Algorithms (22C:031): Lecture for 11/30

Kasturi Varadarajan

Department of Computer Science, University of Iowa

November, 2010

An Efficient Verifier

- Informally, an efficient verifier for decision problem X is a foolproof mechanism for a computationally bounded entity that a computationally unbounded entity (a prover) can use to convince the verifier of yes-instances of X.

Let us now move to the formal definition starting from this informal one. Keep an example of X in mind, say 3CNF-SAT.

Mechanism

- The mechanism is an algorithm B that takes as two inputs s and t.
- The first input is always an instance s of X.
- The second input t is any proof string
- Think of the action of B as: does the proof t convince me that s is a yes-instance of X?

Foolproof Mechanism

- If s is a no-instance of X, then for every string t, $B(s, t)$ must output "No".

This is a requirement of B that captures the aspect of being foolproof.

Yes Instances

- If s is a yes-instance of X, then for some string t, $B(s, t)$ must output "Yes".

This is the feature of the mechanism that the prover can use to convince the verifier that s is a yes-instance. It simply provides the correct proof/witness t.

Computationally Bounded Verifier

- B must run in time that is polynomial in the sum of the lengths (sizes) of s and t.
- If s is a yes-instance of X, then for some string t whose length is bounded by a polynomial in the length of s, $B(s, t)$ must output "Yes".

The Formal Definition

An efficient verifier for a decision problem X is a polynomial-time algorithm that takes two inputs s and t and outputs "Yes/No", with the property that

- If s is a no-instance of X, then $B(s,t)$ outputs "No" for every t.
- if s is a yes-instance of X, there is a t whose length is bounded by a polynomial in the length of s, for which $B(s,t)$ outputs "Yes".

Efficient verifier for 3CNF-SAT

Our verifier B works as follows: its first input s is a 3CNF-formula; if this has n variables, it

- outputs "Yes" if t is an n-bit 0–1 string that is a satisfying assignment for formula s.
- outputs "No" if t is not an n-bit 0–1 string that is a satisfying assignment for s.

A Bogus verifier for 3CNF-SAT

Our verifier, on input 3CNF-formula s, and t,

- outputs "Yes" if t is the string consisting of the bit "1".
- outputs "No" otherwise.

Why is this not an efficient verifier?

Efficient Verifier for Independent Set

Our verifier, on input $s = \langle G, k \rangle$ and t,

- outputs "Yes" if t encodes a set of vertices in the graph G, and this set is an independent set and has size at least k.
- outputs "No" otherwise.

Problems with (apparently) No Efficient Verifiers

Consider the problem 3CNF-UNSAT:

- yes-instances are 3CNF formulae that are not satisfiable (have no satisfying assignment)
- no-instances are 3CNF formulae that are satisfiable (have at least one satisfying assignment)

Efficiently Solvable Problems have Efficient Verifiers

Let X be a decision problem that has a poly-time algorithm A.
Then an efficient verifer for B is:

- On inputs s and t, B ignores t, runs A on s and outputs $A(s)$.

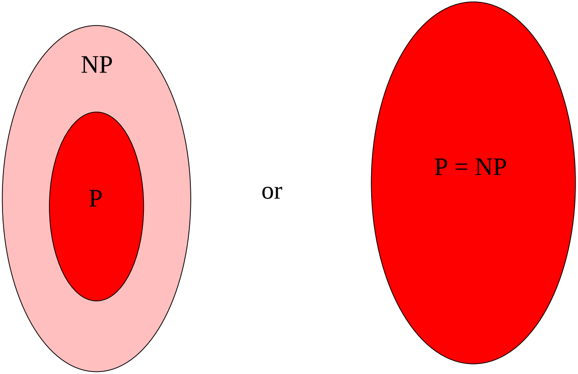
P and NP

- P is the set of all decision problems that have poly-time algorithms.
- Thus, decision versions of weighted interval scheduling, weighted interval covering, and shortest path are in P.
- NP is the set of all decision problems that have efficient verifiers.
- So NP includes not only the above 3 problems and the other known to be in in P, but also ...
- 3CNF-SAT, Independent Set, Colorability, Set Cover, and many other problems we've not looked at.

The $P = NP$ question

- We know that $P \subseteq NP$, but
- Is $NP = P$? That is, are there problems that have efficient verifiers but no efficient algorithms?

