NP-Completeness and Computation Intractability

Pramook Khungurn

December 11, 2019

1 Polynomial Time Reduction

- We say that a computational problem Y is **polynomial-time reducible to** another computational problem X if an arbitrary instance of Y can be solved using a polynomial number of standard computational steps, plus a polynomial number of calls to a black box that solves problem X.

  We write $Y \leq_p X$ to denote that Y is polynomial-time reducible to X.

- If $Y \leq_p X$, we have that:
  - If X is solvable in polynomial time, then so does Y.
  - If Y cannot be solved in polynomial time, then X cannot be solved in polynomial time either.

2 Independent Set and Vertex Cover

- Let $G = (V, E)$ be a graph. A set of vertices $S \subseteq V$ is called **independent** if no two nodes in S are joined by an edge.
- The **Independent Set** problem asks that:

  Given a graph G and a number k, does G contains an independent set of size at least k?

- A set of vertex $S \subseteq V$ is a **vertex cover** if every edge $e \in E$ has at least one end in S.
- The **Vertex Cover** problem asks that:

  Given a graph G and a number k, does G contains a vertex cover of size at most k?

- We will show that Independent Set $\leq_p$ Vertex Cover and Vertex Cover $\leq_p$ Independent Set by showing the following lemma:

  Lemma 2.1. *Let $G = (V, E)$ be a graph. Then S is an independent set if and only if its complement $V - S$ is a vertex cover.*

  Proof. Suppose S is an independent set. Consider an arbitrary edge $e = (u, v)$. Since S is independent, it cannot be the case that both u and v are in S, so one of them must be in $V - S$. This follows that $V - S$ is a vertex cover.

  Conversely, suppose that $V - S$ is a vertex cover. Consider any two nodes u and v in S. If they were joined by edge e, then neither end of e would lie in $V - S$, contradicting our assumption that $V - S$ is a vertex cover. It follows that no two nodes in S are joined by an edge, so S is an independent set. □

- Thus, we can find if a graph has an independent set of size at least k by finding if a graph has a vertex cover of size at most $|V| - k$. Conversely, we can find if a graph has a vertex cover of size at most k by finding if it has an independent set of last at least $|V| - k$.

  It follows that the two problems are polynomial-time reducible to each other.

3 Vertex Cover and Set Cover

- The **Set Cover** problem asks that:

  > Given a set U of n elements, a collection S_1, S_2, ..., S_m of subsets of U, and a number k, does there exists a collection of at most k of these sets whose union is equal to all of U?

- We have that Vertex Cover $\leq_p$ Set Cover. Given a graph G, we let $U = E$. Now, we create as many subsets of U as there are vertices in the graph. The set S_v corresponding to vertex v contains on the edges incident to v. A solution to set cover of k corresponds to a set of k vertices all whose incident edges contain all edges in the graph, which is a vertex cover of size k. Therefore, Vertex Cover is polynomial-time reducible to Set Cover.

- The **Set Packing** problem asks that:

  > Given a set U of n elements, a collection S_1, S_2, ..., S_m of subsets of U, and a number k, does there exist a collection of at least k of these sets with the property that no two of them intersect?

- Set Packing is a generalization of Independent Set, and we can easily show that Independent Set $\leq_p$ Set Packing.

4 Reduction via "Gadgets"

- Let x_1, x_2, ..., x_n be *boolean variables*, each of which can take values from the set $\{0, 1\}$. Let X denotes the set of these boolean variables.

- A **term** over X is either
  - one of the variable x_i, or
  - its negation $\overline{x_i}$, or

- A **clause** is a disjunction of distinct terms: $t_1 \vee t_2 \vee \cdots \vee t_\ell$.

- We can talk about **truth assignment** for X, which you already know what it means.

- An assignment *satisfies* a clause C if it causes C to evaluates to 1.

- An assignment satisfies a set of clauses $C_1, C_2, \ldots, C_k$ if it satisfies all of them. In other words, it causes the conjunction $C_1 \wedge C_2 \wedge \cdots \wedge C_k$ to evaluate to one.

- A set of clauses is said to be **satisfiable** if there is a truth assignment that satisfy all of the clauses.

