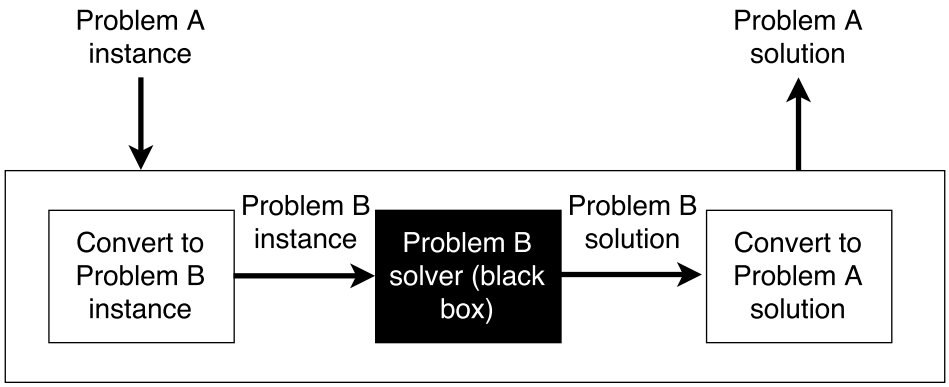
CPSC 320: NP-completeness, SAT and 3SAT Solutions*

We often use reductions to solve new problems based on problems we can already solve. For example, in an earlier worksheet, we saw a reduction from the Resident Hospital problem to the Stable Matching problem.



But... there's another way to use reductions. A more **sinister** way.[1] We'll illustrate this using a famous problem from logic: Satisfiability.

1 Boolean Satisfiability

Boolean satisfiability (SAT) is—as far as Computer Scientists know—a hard problem, in the sense that no-one knows of an algorithm to solve SAT that has worst-case polynomial runtime. In more technical terms, SAT is *NP-complete*. Recall that this problem is defined by:

> The input is a collection of m *clauses* over n boolean variables X_1, X_2, ... X_n. Each clause is a disjunction of some of the variables or their complements.
>
> The problem consists in answering the question "Is there a way to assign truth values to each variable that makes **every** clause of the instance TRUE?"

If there is an assignment of truth values that makes every clause of a SAT instance TRUE, then we call the instance *satisfiable*. Here is a sample instance of SAT:

$$X_1 \vee \overline{X}_2 \vee X_3 \vee X_4$$
$$X_5$$
$$\overline{X}_1$$
$$X_2 \vee \overline{X}_3 \vee \overline{X}_5$$
$$\overline{X}_2 \vee X_3$$

For convenience, we will insist on using the variables $X_1, X_2, \ldots X_n$ for some n, without skipping any. Given an instance I of SAT, we want to know: is the instance I satisfiable or not? The answer is either Yes or No, so we call this a *decision problem*.

[1] Well, OK. Just **another** way.

1. Is the example SAT instance above satisfiable? If not, explain why not. If so, prove it by giving an assignment that makes the statement true.

   SOLUTION:

   Yes, this is satisfiable. Solving it by hand: we need $X_5 = T$ and $X_1 = F$. Once X_5 is true, then the last two clauses indicate that X_2 and X_3 have to have the same truth values. Assigning both true or both false will also handle the first clause. So, for example: $X_1 = F, X_2 = T, X_3 = T, X_4 = T, X_5 = T$ is a truth assignment that satisfies the clause.

2. For a SAT instance I, a truth assignment is a potential solution, and the truth assignment is a *good* solution if it satisfies instance I.

   Suppose in addition to instance I you were given a truth assignment, say represented as an array $T[1..n]$ where $T[i]$ is true if and only if X_i is set to true. How long would it take to certify that truth assignment T is good?

   SOLUTION: We should be able to plug in the truth values to the clause literals and evaluate whether all clauses are true in time that is linear in the length of the instance I. We can scan the clauses from first to last, and for each, check whether at least one literal is true. We can check the truth assignment to each literal in $O(1)$ time by looking in the array. The truth assignment is good if and only if all clauses are satisfied.

   Note: We've just described a *certification* (or *verification*) algorithm for SAT. A certification algorithm takes as input an instance of a problem, plus a potential solution, and checks whether or not the solution is good. If a decision problem has a certification algorithm that runs in polynomial time (as a function of the given instance size), **the problem is in the class NP**. So, we've just shown that SAT is in NP.

