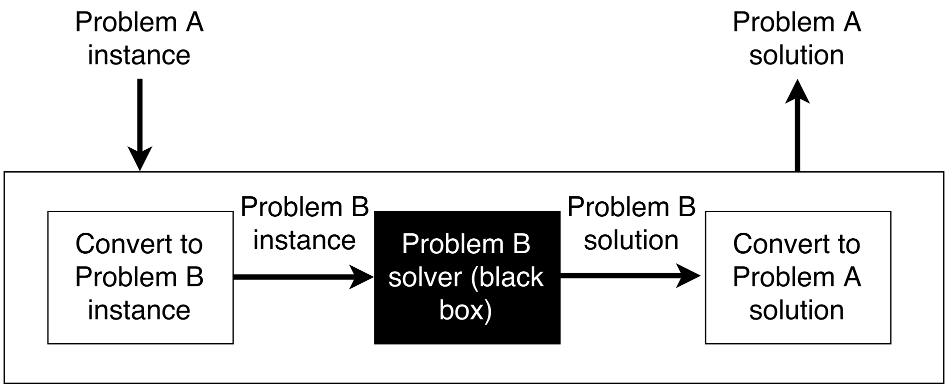
CPSC 320: NP-completeness, SAT and 3SAT*

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, $\ldots X_n$. 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.*

*Copyright Notice: UBC retains the rights to this document. You may not distribute this document without permission.

[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.

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?

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)?

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.

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

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.

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).

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

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

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?

   $$(l_1 \vee l_2 \vee X_{i+1}) \ \wedge \ (\bar{X}_{i+1} \vee l_3 \vee \underline{\hspace{1.5cm}}) \wedge (\bar{X}_{i+2} \vee l_4 \vee X_{i+3}) \wedge \ldots$$

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

   $$\wedge \ (\bar{X}_{i+(k-4)} \vee l_{k-2} \vee \underline{\hspace{1.5cm}}) \wedge (\bar{X}_{i+(k-3)} \vee l_{k-1} \vee \underline{\hspace{1.5cm}}).$$

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.

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.

   Transform instance algorithm:

   Transform solution algorithm:

10. Why is the reduction correct?

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?

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?

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?