- The **Satisfiability** (SAT) problem asks that:

  > Given a set of clauses C_1, C_2, ..., C_k over a set of variable $X = \{x_1, x_2, \ldots, x_n\}$, does there exists a satisfying truth assignment?

- The **3-Satisfiability** (3-SAT) problem asks that:

  > Given a set of clauses C_1, C_2, ..., C_k, each containing exactly 3 terms, over a set of variable $X = \{x_1, x_2, \ldots, x_n\}$, does there exists a satisfying truth assignment?

- **Theorem 4.1.** *3-SAT $\leq_p$ Independent Set*

Proof. Given an instance of 3-SAT with k clauses, construct a graph G with $3k$ vertices, each represent a term in each clause. For terms in the same clause, we connect each pair of them to form a triangle. We also connect a term x_i to any other $\overline{x_i}$ in any other clauses and vice versa.

We claim that, if the clauses are satisfiable, then there exists an independent set of size k.

Suppose that that the clauses are satisfiable. Then, at least one term from each clause evaluates to 1. We select a term that evaluates to 1 from each clause. We claim that the vertices corresponding to them form an independent set. To see why, observe that we choose only one vertex from a triangle. So, no two vertices connected by edges in a triangle cannot be both in the set. Thus, if there are vertices connected by an edge, they must belong to different triangles (which means the corresponding terms are in different clauses). Now, if u and v are connected by an edge and are in different clauses, then $u = x_i$ and $v = \overline{x_i}$ for some variable x_i. Since only one of them can be true according to a truth assignment, only one of them can be chosen. Therefore, no two selected vertices can share an edge, which means that the selected vertices form an independent set of size k.

If there's an independent set of size k, we have that each vertex must correspond to terms in different clauses because all vertices in a clause are connected. We take the vertices in the independent set and make a boolean assignment so that the terms corresponding to them evaluates to 1. Note that this is a valid truth assignment that satisfies all the clauses. This is because each clause has at least one term evaluates to 1. Moreover, there cannot be any variable that we assign it both the value 0 and 1. Because, if there is one, then we pick a vertex corresponding to x and $\overline{x}$, but this is impossible because vertices corresponding to these terms are connected. □

5 Definition of NP

- An input to an algorithm can be encoded as a binary string s. We let $|s|$ denote the length of the binary string.
- A yes/no computational problem can be identified as a set X of "yes" binary string.
- A yes/no computational problem is **polynomial-time solvable** if there is an algorithm A and a polynomial function p such that
  - $A(s) =$ yes iff $s \in X$, and
  - $A(s)$ runs in $O(p(|s|))$ for all s.
- $\mathcal{P}$ is the set of problems that is polynomial-time solvable.
- We say that B is an **efficient certifier** of X if the following properties hold:
  - B is a polynomial-time algorithm that takes two inputs s and t.
  - There is a polynomial function p so that, for every string s, we have that $s \in X$ iff there is a string t such that $|t| \leq p(|s|)$ and $B(s,t) =$ yes.
- $\mathcal{NP}$ is the set of all problems by which there exists an efficient certifier.
- **Lemma 5.1.** $\mathcal{P} \subseteq \mathcal{NP}$.

  Proof. Let X be a problem in $\mathcal{P}$. Define $B(s,t) = A(s)$ where A is the polynomial-time algorithm that solves X. We have that B is an efficient certifier. □

6 NP-Complete Problems

- A problem X is said to be **NP-complete** if
  - $X \in \mathcal{NP}$, and
  - for any problem $Y \in \mathcal{NP}$, $Y \leq_p X$.
- Suppose X is an NP-complete problem. Then X is solvable in polynomial time iff $\mathcal{P} = \mathcal{NP}$.
- Conversely, if there is any problem in $\mathcal{NP}$ that cannot be solved in polynomial time, then no NP-complete problems can be solved in polynomial time.
- A **circuit** K is a labeled, directed acyclic graph such that:
  - The **sources** in K (the nodes with no incoming edges) are labeled either with one of the constants 0 or 1, or with the name of a distinct variable. Nodes of the latter type will be referred to as the *inputs* to the circuits.
  - Every other node is labeled with one of the Boolean operator $\wedge, \vee$, or $\neg$. Nodes labeled with $\neg$ has only one incoming edge, and nodes labeled with $\wedge$ or $\vee$ has two incoming edges.
  - There is a single node with no outgoing edges, and it will represent the *output*.
- The **Circuit Satisfiability** problem asks that:

  Given a circuit, is there an assignment to the values of the inputs that causes the output to take value 1?

