Intractability: Decision Problems and Reductions

1 Introduction

A **decision problem** D is a problem for which the answer is either a YES or a NO. Equivalently, we may write the answer as 1 or 0. We introduced the complexity classes of decision problems **P** and **NP**:

- **P:** is the class of decision problems for which there exists an algorithm which correctly computes the answer on all instances in *polynomial time*.
- **NP:**, which stands for Nondeterministic-Polynomial time, is the class of decision problems D for which there exists a polynomial time verifier V_D which for every input x takes a polynomial sized certificate y and verifies if $D(x) =$ YES or not. More precisely:

  $D(x) =$ YES if and only if there exists a certificate y and a constant c such that $|y| \leq |x|^c$ and $V_D(x, y) =$ YES.

  Informally, **NP** is the class of decision problems for which a YES instance can be verified in polynomial time.

To construct a hierarchy of decision problems (ordered by difficulty), we defined the notion of a reduction. A **Polynomial Time Reduction** (*Karp Reduction*) from problem $\mathcal{A}$ to problem $\mathcal{B}$ is an algorithm F with the following properties,

- F takes an instance a of problem $\mathcal{A}$ and creates $F(a) = b$, which is an instance of problem $\mathcal{B}$.
- F the property that $\mathcal{A}(a) =$ YES if and only if $\mathcal{B}(b) =$ YES. Note that this requires both of the following to happen:
  1. If $\mathcal{A}(a) =$ YES then $\mathcal{B}(b) =$ YES.
  2. If $\mathcal{A}(a) =$ NO then $\mathcal{B}(b) =$ NO.
- F runs in polynomial time in the size of its input, $|x|$.

For two decision problems $\mathcal{A}$ and $\mathcal{B}$, we say that $\mathcal{A}$ is reducible to $\mathcal{B}$, which we write as $\mathcal{A} \leq_p \mathcal{B}$, if there exists a polynomial time reduction F from $\mathcal{A}$ to $\mathcal{B}$ as defined above. Intuitively, if $\mathcal{A} \leq_p \mathcal{B}$, we say that: "$\mathcal{B}$ is at least as hard as $\mathcal{A}$."

Using our notion of reductions, we can defined the notion of being **NP-Hard**. We say that a problem $\mathcal{A}$ is **NP-Hard** if for any problem $X \in$ **NP**, $X \leq_p \mathcal{A}$. In other words, $\mathcal{A}$ is at least as hard as all problems in NP.

Finally, we can define the notion of being **NP-Complete**. We say a decision problem $\mathcal{A}$ is **NP-Complete** if the following two things are true about $\mathcal{A}$: (1) $\mathcal{A}$ is in **NP** and (2) $\mathcal{A}$ is **NP-Hard**. Informally, if $\mathcal{A}$ is **NP-Complete**, then it is among the hardest problems in **NP**. The first problem shown to be **NP**-Complete was the CIRCUIT-SAT problem (see the *Cook-Levin Theorem*) restated below:

- **CIRCUIT-SAT:** Given a boolean circuit composed of a polynomial number of AND, OR and NOT gates, is there an input to the circuit which makes the circuit output 1?

Once we know a candidate NP-complete problem $\mathcal{A}$ to show NP-completeness for any other problem $\mathcal{B}$ we only need to show a polynomial time reduction from $\mathcal{A}$ to $\mathcal{B}$. Since any NP-complete problem is reducible to $\mathcal{A}$ and since $\mathcal{A}$ is reducible to $\mathcal{B}$ this will show that any NP-complete problem is reducible to $\mathcal{B}$ thereby proving that $\mathcal{B}$ is NP-complete.

We will now show some examples of reductions and prove some problems to be NP-Complete.

2 ILP is NP-Complete

We will show that ILP (Integer Linear Programming) is NP-complete by first showing that it is in NP and then showing that it is NP-hard by reducing SAT to ILP.

- **ILP:** (Integer Linear Programming) Given a linear program with polynomially many variables and polynomially many linear constraints is there a feasible integral solution to the LP? Note that the decision version of ILP doesn't have an objective function.

ILP is in NP: Consider a verifier for ILP that takes an assignment of values to the variables involved and verifies each of the constraints involved. Since there are at most polynomially many constraints, this verifier runs in polynomial time, and thus ILP $\in$ NP.

Reducing SAT to ILP: Given a SAT instance, we will construct an ILP such that the SAT instance is satisfiable if and only if the ILP has a feasible solution.

1. For every literal x_i in the SAT instance we define a variable y_i in our ILP such that $y_i = 1$ if x_i =true and $y_i = 0$ if x_i =false. Note that now $1 - y_i$ represents exactly $\neg x_i$. We add constraint of the form $0 \leq y_i \leq 1$ for all i so as to get $y_i \in \{0, 1\}$.
2. For every clause we introduce a constraint as follows. If the clause is $(x_2 \vee \neg x_5 \vee x_7)$ for example, we add the constraint $y_2 + (1 - y_5) + y_7 \geq 1$. Note that this constraint is satisfied if and only if its corresponding clause is satisfied.