3. A brute force algorithm could make a list of the variables $X_1, \ldots, X_n$ in the problem, try every assignment of truth values to these variables, and return `YES` if any satisfies the expression or `NO` otherwise. Asymptotically, how many truth assignments might this algorithm try (in terms of n)?

   SOLUTION: Brute force would need to try 2^n different truth assignments in the worst case. (Note that trying one truth assignment may take more than constant time. If it takes linear time, as in our algorithm above, the brute force algorithm takes $\Omega(n2^n)$ time to run, in the worst case!)

2 3-SAT and SAT

The 3-SAT problem is just like SAT, except that **every** clause must be **exactly** of length 3. Because it's a bit more restricted than SAT, 3SAT is a better problem to use for NP-completeness reductions (recall that we will reduce **from** 3SAT **to** the problem we want to prove is NP-complete). Let's build a reduction from SAT to 3-SAT (so we're solving SAT in terms of 3-SAT). We'll map an instance I of SAT to an instance I' of 3-SAT, working on one clause at a time. Importantly, for our reduction to work, I should be satisfiable if and only if I' is satisfiable.

1. Suppose that I has a clause with two literal, say $(\overline{X}_2 \vee X_3)$. To obtain I' from I, we want to replace this clause by one or more clauses, while ensuring that I is satisfiable if and only if I' is. How can we do this? Hint: introduce a new variable Y.

   SOLUTION: We replace $(\overline{X}_2 \vee X_3)$ by the two clauses $(\overline{X}_2 \vee X_3 \vee Y)$ and $(\overline{X}_2 \vee X_3 \vee \overline{Y})$. A truth assignment that satisfies I must also satisfy I' because clause $\overline{X}_2 \vee X_3$ appears in both clauses of I'. Also, any truth assignment that satisfies I' must set one of $\overline{X}_2$, X_3 to True, thus satisfying I, because no matter which values we assign to Y, the literals in Y in one of the two clauses will be false, requiring $\overline{X_2} \vee X_3$ to be true.

2. What if I has a clause with only one literal, say (X_5)? Hint: introduce two new variables Y and Z.

   SOLUTION: To obtain I' from I, we replace (X_5) with the four clauses $(X_5 \vee \overline{Y} \vee \overline{Z})$, $(X_5 \vee \overline{Y} \vee Z)$, $(X_5 \vee Y \vee \overline{Z})$ and $(X_5 \vee Y \vee Z)$. A truth assignment that satisfies I must also satisfy I' because variable X_5 appears in every one of the four clauses of I'. Also, any truth assignment that satisfies I' must set X_5 to True, thus satisfying I, because no matter which values we assign to Y and Z, the literals in Y and Z in one of the four clauses will both be false, requiring X_5 to be true.

3. Now suppose that I has a clause with four literals, say $(X_1 \vee \overline{X}_2 \vee X_3 \vee X_4)$. What 3-SAT clauses will you put in I' to replace this clause, so that I' is satisfiable if and only if I is? Hints: Break the clause up somehow. Don't try using de Morgan's laws. Instead, create a brand new variable, say Y, and integrate that into your new clauses.

   SOLUTION: Replace the clause above by $(X_1 \vee \overline{X}_2 \vee Y) \wedge (\overline{Y} \vee X_3 \vee X_4)$.

4. For your construction of part 3, show that if I is satisfiable then I' must also be satisfiable (and modify your construction if needed to ensure this).

   SOLUTION: Suppose that I is satisfiable. Then in a truth assignment that satisfies I, at least one of the literals X_1, $\overline{X}_2$, X_3, or X_4 must be true. If one of the first two literals (i.e., X_1 or $\overline{X}_2$) is true, then the first of our two new clauses is satisfied. We can make sure that the second clause, and thus all of I', is satisfied by choosing Y to be false. Alternatively if neither of the first two literals is true, then we set Y to be true, to ensure that the first of our two clauses is satisfied. We can rest assured that the second clause is also satisfied because at least one of X_3, or X_4 must be true.

5. Also for your construction of part 3, show that if I' is satisfiable then I must also be satisfiable.

   SOLUTION: Suppose that I' is satisfiable. A truth assignment that satisfies I', with Y removed, must also satisfy I. Specifically, if Y is set to true in I''s satisfying truth assignment, we know that at least one of X_3 or X_4 must be true (since $\overline{X}_{n+1}$ is false). And if Y is set to false in the satisfying truth assignment, at least one of X_1 or $\overline{X}_2$ must be true. Either way, our original clause $(X_1 \vee \overline{X}_2 \vee X_3 \vee X_4)$ must be true, and since all other clauses of I and I' are identical, all other clauses of I must be satisfied too.

