CS 301

Lecture 24 – Nondeterministic polynomial time

Stephen Checkoway

April 25, 2018

The classes $\mathrm{TIME}(t(n))$ and P

Let $t : \mathbb{N} \to \mathbb{R}^+$ be a function. The time complexity class $\mathrm{TIME}(t(n))$ is the set of languages that are decidable by an $O(t(n))$-time TM

P is the class of languages that are decidable in polynomial time on a TM,

$$\mathrm{P} = \bigcup_{k=0}^{\infty} \mathrm{TIME}(n^k)$$

The classes $\mathrm{NTIME}(t(n))$ and NP

Let $t : \mathbb{N} \to \mathbb{R}^{+}$ be a function. The nondeterministic time complexity class $\mathrm{NTIME}(t(n))$ is the set of languages that are decidable by an $O(t(n))$-time NTM

NP is the class of languages that are decidable in polynomial time on an NTM,

$$\mathrm{NP} = \bigcup_{k=0}^{\infty} \mathrm{NTIME}(n^{k})$$

This is not the most convenient definition of NP; we'll get a better one shortly

Example: Boolean satisfiability

$\mathrm{SAT} = \{\langle\phi\rangle \mid \phi$ is a satisfiable boolean formula$\}$

Previously, we showed that $2\text{-}\mathrm{SAT} \in \mathrm{P}$ and this relied on the formulae in $2\text{-}\mathrm{SAT}$ being in 2-CNF; there's no such restriction here

E.g., $\phi = (x \wedge (y \vee \overline{z})) \wedge \overline{(x \wedge y \wedge \overline{z})}$
Is ϕ satisfiable?

Example: Boolean satisfiability

$\mathrm{SAT} = \{\langle\phi\rangle \mid \phi$ is a satisfiable boolean formula$\}$

Previously, we showed that $2\text{-}\mathrm{SAT} \in \mathrm{P}$ and this relied on the formulae in $2\text{-}\mathrm{SAT}$ being in 2-CNF; there's no such restriction here

E.g., $\phi = (x \wedge (y \vee \overline{z})) \wedge \overline{(x \wedge y \wedge \overline{z})}$
Is ϕ satisfiable?

Yes. $x = T$, $y = F$, $z = F$ satisfies it. Therefore, $\langle\phi\rangle \in \mathrm{SAT}$

Example: $\mathrm{SAT} \in \mathrm{NP}$

We need to construct a NTM that decides SAT in polynomial time
N = "On input $\langle \phi \rangle$,

1. For each variable in ϕ, nondeterministically assign it a truth value
2. Using the assignments, evaluate ϕ. If $\phi = T$, then *accept*; otherwise *reject*"

Example: $\mathrm{SAT} \in \mathrm{NP}$

We need to construct a NTM that decides SAT in polynomial time
N = "On input $\langle \phi \rangle$,

1. For each variable in ϕ, nondeterministically assign it a truth value
2. Using the assignments, evaluate ϕ. If $\phi = T$, then *accept*; otherwise *reject*"

The essential feature of a NTM is the ability to nondeterministically make a choice (choose a path through its tree of computation)

Remember that an NTM accepts w if some branch of its computation accepts and rejects w if every branch rejects (this is a decider, remember)

Example: $\mathrm{SAT} \in \mathrm{NP}$

We need to construct a NTM that decides SAT in polynomial time
N = "On input $\langle \phi \rangle$,

1. For each variable in ϕ, nondeterministically assign it a truth value
2. Using the assignments, evaluate ϕ. If $\phi = T$, then *accept*; otherwise *reject*"

The essential feature of a NTM is the ability to nondeterministically make a choice (choose a path through its tree of computation)

Remember that an NTM accepts w if some branch of its computation accepts and rejects w if every branch rejects (this is a decider, remember)

If ϕ is satisfiable, then some branch of N's computation will select a satisfying assignment so N will accept

If ϕ is not satisfiable, then every branch will reject so N will reject; thus $L(N) = \mathrm{SAT}$

Both steps take polynomial time so $\mathrm{SAT} \in \mathrm{NP}$

$\mathrm{P} \subseteq \mathrm{NP}$

Theorem

For every language $A \in \mathrm{P}$, $A \in \mathrm{NP}$. *I.e.*, $\mathrm{P} \subseteq \mathrm{NP}$

How would we prove this?