Next we show that if there exists a feasible solution to this ILP then there must exist a satisfiable solution to the SAT instance.

Take the feasible solution an for all i such that $y_i = 1$ set x_i = TRUE and set x_i = FALSE for the rest. Suppose there wasn't a feasible solution to the ILP. This implies that there is no satisfying assignment for the SAT instance because if there was one we could use it to construct a feasible solution to the ILP by setting $y_i = 1$ for all i such that $x_i =$ TRUE and $y_i = 0$ for the rest thereby completing the reduction. $\square$

3 Reducing CLIQUE to INDEPENDENT-SET

Recall **CLIQUE** and **INDEPENDENT-SET**:

- **CLIQUE:** Given graph $G = (V, E)$ and integer k, is there a set of vertices $C \subseteq V$ with $|C| \geq k$ that form a complete graph?
- **INDEPENDENT-SET:** Given graph $G = (V, E)$ and integer k, is there a set of vertices $I \subseteq V$ with $|I| \geq k$ such that for any $u, v \in I$, $(u, v) \notin E$?

Recall that CLIQUE is NP-Complete (this can be shown via a reduction from 3-SAT to CLIQUE). We will now see that INDEPENDENT-SET is also NP-Complete by giving a reduction from CLIQUE as follows:

1. **Show that INDEPENDENT-SET is in NP:** To prove this, we need to prove there exists a verifier $\mathcal{V}(x, y)$. Let $x = (G, k)$ be a "yes" input. Let y be I that satisfies the condition.

   It takes $O(|I|)$ to check that $|I| = k$. It takes $O(|I|^2)$ to check that for every $u, v \in I$, $(u, v) \notin E$. This checks in polynomial time that the certificate y proves that x is a valid input. Therefore, INDEPENDENT-SET is in NP.

2. **Show INDEPENDENT-SET is NP-hard:** We prove this by giving a Karp-reduction of CLIQUE to INDEPENDENT-SET.

   (a) Given an input $x = (G, k)$ to CLIQUE, create input G' which has the same vertices, but has edge (u, v) if and only if $(u, v) \notin E$. This takes $O(|E|)$ time, so this reduction takes polynomial time.

   (b) If I is a set of vertices that form a k-INDEPENDENT-SET for G', then $C = I$ is a k-Clique for G because for $u, v \in I$, INDEPENDENT-SET says that $(u, v) \notin E'$, but this implies that $(u, v) \in E$ for the CLIQUE problem due to the method of construction. This shows that there are edges between every pair of nodes in C. In addition $|C| = k$, and so C is a k-clique.

   (c) If C is a set of vertices that form a k-Clique in G, then $I = C$ is a k-Independent set for G'. This is because $u, v \in C$ implies that $(u, v) \in E$ for CLIQUE, and this implies that $(u, v) \notin E'$ for INDEPENDENT-SET. Since $|I| = |C| = k$, this shows that there are k elements in the construction G' that are not adjacent to each other.

This proves CLIQUE reduces to INDEPENDENT-SET in polynomial time, which means that INDEPENDENT-SET is at least as hard as CLIQUE, so k-INDEPENDENT-SET is NP-hard. $\square$

4 Reducing INDEPENDENT-SET to VERTEX-COVER

- **VERTEX-COVER:** Given graph $G = (V, E)$ and integer d, is there a set of vertices $S \subseteq V$ with $|S| \leq d$ such that for any $(u, v) \in E$ either $u \in S$ or $v \in S$ (or both).

We will prove that VERTEX-COVER is **NP**-Complete by giving a reduction from INDEPENDENT-SET, which we proved is NP-Complete.

1. **Show that VERTEX-COVER is in NP:** Given a graph $G = (V, E)$, an instance of VERTEX-COVER for which there exists a VERTEX-COVER of size at most d, we can use the vertex cover set S itself to verity if it is indeed true.

   First we check that $|S| \leq d$ which takes $O(|S|)$ time and then we check for each pair $(u, v) \in E$ either $u \in S$ or $v \in S$ in $O(m)$ time. Overall, since $|S| \leq n$, this is polynomial in the size of the input.

2. **Show that VERTEX-COVER is NP-Hard:** To show this, we reduce INDEPENDENT-SET to VERTEX-COVER.

   (a) For an input $x = (G, k)$ to INDEPENDENT-SET, we simply use the set our reduction to return $y = (G, n-k)$ an instance to VERTEX-COVER. That is, we keep G the same and just set $d = n - k$. We will now prove that G has an INDEPENDENT-SET of size $\geq k$ if and only if G has a VERTEX-COVER of size $\leq n - k$.

