Reductions for NP-Complete Problems

Tao Hou

1 Circuit Satisfiability: A First NP-Complete Problem

Definition 1. Define a *circuit* K to be a labeled, directed acyclic graph such as the one shown in Figure 1:

- The *sources* in K are the nodes with no incoming edges. They are either labeled a fixed value 0 or 1, or need to be assigned a value (thus called *free* sources)
- Every other node is labeled with one of the Boolean operators $\wedge$, $\vee$, or $\neg$; nodes labeled with $\wedge$ or $\vee$ have two incoming edges, and nodes labeled with $\neg$ have one incoming edge.
- There is a single node with no outgoing edges, and it represents the output: the result computed by the circuit.

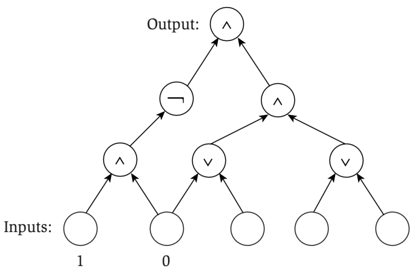

Figure 1: A circuit with 5 sources (two of them have fixed truth values and three are free sources) and one output. [Figure from [1]]

Problem 1 (CIRCUIT-SAT)**.** Given a circuit K, is there an assignment of 0-1 values to the free sources that causes the output to have the value 1?

Proposition 1. *CIRCUIT-SAT is NP-complete.*

Sketch of proof.

- Any algorithm that takes an input of n bits and produces a yes/no answer can be represented by a circuit having n sources
- Moreover, if the algorithm takes poly-time, then circuit is of poly-size w.r.t n.
- Consider some problem X in NP, which has a poly-time certifier $C(s, t)$.
- An instance s of X is an "yes" instance if and only if there exists a certificate t of length $p(|s|)$ such that $C(s, t) =$"yes".
- Notice $C(s, t)$ is an algorithm taking an input of $|s| + |t|$ bits
- So we convert it into a poly-size circuit K with $|s| + |t|$ sources:
  - First $|s|$ sources are *hard-coded* with values of s
  - Remaining t sources are *free* and represent bits of the certificate t
- Circuit K is satisfiable there iff exists a certificate t making $C(s, t) =$"yes".

□

2 SAT and 3-SAT

Definition 2. A Boolean formula is made up of the Boolean variables $x_1, \ldots, x_n$, operators including $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT), $\Rightarrow$ (implication), $\Leftrightarrow$ (if and only if), and composite (combinations) of them possibly with parenthesis. E.g.,:

$$((x_1 \Rightarrow x_2) \vee \neg((\neg x_1 \Leftrightarrow x_3) \vee x_4)) \wedge \neg x_2.$$

Problem 2 (SAT)**.** Given a Boolean formula with n Boolean variables $x_1, \ldots, x_n$, is there an assignment (of true/false values) to $x_1, \ldots, x_n$ making the whole formula true?

Definition 3. Given n Boolean variables $x_1, \ldots, x_n$:

- *literal*: one of the variables x_i or its negation $\neg x_i$
- *clause*: a disjunction of distinct literals, e.g., $x_1 \vee \neg x_2 \vee \neg x_1 \vee x_5$

Problem 3 (3-SAT)**.** Given m clauses $C_1, \ldots, C_m$ each containing exactly *three* literals over the Boolean variables $x_1, \ldots, x_n$, is there an assignment of truth values to the variables making

$$C_1 \wedge \cdots \wedge C_m$$

to be true (i.e., all the clauses are true)?

Proposition 2. *3-SAT is NP-complete.*

Proof. We shall omit the proof that 3-SAT $\in$ NP. We then prove that 3-SAT is NP-hard by reducing CIRCUIT-SAT to 3-SAT. Given a circuit K as an instance of CIRCUIT-SAT, we construct an instance of 3-SAT as follows:

- To construct the Boolean variables: Have a variable x_v for each node v of the circuit, which encodes the out-going truth value of the node v. (This finishes the construction of the Boolean variables)
- To construct the clauses:
  - A node v labeled with "$\neg$" with its only incoming edge from node u: Notice that we need to have $x_v = \neg x_u$. We guarantee this by adding the two clauses to the 3-SAT instance:

    $$(x_v \vee x_u), \text{ and } (\neg x_v \vee \neg x_u).$$

  - A node v is labeled with "$\vee$" with its two incoming edges from nodes u and w: We need to have $x_v = x_u \vee x_w$. We guarantee this by adding the following clauses:

    $$(\neg x_v \vee x_u \vee x_w), (\neg x_u \vee x_v), \text{ and } (\neg x_w \vee x_v).$$

    Why? Because:

    $$\begin{aligned} x_v \Leftrightarrow x_u \vee x_w &= (x_v \Rightarrow x_u \vee x_w) \wedge (x_u \vee x_w \Rightarrow x_v) \\ &= (\neg x_v \vee x_u \vee x_w) \wedge (\neg(x_u \vee x_w) \vee x_v) \\ &= (\neg x_v \vee x_u \vee x_w) \wedge ((\neg x_u \wedge \neg x_w) \vee x_v) \\ &= (\neg x_v \vee x_u \vee x_w) \wedge (\neg x_u \vee x_v) \wedge (\neg x_w \vee x_v) \end{aligned}$$

  - A node v labeled with "$\wedge$" with two incoming edges from nodes u and w: We need to have $x_v = x_u \wedge x_w$. We add the following clauses:

    $$(\neg x_v \vee x_u), (\neg x_v \vee x_w), \text{ and } (x_v \vee \neg x_u \vee \neg x_w).$$

  - A source v labeled with a fixed value 1: Add a clause (x_v) with a single literal, forcing the variable x_v to take the designated value 1.
  - A source v labeled with a fixed value 0: Add a clause $(\neg x_v)$ with a single literal, forcing the variable x_v to take the designated value 0.
  - The output node o: Add the single-literal clause (x_o), which requires that o take the value 1.

We then claim that **all the clauses we have constructed for 3-SAT are satisfiable iff the circuit K can be satisfied**:

- Since free sources in K correspond to Boolean variables but *do not* correspond to any clauses, we are free to choose values for the Boolean variables corresponding to these free sources
- The clauses constructed for each internal node v guarantee that the Boolean variable x_v has the same value as the outgoing edge in K given a certain assignment on the free sources
- The output value also has to be 1 by the single-literal clause (x_o)

Notice that our goal was to create an instance of 3-SAT where all clauses have **exactly 3 literals**, while in the instance we constructed, some clauses have lengths of 1 or 2. So we need to convert this instance of SAT to an equivalent instance in which each clause has exactly three literals.

To do this:

- Create four new variables:
$$z_1, z_2, z_3, z_4,$$
- And the four clauses:
$$(\neg z_i \vee z_3 \vee z_4), (\neg z_i \vee \neg z_3 \vee z_4), (\neg z_i \vee z_3 \vee \neg z_4), (\neg z_i \vee \neg z_3 \vee \neg z_4)$$
  for each of $i = 1$ and $i = 2$. In order for all four clauses to be true, we must have $z_i = 0$ for $i = 1$ and $i = 2$.

  Then:
- $(t) \Rightarrow (t \vee z_1 \vee z_2)$
- $(t \vee t') \Rightarrow (t \vee t' \vee z_1)$ ☐

Proposition 3. *SAT is NP-complete.*

Proof. This is easy because each instance of 3-SAT is an instance of SAT. ☐

3 Independent set

Definition 4. In a graph $G = (V, E)$, we say a set of vertices $S \subseteq V$ is an *independent set* if no two vertices in S form an edge in G.

Remark 4. By default, graphs are *undirected* in this topic.

Remark 5. It is easy to find small independent sets in a graph (for example, a single node forms an independent set); the hard part is to find a large independent set, since when you add more and more points into a set, it becomes more probable that two vertices from the set are connected by an edge.

Problem 4 (IND-SET). Given a graph G and a number k, does G contain an independent set of size at least k?

Remark 6. The "optimization" version of the above problem is to find the independent set with the maximum size.

Proposition 7. *IND-SET is NP-complete.*

Proof. **IND-SET is in NP.** To design a certifier, given a certificate t, you try to decode k vertices from the bits in t. If the decoding cannot be done, you simply return "no". Otherwise, you check whether the k vertices form an independent set (which is easy), and return "yes" or "no" accordingly.