The $P = NP$ question



NP-Complete Problems

A decision problem X is said to be NP-complete if

1. $X \in NP$, that is, X has an efficient verifier
2. For every decision problem $Y \in NP$, $Y \leq_P X$ (Y is polynomial time reducible to X)

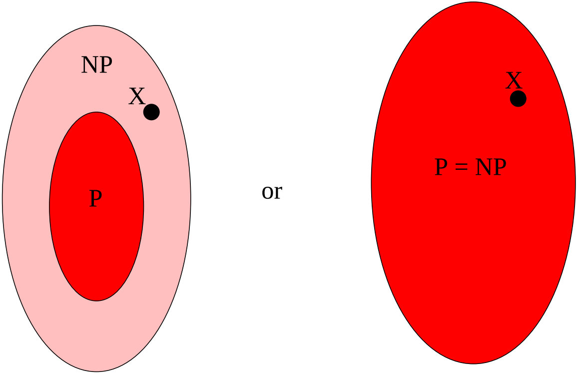
NP-Complete Problems

Claim: Suppose X is NP-complete. Then $X \in P$ implies $NP \subseteq P$.

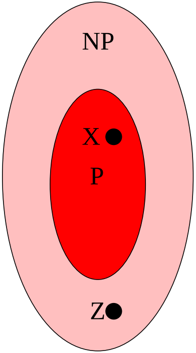
- Proof: Suppose $Y \in NP$. Since X is NP-complete, we know $Y \leq_P X$. Since $Y \leq_P X$ and $X \in P$, we have $Y \in P$.

This claim explains the sense in which NP-complete problems are the hardest ones in NP.

If X is NP-Complete:



If X is NP-Complete, this can't hold:



NP-Completeness

- Notice that if X and Y are two NP-complete problems, then we have $X \leq_P Y$ and $Y \leq_P X$
- Either both problems are in P, or neither is.
- So, all NP-complete problems share the same fate, though we don't know what that fate is.

Excuse Me

That's all very well, but are there actual problems that are NP-complete?

3CNF-SAT is NP-Complete

Theorem: 3CNF-SAT is NP-complete.
To show this, we need to show two things:

- 3CNF-SAT is in NP. We already did that.
- For any $Y \in NP$, $Y \leq_P$ 3CNF-SAT. We won't show this. It has been shown to be true by others, and we'll just assume it, at least for now.

INDEPENDENT SET is NP-Complete

- We need to show INDEPENDENT SET is in NP. We already did that.
- We need to show that for any $Y \in NP$, $Y \leq_P$ INDEPENDENT SET. To do this, we'll simply show 3CNF-SAT $\leq_P$ INDEPENDENT SET.
- This suffices. Why? Let $Y \in NP$. Since 3CNF-SAT is NP-complete, $Y \leq_P$ 3CNF-SAT. Since 3CNF-SAT $\leq_P$ INDEPENDENT SET, and poly-time reducibility is transitive, $Y \leq_P$ INDEPENDENT SET.

NP-Completeness Recipe

In general, to show a brand new problem X to be NP-complete, we will

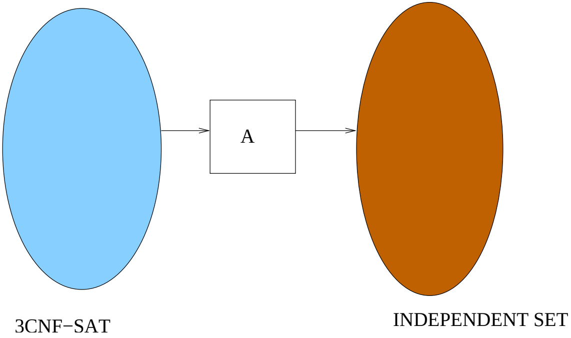
1. show that $X \in NP$. This is typically easy (at least for the homework problems).
2. choose an appropriate known NP-complete problem Z, and show that $Z \leq_P X$. (Not $X \leq_P Z$!!!) This is less easy, but one can become good at it (that's the point of the homework).