   (b) If G has an independent set S with $|S| \geq k$, then we show that the set $V - S$ forms a vertex cover. Note that $|V - S| \leq n - k$. Let $(u, v) \in E$ be an edge in G, because S is an independent set, at most one of u or v can be in S. Which means that at least one of u or v is in $V - S$.

   (c) Similarly, let T be a vertex cover in G with $|T| \leq n-k$. Then we claim that $S = V-T$ forms an independent set in G of size at least k. First notice that $|V-T| \geq n-n+k = k$. Secondly, for any edge $(u, v) \in E$, at most one of u or v are in S. This is because otherwise, if both u and v were in S, then the edge (u, v) would not have been covered in T.

Hence, VERTEX-COVER is NP-Complete. □

5 Reducing VERTEX-COVER to Set Cover

- **Set Cover:** Given a set S of n elements $\{1, 2, \ldots, n\}$ and m sets $S_1, \ldots, S_m$ where $S_i \subseteq S$, does there exist a set of k sets $S_{i_1}, \ldots, S_{i_k}$ such that $S_{i_1} \cup \cdots \cup S_{i_k} = S$?

We will give a proof that Set Cover is NP-Complete by giving a reduction to Set Cover. Given that VERTEX-COVER is NP-complete, we prove that Set Cover is NP-Complete.

1. **Show that Set Cover is in NP:** To prove this, we need to prove there exists a verifier $\mathcal{V}(x, y)$. Let $x = S, S_1, \ldots, S_m$ be a "yes" input. Let y be $S_{i_1}, \ldots, S_{i_k}$ that satisfies the condition.

   In $O(k)$ time, we can figure whether or not we have exactly k sets. In $O(kn)$ time we can determine whether or not all all elements in S is in the union. This proves we have a polynomial time verifier, which means that Set Cover is in NP.

2. **Show that Set Cover is NP-Hard:** To prove this, we reduce VERTEX-COVER to Set Cover.

   (a) For an input $x = (G, k)$ to VERTEX-COVER, we make $R(x) = S, S_1, \ldots, S_m$. Let S be the set of all edges $e_j \in E$. For each $v_i \in V$, create set S_i. This set contains the set of edges e_j that touch v_i. This new input is polynomial because $|S| = |E|$ and each set S_i has size at most $|E|$ and there are $|V|$ sets.

   (b) If there is a k-VERTEX-COVER, there is a k-Set Cover. If the vertex $v_i \in V'$ is part of the vertex cover, then S_i is part of the set cover. Since every edge $e_j \in E$ is incident to some vertex $v_i \in V'$, this means that every element $e_j \in S$ is covered the set S_i.

   (c) If there is a k-Set Cover, there is a k-VERTEX-COVER. If S_i is in the set cover, choose v_i to be in the vertex cover. Every element e_j is contained in some set S_i. By construction, this means every edge e_j is incident to the vertex v_i that got chosen. Since there are k sets, there will be k vertices chosen for the vertex cover.

□

6 Reducing Hamiltonian Cycle to Hamiltonian Path

Given a directed graph $G = (V, E)$:

- **Hamiltonian Cycle:** Is there a cycle that visits every vertex exactly once?
- **Hamiltonian Path:** Is there a path that visits every vertex exactly once?

Given that Hamiltonian Cycle is NP-Complete, we will prove that Hamiltonian Path is NP-Complete.

1. **Show Hamiltonian Path is in NP:** To prove this, we need to prove that there exists a verifier $\mathcal{V}(x, y)$. Let $x = G$ be a "yes" input. Let y be a path P that satisfies the condition.

   We can verify that the path traverses every vertex exactly once, then check the path to ensure that every edge in the path is an edge in the graph. Naively, it takes O(n^2) to check that every vertex is traversed exactly once and O(nm) to check that every edge in the path is in the graph.

2. **Show that Hamiltonian Path is NP-Hard:** We prove this by giving a Karp-reduction of Hamiltonian Cycle to Hamiltonian Path.

(a) Given an instance of Hamiltonian Cycle G, choose an arbitrary node v and split it into two nodes v' and v''. All directed edges into v now have v' as an endpoint, and all edges leaving v leave v'' instead. We call this new graph G'. The transformation takes at most O(E).

(b) If there is a Hamiltonian Cycle in G, there is a Hamiltonian Path on G'. We can use the edges of the cycle on G as the path on G', but our path must begin on v'' and end on v'.

(c) If there is a Hamiltonian Path on on G', there is a Hamiltonian Cycle on G. The path must begin at v'' and end at v', since there are no edges into v'' or out of v'. Thus we can use the path on G' as a cycle on G once v' and v'' are remerged.

