P, NP, PH and PSPACE

NP and PSPACE

A language L is said to be in $SPACE(s(n))$ if there exists a turing machine that decides membership in L using space $O(s(n))$ on inputs of length n. Simlarly, a language L is said to be in $TIME(t(n))$ if there exists a turing machine that decides membership in L in time $O(t(n))$ on inputs of length n. 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 representations of all graphs and two vertices A and B in the graph such that there exists a path between A and B in the graph. The language $PATH$ has a polynomial-time decider. We have come across several problems with a polynomial-time decider: set of arrays such that median is at least k (finding median), calculating a min/max weight spanning tree, etc.

We define P to be the class of languages with polynomial time deterministic TMs. In other words,

$$P = \cup_k DTIME(n^k).$$

Now, do all decidable languages belong to P? Let us consider a couple of languages.

- *HAM-PATH*: Set of graphs G and vertices $s, t \in V(G)$ such that there exist a path from s to t that visits every vertex in G exactly once. Or equivalently, given a graph and two vertices s, t in the graph, does ther exist a path from s to t that visits every vertex in G exactly once?

- *SAT*: Set of boolean formulas so that there exists a boolean assignment to the variables that make the formula true. 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.

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

  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 accept. The path amounts to a verification of the YES answer. Using such NTMs, we can define the class of languages $NSPACE(s(n))$ and $NTIME(t(n))$.

Now, NP is the class of languages that can be decided by polynomial-time NTMs. That is,

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

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 $HAM-PATH$, if G has a Hamiltonian path, then the sequence of vertices visited is the certificate.

- Clearly, $P \subseteq NP$.

- By Savitch's Theorem, $NPSPACE = PSPACE$ since the space requirement for $PSPACE$ only squares that for $NPSPACE$.

Also, we have $NTIME\left(t\left(n\right)\right) \subseteq DTIME(2^{O(t(n)}$. 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 $L \in PSPACE$ and $L \notin NP$?
- or $L \in NP$ and $L \notin P$?

We do know that $P \subsetneq EXP$ by the time hierarchy theorem.

coNP

Now, by the definition of NP,

$$L \in NP \iff \exists NTM\ M \text{ s.t. } L = \{x \mid \exists \text{ accepting path in } M \text{ on input } x\}.$$

The class of languages that are complements of languages in NP is called $coNP$. That is, $coNP = \{L : \overline{L} \in NP\}$ where $\overline{L} = \{x \mid x \notin L\}$. Or equivalently,

$L \in coNP \Leftrightarrow \exists NTM\ M$s.t. $L = \{\ x \mid$ Every valid computation path of M is a rejecting path for $x\ \}$.

For the proof of equivalence, we observe the following.

$$L \in coNP \Leftrightarrow \overline{L} \in NP \Leftrightarrow L = \{x \mid x \text{ is not accepted by the NTM for } \overline{L} \text{ on any path}\}.$$

How can we verify membership of a language in $coNP$? We need a short (polynomial-sized) certificate that $x \notin L$. For example, is there a polynomial-sized certificated to show that a graph G does not belong in $HAMPATH$, or that a boolean formula F does not have a satisfying assignment?

P, NP and coNP - what is the relationship?

What is the hierarchy of P, NP, and coNP? We know that every language that is in P can be solved in polynomial time. So we certainly have a certificate that shows that a language belongs to P – it is 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 us revisit the definitions of 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 $\overline{SAT}$ 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. Whenever we see $\forall$, we are asking if all variable settings yield a satisfying assignment. Now, we may ask that some assignment exists for x_1 so that for every possible assignment x_2, the formula is satisfied. If any of the branches under a universal quantifier fail, then the existential quantifier also fails.

More generally, consider

$$\Sigma_2 SAT = \{\text{Formulas of the form } \exists X_1 \forall X_2\ F(x_1, x_2) \text{ which evaluate to true}\}.$$

That is, it is the set of all formulae such that there is some assignment to the first set of variables X_1 such that all assignments to the second set of variables X_2 will satisfy the formula.

