Computational Hardness and Coping Strategies

So far in this course:

- Efficient algorithms for many many problems
- General principles for designing efficient algorithms.

Q. Do all problems have efficient algorithms?

Next few weeks

1. Do all problems have efficient algorithms?
2. What does 'efficient' mean?
3. What are some problems that do not seem to have efficient algorithms?
4. Why do we think so?
5. What do we do if we encounter these problems in life?
6. Why do we care about all this?

Efficiently Solvable computational Problems

Solvable in polynomial time: a computational problem C is said to solvable in polynomial time (C is in P) if there is a (deterministic) algorithm A s.t

1. A solves C
2. The time complexity of A is at most $O(n^d)$ where n is the input size and d is an absolute constant (independent of n)

Examples:

shortest path
polynomial mult
integer mult
max flow

} what is d in these cases?

Efficiently solvable $\equiv$ polynomial time algorithms

But is n^{100} time algorithm really that efficient?

Theoretically: this notion of efficiency is quite robust. to precise models of computation, and has an intricate and consistent theory based on it that has stood the test of time.

In practice: for most problems in P, the constant happens to be small.

so that algorithms are indeed pretty efficient

So going forward: efficiency $\longleftrightarrow$ poly time solvable (membership in P)

A fundamental quest in computer science is to understand and classify computational problems based on their time complexity.

For a start (and quite worthwhile)

Is a certain problem solvable in polynomial time?

or

Must any algorithm for it require superpoly time?

Efficiently Verifiable Computational Problems

Minimum Vertex Cover

$G(V, E)$ — undirected graph

$S \subseteq V$ is a Vertex Cover if for every edge in the graph at least one of its end points is in S.

MinVC : Given a graph G, and integer k, does G have a vertex cover of size at most k?

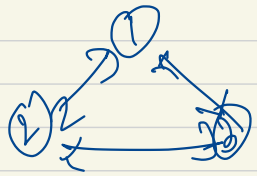
(Yes-no answer or the search version)

(1) — optimization question

(2) — a 'decision problem' — Yes/No answer.

(3) — search problem

Often,



compute the shortest path from u to v

↕

Is there a shortest path of length at most k.

compute the max flow

↕

Is there a flow of value at least k.

Convenient to work with search problems.

Min VC

G, k : output a vertex cover of size $\leq k$, if it exists.

Naive algorithm :

For all $S \subseteq V$, $|S| \leq k$

check if S is a VC

Running time : $\sim \binom{n}{k} \sim \left(\frac{n}{k}\right)^k$

— Not polynomial time.

Can we do significantly better?

A verification algorithm:

Given a subset $S \subseteq V$, can we efficiently check if S is a VC of G of size $\leq k$?

- Such a set S is a 'certificate' that the answer to the minVC question on (G, k) is Yes.
- Given such a candidate certificate, easy to check the validity of the certificate.

minVC : seems difficult to solve efficiently

BUT

there is an efficient algorithm to verify proposed solutions.

Example 2: SATISFIABILITY (SAT)

Boolean formula: variables $x_1, x_2, \ldots x_n$

Operations $\wedge$ (AND)

$\vee$ (OR)

$\neg$ (NOT)

Literal x_i, $\neg x_i$

Clause: AND of literals

$(x_i \vee x_j)$,

$(x_1 \vee \neg x_2)$, $(\neg x_3 \vee x_4 \vee \neg x_5)$

(CNF) formula

OR of clauses

$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_3 \vee x_4) \wedge (x_1) \wedge (x_7 \vee \neg x_8)$

Assignment: a function from variables $\{x_1, \ldots, x_n\}$ to $\{0,1\}$ (True or False)

Value of a literal: $A : \{x_1, \ldots, x_n\} \to \{0, 1\}$

$A(0) = 0$

$A(1) = 1$

$\neg 0 \equiv 1$
$\neg 1 \equiv 0$

$A(\neg x_i) = \neg A(x_i)$

Value of a clause:

$C = y_1 \vee y_2 \vee y_3 \ldots \vee y_r$

y_i – literal.