6. Extend your 4-literal clause plan above to a 5-literal clause like $(X_1 \vee \overline{X}_2 \vee X_3 \vee X_4 \vee \overline{X}_5)$.

   SOLUTION: We can introduce two new variables, say Y and Z, and replace the above 5-literal clause by

   $$X_1 \vee \overline{X}_2 \vee Y$$
   $$\overline{Y} \vee X_3 \vee Z$$
   $$\overline{Z} \vee X_4 \vee \overline{X}_5$$

7. Show, by filling in the blanks below, how you would transform any clause with $k > 3$ literals

   $$(l_1 \vee l_2 \vee \ldots \vee l_k)$$

   into clauses with three literals (keeping in mind overall reduction correctness). You can use new variables that have not already been "used up", starting with X_{i+1} (where $i \geq n$). How many clauses do you get? What would be the runtime of an algorithm to do this, as a function of k?

   SOLUTION:

   $$(l_1 \vee l_2 \vee X_{i+1}) \wedge (\bar{X}_{i+1} \vee l_3 \vee X_{i+2}) \wedge (\bar{X}_{i+2} \vee l_4 \vee X_{i+3}) \wedge \ldots$$

$$\wedge\ (\overline{X}_{i+(j-2)} \vee l_j \vee X_{i+(j-1)})\ \wedge \ldots$$
$$\ldots \wedge\ (\bar{X}_{i+(k-4)} \vee l_{k-2} \vee X_{i+(k-3)}) \wedge (\bar{X}_{i+(k-3)} \vee l_{k-1} \vee l_k)$$

There are $k-2$ new clauses in total, using $k-3$ new variables $X_{i+1}, X_{i+1}, \ldots, X_{i+(k-3)}$ to "link" the clauses together. Overall, an algorithm to do this takes time $\Theta(k)$.

8. Let's use the name TRANSFORM-CLAUSE to refer to the algorithm for transforming a clause, as described in part 7. Suppose that I' is obtained from I by transforming clause $(l_1 \vee l_2 \ldots \vee l_k)$ using algorithm TRANSFORM-CLAUSE. Explain why I is satisfiable if and only if I' is satisfiable.

   SOLUTION: This generalizes our arguments in parts 4 and 5

   If direction: First suppose that I is satisfiable, and choose a satisfying truth assignment to the variables $X_1 \ldots X_n$. We need to extend this to a satisfying truth assignment for I', by assigning truth values to the newly created variables $X_{i+1}, \ldots X_{i+(k-3)}$.

   Suppose that l_j is the first literal in the clause $(l_1 \vee l_2 \ldots \vee l_k)$ which is true with respect to the chosen truth assignment. Then we set $X_i, \ldots, X_{i+(j-2)}$ to true. These ensure that all clauses created *before* clause

   $$(\overline{X}_{i+(j-2)} \vee l_j \vee X_{i+(j-1)})$$

   are true. Literal l_j ensures that clause $(\overline{X}_{i+(j-2)} \vee l_j \vee X_{i+(j-1)})$ is true. So, we can set the remaining newly created variables $X_{i+(j-1)}, \ldots, X_{i+(k-3)}$, to false, ensuring that the remaining newly created clauses are true. This ensures that all clauses in I' are true.

   Only if direction: Conversely suppose that I' is satisfiable. A truth assignment obtained by removing the newly created variables from a satisfying truth assignment of I' must satisfy all clauses of I other than $(l_1 \vee l_2 \ldots \vee l_k)$, since all of these clauses are also in I'.

   Suppose that $(l_1 \vee l_2 \ldots \vee l_k)$ is not satisfied by the chosen truth assignment. Then all of the l_i are false. In this case, in the truth assignment satisfying I', it **must** be that all of $X_{i+1}, \ldots, X_{i+(k-3)}$ are true, in order to satisfy the first $k-3$ newly created clauses. In this case, the last clause, namely $(\overline{X}_{i+(k-3)} \vee l_{k-1} \vee l_k)$, will not be satisfied. This contradicts the fact that I' is satisfied by the chosen truth assignment, and so we conclude that $(l_1 \vee l_2 \ldots \vee l_k)$ is indeed satisfied by the chosen truth assignment.