$\mathrm{P} \subseteq \mathrm{NP}$

Theorem

For every language $A \in \mathrm{P}$, $A \in \mathrm{NP}$. *I.e.,* $\mathrm{P} \subseteq \mathrm{NP}$

How would we prove this?

Proof.

If $A \in \mathrm{P}$, then it is decided by a deterministic TM M in polynomial time.

We can construct an NTM N that has identical behavior to M; i.e., N doesn't use nondeterminism.

Thus $L(N) = L(M)$ and N runs in polynomial time $\square$

NP $\subseteq$ EXPTIME

Theorem

For every language $A \in \mathrm{NP}$, $A \in \mathrm{EXPTIME} = \bigcup_{k=0}^{\infty} \mathrm{TIME}(2^{n^k})$. *I.e.,* $\mathrm{NP} \subseteq \mathrm{EXPTIME}$

How would we prove this?

NP ⊆ EXPTIME

Theorem

For every language $A \in \mathrm{NP}$, $A \in \mathrm{EXPTIME} = \bigcup_{k=0}^{\infty} \mathrm{TIME}(2^{n^k})$. *I.e.,* $\mathrm{NP} \subseteq \mathrm{EXPTIME}$

How would we prove this?

Proof.

If A is decided by an NTM N in nondeterministic polynomial time $O(n^k)$, then we can construct a TM M that simulates N in (deterministic) time $2^{O(n^k)}$. □

$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXPTIME}$

It's true, although we haven't proved it, that $\mathrm{P} \neq \mathrm{EXPTIME}$. I.e., there are problems that we can solve in exponential time that we know can't be solved in polynomial time

Thus at least one of the subsets in $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXPTIME}$ must be strict

$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXPTIME}$

It's true, although we haven't proved it, that $\mathrm{P} \neq \mathrm{EXPTIME}$. I.e., there are problems that we can solve in exponential time that we know can't be solved in polynomial time

Thus at least one of the subsets in $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXPTIME}$ must be strict

Put another way, one of the following statements is true

- $\mathrm{P} = \mathrm{NP}$ and $\mathrm{NP} \neq \mathrm{EXPTIME}$;
- $\mathrm{P} \neq \mathrm{NP}$ and $\mathrm{NP} \neq \mathrm{EXPTIME}$; or
- $\mathrm{P} \neq \mathrm{NP}$ and $\mathrm{NP} = \mathrm{EXPTIME}$

Which one is true?

$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXPTIME}$

It's true, although we haven't proved it, that $\mathrm{P} \neq \mathrm{EXPTIME}$. I.e., there are problems that we can solve in exponential time that we know can't be solved in polynomial time

Thus at least one of the subsets in $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXPTIME}$ must be strict

Put another way, one of the following statements is true

- $\mathrm{P} = \mathrm{NP}$ and $\mathrm{NP} \neq \mathrm{EXPTIME}$;
- $\mathrm{P} \neq \mathrm{NP}$ and $\mathrm{NP} \neq \mathrm{EXPTIME}$; or
- $\mathrm{P} \neq \mathrm{NP}$ and $\mathrm{NP} = \mathrm{EXPTIME}$

Which one is true?

Fun fact: We don't know which is true!

Partitioning a multiset

$$\textsc{Partition} = \{\langle S\rangle \mid S \text{ is a multiset of positive integers and } \exists A \subseteq S \text{ s.t. } \textstyle\sum_{x\in A} x = \sum_{x\in S\setminus A} x\}$$

Consider the multiset $S = \{1, 1, 2, 3, 5\}$. Is $\langle S\rangle \in \textsc{Partition}$?

Partitioning a multiset

$$\textsc{Partition} = \Big\{\langle S\rangle \mid S \text{ is a multiset of positive integers and } \exists A \subseteq S \text{ s.t. } \textstyle\sum_{x\in A} x = \sum_{x\in S\setminus A} x\Big\}$$

Consider the multiset $S = \{1, 1, 2, 3, 5\}$. Is $\langle S\rangle \in \textsc{Partition}$?

