CS 6505: Computability & Algorithms
Lecture Notes for Week 5, Feb 8-12

P, NP, PSPACE, and PH

A deterministic TM is said to be in $SPACE\left(s\left(n\right)\right)$ if it uses space $O\left(s\left(n\right)\right)$ on inputs of length n. Additionally, the TM is in $TIME(t\left(n\right))$ if it uses time $O(t\left(n\right))$ on such inputs. A language L is polynomial-time decidable if $\exists k$ and a TM M to decide L such that $M \in TIME(n^k)$. (Note that k is independent of n.)

For example, consider the langage $PATH$, which consists of all graphsdoes there exist a path between A and B in a given graph G. The language $PATH$ has a polynomial-time decider. We can think of other problems with polynomial-time decider: finding a median, calculating a min/max weight spanning tree, etc.
P is the class of languages with polynomial time TMs. In other words,

$$P = \ \cup_k TIME(n^k)$$

Now, do all decidable languages belong to P? Let's consider a couple of languages:
HAM PATH: Does there exist a path from s to t that visits every vertex in G exactly once?
SAT: Given a Boolean formula, does there exist a setting of its variables that makes the formula true? For example, we could have the following formula F :

$$F = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$$

The assignment $x_1 = 1, x_2 = 0, x_3 = 0$ is a satisfying assignment.

No polynomial time algorithms are known for these problems – such algorithms may or may not exist. They can be solved (i.e., decided) by polynomial-time Nondeterministic TMs (NTMs). The idea behind this polynomial time algorithm is that a nondeterministic TM "guesses" the correct path for the $HAM - PATH$ language, and it "guesses" the correct assignment for the SAT language.

Recall that an NTM accepts iff any one of its computation paths accepts. The path amounts to a verification of the YES answer. We have $NSPACE(s\left(n\right))$ and $NTIME(t\left(n\right))$

NP is the class of languages that can be decided by polynomial-time NTMs:

$$NP = \cup_k NTIME(n^k)$$

Alternately, NP is the class of languages with the property that the membership ("YES") can be verified in polynomial-time using a polynomial-sized *certificate*. For example, consider SAT: if F is satisfiable, a valid assignment is the certicate. In $HAMPATH$, if G has a Hamiltonian path, then the sequence of vertices visited is the certificate.
Clearly $P \subseteq NP$. From Savitch's theorem,

$$NSPACE = PSPACE$$

Since the space requirement for $PSPACE$ only squares that for $NSPACE$.
Also, we have $NTIME\left(t\left(n\right)\right) \subseteq DTIME(2\{O(t\left(n\right)\})$
We define EXP or $EXPTIME$ = Languages that can be decided in exponential time.
So

$$P \subseteq NP \subseteq PSPACE \subseteq EXP$$

Amazingly, we do not know if any of these containments is strict. In other words, does there exist a language L such that $L \in EXP$ and $L \notin PSPACE$ or even $L \notin P$?
$L \in NP \iff \exists NTM\ M$ s.t. $L = \{x \mid \exists \text{ accepting path in } M \text{ on input } x\}$

The class of languages that are complements of languages in NP is called $coNP$.
$L \in coNP \Leftrightarrow \exists NTM\ M$ such that
$L = \{\ x \mid \underline{\text{Every}} \text{ valid computation path of } M \text{ is an accepting path for } x\ \}$.

$$L \in coNP \ \Leftrightarrow\ T \in NP \ \Leftrightarrow\ T = \{x \mid x \notin L\},$$

$L = \{x \mid x \text{ is not accepted by a TM for } L \text{ on any path }\}$

In other words, L is rejected on every path.
How do we verify membership of a language in coNP? We need a short (polynomial-sized) certificate that, if $x \notin L$. For example, we need to show that a graph G does not belong in $HAMPATH$, or we need to show that a formula F does not have a satisfying assignment.

What is the hierarchy of P, NP, and coNP? We know that every language that's in P can be solved in polynomial time. So we certainly have a certificate that shows that a language belongs to P – it's the TM that decides the language! So P is in NP. Likewise, the same TM that decides a language L in P will also reject its input in polynomial time, so P is also in NP. What about NP and coNP? Well, we don't know.

Polynomial Hierarchy (PH)

We would like to construct a hierarchy of problems within PSPACE that are successively more difficult. First, let's revisit definitions for a couple of languages:

$$SAT : \{\ F \mid \exists x : F(x) =\ 1\}$$

$$\overline{SAT} : \{F \mid \forall x : F(x) =\ 0\}$$

So SAT is the set of all formulae such that there is some satisfying assignment for each formula, and is the set of all formulae such that all assignments are not satisfying (i.e., there are no satisfying assignments).

We have used a computation tree to visualize finding a solution to a problem. We can imagine the same kind of computation tree for solving SAT: we can set x_1 to 0 or 1, then we can set x_2 to 0 or 1, and so on. Whenever we see $\exists$, the existential quantifier, we are asking if one of those two settings will yield a satisfying assignment. So setting $x_1 = 0$, Whenever we see $\forall$, we are asking if all of the branches yield a satisfying assignment. For example, we may ask that some satisfying exists for $x_2 = 0$ and $x_2 = 1$. If any of the branches under a universal quanitifer fail, then the quanitifier will fail.

