1 3-Sat

After Cook’s paper [Coo71] published, Dick Karp immediately realized that the notion of NP-hardness captures a large amount of intractable combinatorial optimization problems. In [Kar72], he showed 21 problems to be NP-complete. This list quickly increased and by the time of 1979, Garey and Johnson [GJ79] wrote a whole book on NP-complete problems. This book has became a classic nowadays, and thousands of NP-hard problems were discovered during the past four decades. These intractable problems spread over all kinds of areas, even beyond computer science.

The canonical hard problem $L_{\text{NP}}$ defined last time is not very useful to show NP-hardness of other problems, and SAT is much more handier in this sense. What is even more useful is the following variant of SAT. Let $k$-CNF formulas be those whose clauses involve at most $k$ literals. For example, $(x_1 \lor \overline{x_2} \lor \overline{x_3} \lor x_4) \land (x_2 \lor x_5)$ is a 4-CNF.

**Name:** $k$-SAT

**Input:** A $k$-CNF formula $\varphi$.

**Output:** Is $\varphi$ satisfiable?

**Theorem 1.** 3-SAT is NP-complete.

**Proof.** We give a reduction SAT $\leq_p$ 3-SAT. Namely, given a CNF formula $\varphi$, we construct (in polynomial time) another 3-CNF formula $\varphi'$, such that $\varphi$ is satisfiable, if and only if $\varphi'$ is satisfiable.

The only thing we need to do is for every clause $c$ in $\varphi$, we replace it by a conjunction of clauses of size at most 3 and preserve satisfying assignments. We do this inductively. For a clause $c$ of size $k > 3$, say its first two literals are $x_1$ and $x_2$. So $c$ has the form $x_1 \lor x_2 \lor c'$, where $c'$ is a clause of size $k - 2$. We introduce a new variable $y_1$ and consider the following formula: $\varphi_c := (x_1 \lor x_2 \lor y_1) \land (\overline{y_1} \lor c')$.

- If an assignment $\sigma$ satisfies $c$, then at least one of $x_1$, $x_2$, and literals in $c'$ is true under $\sigma$. Hence, we can assign $y_1$ accordingly to make $\varphi_c$ true. For example, if $x_1$ is true, then we assign $y_1$ to be false.

- If an assignment $\sigma$ satisfies $\varphi_c$, then depending on $y_1$’s value, at least one of $x_1 \lor x_2$ and $c'$ is true. Hence, $c$ is satisfied under $\sigma$. 

In this construction, the sizes of the new clauses (namely, $x_1 \vee x_2 \vee y_1$ and $\overline{y}_1 \vee c'$) decrease at least 1. To continue, we apply this construction to $\overline{y}_1 \vee c'$ and reduce clause sizes by 1 again (if still $> 3$). (Or equivalently, invoke the induction hypothesis.)

We finish with a 3-CNF formula $\varphi'$ which contains a number of new variables, and $\varphi$ is satisfiable if and only if $\varphi'$ is. It is also easy to see that the construction only takes linear time. \qed

There are two key points of the reduction above: 1. local transformations (from a clause to a conjunction of clauses, without affecting other clauses); 2. introducing new variables.

Since trivially 3-SAT $\leq_P$ k-SAT for any $k \geq 3$, k-SAT is NP-hard for any $k \geq 3$. On the other hand, the proof above does not work for 2-SAT. In fact, 2-SAT $\in$ P.

## 2 More Reductions

We will show that from 3-SAT, it is relatively easy to derive NP-hardness for other combinatorial problems.

For an optimization problem, we often consider their “threshold” version as the decision problem. Recall that for a graph $G = (V, E)$, a vertex cover $C \subseteq V$ is a subset of vertices so that every edge is adjacent to at least one vertex in $C$.

**Name:** VC

**Input:** A graph $G$ and a number $k$.

**Output:** Does $G$ contain a vertex cover of size at most $k$?

**Theorem 2.** VC is NP-hard.

Once again, we will build a reduction, this time from 3-SAT to VC. The key to this reduction, is to transform the “local” constraint in 3-SAT to the local constraint in VC.