$A(C) = 1$ if at least one literal $y_i \in C$ s.t $A(y_i) = 1$

$= 0$ Otherwise

CNF formula:

$$\phi \equiv C_1 \wedge C_2 \wedge \cdots \wedge C_L$$

$$A(\phi) = 1 \quad \text{if } \forall i \in [L],\ A(C_i) = 1$$

$$= 0 \quad \text{otherwise}$$

Variables $x_1\ x_2\ x_3$ $\qquad$ A: $x_1 = 0$, $x_2 = 1$, $x_3 = 1$

$$\phi \equiv (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee \neg x_3)$$

$A(\phi) =$

A CNF formula ϕ is said to be satisfiable if there exists an assignment $A : \{x_1, \ldots x_n\} \to \{0,1\}$ s.t $val(\phi)$ on A is 1.

$$\phi_1 \equiv (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (x_1 \wedge \neg x_2)$$

$$\phi_2 \equiv (x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2)$$

Are ϕ_1, ϕ_2 satisfiable?

CNF-SAT : given as input a CNF formula ϕ, output a satisfying assignment (if it exists)

ϕ: n variables, m clauses

A fundamental computational problem both in practice and in theory (will see some reasons)

Naive solution?

Running time?

Again, (as with (minVC)), given an assignment to the variables, there is an efficient algorithm to verify if this is a satisfying assignment.

. There is a certificate asserting that ϕ is satisfiable, that can be efficiently verified.

Example 8: MaxClique

Given a graph G, integer k, output a clique on k vertices

Naive algorithm:

Easily verifiable certificate:

Example 4: Max-flow:

Given a flow network N with integer capacities, integer k, output a feasible s-t flow of value at least k !

Algorithm ?

what about Easy-to-verify-certificates for YES instances?

NP

Non-deterministic polynomial time

- justifying the name takes a fair bit of work - generally done in the course on Theory of computation.

Keywords : Turing Machines

Deterministic Turing Machines

Non-deterministic Turing Machines

Alan Turing

fun fact: Many of Turing's original papers are quite readable.

NP : a problem is in NP if there is an efficiently-verifiable certificate for the yes instances.

Ex: minVC, maxClique, SAT, Maxflow shortest path.

P vs NP

Theorem:

$$P \subseteq NP$$

(Efficiently solvable problems have efficiently verifiable certificates)

Proof:

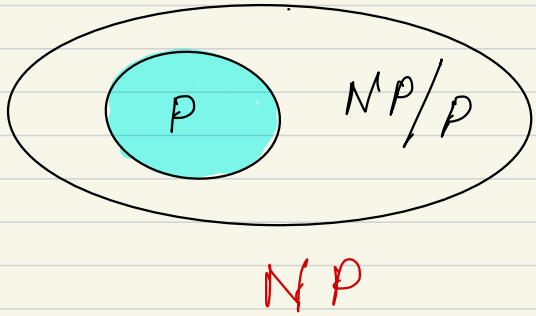
think of Max cut, shortest Path examples.

P vs NP problem

Is $P = NP$?

Can every problem with efficiently verifiable certificates have an efficient algorithm?

Cook's conjecture: $P \neq NP$.



Polynomial Time Reductions

Problem A is polynomial time reducible to problem B if:

There is an efficient algorithm for A that makes function calls to an algorithm for B.

Examples:

1. APSP is polynomial time reducible to single source shortest path.
2. Testing connectivity is poly time reducible to BFS.
3. Reachability is poly time reducible to BFS

④ Topological sort is reducible to DFS

⑤ strongly connected component is poly time reducible to DFS

.
.
.

Reductions are a fundamental paradigm for algorithm design.

Some more examples:

① Yes-No version of minVC poly time reduces to optimization version.

② Yes-No ~~~ max clique ~~~
~~~ optimization version

③ Yes-No ~~~ max flow ~~~
~~~ optimization version.

Q: What about the other direction?

