CS-E3190 Principles of Algorithmic Techniques

08. Hardness – Tutorial Exercise

This tutorial goes over the method of proving that a problem is NP-complete. Throughout this course we have showed many problems to be in P by developing efficient algorithms; this week we focus on proving membership in other complexity-classes. In particular we will first prove that VERTEXCOVER is in NP, and then prove that it is also NP-hard. We define and apply the so-called *polynomial-time reduction (aka Karp reduction)* - perhaps the most important tool for proving that problems are NP-hard.

Definition: The VERTEXCOVER decision problem is given as follows: Given an undirected graph $G = (V, E)$ and an integer $k \geq 0$, decide if there exists a vertex cover of size at most k.

Recall that a *vertex cover* is a subset of vertices $U \subseteq V$ such that every edge e in E is adjacent to some vertex in U. Formally, U is a vertex cover if for all $\{u, v\} \in E$ either $u \in U$ or $v \in U$.

1. **Proving VERTEXCOVER is in NP.** Recall that a decision problem Π is in NP if every Yes-instance has a *certificate* that can be *verified* in polynomial-time. I.e. membership in NP has sufficient conditions:

   - For all Yes-instances $I \in \Pi$ there is a Yes-certificate C.
   - There is a polynomial time algorithm that correctly verifies (I, C) of Yes-instances.

   Here "polynomial time" means polynomial in the encoding size of the input I.

   (a) Show that every Yes-instance of VERTEXCOVER has a Yes-certificate.

   Solution. We can use the vertex cover $U \subseteq V$ as the Yes-certificate.

   Suppose I is a Yes-instance. By definition it has a vertex cover $U \subseteq V$ of size at most k, so each Yes-instance has a certificate as required.

   (b) Give a verification algorithm. Prove that it correctly verifies instance-certificate pairs (I, C), and that it runs in polynomial time on Yes-instances' (I, C)'s.

   Solution. Consider the following algorithm, called Verify(C, I):

   Algorithm 1: Verify(I, C)

   ```
   if length(C) > k then
       return FALSE
   for all v ∈ C do
       Remove all edges in E containing v
   if E is empty then
       TRUE
   FALSE
   ```

   In words, the procedure goes over all nodes in C once, removing their adjacent edges from E one by one. The procedure returns TRUE only if all edges are removed and the length of C was no more than k.

   Correctness. We need to show that Verify(I, C) = TRUE if and only if I is a Yes-instance. We assume throughout that C is as given in part (a).

   [$\Longrightarrow$] Suppose I is a yes instance. Then, by what we previously stated, Verify$(I, C) =$ TRUE.

[$\Longleftarrow$] Suppose Verify$(I, C) = $ TRUE. Then C contains at most k vertices, and all edges in E are removed. Hence each edge is covered by some vertex in C, so C is a valid set-cover of size no more than k, i.e. I is a Yes-instance, and C is correct.

Runtime. We only need to prove a polynomial runtime for Yes-instfances. Note that the encoding length of I is $O(n + m)$, with $n := |V|$ and $m := |E|$.

The runtime is dominated by the for-loop. In a very conservative worst-case, the algorithm scans the full set of edges for each node. This takes mn time, which is clearly polynomial in the input size $n + m$.

2. **Proving VERTEXCOVER is NP-hard.** A problem Π is NP-hard if all problems in NP polynomially reduce to Π. This is impractical to prove directly. However a valid proof method is to select one problem Π_{hard} that is known to be NP-hard, then proving only one *polynomial-time reduction* from Π_{hard} to Π.

   Definition. We say decision problem B has a polynomial-time reduction (aka Karp reduction) to problem A if there exists a polynomial-time conversion $f : B \to A$ that, for every instance I_B of B generates a polynomial-size instance $I_A = f(I_B)$ of A that satisfies

$$f(I_B) \text{ is a Yes-instance of A} \iff I_B \text{ is a Yes-instance of B.}$$

   Here we will show that the NP-hard problem 3SAT reduces to VERTEXCOVER.

