CSE 431 Theory of Computation
Scribes: Scott Daley and Daniel Noteboom
Lecture 10 May 1

P vs NP

P = $\bigcup_{k\geq 1} Time(n^k)$ (*the set of problems solvable in time $\leq cn^k$ for some fixed c,k*)

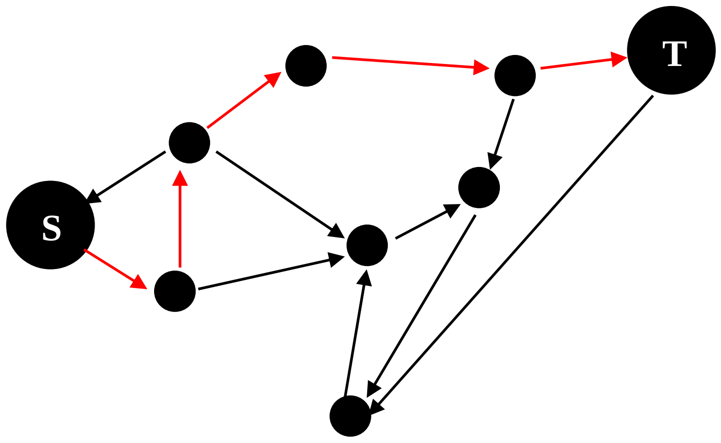
NP = $\bigcup_{k\geq 1} NTime(n^k)$ (*same as above but in non-deterministic time*)

- It seems as if a non-deterministic machine is more powerful than a deterministic machine, but no-one knows.

Algorithms:

Language: PATH = $\{< G, s, t >$: G is a directed graph and s,t are two nodes where there is a path from s->t

We can't just use bruteforce search to solve this problem because this would be time m^m(where m is the number of edges). We have to be a little smarter.



Theorem: $\boldsymbol{PATH \in P}$

1. Mark s O(1)
2. While there are more newly marked nodes O(n)
   For every edge (u,v) in G O(n^2)
   If u is marked and v is unmarked, then mark v O(1)
3. Accept if t is marked O(1)

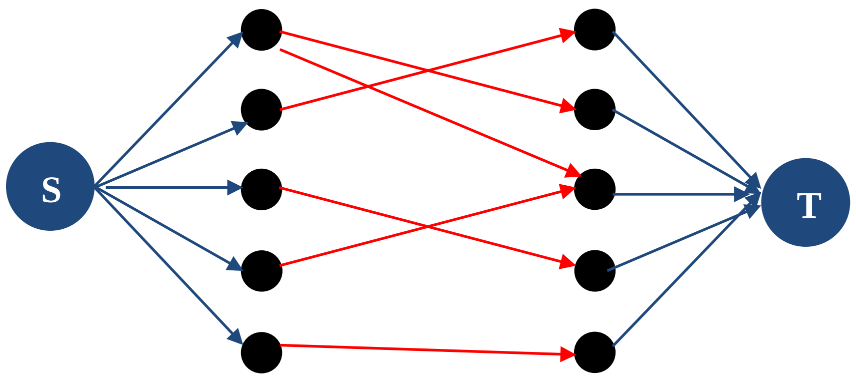
At worst, the running time is O(n^3). It's true that this running time is slower than it needs to be, and this algorithm is not the best algorithm to solve this problem (Breadth first search would solve the problem in O(m +n) time). However, for our purpose, all we wanted to do was prove that PATH was decidable in polynomial time and we accomplished this task.

Problem: Bipartite Matching

A Bipartite Graph G = (V,E) is a graph in which the vertex set V can be divided into two disjoint subsets X and Y such that every edge $e \in E$ has one end point in X and the other end point in Y. A matching M is a subset of edges such that each node in V appears in at most one edge in M.[1]

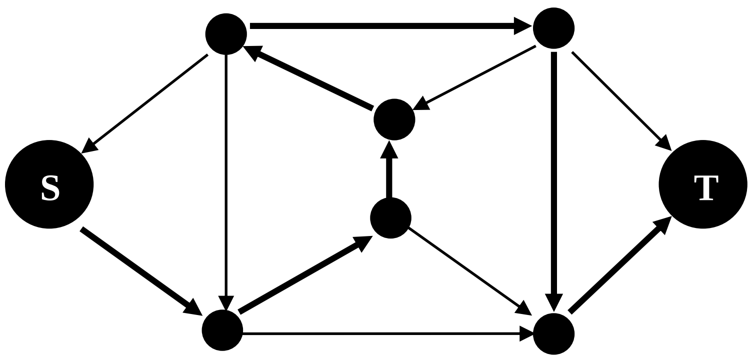
A maximum matching is a matching with the largest possible number of edges. The solution is to add two nodes, s and t to the graph where s points to all the nodes in X and t is pointed to by all the nodes in y. Each edge has a capacity of 1. Finding the maximum flow will also find the maximum matching.