Reductions and computational hardness

A is poly time reducible to B

$\Rightarrow$

(If B has a poly time algorithm, then A has a poly time algorithm.)

Equivalently,

if A does not have a polynomial time algorithm, then B does not have a poly time algorithm.

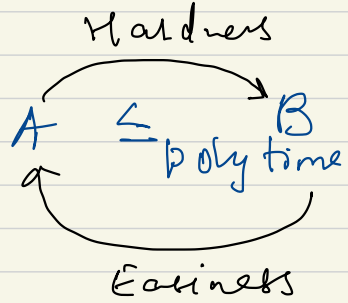
A is poly time reducible to B

$\Rightarrow$

'Easiness' of B $\Rightarrow$ 'Easiness' of A.

Equivalently:

Hardness of A $\Rightarrow$ hardness of B



B is at least as hard as A.

NP-hardness

A problem B is said to be NP hard if every problem in NP is polynomial time reducible to B.

From the earlier discussion:

B has a polynomial time algorithm

$\Leftrightarrow$ P = NP

Theorem (Cook-Levin)

CNF SAT is NP-hard.

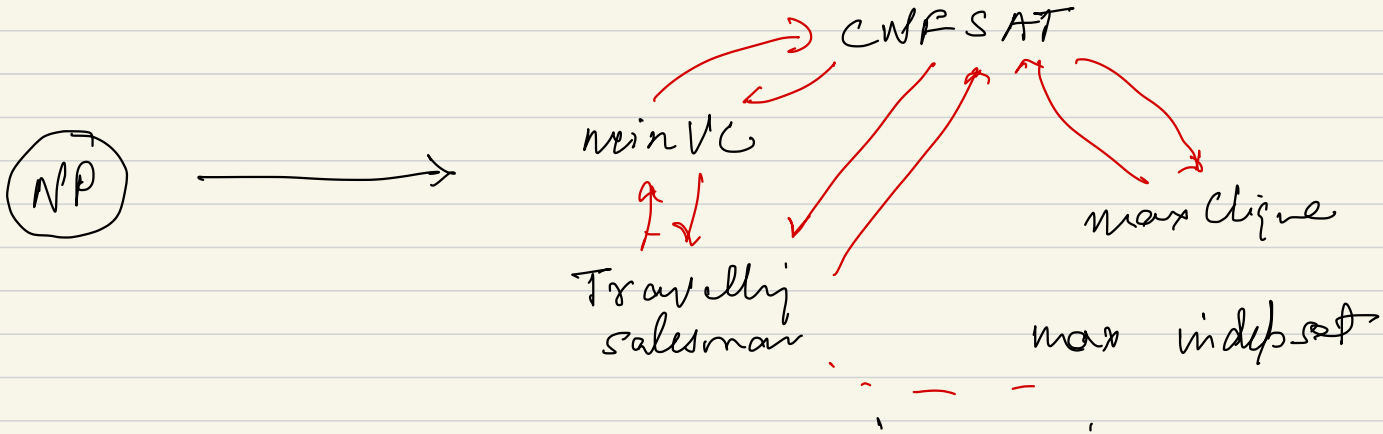
$\Rightarrow$ P vs NP problem is equivalent to understanding whether CNF SAT has a polynomial time algorithm.

In fact:

MinVC is NP-hard

max clique is NP-hard

So, a web of reductions



If one of these has a polynomial time algo, then all of them (and all of NP) has polynomial algo.

Proof of the Cook-Levin Theorem

Won't see it here.

Not difficult, but needs care with formal definitions.