This gives us the concept for an <u>Alternating Turing Machine</u>. An Alternating Turing Machine is one that can, at each node of computation, <u>accept</u> if any one path emanating from the node accepts (i.e., we have an existential quantifier $\exists$) or if <u>all</u> paths accept (i.e., we have a universal quantifier). The Alternating Turing Machine must also alternate between $\exists$ and $\forall$ quantifiers.

We will use this same idea to build a hierarchy of problems in PSPACE. Let's define the following:

$\mathbf{\Sigma_i}$ is an alternating Turing Machine that alternates i times between existentially-quantified ($\exists$) and universally-quantified ($\forall$) stages, starting with an existentially-quantified stage.

$\mathbf{\Pi_i}$ is an alternating Turing Machine that alternates i times between universally-quantified and existentially-quantified stages, starting with a universal quantifier.

So $\Sigma_2 SAT = \{\ F \mid \exists x_1 \forall x_2,\ F(x_1, x_2) =\ 1\ \}$. That is, it is the set of all two-variable formulae such that there is some assignment to the first variable such that all assignments to the second variable will satisfy the formula. Similarly, $\Pi_2 SAT = \{\ F \mid \forall x_1 \exists x_2,\ F(x_1, x_2) =\ 0\}$. We can also alternate up to as many

variables as we like: $\Sigma_i SAT = \{F \mid \exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots,\ F(x_1, \ldots,\ x_i) = 1\}$, and we can have $\Pi_i SAT = \{F \mid \forall x_1 \exists x_2 \forall x_3 \exists x_4 \ldots,\ F(x_1, \ldots,\ x_i) = 0\ \}$.
We define the Polynomial Hierarchy, PH, as follows:

$$PH = \cup_i \Sigma_i = \cup_k TIME\left(n^k\right) = \ \cup_i \Pi_i \cup_k TIME(n^k)$$

$$PH \subseteq PSPACE$$

PSPACE Complete
We have an idea of what's in PSPACE, but what are the "hardest" algorithms in PSPACE? We want to know if solving one problem in PSPACE will somehow yield a solution to another problem in PSPACE. For that, we have the notion of PSPACE-completeness:
A language L is ***PSPACE-complete*** if it satisfies two conditions:

1. L is in $PSPACE$;

2. All languages in $PSPACE$ are polynomial-time reducible to L.

We know that all Turing Machines that use $O(n^k)$ space for some k are in $PSPACE$, but we don't have a PSPACE-complete language yet. For that, we will use the language $TQBF$: $TQBF = \{\ \phi | \phi$ is a fully quantified Boolean formula $\}$.

A fully-quantified Boolean formula is a Boolean formula in which every variable has a quantifier. For example, the following formula is fully-quantified:

$$\phi = \exists x_1 \forall x_2 \exists x_3 [\ (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_3})$$

Each variable, x_1, x_2, and x_3, has a quantifier that precedes it.

We want to show that TQBF is PSPACE-complete. Again, we do not have any other languages that are PSPACE-complete, so we must turn the execution of any Turing Machine M with input w into a quantified Boolean formula. If M accepts w, then the quantified Boolean formula has a satisfying assignment. Otherwise, the quantified Boolean formula does not have a satisfying assignment. Here is a sketch of how we proceed:

We construct a *tableau* that describes the Turing Machine's computation from a configuration A to a configuration B. We want to go from the starting configuration A to the accepting configuration B in at most t steps. Now we construct a Boolean formula for the Turing Machine such that each tape character and Turing Machine state corresponds to a literal. Each clause is equivalent to determining the possible values on the tape and the Turing Machine's state. The Boolean formula is true if the Turing Machine M will accept w in at most t steps.

We proceed recursively like we did in solving the *PATH* problem: we cut the distance from t to $t/2$ and look for a configuration C such that we can find a path from A to C in at most t/2 steps and another path from C to B in at most t/2 steps. So we can actually find a solution, but we may end up with an exponentially large formula. We avoid this by introducing quantifiers that help us cut down the size of the formula. (For a more detailed description, see Sipser section 8.3.)
For more information on showing that TQBF is PSPACE-complete, see chapter

9 of Kleinberg/Tardos or see Sipser section 8.3.

References

1. _Sipser, Sections 7.3 (P and NP), 8.2 and 8.3 (PSPACE), and 10.3(PH)
2. _Papadimitriou, chapter 7 (P, NP, and PSPACE), section 17.2 (PH)
3. _Arora and Barak (Draft at http://www.cs.princeton.edu/theory/index.php/Compbook/Draft), Chapters 2 (NP and NP completeness) and 5 (PH)
4. Kleinberg and Tardos, chapter 9 (PSPACE). Kleinberg and Tardos present a more in-depth version of PSPACE that may be preferred by those who prefer a more conversational style.