   Definition. The 3SAT decision problem is defined as:

   Input: n variables x_i, and a Boolean formula in conjunctive normal form with m clauses, each of size exactly 3.

   Goal: Decide if there exists an assignment $x \in \{\text{TRUE}, \text{FALSE}\}^n$ such that the Boolean formula evaluates as true.

   Example: The following is a 3SAT Boolean formula with $n = 4$ and $m = 3$.

$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4})$$

   The assignment $\sigma = (\text{TRUE}, \text{FALSE}, \text{FALSE}, \text{FALSE})$ satisfies the formula above.

   Prove that VERTEXCOVER is NP-hard by reducing from 3SAT. A proof should:

   (i) Give a conversion method f.

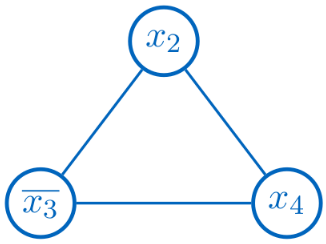
   (ii) Prove the if-and-only if correctness of f.

   (iii) Argue all times and the generated instance-sizes are polynomial.

   Solution.

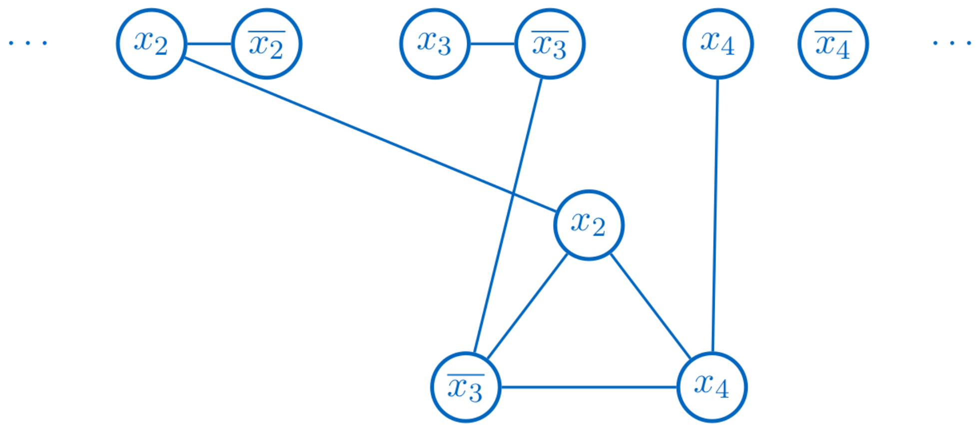
   - ***Instance conversion***. Let f be a map from 3SAT to VERTEXCOVER instances. The procedure is the following:

     (a) For every clause $(x_{j_1} \vee x_{j_2} \vee x_{j_3})$, $j = 1, \ldots, m$ generate a clause gadget. This is a complete size 3 graph where each node represents a literal in j. For instance, from clause $(x_2 \vee \overline{x_3} \vee x_4)$ generates the following clause gadget.



(b) For every variable x_i, $i = 1, \dots, n$ generate one node for x_i node and one for its negation $\overline{x_i}$. Connect each pair with an edge. Call these literal gadgets.

(c) For every clause j, connect each node x_{j_t}, $t = 1, 2, 3$, in the clause gadget to its corresponding node in the literal gadget with an edge.
The clause gadget example, would for instance connect like so:



(d) Finally, set the target parameter $k = n + 2m$.