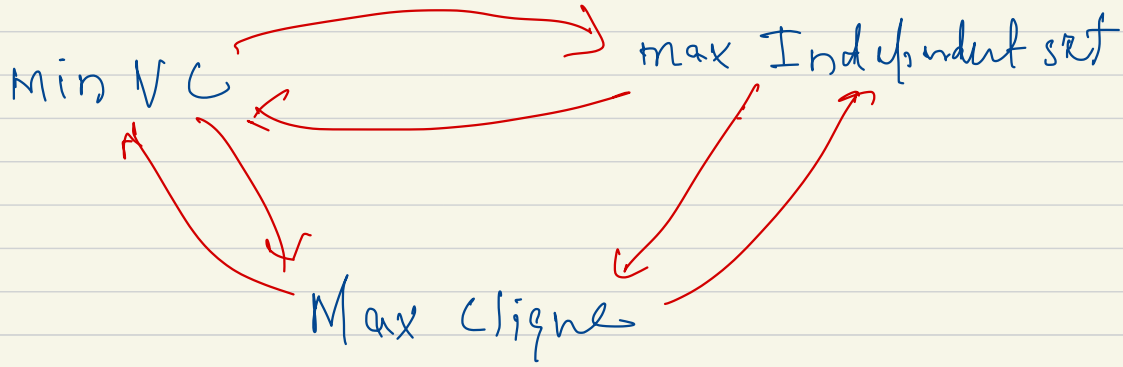
Look up online.

Instead: we will assume the theorem and prove a problem B is NP-hard, will show a reduction from SAT / minVC — / known NP-hard problem to B.

Some reductions

- $\leq_{NP}$ algo.
- Coping with hardness
- Approx
  - Parameterized
  - Randomized
  - Local search
  - ⋮

Examples of reductions



Ind set: $G(V, E)$

$S \subseteq V$ is an independent set if

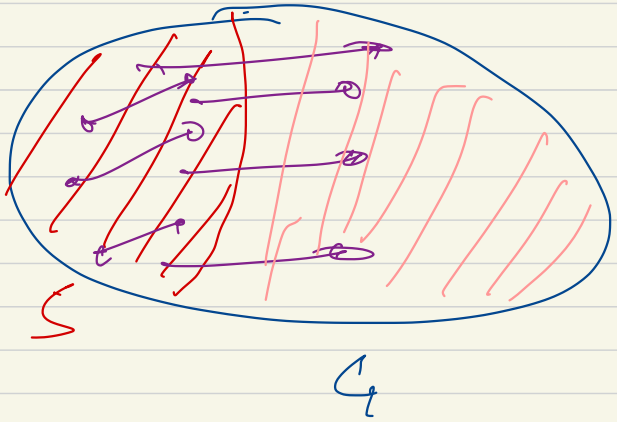
$\forall\ u_1, u_2 \in S$

$(u_1, u_2) \notin E$

① min VC $\longrightarrow$ poly time reducible to max Ind Set.

Pf: - Graph $G(V, E)$, $k \in \mathbb{N}$

- Want to know if G has a VC of size $\leq k$.



$V \setminus S$

S — vertex cover of G

Obs: G has a VC of size $\leq k$

iff

G has an Ind Set of size $\geq n-k$.

Algo for min VC

(1) Invoke Algo for max Ind set with graph G, size parameter $n-k$.

(2) Answer accordingly.

(5) max Ind set poly time reduces to max Clique.

$G(V, E)$

$\bar{G}(V, \bar{E})$ → complement of G

$\forall (u_1, u_2) \in V \times V$,

$(u_1, u_2) \in \bar{E}$ iff $(u_1, u_2) \notin E$

Obs: $G(V, E)$, $\bar{G}(V, E)$ - complement of G.

$\forall S \subseteq V$,

S is an ind set in G

iff

S is a clique in $\bar{G}$.

max Ind set: (G, k)

⓪ construct $\bar{G}$

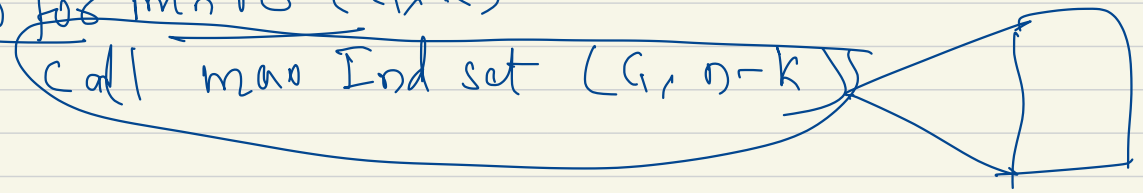
① Invoke a subroutine for max clique on $(\bar{G}, k)$

