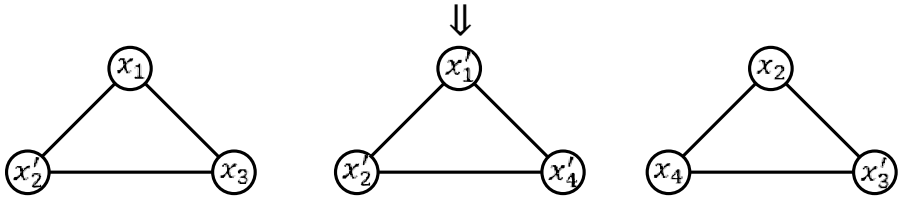
CS 331 NP Notes 2

Section 8.2: Satisfiability Problem

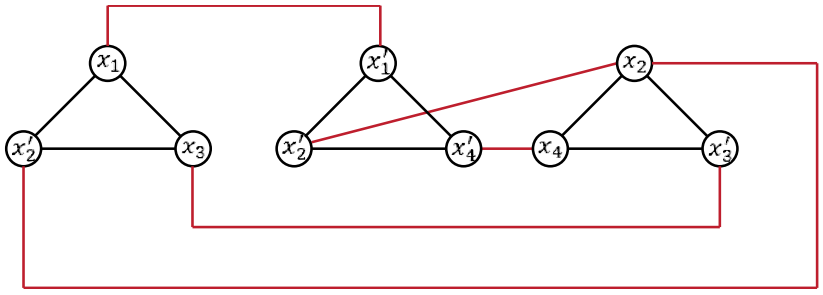
- **Input:** given a set X of n Boolean variables $x_1, x_2, \dots, x_n$, each can take the value 0 or 1 (equivalently to false or true).
- A **clause** is a **disjunction** of **distinct** terms where every term contains the variable x_i or x_i'.
- **F** is a formula consists of **conjunction** of **clauses**.
- E.g. $F = (x_1 \vee x_2 \vee x_3') \wedge (x_4 \vee x_5' \vee x_1) \wedge (x_3' \vee x_4' \vee x_6)$
  - F is satisfiable if we can assign truth values to variables (not literals -- a var or it's negation) to make the entire formula true. In this example ($x_1 = 1$ and $x_2 = x_3 = x_4 = x_5 = x_6 = 0$)
- **Satisfiability problem:** given a set of clauses $C, C_2, \dots, C_k$ over a set of variables $X = \{x_1, x_2, \dots, x_n\}$ is there a satisfying truth assignment?
- More notes
  - The problem contains a set x of n Boolean variables $x_1, x_2, \dots, x_n$ which can each take on the values 0 or 1 only
  - A clause is a disjunction of distinct terms where each term contains either x_i or its negation x_i'
  - X consists of a conjunction of clauses
  - 3-SAT is equivalently difficult to SAT and is easier to think about
  - In 3-SAT each clause has a length of 3
    - i.e. each clause is restricted to EXACTLY 3 literals but we can have any number of clauses
  - 3-SAT problem can be solved by using a brute force algorithm.

Is 3-SAT $\leq_P$ I.S.?

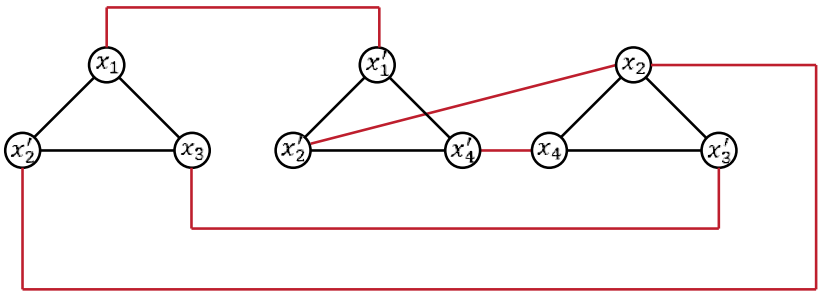
- E.g. $F = (x_1 \vee x_2' \vee x_3) \wedge (x_1' \vee x_2' \vee x_4') \wedge (x_2 \vee x_3' \vee x_4)$
  - To reduce the formula F we need to produce the graph G (below)



