## 1 Introduction

The next objective of our development will be to find a single concrete problem that we can show is **NP**-complete. Once we have that, then we can use the mechanisms of reducibility to show many other problems are **NP**-complete (by reducing the initial example to them).

The issue is that getting the ball rolling is a little tough since the first time around we have to show that *every* problem in **NP** can be reduced to the problem that we're considering. That problem is going to be SATISFIABILITY or, as it is usually known, SAT. So let's begin by introducing that.

## 2 Propositional Logic

In order to define SAT we need to (re?)-introduce some basic terminology from propositional logic.

A *Boolean variable* $x$ is a variable that takes on values t or f (for true and false, obviously). There are a number of operations defined on Boolean variables to form *Boolean-valued expressions*. So, in the following $x$ and $y$ might refer to variables or more complex expressions (defined recursively in the usual way).

*Negation* of Boolean-valued expressions

| $x$ | $\neg x$ |
|---|---|
| t | f |
| f | t |

*Disjunction* of Boolean valued expressions:

| $x$ | $y$ | $x \vee y$ |
|---|---|---|
| t | t | t |
| t | f | t |
| f | t | t |
| f | f | f |

*Conjunction* of Boolean valued expressions:

| $x$ | $y$ | $x \wedge y$ |
|---|---|---|
| t | t | t |
| t | f | f |
| f | t | f |
| f | f | f |

A *clause* is a disjunction of Boolean variables and negated Boolean variables.

A *formula in conjunctive normal form* (CNF) is a conjunction of clauses.

A *truth assignment* for a set of Boolean variables $V$ is a map:

$$V \to \{\mathsf{t}, \mathsf{f}\}.$$

A formula is *satisfiable* if it evaluates to t for *some* truth assignment.

Two formulas over the same set of variables are *equivalent* if, for any truth assignment of the variables, their values are the same. There are a number of basic equivalences that are easily checked using truth-tables (left for the tutorial problems). These include the distributive laws and DeMorgan's laws:

$$\begin{aligned} a \wedge (b \vee c) &\text{ is equivalent to } (a \wedge b) \vee (a \wedge c),\\ a \vee (b \wedge c) &\text{ is equivalent to } (a \vee b) \wedge (a \vee c),\\ \neg(a \vee b) &\text{ is equivalent to } \neg a \wedge \neg b,\\ \neg(a \wedge b) &\text{ is equivalent to } \neg a \vee \neg b. \end{aligned}$$

Using these, one can show that every Boolean formula is equivalent to one in conjunctive normal form (also left for the tutorial problems). However, the size of the CNF-formula can be exponentially larger than the original.

## 3 Satisfiability

Recall that our aim is to find a single concrete example of an **NP**-complete problem. Once we've got that, we can hope to construct a catalogue of **NP**-complete problems by

finding other problems in **NP** that it reduces to. That problem is:

SATISFIABILITY or SAT
*Instance*: A CNF-formula over a set of variables $V$.
*Problem*: Does the formula have a satisfying assignment?

Let's begin by convincing ourselves that SAT is in **NP**. First of all, how do we represent instances of SAT in a Turing machine? I'll do this in relatively fine-grained detail, but it should be clear – we could imagine doing it in a text file. Perhaps the first line specifies the number of variables and then each additional line represents a clause of the SAT instance using the index of the variable to stand for itself and a negative index to stand for the negated variable. Thus, for instance (with variables $x_0$ through $x_7$):

$$(x_0 \vee x_2 \vee \neg x_6) \wedge (x_1 \vee \neg x_3 \vee x_4 \vee \neg x_6) \wedge (x_1 \vee x_3 \vee x_5) \wedge (x_0 \vee \neg x_7)$$

would become:

```
8
0 2 -6
1 -3 4 -6
1 3 5
1 -7
```

So, how much space do we need to represent an instance of SAT? If there are $n$ possible variables, then we need $O(\log n)$ space to represent a variable. That $O$ can also absorb the extra space we need for the negative sign, the space between variables, and the line breaks between clauses. We should never repeat a variable in a clause (since $x \vee x$ and $x$ are equivalent) nor include both a variable and its negation in a clause (since $x \vee \neg x$ is always true, so all such clauses are automatically satisfied). So each clause can be represented in $O(n \log n)$ space. Therefore, the total amount of space we need is $O(kn \log n)$ where $k$ is the number of clauses.