IND-SET is NP-complete. We show that 3-SAT reduces to IND-SET. Suppose we are given an instance Φ of 3-SAT:

- Variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_k$.

We construct a graph $G = (V, E)$ consisting of $3k$ vertices with two types of edges:

- G contains 3 vertices for each clause, one for each literal.
- (Triangle edges) Connect the 3 literals in a clause in a triangle.
- (Cross edges) Connect literals to their negations.

Example:

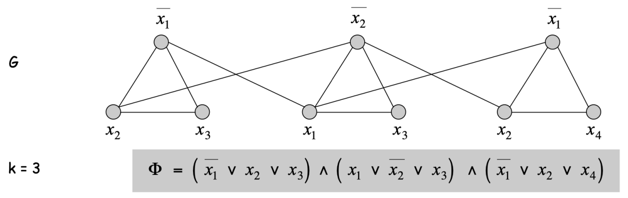
$\overline{x_1}$
$\overline{x_2}$
$\overline{x_1}$
G
x_2
x_3
x_1
x_3
x_2
x_4
k = 3
$\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$

Figure 2: [taken from [1]]

Idea of construction:

- To have an independent set of size $\geq k$ in G, you have to choose at most one vertex from each triangle (BTW, the size of the independent set has to be **exactly** k)
- This corresponds to choosing a literal from a clause to be "true" which makes the clause "true"
- Connecting the literals to their negations make sure that values for the variables are "consistent" across the different literals

We then show that Φ **is satisfiable iff** G **contains an independent set of size** $\geq k$:

- $\Rightarrow$: Given a satisfying assignment to Φ, select a "true" literal from each triangle in G. This is an independent set S of size k.
  - It's clear that no triangle edge connects two vertices in S
  - A cross edge connecting two vertices $x_i, \neg x_i$ in S also cannot happen: $x_i, \neg x_i$ in S are two "true" literals from two different clauses, but they cannot be both "true".
- $\Leftarrow$: Let S be independent set of size k in G.
  - S must contain exactly one vertex (literal) from each triangle (clause).
  - Set these literals in S to "true", which can be done because there are no two literals such as $x_i, \neg x_i$ in S
  - We then have that each clause is true because we have a "true" literal from each triangle ☐

4 Vertex cover

Definition 5. Given a graph $G = (V, E)$, we say that a set of vertices $S \subseteq V$ is a *vertex cover* of G if for every edge $e \in E$, at least one vertex of e is in S.

Remark 8. It is easy to find large vertex covers in a graph (for example, the whole vertex set is one); the hard part is to find small ones

Problem 5 (VERTEX-COVER). Given a graph G and an integer k, does G contain a vertex cover of size at most k?

Remark 9. The "optimization" version of the problem is to find a vertex cover of the smallest size.

Proposition 10. *Let $G = (V, E)$ be a graph. Then S is an independent set of G if and only if its complement $V - S$ is a vertex cover of G.*

Proof.

- $\Rightarrow$:
  - Suppose that S is an independent set.
  - Consider an arbitrary edge e.
  - Since S is an independent set, the two vertices of e cannot be both in S.
  - So one of the vertex of e is in $V - S$.
  - Since e is arbitrary, $V - S$ is a vertex cover
- $\Leftarrow$:
  - Suppose that $V - S$ is a vertex cover.
  - Consider any two nodes u and v in S.
  - We want to show that $(u, v) \notin E$ so that S is independent.
  - Use proof by contradiction, suppose that $(u, v) \in E$.
  - Since $V - S$ is a vertex cover, at least one of u, v is in $V - S$
  - Say $u \in V - S$
  - We have $u \notin S$ (a contradiction). ☐

Proposition 11. *VERTEX-COVER is NP-complete.*

Proof. Proof of VERTEX-COVER $\in$ NP is omitted.
To prove that it's NP-complete, we reduce IND-SET to VERTEX-COVER:

- Given an instance (G, k) of IND-SET, we construct an instance $(G, n-k)$ of IND-SET, where n is the number of vertices of G.
- Then, by Proposition 10, G has an independent set of size $\geq k$ if and only if G has a vertex cover of size $\leq n-k$. $\square$

5 Graph coloring

Definition 6 (k-coloring)**.** Given a graph G and k colors (labels), a k-coloring of G is an assignment of the k colors to each vertex such that no two adjacent vertex have the same color.