Similarly,

$$\Pi_2 SAT = \{\text{Formulas of the form } \forall X_1 \exists X_2\ F(x_1, x_2) \text{ which evaluate to true}\}.$$

We can also alternate upto i sets of variables.

$$\Sigma_i SAT = \{\text{Formulas of the form } \exists X_1 \forall X_2 \ldots\ F(x_1, x_2, \ldots, x_i) \text{ which evaluate to true}\}.$$

and

$$\Pi_i SAT = \{\text{Formulas of the form } \forall X_1 \exists X_2 \ldots\ F(x_1, x_2, \ldots, x_i) \text{ which evaluate to true}\}.$$

This leads us towards the concept of an Alternating Turing Machine. An Alternating Turing Machine is one that can, at each node of computation, accept if any one path emanating from the node accepts (i.e., we have an existential quantifier $\exists$) or if all 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 us define the following.

Σ_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.

Π_i is an alternating Turing Machine that alternates i times between universally-quantified and existentially-qualified stages, starting with a universal quantifier.

We now define the Polynomial Hierarchy (PH) as

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

By the definition of PH and alternating turing machines, it follows that $PH \subseteq PSPACE$. This is because, we can simulate any alternating turing machine that runs in polynomial time using a deterministic turing machine that uses polynomial space.

PSPACE-completeness

We have an idea of what is 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 $PSPACE$ and
2. All languages in $PSPACE$ are polynomial-time reducible to L. That is, given any language B in PSPACE, we can decide membership in B using L as an oracle/procedure polynomial number of times and in polynomial additional time.

We know that all Turing Machines that use $O(n^k)$ space for some k are in $PSPACE$, but we donot have a PSPACE-complete language yet. For this, we will use the following language.

$$TQBF = \{F | F \text{ is a fully quantified Boolean satisfiable formula}\}.$$

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

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

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

TQBF is PSPACE-complete

We want to show that TQBF is PSPACE-complete.

We first have to show that TQBF is in PSPACE. For this, we just give an alternating turing machine that matches the quantifiers of the given formula. The depth of the computation tree is the number of variables and hence the matchine uses polynomial space.

Next we have to reduce any problem $B \in PSPACE$ to a TQBF. The idea is to 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.

Since $B \in PSPACE$, there exists a turing machine M that decides B. Now, examine the computation tableau of M on input w. We can write a boolean formula F_B to check whether the computation is valid, the start state is valid and the end state is ACCEPT. Therefore, $w \in B$ if and only if F_B is true. But F_B has exponential size. However, we need a polynomial time reduction.

Now, we will write a Boolean formula $F_{\text{start,accept,t}}$ which will evaluate to true if the Turing Machine M accepts the input 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.

$$F_{s,a,t} = \exists u \left(\phi_{s,u,\lceil \frac{t}{2} \rceil} \wedge \phi_{u,a,\lfloor \frac{t}{2} \rfloor} \right)$$

For the base case, if $t = 1$ or 0, we can write an explicit formula. But we may still end up with an exponentially large formula – every level of the recursion cuts t in half but doubles the size of the

formula; hence we might end up with a formula of size t. We avoid this by introducing quantifiers that help us cut down the size of the formula. We use universal quantifiers:

$$F_{s,a,t} = \exists u \forall (x,y) \in \{(s,u),(u,a)\} F_{x,y,\lceil \frac{t}{2} \rceil}.$$

Finally, we can express this as a boolean expression due to the following equivalence.

$$\begin{aligned} \forall x \in \{y,z\} F &\equiv \forall x (x = y \vee x = z) \Rightarrow F \\ &\equiv \forall x \dashv (x = y \vee x = z) \vee F \\ &\equiv \forall x \dashv (((x \wedge y) \vee (\dashv x \wedge \dashv y)) \vee ((x \wedge z) \vee (\dashv x \wedge \dashv z))) \vee F. \end{aligned}$$

Now, the size of the final formula is polynomial in the size of the original formula.