  - G consists of a triangle for each clause in F
    - The nodes in each triangle are labeled by the literals in the clause
  - Not done yet, now we need to add edges between all literals and their compliments



- Note: The red lines represent possible conflicts



- Claim: The original 3-SAT formula F is satisfiable if and only if the graph G we have constructed has an independent set of size k where k is equal to the number of clauses.

  (**Observations:** (1) we can choose at most one node from every triangle. Why? since an independent set is a set of nodes such that no two nodes are adjacent. (2) we cannot select two nodes labeled by complimentary literals because they are connected by an edge).

- **Proof:** Given a 3-SAT formula, if you find a satisfiable 3-SAT instance, then each triangle in our graph contains at least one node whose label evaluates to 1. Note that in each clause, you might have more than one true literal, you can go ahead a circle the corresponding nodes on the graph. After you are done circling the nodes that corresponds to true literals, choose one from each triangle in the graph such that no literal and its compliment are connected with an edge.
  Therefore, let S be a set consisting of one such node from each triangle. Since all nodes in S evaluate to 1, it must be the case that S is independent otherwise there would be some arbitrary edge e between two nodes u and v in S. However, if this were true, it would indicate a conflict meaning that one of the nodes in S would have to evaluate to zero; a contradiction.

  Conversely, suppose our graph G has an independent set S of size at least k. However, if this is true it must be the case that the size of S is exactly k and contains one node from each triangle. This occurs because if the size of S were greater than k, it would imply that there were two nodes u and v which belonged to the same triangle and therefore the two would have an edge between them which would imply that S is not independent, a contradiction.

Our claim, is that there is some truth assignment T for the variables in the 3-SAT instance such that the labels of all the nodes in S evaluate to 1. Therefore, to achieve the correct assignment for T we check to see if either x_i or x_i' exists in S. If neither x_i nor x_i' exist in S, arbitrarily set $T(x_i) = 1$. If x_i appears in S we set $T(x_i) = 1$ Otherwise, we set $T(x_i) = 0$. Note, that it cannot be the case that both x_i and x_i' exist in S as by the construction of G there would be an edge between the two and therefore S would not be independent, which is a contradiction.

Therefore, by constructing T in this way it must be the case that all nodes in S will evaluate to 1. Therefore, since G has an independent set of size at least k if and only if the original 3-SAT instance is satisfiable. ■

Wrap up: the following is a reduction **3-SAT $\leq_P$ I.S.**

- It builds up a graph (can be done in polynomial time)
- call the black box once -- feed the graph and the target (# of clauses) into the black box.
- The outcome of the black box is either yes or no.

Lemma 8.9: If $Z \leq_P Y$, and $Y \leq_P X$, then $Z \leq_P X$

- Once again this intuitively makes sense since the time it takes to reduce Z to X would be the time taken to reduce Z to Y compounded with the time taken to reduce Y to X.
    - i.e. suppose you can reduce Z to Y using the function $f(|Z|)$ and that you can reduce Y to X using the function $g(|Y|)$. Therefore, to reduce Z to X you could simply call $f\big(g(|Y|)\big)$

P, NP, and NP-Complete

$\mathcal{P}$ – A class of problems which can be solved in polynomial time. The set of all decision problems which are solvable by an algorithm[1] whose worst case running time is bounded by some polynomial function of the input size

Unfortunately not all problems are solvable by a polynomial time algorithm. Therefore, they are not in class P. For example, Independent Set, Vertex Cover, and 3-SAT. There are many problems that are not known to be in P.