- ***Correctness***. We need to prove that $I_{3\text{SAT}}$ is Yes $\iff$ $f(I_{3\text{SAT}})$ is Yes.
  [$\Longrightarrow$]: $I_{3\text{SAT}}$ is a Yes-instance $\Longrightarrow$ $f(I_{3\text{SAT}})$ is a Yes-instance.
  Suppose $I_{3\text{SAT}}$ is a yes instance so there exists a satisfying assignment $x^* \in \{\text{TRUE}, \text{FALSE}\}^n$.
  We can construct a vertex cover U in $f(I_{3\text{SAT}})$ from x^* as follows:
  - For each variable $i = 1, \dots, n$ we select one node in the literal gadget into U;
    If $x_i^* = \text{TRUE}$ select the node corresponding to x_i, otherwise select $\overline{x_i}$.
  - Because $I_{3\text{SAT}}$ is a Yes-instance every clause j must contain at least one satisfied literal (i.e. one that 'agrees' with the assignment x^*).
    For every clause j select one such satisfied literal. Identify the two *other* literals in clause j and add their nodes in the corresponding clause gadget to U.
    E.g. if $(x_2^*, x_3^*, x_4^*) = (\text{TRUE}, \text{FALSE}, \text{FALSE})$ and we have gadget $(x_2 \vee \overline{x_3} \vee x_4)$, both x_2 and $\overline{x_3}$ are satisfied under x^*. We can select either, and put the two unselected literals' respective nodes into set U.

Claim: The resulting vertex set $U \subseteq V$ is a vertex cover.

Every edge inside a literal gadget is covered by U because one node in each node-pair is selected. This is because each x_i is either TRUE or FALSE.

Every edge inside a clause gadget is covered by U because two nodes in each triplet are selected; any two nodes cover all edges in a complete size-3 subgraph.

Only edges between literal and clause gadgets may remain uncovered. Consider one such edge. If the edge's endpoint in the clause gadget is selected in U the edge is covered. If the endpoint in the clause gadget is not in U, we can argue its other endpoint must be. If a clause gadget node is not in U its literal must be satisfied under x^*. In this case the edge's endpoint in the literal gadget is in U, so the edge is covered. This proves that the set U is a vertex cover.

Finally, note that the vertex cover U contains exactly $n + 2m$ nodes; n from the literal gadgets and $2m$ from the clause gadgets. So $|U| = k$ as required.

Hence the procedure f correctly maps Yes-instances of 3SAT to Yes-instances of VERTEXCOVER.

$[\Longleftarrow]$: $f(I_{3\text{SAT}})$ is a Yes-instance $\implies$ $I_{3\text{SAT}}$ is a Yes-instance.

Suppose $f(I_{3\text{SAT}})$ is a Yes-instance, i.e. there is a vertex cover U of size at most k.

Claim: U contains exactly one node in each literal and two in each clause gadget.

Since U is a vertex cover it covers all edges. To cover all edges inside the literal gadgets takes a minimum of n nodes. To cover all edges inside the clause gadgets takes a minimum of $2m$ nodes. But $|U| \leq k = n + 2m$. So the size of U is exactly $n + 2m$. Now, if one gadget has more nodes in U than the allotted 1 or 2, some other gadget would have too few, contradicting U being a set cover. The claim follows.

We can construct a satisfying assignment x^* by scanning literal gadgets. Let x_i^* be TRUE if the node representing $x_i =$ TRUE is in U; otherwise set $x_i^* =$ FALSE.

Suppose by contradiction that some clause j is not satisfied under x^*. Consider the clause gadget of j. The gadget has 2 nodes in U Then there is a clause gadget j for which all 3 nodes connect to unselected nodes in the literal gadgets. Because there is no more than 2 selected nodes in each clause gadget, there is an unseleced node connected to an unselected literal-node; this is an uncovered edge. This contradicts the assumption that U is a size k set cover.

- ***Polynomial time and space***.

  The instance conversion is a polynomial-time operation. We can assume that creating nodes and edges are $O(1)$ operations (this could be perhaps be up to $O(nm)$ for some data structures, but we will see it does not matter).

  The process creates $O(n)$ time literal gadgets and $O(m)$ clause gadget . One can generate the edges between the two gadget types by looping over the clauses and searching the literal-nodes one-by-one, which takes $O(mn)$ time. Assigning k is no slower.

  Finally, note that the resulting instance has $2n + 3m$ nodes, $n + 6m$ edges, and $k = n + 2m$. The size of such an instance is clearly bounded by a polynomial size in the encoding size of $I_{3\text{Sat}}$.

  This shows that 3SAT reduces to VERTEXCOVER.

Since 3SAT is NP-hard, it follows that VERTEXCOVER is as well. $\square$