Yes, $A = \{1, 2, 3\}$, $S \setminus A = \{1, 5\}$ both sum to 6

Show $\textsc{Partition} \in \mathrm{NP}$

We need to construct an NTM that decides $\textsc{Partition}$ in polynomial time

N = "On input $\langle S \rangle$,

1. Set $a \leftarrow 0$, $b \leftarrow 0$
2. For each $x \in S$
3.   Nondeterministically pick $c \in \{0, 1\}$
4.   If $c = 0$, then set $a \leftarrow a + x$; otherwise set $b \leftarrow b + x$
5. If $a = b$, then *accept*; otherwise *reject*"

The elements where $c = 0$ are in A and a is their sum; the elements where $c = 1$ are in $S \setminus A$ and b is their sum

Show $\textsc{Partition} \in \mathrm{NP}$

We need to construct an NTM that decides $\textsc{Partition}$ in polynomial time
N = "On input $\langle S\rangle$,

1. Set $a \leftarrow 0$, $b \leftarrow 0$
2. For each $x \in S$
3. Nondeterministically pick $c \in \{0, 1\}$
4. If $c = 0$, then set $a \leftarrow a + x$; otherwise set $b \leftarrow b + x$
5. If $a = b$, then *accept*; otherwise *reject*"

The elements where $c = 0$ are in A and a is their sum; the elements where $c = 1$ are in $S \setminus A$ and b is their sum

If $\langle S\rangle \in \textsc{Partition}$, then some branch of the computation will pick the correct A such that $a = b$ and N accepts
If $\langle S\rangle \notin \textsc{Partition}$, then every branch will select an A such that $a \neq b$ so N rejects

Each step takes polynomial time and the loop happens $|S|$ times so $\textsc{Partition} \in \mathrm{NP}$

Verifiers

A verifier for a language A is a deterministic TM V such that

$$A = \{w \mid V \text{ accepts } \langle w, c\rangle \text{ for some string } c\}$$

A polynomial time verifier is a verifier that has running time polynomial in the length of w but not c

c is called a certificate (or proof or witness)

Verifiers

A verifier for a language A is a deterministic TM V such that

$$A = \{w \mid V \text{ accepts } \langle w, c\rangle \text{ for some string } c\}$$

A polynomial time verifier is a verifier that has running time polynomial in the length of w but not c

c is called a certificate (or proof or witness)

The idea behind verifiers is given an instance of a problem w and some extra information about the solution of the problem c, V verifies $w \in A$

Verifiers need to be designed such that if $w \notin A$, then *no* certificate exists such that V accepts $\langle w, c\rangle$

Polynomial time verifier for SAT

An instance of SAT is (the representation of) a boolean formula ϕ
A certificate is an assignment of variables to truth values

E.g., $\phi = (x \wedge (y \vee \overline{z})) \wedge \overline{(x \wedge y \wedge \overline{z})}$
One possible certificate c is the assignment $x = T$, $y = F$, and $z = F$

Polynomial time verifier for SAT

An instance of SAT is (the representation of) a boolean formula ϕ
A certificate is an assignment of variables to truth values

E.g., $\phi = (x \wedge (y \vee \overline{z})) \wedge \overline{(x \wedge y \wedge \overline{z})}$
One possible certificate c is the assignment $x = T$, $y = F$, and $z = F$

We can construct a polynomial time verifier for SAT:
V = "On input $\langle \phi, c \rangle$,

1. Using the assignment c, evaluate ϕ
2. If $\phi = T$, then *accept*; otherwise *reject*"

Polynomial time verifier for SAT

An instance of SAT is (the representation of) a boolean formula ϕ
A certificate is an assignment of variables to truth values

E.g., $\phi = (x \wedge (y \vee \overline{z})) \wedge \overline{(x \wedge y \wedge \overline{z})}$
One possible certificate c is the assignment $x = T$, $y = F$, and $z = F$

We can construct a polynomial time verifier for SAT:
V = "On input $\langle \phi, c \rangle$,

1. Using the assignment c, evaluate ϕ
2. If $\phi = T$, then *accept*; otherwise *reject*"

If $\langle \phi \rangle \in \mathrm{SAT}$, then ϕ is satisfiable so there is some assignment c that satisfies ϕ and V will accept $\langle \phi, c \rangle$

If $\langle \phi \rangle \notin \mathrm{SAT}$, then ϕ is unsatisfiable so no matter what c is, it can't satisfy ϕ, so V will reject $\langle \phi, c \rangle$

V runs in time polynomial in $|\langle \phi \rangle|$

Polytime verifier for PARTITION

What should the certificate for an instance of PARTITION be?

