R}^+$ with $T(n) \geq n+2$,

- $\mathsf{TIME}_{\mathsf{search}}(T(n))$ is

- $\mathsf{TIME}(T(n))$ is

- (Polynomial time)

$$\mathsf{P}_{\mathsf{search}} = \qquad\qquad\qquad \mathsf{P} = \qquad\qquad\qquad .$$

- (Exponential time)

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

Note that $\mathsf{P}_{\mathsf{search}}$ and $\mathsf{EXP}_{\mathsf{search}}$ would be the same if we replace Word-RAM programs with algorithms described using any strongly Turing-equivalent model, like Turing Machines.

The algorithms that we have covered this semester demonstrate that a number of problems are in $\mathsf{P}_{\mathsf{search}}$. For some other problems, we only know 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 us some hope:

Theorem 3.2.

Unfortunately, we do not know how to prove that many of the problems we care about (like the ones mentioned above) are in $\mathsf{EXP}_{\mathsf{search}} \setminus \mathsf{P}_{\mathsf{search}}$. So our state of knowledge is as follows:

To get a handle on why these problems seem hard, we will introduce another complexity class.

4   NP$_{\mathsf{search}}$

Roughly speaking, NP$_{\mathsf{search}}$ consists of the computational problems where valid outputs can be *verified* in polynomial time. This is a very natural requirement; what's the point in searching for something if we can't recognize when we've found it? See Figures 1 for an illustrative example; note that verifying a solution appears to be much easier than finding one.

Figure 1: Can you find a cat?

Formally:

Definition 4.1. A computational problem $\Pi = (\mathcal{I}, \mathcal{O}, f)$ is in NP$_{\mathsf{search}}$ if the following conditions hold:

1. All valid outputs are of polynomial length:

2. All valid outputs are verifiable in polynomial time:

Let's see some examples of problems in NP$_{\mathsf{search}}$.

4

Proposition 4.2. SATISFIABILITY, GRAPH COLORING, *and* INDEPENDENT SET — THRESHOLD SEARCH *are all in* $\mathsf{NP}_{\mathsf{search}}$.

Proof. For SATISFIABILITY, we have

For GRAPH COLORING, we have

For INDEPENDENT SET — THRESHOLD SEARCH, we have

$\square$

As a *non*-example, consider INDEPENDENT SET — OPTIMIZATION SEARCH, where

The following proposition shows that every problem in $\mathsf{NP}_{\mathsf{search}}$ can be solved in exponential time.

Proposition 4.3. $\mathsf{NP}_{\mathsf{search}} \subseteq \mathsf{EXP}_{\mathsf{search}}$.

Proof.

$\square$

So now our diagram of complexity classes is the following:

Remark 4.4. On $\mathsf{P}_{\mathsf{search}}$ vs $\mathsf{NP}_{\mathsf{search}}$: Somewhat counterintuitively, $\mathsf{P}_{\mathsf{search}} \nsubseteq \mathsf{NP}_{\mathsf{search}}$. This due to artificial examples, but most of the natural problems in $\mathsf{P}_{\mathsf{search}}$ are also in $\mathsf{NP}_{\mathsf{search}}$.

Remark 4.5. Every problem in $\mathsf{NP}_{\mathsf{search}}$ has a corresponding decision problem (deciding whether or not there are any valid outputs). The class of such decision problems is called NP. We will discuss the class NP more next week.

Remark 4.6. The 'N' in $\mathsf{NP_{search}}$ and NP stands for 'nondeterministic,' because problems in these classes can be solved by "nondeterministically guessing" a solution and efficiently verifying that it is valid. We will not make precise this notion of nondeterministic computation.

5 Polynomial-Time Reductions

Although we do not know how to prove that many problems we care about are not in $\mathsf{P_{search}}$, it turns out that we can get a much better understanding of their complexity via *reductions*. Since we have adopted "polynomial time" as our coarse notion of "efficiently solvable," it makes sense to do the same for reductions:

Definition 5.1. For computational problems Π and Γ, we write $\Pi \leq_p \Gamma$ if

It is useful to note that polynomial-time reductions make at most polynomially many oracle calls, each of at most polynomial length:

Lemma 5.2. *For computational problems Π and Γ, there is a polynomial-time reduction from Π to Γ if and only if there is a constant $c \geq 0$ such that $\Pi \leq_{O(n^c), O(n^c) \otimes O(n^c)} \Gamma$.*

Proof.

$\square$

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

Lemma 5.3. *Let Π and Γ be computational problems such that $\Pi \leq_p \Gamma$. Then:*

1.

2.

Proof.   1.

2. Contrapositive of Item 1

$\square$

The vast majority of reductions we have seen in this book so far have been polynomial-time reductions. Typically, we have used these for *positive* results:

However, Item 2 shows that we can also use reductions for *negative* results, to give evidence that problems are not in $\mathsf{P_{search}}$. For example,

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 5.4. *If $\Pi \leq_p \Gamma$ and $\Gamma \leq_p \Theta$ then $\Pi \leq_p \Theta$.*

This follows by similar reasoning to Lemma 5.3.