Hardness[1]

What should we do when we *cannot* find an efficient algorithm for a problem Π? Ideally, we would like to then show that no one else can find an algorithm for Π because it is *mathematically impossible* to design a fast algorithm which will always work correctly. Unfortunately, we are faaaaaaaaaaaar from proving any such theorem. Instead, in computer science we have recognized a *huge* collection of difficult problems for which we *believe* that there are no fast algorithms and maybe one day in the future someone will prove this belief correct. What is fascinating about this collection is that the problems in this collection are all "equivalent", in that if (a) one proves there are not polynomial time algorithms for any one of these problems, it automatically proves hardness of *all* of these problems, and conversely, (b) if one finds a polynomial time algorithm for any one of these problems, it automatically implies polynomial time algorithms for *all* of these problems. Reflect on this rather marvelous situation.

This collection of problems are called[2] ***NP-complete problems***. And as of now, there are *hundreds* of problems in this collection.

There is a whole course (COSC 39) at Dartmouth which builds the theory of computation rigorously culminating in the theory of NP-completeness. We have around an hour. So instead of telling what NP, P, and all that jazz is, let me instead tell you *how* someone establishes "hardness".

1 A Hard Problem, and the Idea of Polynomial Time Reductions

Recall what Boolean formulas are. There are Boolean variables and their negations; together these are called *literals*. Examples are x_1, $\neg x_2$, and so on. Then there are *clauses* where each clause is a collection of literals OR-red with each other. Examples being $x_1 \vee \neg x_2$, $x_2 \vee x_3 \vee \neg x_4$, and so on. A *SAT formula* is the AND of a collection of such clauses. A formula is *satisfiable* if there exists a setting of {true, false} to the variables such that in *every* clause *at least* one of the literals evaluates to true. For example, the formula could be

$$\phi = (x_1 \vee \neg x_2) \ \wedge \ (x_2 \vee \neg x_1)$$

in which case it is satisfiable by setting both x_1 and x_2 to true. On the other hand, the formula

$$\phi' = (x_1 \vee \neg x_2) \ \wedge \ (x_2 \vee \neg x_1) \ \wedge \ (\neg x_1 \vee \neg x_2) \ \wedge \ (x_1 \vee x_2)$$

is *unsatisfiable*. You can check this by going over all the 4 possibilities of setting {true,false} to the x_1, x_2 variables. Now consider the following computational problem. This is the "seed" hard problem.

SAT
Input: A SAT formula with n variables and m clauses.
Output: Decide whether or not it is satisfiable.

The following conjecture is our belief that the SAT problem is hard. This conjecture is (equivalent to) the $\mathbf{P} \neq \mathbf{NP}$ conjecture.

[1] *Lecture notes by Deeparnab Chakrabarty. Last modified : 19th Mar, 2022*
These have not gone through scrutiny and may contain errors. If you find any, or have any other comments, please email me at deeparnab@dartmouth.edu. Highly appreciated!

[2] You must agree it's a rather poor name for such a fascinating collection.

Conjecture 1. There is no polynomial time algorithm for SAT.

Remark: *At this point, you may be inclined to say, "Sure! This problem is so unstructured, what can you do but check all assignments? And there are 2^n many of them." Well, don't! First, there are many heuristics which do better than pure guessing and do well in practice. Unfortunately, none of them provably give fast algorithms all the time. Second, the argument above can be made for many problems. And if something we have learned in this course is that the study of algorithms is riddled with surprises (and we are surprised by many riddles).*

We are very far from either proving or disproving this conjecture. On the one hand, the best algorithm for SAT runs in time "essentially" 2^n (formally, it runs in $2^{n-o(n)}$ time where recall $o(n)$ is the set of functions whose ratio with n tend to 0 as n tends to infinity). On the other hand, $O(n+m)$-time algorithms for SAT are not ruled out. On a different track, as mentioned above, there are many fast heuristics for SAT which solve the SAT formulas arising in practice very well. All in all, SAT is a very interesting beast which hasn't been tamed.

Polynomial Time Reductions. How is SAT being hard related to the problem Π that I cannot solve? For simplicity, let us assume Π is a YES/NO problem, that is, for any input I to Π we need to either answer YES or NO (think SUBSET-SUM, CYCLE?, etc). Now suppose we had an *efficient method* $\mathcal{R}$ which took any SAT formula ϕ and returned an input I_ϕ of our problem Π such that ϕ is satisfiable implies I_ϕ has the answer YES, and if ϕ is unsatisfiable then I_ϕ has the answer NO. In that case, if we believe Conjecture 1, we would be forced to conclude there ***cannot*** be any efficient algorithm $\mathcal{A}$ to solve Π (thereby justifying our failures). Why?