**Proof.** Given a 3-CNF formula $\varphi$ with $n$ variables and $m$ clauses, we construct a graph $G_\varphi = (V, E)$ as follows. We introduce a vertex $x_i$ and $\overline{x}_i$ for each variable $x_i$, and connect the two. For each clause $c_j = \ell_{j,1} \vee \ell_{j,2} \vee \ell_{j,3}$, we introduce a vertex for each of its literals, and connect all of them pairwise. In addition, connect all literals to the corresponding variable vertex. For example, if $\ell_{4,1}$ is $\overline{x}_5$, then $(\ell_{4,1}, \overline{x}_5) \in E$. So $|V| \leq 2n + 3m$ and $|E| \leq n + 6m$.

Now, we should step back and think about the construction. To get a small vertex cover, we want to occupy as few vertices as possible. We have to cover at least one of $(x_i, \overline{x}_i)$. We also have to cover at least two of $\ell_{j,1}$, $\ell_{j,2}$, and $\ell_{j,3}$ for a clause $c_j$. Hence, the vertex cover has size at least $n + 2m$.

If $\varphi$ has a satisfying assignment $\sigma$, then we interpret $\sigma$ as a vertex cover as follows. We cover $x_i$ or $\overline{x}_i$ according to $\sigma(x_i)$’s value. We also cover false literals in a clause. Since $\sigma$ is a satisfying assignment, there are at most two literals covered this way. If a clause has less than two literals, we cover some arbitrary vertices (still at most 2 in total) so that the whole triangle is covered. The only edges in question are those connecting the uncovered vertex
in each triangle, and the corresponding variable vertex. We know that the uncovered literal must be true, and hence the corresponding variable vertex is covered. So all edges are fine and we have a vertex cover of size \( n + 2m \).

On the other hand, suppose we do have a vertex cover of size \( n + 2m \). As we have argued earlier, it must contain exactly one for each pair of variable vertices, and two of every clause triangle. We interpret the covered variable vertex as true. If a clause is not satisfied, then all literals must be false. Hence, the one uncovered is connected to a uncovered variable vertex, contradicting to the assumption of a vertex cover.

From VC, we can further show other problems to be \textsc{NP}-hard. Recall that for a graph \( G = (V, E) \), an independent set \( I \subseteq V \) is a subset of vertices so that no two vertices in \( I \) are adjacent.

**Name:** \textsc{IndSet}

**Input:** A graph \( G \) and a number \( k \).

**Output:** Does \( G \) contain an independent set of size at least \( k \)?

**Theorem 3.** \textsc{IndSet} is \textsc{NP}-hard.

**Proof.** We show a reduction \( \text{VC} \leq_p \text{IndSet} \). The graph \( G = (V, E) \) is kept as the same, but for the input \( k \), we replace it with \( n - k \). It is easy to see that, for a vertex cover \( C \) of size \( t \), its complement \( \overline{C} := V \setminus C \) is an independent set of size \( n - k \). Hence the reduction is valid.

Recall that for a graph \( G = (V, E) \), a clique \( C \subseteq V \) is a subset of vertices so that any two vertices in \( C \) are adjacent.

**Name:** \textsc{Clique}

**Input:** A graph \( G \) and a number \( k \).

**Output:** Does \( G \) contain a clique of size at least \( k \)?

**Theorem 4.** \textsc{Clique} is \textsc{NP}-hard.

**Proof.** We reduce \( \text{IndSet} \leq_p \text{Clique} \). Given \( G = (V, E) \), we construct \( G' = (V, \overline{E}) \), where \( \overline{E} \) is the complement of \( E \). Namely, \( (i, j) \in \overline{E} \) if and only if \( (i, j) \notin E \). It is easy to verify that an independent size in \( G \) is a clique in \( G' \). Hence the reduction is valid.

**Remark** (Bibliographic). These reductions were first shown by Karp [Kar72]. Relevant chapters are [AB09, Chapter 2] and [Pap94, Chapter 8 and 9].
References