Polytime verifier for PARTITION

What should the certificate for an instance of PARTITION be?

The certificate is subset A such that $\sum\limits_{x \in A} x = \sum\limits_{x \in S \setminus A} x$

V = "On input $\langle S, A \rangle$,

1. If $A \not\subseteq S$, then *reject*
2. Compute $a = \sum_{x \in A} x$ and $b = \sum_{x \in S \setminus A} x$
3. If $a = b$, then *accept*; otherwise *reject*"

Polytime verifier for PARTITION

What should the certificate for an instance of PARTITION be?

The certificate is subset A such that $\sum_{x \in A} x = \sum_{x \in S \setminus A} x$

V = "On input $\langle S, A \rangle$,

1. If $A \not\subseteq S$, then *reject*
2. Compute $a = \sum_{x \in A} x$ and $b = \sum_{x \in S \setminus A} x$
3. If $a = b$, then *accept*; otherwise *reject*"

If $\langle S \rangle \in$ PARTITION, then there is some $A \subseteq S$ that makes the equality hold so V will accept $\langle S, A \rangle$

If $\langle S \rangle \notin$ PARTITION, then no $A \subseteq S$ will make the equality hold so V will reject $\langle S, A \rangle$

Computing the sums takes polynomial time so V is a polytime verifier for PARTITION

A better characterization of NP

Theorem

Language A is in NP iff there is a polytime verifier for A.

This gives a better characterization of NP: NP is the class of languages for which a polynomial time verifier exists

P The class of languages that can be decided in polynomial time

NP The class of languages that can be verified in polynomial time

Proof

We need to prove to things

1. $\Longrightarrow$ If $A \in \mathrm{NP}$, then there is a polytime verifier V for A
2. $\Longleftarrow$ If there is a polytime verifier V for A, then $A \in \mathrm{NP}$

Start with $\Longrightarrow$: If A is in NP, then it is decided by an NTM N in polynomial time

For each $w \in A$, N makes a sequence of nondeterministic choices when it is run on w. (This sequence is the address tape in our NTM simulator)

Let c be the sequence of choices N makes for one branch of computation

Proof continued

V = "On input $\langle w, c\rangle$,

1. Simulate N on w using each symbol of c as the choice to take in each step, if there aren't enough symbols in c, then *reject*
2. If N accepts, then *accept*; otherwise *reject*"

Since N takes polytime on each branch, V takes polytime on the branch selected by c

If $w \in A$, then some sequence of choices c will cause N to accept w and thus V will accept $\langle w, c\rangle$

If $w \notin A$, then no matter what sequence of choices c that N makes, N will reject and thus V will reject $\langle w, c\rangle$ for all c

Proof continued

Now for $\Longleftarrow$: If V is a polynomial time verifier for A, then we need to construct a polynomial time TM N such that $L(N) = A$.

V runs in time $t(n) = a \cdot n^k$ for some $a, k \in \mathbb{N}$ (because it's a polytime verifier)

N = "On input w,

1. Nondeterministically select a string c of length at most $a \cdot n^k$
2. Run V on $\langle w, c \rangle$. If V accepts, then *accept*; otherwise *reject*"

Picking a string of polynomial length takes polynomial time; running a polytime verifier takes polynomial time so N runs in nondeterministic polynomial time

If $w \in A$, then there is some certificate c of length at most $a \cdot n^k$ [why?] such that V accepts $\langle w, c \rangle$. Thus some branch of N's computation will pick the correct c such that V accepts so N will accept

If $w \notin A$, then V rejects $\langle w, c \rangle$ for every c so N will reject. Therefore, $L(N) = A$ $\square$

Example: Hamiltonian path

A Hamiltonian path in a directed graph G is a directed path that goes through every vertex exactly once

$\textsc{HamPath} = \{\langle G, s, t\rangle \mid G \text{ has a Hamiltonian path from } s \text{ to } t\} \in \mathrm{NP}$

What should we pick for the certificate?