3CNF-SAT $\leq_P$ INDEPENDENT SET

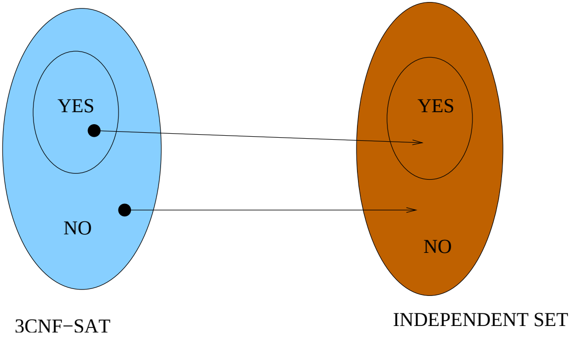
- We need an algorithm, A, that takes as input an instance ϕ of 3CNF-SAT (ϕ is a 3CNF-formula)
- A must output an instance $A(\phi)$ of INDEPENDENT SET
- A must guarantee that ϕ is a Yes-instance of 3CNF-SAT if and only if $A(\phi)$ is a Yes-instance of INDEPENDENT SET

Imagine some $\phi = (x_1 \vee \bar{x_2} \vee x_3), (x_2 \vee \bar{x_3} \vee x_4), \ldots$, with m clauses and n variables.

3CNF-SAT $\leq_P$ INDEPENDENT SET

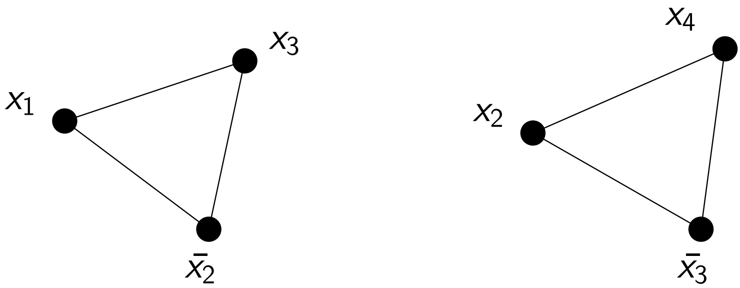


3CNF-SAT $\leq_P$ INDEPENDENT SET



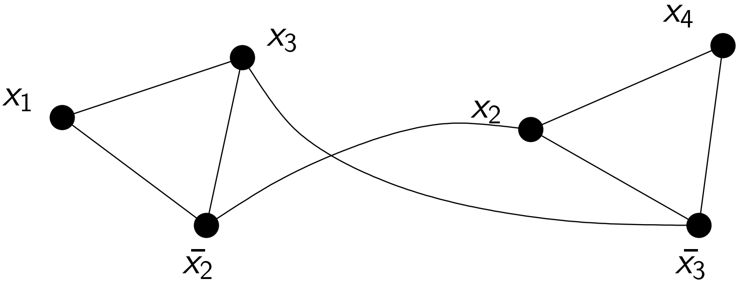
The Algorithm A

- Imagine some input $\phi = (x_1 \vee \bar{x_2} \vee x_3), (x_2 \vee \bar{x_3} \vee x_4), \ldots$, with m clauses and n variables.
- For each clause, A creates 3 vertices, labelled by corresponding literals, and adds edges between them



The Algorithm A

- A adds an edge between two vertices in different clauses if they are labelled by a literal and its complement literal,



The Algorithm A

- This completes the graph construction.
- The INDEPENDENT SET instance $A(\phi)$ that is generated is: Does this graph have an independent set of size at least m (the number of clauses in ϕ)

Yes mapped to Yes

- Suppose ϕ was a satisfiable instance
- We need to argue that the graph constructed has an independent set of size m:
- Fix a satisfying assignment for ϕ.
- It makes true at least one literal in each clause. Pick one such literal from each clause.
- The corresponding vertices in the graph form an independent set of size m.

No mapped to No

- Suppose ϕ was not a satisfiable instance
- We need to argue that the graph constructed does not have an independent set of size m.
- To do this, we'll argue: if the graph does have an independent set of size m, then ϕ is satisfiable.

No mapped to No

- Suppose the graph does have an independent set of size m.
- The independent set cannot have two vertices from the same "clause"
- So the independent set has one vertex from each "clause".
- Take the labels of these vertices
- These literals do not include both x_i and $\bar{x}_i$ for any i.
- Thus there is an assignment that makes these literals true.
- This assignment makes every clause true. Thus, ϕ is satisfiable.