- Determining that converting the graph into a flow network where finding the maximum flow produces the correct answer in polynomial time is non-trivial. The point is that the first problem was easy, and this problem is harder. We might suspect that a problem is hard, but it might not be hard.



Problem: Hamiltonian Path-A directed graph that goes through each node exactly once.

Language: HAMPATH = {$\langle G, s, t\rangle$: G is a directed graph with a Hamiltonian path from s to to t}.



- No known polynomial time algorithm exists for Hamiltonian path
- We need to have some way of talking about whether a problem is hard or easy. (even if we don't know the truth yet)
- There exists many unsolved problems that would revolutionize their respective fields if a solution was found. Are these problems related?

Language: COMPOSITES = $\{\langle n\rangle$: n is not prime$\}$
Solution: Check for i = 1,2,3, … $\sqrt{n}$. Does i divide n evenly? If the answer is yes to any i then n is not prime. Running time is O($\sqrt{n}$(log n)^3)

  - Input size ≈ log n
  - $\sqrt{n}$ part is slow, so this algorithm does not run in polynomial time

In both HAMPATH and COMPOSITES, it is very hard to find an efficient algorithm but the solutions can be checked very efficiently. In some problems the solutions cannot be checked efficiently.

Problem: $\overline{HAMPATH}$: $\{\langle G, s, t\rangle$: G is a directed graph with no Hamiltonian path from s to to t$\}$.

Problem:$\overline{COMPOSITES}$ $\{\ \langle x\rangle\ x\ is\ prime\}$

- It turns out there are efficient verifiers for $\overline{COMPOSITES}$ but there are no known efficient verifiers for $\overline{HAMPATH}$.

A formal way to talk about efficient verifiers is as follows

Definition: For a language L an algorithm V is a polynomial-time verifier for L if

$$L = \{w\!: V\ accepts\ \langle w, c\rangle\ for\ some\ string\ c\ and\ v\ runs\ in$$

$$polynomial\ time\ in\ length\ of\ w, i.e. V\ runs\ in\ time\ O(|w|^c)$$

$$for\ some\ c$$

Theorem: $HAMPATH$ has a polynomial time verifier (and thus is in NP as we will see).

Proof idea: The description of the Hamiltonian is the certificate.

Proof: The following is a verifier V for $HAMPATH$.

V = "On input $\langle\langle G, s, t\rangle, c\rangle$:

1. Check that c describes a Hamiltonian path from s to t in G.

   For example:

   $$c = (9,7,13,4,1,8,6,5,2,11,10,12)$$

   Are s and t in G?

   Does s = 9 and t = 12?

   Do all nodes in G appear exactly once?

   Is each pair of adjacent numbers an edge in G?

2. If all are true, $accept$.
3. Otherwise, $reject$."

$\langle G, s, t\rangle \in HAMPATH \leftrightarrow \exists c$ such that V accepts $\langle\langle G, s, t\rangle, c\rangle$

Example: $CLIQUE = \{\langle G, k\rangle : G$ is a graph that has a clique of size k$\}$

Clique: all noes in a clique have an edge to one another.

Theorem: $CLIQUE$ has a polynomial time verifier (and thus is in NP as we will see).

Proof idea: The list of nodes in the clique is the certificate.

Proof: The following is a verifier V for $CLIQUE$.

V = "On input $\langle\langle G, k\rangle, c\rangle$:

```
         1. Check that $c$ has $k$ nodes.
         2. Check that $\forall\ u, v$ in $c$ that there is an edge $\{u, v\}$ in $G$.
$O(n^2)$ 3. If both are true, $accept$.
         4. Otherwise, $reject$.
```

$\langle G, k\rangle \in CLIQUE \leftrightarrow \exists c$ such that V accepts $\langle\langle G, k\rangle, c\rangle$

Theorem: NP is exactly the class of polynomially verifiable languages.