The plan for showing that SAT is in **NP** is simply to have the genie write down $n$ bits on the oracle tape representing a satisfying assignment if there is one. How long does it take us to check?

We need to process each line (after the first) of the input file. For each variable index we see, we need to go look it up on the oracle tape (time $O(n)$) and if it creates a satisfying

assignment for this clause we can move on to the next one. If not, we need to look up the next variable etc. If the clause fails to be satisfied we can terminate (and reject). In worst case it's always the last variable on each line that winds up satisfying the clause so we spend $O(n^2)$ time per clause and $O(kn^2)$ time in total. Regardless of the relationship between $k$ and $n$ this is polynomial in the input size since for instance

$$(kn \log n)^2 > kn^2.$$

Therefore, the time required to check is at worst quadratic in the size of the input.

So,

**Theorem 3.1.** SAT *is in* **NP**.

Why is it **NP**-complete? That's a wee bit harder.

## 4 Making a plan

Since we already know that SAT is in **NP** all we need to do now is to prove that it's **NP**-hard. To do this we must establish a polynomial time reduction from *any* problem in **NP** to SAT. That's a daunting task:

- Where can we start?
- What's the handle?
- How do we turn the crank?

Let's look at what we have to work with: a language $L \in \mathbf{NP}$ (whose decision-problem we want to reduce to deciding SAT). Since this language is in **NP** it comes with a non-deterministic Turing machine $M$ that accepts $L$ and has a time bound $An^c$ for inputs of length $n$.

**That's all we know!**

We're thinking about non-determinism in the "oracle tape" model, the underlying operation of $M$ is deterministic. It's just the contents of the oracle tape that are "free". As $M$ runs we could imagine taking a snapshot of its configuration at each time step.

What's a snapshot?

- A record of the position of the read-write heads (main tape, oracle tape)
- The current state
- The complete contents of the tapes

That's a discrete pile of information which we can take to be of polynomially-bounded size in the original intput (since we're only going to have $M$ run for $An^c$ steps we only need to consider $An^c$ cells on each tape - the rest are inaccessible).

So, we need to describe an encoding of such a snapshot in Boolean variables and also describe a way to enforce consistency (both of the individual images and the relationships between them) by means of a CNF-formula of polynomially bounded size.

And that's what we'll do next time.

## 5 Tutorial problems

1. Which of the following formulas are satisfiable?
   (a) $(x \vee y) \wedge (\neg x \vee \neg y)$
   (b) $(x \vee y) \wedge (\neg x \vee \neg y) \wedge (\neg x \vee y)$
   (c) $(x \vee y) \wedge (\neg x \vee \neg y) \wedge (\neg x \vee y) \wedge (x \vee \neg y)$
   (d) $(\neg a \vee b \vee c) \wedge (a \vee \neg b) \wedge (a \vee \neg c)$
2. Verify that the following pairs of formulas are equivalent:
   (a) $a \wedge (b \vee c)$ and $(a \wedge b) \vee (a \wedge c)$,
   (b) $a \vee (b \wedge c)$ and $(a \vee b) \wedge (a \vee c)$,
   (c) $\neg(a \vee b)$ and $\neg a \wedge \neg b$,
   (d) $\neg(a \wedge b)$ and $\neg a \vee \neg b$.
3. Show that for any Boolean formula $\Phi(x_1, x_2, \ldots, x_k)$ there is an equivalent formula $\Psi(x_1, x_2, \ldots, x_k)$ in conjunctive normal form.
4. Find a CNF-formula equivalent to
   $$(a \wedge b) \vee (c \wedge d) \vee (e \wedge f)$$
   This should suggest why we can have an exponential blow-up in size when we convert to CNF (though it doesn't prove it as it stands – that's a little technical, since you need to provide a lower bound on the length of the shortest CNF-formula equivalent to a given formula.