9. Give a reduction from SAT to 3-SAT. Recall that a reduction consists of two algorithms that "connect" one problem to another, as in the diagram from page 1.

   SOLUTION:

   Transform instance algorithm: This algorithm converts an instance I of SAT to an instance I' of 3-SAT, working clause by clause, by calling the TRANSFORM-CLAUSE algorithm described earlier on clauses of size greater than three, and using the transformations of parts 1 and 2 for clauses of sizes one and two respectively.

For instance:

SAT instance		3SAT instance
$X_3 \vee \overline{X_5}$	$\Rightarrow$	$\begin{cases} X_3 \vee \overline{X_5} \vee Y_1 \\ X_3 \vee \overline{X_5} \vee \overline{Y_1} \end{cases}$
$\overline{X_1} \vee X_2 \vee X_4 \vee X_5$	$\Rightarrow$	$\begin{cases} \overline{X_1} \vee X_2 \vee Y_2 \\ \overline{Y_2} \vee X_4 \vee X_5 \end{cases}$
$\overline{X_4}$	$\Rightarrow$	$\begin{cases} \overline{X_4} \vee \overline{Y_3} \vee \overline{Y_4} \\ \overline{X_4} \vee \overline{Y_3} \vee Y_4 \\ \overline{X_4} \vee Y_3 \vee \overline{Y_4} \\ \overline{X_4} \vee Y_3 \vee Y_4 \end{cases}$
$X_1 \vee \overline{X_2} \vee \overline{X_3} \vee X_4 \vee \overline{X_5} \vee \overline{X_6}$	$\Rightarrow$	$\begin{cases} X_1 \vee \overline{X_2} \vee Y_5 \\ \overline{Y_5} \vee \overline{X_3} \vee Y_6 \\ \overline{Y_6} \vee X_4 \vee Y_7 \\ \overline{Y_7} \vee \overline{X_5} \vee \overline{X_6} \end{cases}$

Transform solution algorithm: This algorithm to transform a solution (YES or NO) to the 3-SAT instance I' back to a solution for I simply uses the exact same solution (YES OR NO).

Algorithm SAT-to-3-SAT(I) takes time linear in the length of I, and the algorithm to translate the solution back takes $O(1)$ time.

10. Why is the reduction correct?

    SOLUTION: We've shown in the previous parts that when we transform clauses with one literal, two literals, or at least four literals, we preserve satisfiability: I is satisfiable if and only if I' is. Since our algorithm transforms one clause at a time, it preserves satisfiability at every step, so the final 3-SAT instance I' is satisfiable if and only if I is. As a result, the second algorithm that uses the solution for I' as the solution for I produes the correct answer for I.

3 What does a reduction tell us?

Here, consider a reduction from problem A to problem B, as illustrated in the figure of page 1.

1. **SCENARIO #1 (how we've used reductions prior to this worksheet):** Say our reduction's two algorithms take $O(f(n))$ time and the black box solver for B also takes $O(f(n))$ time. What can we say about the running time to solve problem A?

   SOLUTION: In this scenario, the reduction and the solution to problem B problem together yield an $O(f(n))$ solution to problem A. (This is how we'd normally use reductions in practice!)

2. **SCENARIO #2 (what we usually think of NP-completeness as meaning):** Say our reduction's two algorithms take $O(g(n))$ time and we know that there is **no algorithm** for problem A that runs in $O(g(n))$ time. What do we know about the running time for problem B? Why?

   SOLUTION: In this situation, there canot be an algorithm for problem B that runs in $O(g(n))$ time. Why not? Well, say there were, then as in the previous part, we could use that to generate an $O(g(n))$ solution for problem A, but we said no such algorithm exists!

If we assume P $\neq$ NP, this is how we use reductions in an NP-completeness proof. (Although our use is technically more similar to what we do in the next part!)

3. **SCENARIO #3 (what NP-completeness technically means):** Say that we know (which we do) that if SAT can be solved in polynomial time, then **any** problem in the large set called "NP" can also be solved in polynomial time. What does our reduction from SAT to 3-SAT tell us? Why?

   SOLUTION: It tells us that if 3-SAT can be solved in polynomial time, then so too can **any** problem in NP be solved in polynomial time.