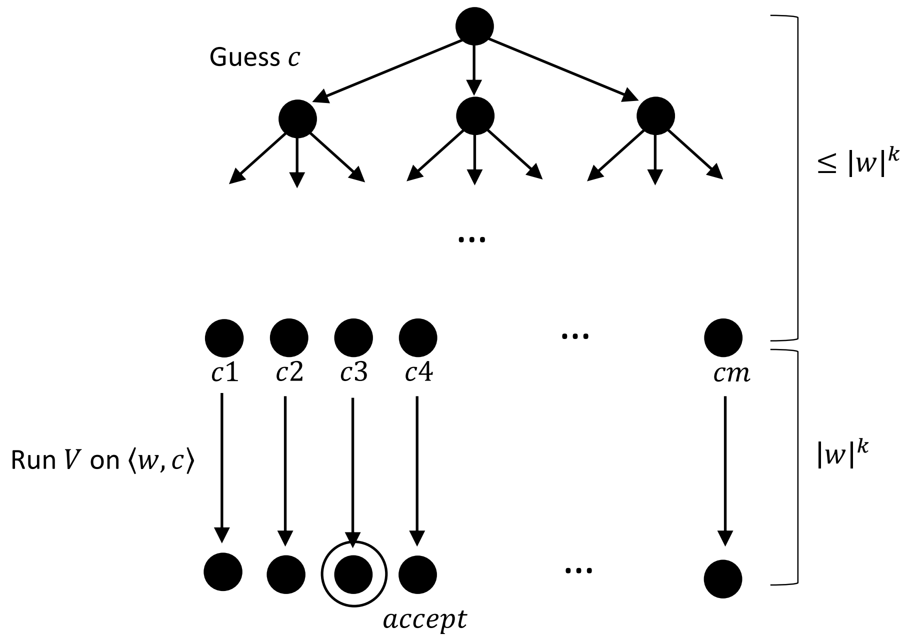
Proof: L with polynomial time verifier $\rightarrow\ L \in NP$.

Suppose L has a polynomial time verifier V.

$$L = \{w\colon V \text{ accepts } \langle w, c\rangle \text{ for some } c\}$$

V runs in polynomial time in $|w|$. Assume certificate has $|c| \leq |w|^k$ for some k.

Guess first bit of c, then next bit of c, and so on. The non-deterministic tree that does this has height $|w|^k$. For all certificates c of length 1 to $|w|^k$, run V on $\langle w, c\rangle$.


Guess c
$\leq |w|^k$
...
$c1$ $c2$ $c3$ $c4$ ... cm
Run V on $\langle w, c\rangle$
$|w|^k$
$accept$

Because this non-deterministic Turing machine can decide L in polynomial time, then $L \in NTIME(n^k)$ for some k and $L \in NP$.

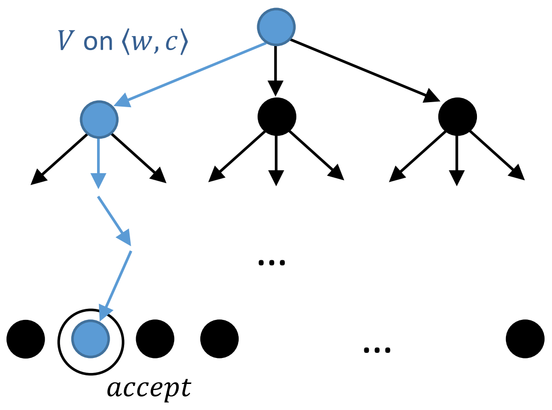
Proof: $L \in NP \;\rightarrow L$ has polynomial time verifier.

Now suppose that $L \in NP$. Then $L \in NTIME(n^k)$ for some k. Then there is some non-deterministic Turing machine N with height n^k that decides L.

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

1. Simulate N on w with choices given by c. *(deterministic)*
2. If N accepts w, $accept$."

The deterministic Turing machine V decides L in polynomial time.



Reference Site:

1. http://pages.cs.wisc.edu/~shuchi/courses/787-F09/scribe-notes/lec5.pdf