$\mathcal{NP}$ –The problems which can be *verified* using a polynomial time algorithm

- Verifiable means that given a certificate (i.e. a solution), we could verify that the certificate is correct in polynomial time

Lemma 8.10: $\mathcal{P} \subseteq \mathcal{NP}$

- Any problem which can be solved in polynomial time can be verified in polynomial time.

Lemma 8.11: Is there a problem in $\mathcal{NP}$ that does not belong to $\mathcal{P}$? Does $\mathcal{P} = \mathcal{NP}$?

- We don't know
- General belief is that $\mathcal{P} \neq \mathcal{NP}$ – though there is no actual evidence to support this
- $\mathcal{NP}$ = Non-Deterministic Polynomial (Does NOT stand for Non-Polynomial!)

In the absence of progress on the P VS NP question, people turned to a realted question but more approachable question: **what are the hardest problems in NP?**

$\mathcal{NP}$-Complete–A sub-class of $\mathcal{NP}$, the Hardest problems in $\mathcal{NP}$.

- A Set of hard problems such that all other $\mathcal{NP}$-Complete problems can be reduced to these in polynomial time.
- A problem X is $\mathcal{NP}$-Complete if and only if
  - $X \in \mathcal{NP}$
    - Can be verified in polynomial time
  - For all $Y \in \mathcal{NP}$, $Y \leq_P X$
    - i.e. Every problem in $\mathcal{NP}$ can be reduced to X -- X is very expressive. If you have a problem $Y \in \mathcal{NP}$, then you can solve it using a black box for problem X.

Lemma 8.12: Suppose X is an $\mathcal{NP}$-Complete problem. Then X is solvable in polynomial time if and only if $\mathcal{P} = \mathcal{NP}$.

Proof:

$\rightarrow$: Suppose $\mathcal{P} = \mathcal{NP}$, then X can be solved in polynomial time since it belongs to $\mathcal{NP}$.

$\leftarrow$: Suppose X is solvable in polynomial time. Given that X is also an $\mathcal{NP}$-Complete problem, take an arbitrary problem Y $\in \mathcal{NP}$, then by definition Y $\leq_P$ X. This means that Y can also be solved in polynomial time -- as stated by lemma 8.1. ■

[1] the algorithm is assumed to deterministic and always correct on all input.

- **Consequence of Lemma 8.12:** If there is any problem in $\mathcal{NP}$ that cannot be solved in polynomial time, then no $\mathcal{NP}$-Complete problem can be solved in polynomial time massive effort has been invested to come up with a polynomial time algorithm for $\mathcal{NP}$-Complete problems and everybody has failed so far. , Independent Set, Vertex Cover, and 3-SAT are all $\mathcal{NP}$-Complete problems.
  If a person worked on SAT trying to find a polynomial time algorithm and failed and another person worked on Independent Set trying to find a polynomial time algorithm and failed. The result is two separate effort provides a combined evidence that $\mathcal{P} \neq \mathcal{NP}$.
  If you have a polynomial time algorithm for one $\mathcal{NP}$-Complete problem, you have a polynomial time algorithm for all $\mathcal{NP}$-Complete problems. So, either every $\mathcal{NP}$-Complete problem can be solved in polynomial time or none can.

- $\mathcal{NP}$-Hard problems are at least as hard as $\mathcal{NP}$ problems
  - Do not have to be $\mathcal{NP}$
  - Do not have to be decision problems

- How to show that a problem $X \in \mathcal{NP}$-Complete?
  - show that X $\in \mathcal{NP}$
  - Choose a known $\mathcal{NP}$-Complete problem Y and show that $Y \leq_P X$

  For example, we have to use one known $\mathcal{NP}$-Complete problem such as 3-SAT and we want to use it to show that the problem X is $\mathcal{NP}$-Complete. We know that every problem in $\mathcal{NP}$ can be reduced in polynomial time to 3-SAT. The 3-SAT is $\mathcal{NP}$-complete. If $3 - SAT \leq_P X$, then by transitivity all $\mathcal{NP}$ problems can be reduced to X.