Well suppose not, and there was indeed an efficient algorithm $\mathcal{A}$ for solving Π. Then, consider the following algorithm $\mathcal{B}$ for SAT:

a. Upon input ϕ, first run efficient procedure $\mathcal{R}(\phi)$ to obtain I_ϕ.

b. Run the efficient procedure $\mathcal{A}(I_\phi)$ to get answer YES or NO.

c. Return the same answer for ϕ.

The algorithm is correct due to the property of $\mathcal{R}$. The algorithm is efficient since both $\mathcal{R}$ and $\mathcal{A}$ are efficient (the latter by assumption, the former by construction). Contradicting Conjecture 1. Therefore, if we believe Conjecture 1, we must conclude $\mathcal{A}$ cannot exist.

This kind of algorithm $\mathcal{R}$ which takes instance of one problem and gives us an instance of another problem is called a ***reduction algorithm***. We have already met them in our applications of flows-and-cuts.

A piece of notation: if such an efficient algorithm $\mathcal{R}$ as above existed which takes an instance of the SAT problem to an instance of our problem Π, then we denote it as SAT $\preceq_{\text{poly}}$ Π. What we just established is the following lemma.

Lemma 1. If SAT $\preceq_{\text{poly}}$ Π and Conjecture 1 is true, then there can be no polynomial time algorithm for Π.

And there was nothing special about SAT really. After establishing SAT $\preceq_{\text{poly}}$ Π, suppose we next established Π $\preceq_{\text{poly}}$ Ψ for some other problem Ψ, then we would again be able to conclude there is no polynomial time algorithm for Ψ, unless Conjecture 1 is false. Such problems which can be reduced from SAT are called ***NP-hard*** problems.

2 An Example of Reduction : The Independent Set Problem

Let us illustrate one reduction algorithm. Recall the independent set problem : give an undirected graph $G = (V, E)$, a subset $I \subseteq V$ of vertices in a graph is *independent* if for every pair of vertices u and v in I, (u, v) is ***not*** and edge. The independent set problem asks us to find as large an independent set as possible. We look at the following YES/NO version of the problem. Clearly, if we could solve the *optimization* version, we would be able to solve this YES/NO version.

IS
Input: An undirected graph $G = (V, E)$ and a positive integer K.
Output: Decide whether or not there exists an independent set I in G with $|I| \geq K$?

Lemma 2. SAT $\preceq_{\text{poly}}$ IS

Proof. Let's first state what we need to show. We want to describe an algorithm $\mathcal{R}$ which takes input a SAT formula ϕ and outputs (a) an undirected graph $G_\phi = (V_\phi, E_\phi)$, and (b) a positive integer K_ϕ such that

R1. If ϕ is satisfiable, then there exists an independent set $I \subseteq V_\phi$ with $|I| \geq K_\phi$.

R2. If there exists an independent set $I \subseteq V_\phi$ with $|I| \geq K_\phi$, then ϕ is satisfiable.

Note that [R2.] is the "contrapositive" of what we want : we would like to prove "ϕ unsatisfiable $\Rightarrow$ "all independent sets $I \subseteq V_\phi$ have size $< K_\phi$.". Which is equivalent (remember logic from COSC 30?) to [R2.].

Let $\phi := C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each C_i is a clause which contains literals $C_i := \alpha_{i,1} \vee \alpha_{i,2} \vee \cdots \vee \alpha_{i,d_i}$. Each $\alpha_{i,j}$ is a *literal*, that is, it is either x_k or $\neg x_k$ for some variable x_k.

Now we describe the graph. The graph has $2n + \sum_{i=1}^{m} d_i$ vertices. These are divided into two classes: V_{var} and V_{cl}. For each variable x_i of the formula ϕ, we add two vertices, named x_i and $\neg x_i$, into V_{var}. We add the edge $(x_i, \neg x_i)$ between them. For each clause $C_i = \alpha_{i,1} \vee \alpha_{i,2} \vee \cdots \vee \alpha_{i,d_i}$, we add d_i vertices named $\alpha_{i,1}, \ldots, \alpha_{i,d_i}$ into V_{cl}. We add the *complete* graph between them. That is, we add edges $(\alpha_{i,j}, \alpha_{i,k})$ for all $1 \leq i < k \leq d_i$.

Now fix a literal γ. Say $\gamma = \neg x_i$ for concreteness. Note it appears *exactly* once in V_{var}, but there can be many appearances in V_{cl} as different $\alpha_{i,j}$'s. In fact it appears as many times this literal appears in the formula. Next comes the crucial *connector* edges. For every literal γ, we add an edge between the unique version in V_{var} to *every* appearance of the literal $\alpha_{i,j} = \gamma$ in V_{cl}. We give an example in Figure 1. Perhaps you should try out one for yourself too.