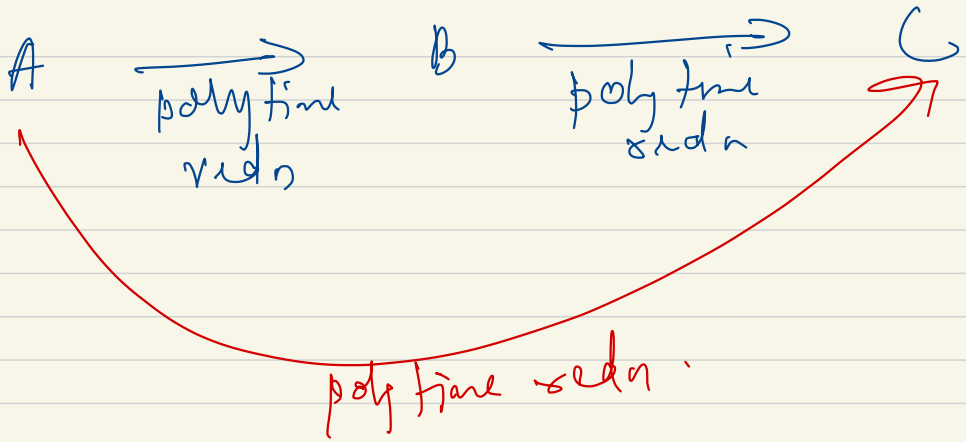
② Answer accordingly

③ min VC $\xrightarrow{\text{poly time reduces}}$ to max clique.

Algo for min VC (G, k)

① Call max Ind set $(G, n-k)$




A
B
C
poly time reduction
poly time reduction
poly time reduction.

(Cook-Levin)

0) SAT is NP-hard

+

1) SAT is poly time reducible to 3SAT

$\Rightarrow$ 3SAT is NP hard

+

2) 3SAT is poly time reducible to max Ind set

$\Rightarrow$ max Ind set is NP hard.

+ Karlin reduc

$\Rightarrow$ min VC, max clique is NP-hard.

Claim: 3SAT is polytime reducible to max Ind Set.

pf: 3SAT:

$(l_{11} \vee l_{12} \vee l_{13}) \wedge (l_{21} \vee l_{22} \vee l_{23}) \wedge \cdots$

$C_1, \ldots, C_m \rightarrow$ 3-clauses.

$C_i \equiv l_{i1} \vee l_{i2} \vee l_{i3}$

$l_{ij} \rightarrow$ literals. $\in \{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$

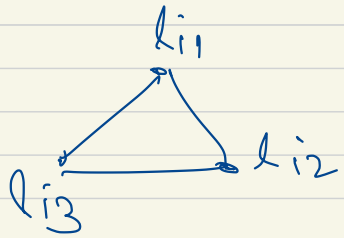
Graph G: $\{l_{ij} : i \in [m], j \in [3]\}$ → Vertices

① Edges:

1.1 clause edges

1.2 conflicting edges.

① $c_i \equiv l_{i1} \vee l_{i2} \vee l_{i3}$



② connect each literal to its complement

$$\begin{cases} l_{ij} = x_i \\ l_{ij'} = \neg x_i \end{cases}$$

Claim: Formula ϕ is satisfiable

iff

G_ϕ has an independent set of size m.

Algo for 3SAT (Assuming the claim)

① Construct the graph G_ϕ

② Check if there is an independent set of size m.

③ Answer accordingly.

Proof of Claim:

① ϕ is satisfiable $\Rightarrow$ ind set of size m.

$\Rightarrow$ ∃ sat assignment s.t for every clause there is a true literal.

Ind set: look at the Δ for each clause pick A vertex labelled by a true literal.

② Ind set of size m $\Rightarrow$ ϕ is satisfiable.