- **Theorem 6.1 (Cook–Levin).** *Circuit satisfiability is NP-complete.*

  Proof. Let X be a problem in $\mathcal{NP}$. Then, X has an efficient verifier $B(s,t)$. Given a fixed length n of s, we can convert the execution of $B(s,t)$ with $|t| \leq p(n)$ to a circuit K with $n + p(n)$ sources. We put the bits of s in the first n sources, and leave the $p(n)$ bits as inputs to the circuit. We then feed the circuit to the oracle solving Circuit Satisfiability. The result is "yes" if and only if $s \in X$. □

- We can discover more NP-complete problems using the following principle: if Y is an NP-complete problem and X is a problem in $\mathcal{NP}$ such that $Y \leq_p X$, then X is NP-complete.
- **Theorem 6.2.** *3-SAT is NP-complete.*

  Proof. We will create a boolean formula that is an input of 3-SAT from a given circuit K. For each vertex v in K, we let x_v be a variable corresponding to v's output value.

  We impose constraints based on the label on each vertex.

  - If v's label is $\neg$ and v takes input from vertex u, then $x_v = \neg x_u$. We impose this constraint by two clauses $(x_v \vee x_u)$ and $(\overline{x_v} \vee \overline{x_u})$.
  - If v's label is $\wedge$ and v takes input from vertex u and vertex w, then $x_v = x_u \wedge w_v$. This means that

$$\begin{aligned} x_v \Leftrightarrow (x_u \wedge x_v) &\equiv (x_v \Rightarrow (x_u \wedge x_v)) \wedge ((x_u \wedge x_v) \Rightarrow x_v) \\ &\equiv (\overline{x_v} \vee (x_u \wedge x_v)) \wedge (\overline{x_u \wedge x_v} \vee x_v) \\ &\equiv (\overline{x_v} \vee x_u) \wedge (\overline{x_v} \vee x_w) \wedge (\overline{x_u} \vee \overline{x_w} \vee x_v). \end{aligned}$$

    So, we create the above three clauses.

– If v's label is $\vee$ and v takes input from vertex u and vertex v, then $x_v = x_u \vee w_v$. This means that

$$\begin{aligned} x_v \Leftrightarrow (x_u \vee x_w) &\equiv (x_v \Rightarrow (x_u \vee x_w)) \wedge ((x_u \vee x_w) \Rightarrow x_v) \\ &\equiv (\overline{x_v} \vee x_u \vee x_w) \wedge (\overline{(x_u \vee x_w)} \vee x_v) \\ &\equiv (\overline{x_v} \vee x_u \vee x_w) \wedge (\overline{x_u} \vee x_v) \wedge (\overline{x_w} \vee x_v). \end{aligned}$$

So, we create the above three clauses.

– If v's label is 0, we add clause $\overline{x_v}$.

– If v's label is 1. we add caluse x_v.

It follows that the formula is satisfiable if there is an assignment to the inputs of the circuit that cause it to evaluate to 1.

However, some of the clauses we created have 1 or 2 terms. We need to convert all clauses to have exactly 3 terms. This is done by introducing 4 new variables z_1, z_2, z_3, and z_4. For each $i \in \{1, 2\}$, we add clauses $(\overline{z_i} \vee z_3 \vee z_4)$, $(\overline{z_i} \vee \overline{z_3} \vee z_4)$, $(\overline{z_i} \vee z_3 \vee \overline{z_4})$, and $(\overline{z_i} \vee \overline{z_3} \vee \overline{z_4})$, which requires that $z_i = 0$ for all the clauses to be true. Now, we can change a 1-term clause t to $(t \vee z_1 \vee z_2)$, and a 2-term clause $(t_1 \vee t_2)$ to $(t_1 \vee t_2 \vee z_1)$. □

- We now know that 3-SAT, Independent Set, Vertex Cover, and Set Cover are all NP-complete.

References