To complete the reduction we also need to specify the parameter K_ϕ. We set $K_\phi = n + m$, that is, the sum of the number of variables and clauses. Now that the reduction is defined, we need to prove that it does what it is supposed to do. That is, establish [R1.] and [R2.]

- *Establishing [R1.]*: Suppose ϕ is satisfiable. We now construct an independent set $I \subseteq V_\phi$ of size $n + m$. Note that ϕ assigns some variables x_i to true and some to false. For every variable x_i that is set to true, we pick the vertex corresponding to the *complement* $\overline{x}_i$ from V_{var} into I. Similarly, for every variable x_i that is set to false, we pick the vertex corresponding to x_i from V_{var} into I. At this point note I is independent and is of size n.

  Next, for every clause $C_i = (\alpha_{i,1} \vee \alpha_{i,2} \vee \cdots \vee \alpha_{i,d_i})$ we know *at least* one of the literals must be set to true. We *arbitrarily* choose one of them, say $\alpha_{i,j}$, and pick the corresponding vertex in V_{cl} for this

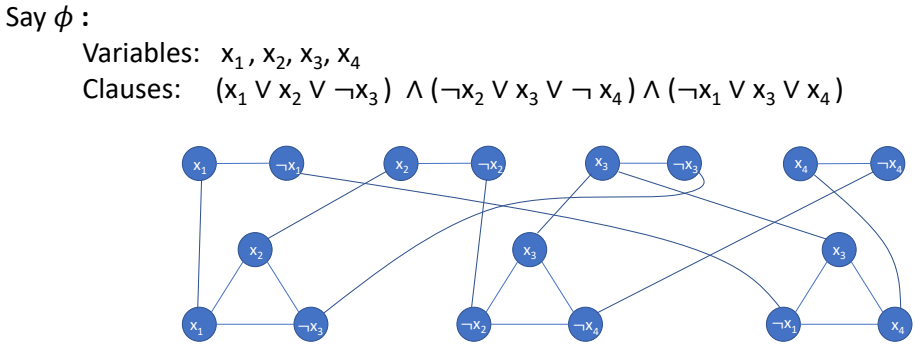

Figure 1: Example for the reduction SAT $\preceq$ IS. For simplicity, we have drawn an example where every clause has exactly 3 literals.

clause into I. Note that $|I|$ is now exactly $n + m$. We now claim that this is independent. Note that the *only* edge $\alpha_{i,j}$ has to a vertex which is *not* of the form $\alpha_{i,k}$ is to the vertex x_t or $\overline{x}_t$ in V_{var} which corresponds to the literal $\alpha_{i,j}$. Furthermore, since $\alpha_{i,j}$ evaluates to true, by design the vertex to which it is connected is *not* in I. Thus, adding the $\alpha_{i,j}$'s doesn't violate independence. This establishes [R1.].

- *Establishing [R2.]*: Now suppose there is an independent set $I \subseteq V_\phi$ of size $\geq n + m$. Note that for every pair of vertices $\{x_t, \overline{x}_t\}$ in V_{var}, we can pick at most one of them into I. Thus, $|I \cap V_{\mathsf{var}}| \leq n$. Similarly, for every collection $\{\alpha_{i,1}, \ldots, \alpha_{i,d_i}\} \subseteq V_{\mathsf{cl}}$, we can pick at most one vertex into I (since it's a complete graph between them). Thus, $|I \cap V_{\mathsf{cl}}| \leq m$. Which means, if $|I| \geq n + m$, we must have equality, *and* I contains exactly one vertex from $\{x_t, \overline{x}_t\}$ for all $1 \leq t \leq n$, and exactly one vertex from $\{\alpha_{i,1}, \ldots, \alpha_{i,d_i}\}$ for all $1 \leq i \leq m$.

  We now describe an assignment to the variables of ϕ. For every pair $\{x_t, \overline{x}_t\}$, if $x_t \in I$, then we set x_t to *false*, and if $\overline{x}_t \in I$, we set x_t to *true*. We claim this is a satisfying assignment. Suppose not. Then there must exist a clause $C_i = (\alpha_{i,1} \vee \alpha_{i,2} \vee \cdots \vee \alpha_{i,d_i})$ such that all the literals have been set to false. However, consider $I \cap \{\alpha_{i,1}, \ldots, \alpha_{i,d_i}\}$; it contains a single vertex $\alpha_{i,j} \in I$. Suppose $\alpha_{i,j} = x_t$. Note $(\alpha_{i,j}, x_t)$ is an edge in G_ϕ. Since we set x_t to false, it must be that $x_t \in I$. But this contradicts that I is independent. The case of $\alpha_{i,j} = \overline{x}_t$ is analogous.

  Therefore, our assignment must satisfy all clauses thereby establishing [R2.]. □

Theorem 1. If Conjecture 1 is true, then there is no polynomial time algorithm for the independent set problem.