Example: Hamiltonian path

A Hamiltonian path in a directed graph G is a directed path that goes through every vertex exactly once

$\textsc{HamPath} = \{\langle G, s, t\rangle \mid G$ has a Hamiltonian path from s to $t\} \in \mathrm{NP}$

What should we pick for the certificate? The certificate should be the Hamiltonian path $c = \langle n_1, n_2, \ldots, n_k\rangle$ itself!

V = "On input $\langle G, s, t, \langle n_1, n_2, \ldots, n_k\rangle\rangle$ where $G = (V, E)$,

1. If $V \neq \{n_1, n_2, \ldots, n_k\}$, $s \neq n_1$, or $t \neq n_k$, then *reject*
2. For $i = 1$ up to $k - 1$,
3. If $(n_i, n_{i+1}) \notin E$, then *reject*
4. Otherwise, *accept*"

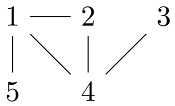
As usual, we need to show that V accepts only when the certificate is a valid Hamiltonian path and rejects everything else

We also need to show that V runs in time polynomial in $\langle G, s, t\rangle$

Vertex cover

A vertex cover for an undirected graph $G = (V, E)$ is a set $C \subseteq V$ such that for all $(a, b) \in E$, either $a \in C$ or $b \in C$

E.g., G:



$C = \{1, 4\}$ is a vertex cover of G of size 2

Vertex cover

A vertex cover for an undirected graph $G = (V, E)$ is a set $C \subseteq V$ such that for all $(a, b) \in E$, either $a \in C$ or $b \in C$

E.g., G:

$$\begin{tikzcd} 1 \arrow[r, dash] \arrow[d, dash] \arrow[dr, dash] & 2 \arrow[d, dash] & 3 \arrow[dl, dash] \\ 5 & 4 & \end{tikzcd}$$

$C = \{1, 4\}$ is a vertex cover of G of size 2

$\textsc{VertexCover} = \{\langle G, k\rangle \mid G \text{ has a vertex cover of size } k\} \in \mathrm{NP}$

What is the certificate?

Vertex cover

A vertex cover for an undirected graph $G = (V, E)$ is a set $C \subseteq V$ such that for all $(a, b) \in E$, either $a \in C$ or $b \in C$

E.g., G:

$$\begin{array}{ccccc} 1 & — & 2 & & 3 \\ | & \diagdown & | & \diagup & \\ 5 & & 4 & & \end{array}$$

$C = \{1, 4\}$ is a vertex cover of G of size 2

$\textsc{VertexCover} = \{\langle G, k \rangle \mid G \text{ has a vertex cover of size } k\} \in \mathrm{NP}$

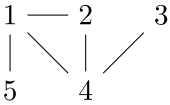
What is the certificate?

The certificate is a vertex cover of size k. The verifier checks that the certificate is a valid vertex cover and has size k

Clique

A clique in an undirected graph $G = (V, E)$ is a set $C \subseteq V$ such that every pair of (distinct) vertices in C is connected by an edge

E.g., G:



$C = \{1, 2, 4\}$ is a clique of size 3

Clique

A clique in an undirected graph $G = (V, E)$ is a set $C \subseteq V$ such that every pair of (distinct) vertices in C is connected by an edge

E.g., G:

$$\begin{array}{ccccc} 1 & \text{—} & 2 & & 3 \\ | & \diagdown & | & \diagup & \\ 5 & & 4 & & \end{array}$$

$C = \{1, 2, 4\}$ is a clique of size 3

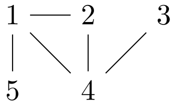
$\textsc{Clique} = \{\langle G, k\rangle \mid G \text{ has a clique of size } k\} \in \mathrm{NP}$

What is the certificate?

Clique

A clique in an undirected graph $G = (V, E)$ is a set $C \subseteq V$ such that every pair of (distinct) vertices in C is connected by an edge

E.g., G:



$C = \{1, 2, 4\}$ is a clique of size 3

$\textsc{Clique} = \{\langle G, k\rangle \mid G \text{ has a clique of size } k\} \in \mathrm{NP}$

What is the certificate?

The certificate is a clique of size k. The verifier checks that the certificate is a valid clique of size k