Problem 6 (k-COLORING)**.** Given a graph G and an integer k, does G have a k-coloring?

Fact 12. *A 2-colorable graph is also called a* bipartite graph*. Checking whether a graph is bipartite can be done in* $O(m+n)$ *time.*

Proposition 13. *3-COLORING is NP-complete.*

Proof.

- We again reduce 3-SAT to 3-COLORING.
- Suppose we are given an instance of 3-SAT with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_k$.
- We going to construct a graph G as an instance of 3-COLORING.

Nodes:

- For each variable x_i, we define nodes v_i and $\overline{v_i}$ corresponding to x_i and its negation $\neg x_i$.
- We also define three "special nodes" T, F, and B.

Triangle edges:

- For each variable x_i, we let v_i, $\overline{v_i}$, and B form a triangle.
- We also let T, F, and B form a triangle (see figure below)

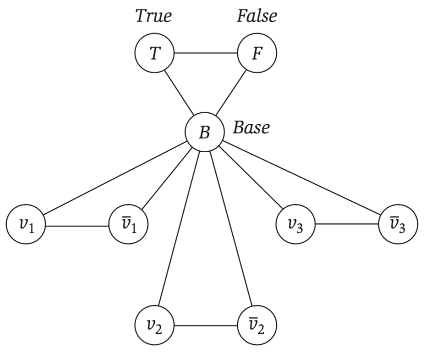

Figure 3: Triangles constructed for three variables x_1, x_2, x_3 [taken from [1]]

Explanation:

- The idea of this construction is to let either v_i or $\overline{v_i}$ to get T's color and let the other get F's color.
- This provides a consistent "true"/"false" assignment to the variable x_i.

Additional nodes and edges:

- We also add more edges and nodes for each clause C_j.
- Take a clause $C_j = (x_1 \vee \neg x_2 \vee x_3)$ as an example.
- What we want to achieve is that, C_j is satisfiable iff at least one of the nodes v_1, $\overline{v_2}$, and v_3 get T's color.
- We attach the following nodes (gray ones) and edges to the existing nodes v_1, $\overline{v_2}$, v_3, T and F:

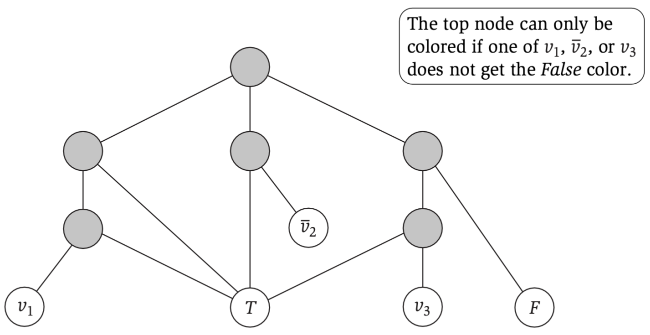
The top node can only be colored if one of v_1, $\overline{v}_2$, or v_3 does not get the *False* color.

Figure 4: Attaching a subgraph to represent the clause $C_j = (x_1 \vee \neg x_2 \vee x_3)$ [taken from [1]]

- We have that the top gray node can be colored iff one of the nodes v_1, $\overline{v_2}$, and v_3 get T's color (i.e., C_j is satisfied).
- Adding the above nodes and edges for each clause, we have a graph G which is 3-colorable iff the instance of 3-SAT is satisfiable. $\square$

6 Summary

Three strategies for NP-completeness reductions:

1. Reduction by simple equivalence: IND-SET $\leq_p$ VERTEX-COVER.
2. Reduction from special case to general case: 3-SAT $\leq_p$ SAT.
3. Reduction by encoding with gadgets: 3-SAT $\leq_p$ VERTEX-COVER; 3-SAT $\leq_p$ 3-COLORING.

References

[1] J. Kleinberg and E. Tardos. *Algorithm Design.* Addison Wesley, 2006.