Thus we have given a polynomial time reduction from Hamiltonian Cycle to Hamiltonian Path, which means that Hamilitonian Path is NP-Complete. □

7 Reducing 3-SAT to 3-COLOR

- **3-COLOR:** Given an undirected graph G, is there a valid 3-coloring of its nodes. (i.e. is it possible to color the nodes with at most three different colors such that no two adjacent nodes are the same color)

We will prove that 3-COLOR is NP-Complete by giving a reduction from 3-SAT.

1. **Show that 3-COLOR is in NP:** Consider a verifier that returns "YES" if and only if the following two criterion hold: (a) that at most three distinct colors are used and (b) that no two adjacent nodes are the same color - this can be done in polynomial-time with respect to the number of edges in the graph. Thus 3-COLOR is in **NP**.

2. **Show that 3-COLOR is NP-Hard:** We prove this by giving a reduction of 3-SAT to 3-COLOR. This is done by constructing a graph G for a given 3-SAT problem that is 3-colorable if and only if the 3-SAT problem is satisfiable.

   (a) Create three nodes, labeled v_T, v_F and v_N and establish an edge between each pair of nodes. Note that these three nodes *must* each be a different color. We will associate the color assigned to v_T as TRUE and the color assigned to v_F as FALSE.

   (b) For each variable x_i:

      i. Add two nodes, v_{x_i} and $v_{\neg x_i}$ to G.

      ii. Add an edge between v_{x_i} and $v_{\neg x_i}$. This ensures that v_{x_i} and $v_{\neg x_i}$ cannot be assigned the same color (and thus value).

      iii. Add two edges: one from v_N to v_{x_i} and another from v_N to $v_{\neg x_i}$. This forces v_{x_i} and $v_{\neg x_i}$ to take on values of either TRUE or FALSE.

(c) **A Gadget for Clause Satisfiability:** Finally, given a clause, we need a *gadget* that we can add to G such that it will be colorable if and only if there exists an assignment of variables such that the clause is satisfied.

Intuitively, we will construct an OR-gate gadget in which the output is hard-wired to "TRUE" and the nodes in the gadget can be colored iff both inputs are not "FALSE." Since our OR-gate would need to take in three values, we break this down into two 2-input OR gates, an "alpha" gate and a "beta" gate, with the "beta" gate serving as an input to the "alpha" gate.

Without loss of generality, suppose that we have a clause of the form $(x_1 \vee x_2 \vee \neg x_3)$. (See the included diagram)

i. Here is the "alpha" gate:

   A. Add three nodes to G: $v_{\alpha,out}$, $v_{\alpha,1}$ and $v_{\alpha,2}$.

   B. Add edges connecting $v_{\alpha,out}$ to v_N and $v_{\alpha,out}$ to v_F. This forces $v_{\alpha,out}$ to be colored the same as v_T.

   C. Add edges connecting $v_{\alpha,out}$, $v_{\alpha,1}$ and $v_{\alpha,2}$ pairwise; thus they must all be different colors, which prohibits $v_{\alpha,1}$ and $v_{\alpha,2}$ from being colored the same as v_T).

ii. Here is the "beta" gate:

   A. Add three nodes to G: $v_{\beta,out}$, $v_{\beta,1}$ and $v_{\beta,2}$.

   B. Add edges connecting $v_{\beta,out}$, $v_{\beta,1}$ and $v_{\beta,2}$ pairwise.

iii. Finally we must connect the gates and the variables:

   A. Add an edge connecting $v_{\beta,out}$ to $v_{\alpha,2}$. (this is how the "beta" OR-gate is an input to the "alpha" OR-gate).

   B. Add an edge connecting $v_{\alpha,1}$ to v_{x_1}, an edge connecting $v_{\beta,1}$ to v_{x_2} and $v_{\beta,2}$ to $v_{\neg x_3}$. (this connects the gadget to the variables)

Next we show that $v_{\alpha,out}$ is colorable if and only if at least one of v_{x_1}, v_{x_2} or $v_{\neg x_3}$ is colored the same as v_T. First note that if $v_{\alpha,out}$ is colorable, it must have the same color as v_T. This means that if v_{x_1} and $v_{\beta,out}$ are both colored the same as v_F, then $v_{\alpha,1}$ and $v_{\alpha,2}$ must both be colored the same as v_N, which is not possible since they have an edge connecting them. If at least one of v_{x_1} and $v_{\beta,out}$ is colored the same as v_T (WLOG let us say $v_{\alpha,1}$ is colored the same as v_T), then this means that we can color $v_{\alpha,1}$ or $v_{\alpha,2}$ the same as v_F and v_N respectively. By the same reasoning, we can see that if we require $v_{\beta,out}$ to be colored the same as v_T, then the above logic applies the inputs v_{x_2} and $v_{\neg x_3}$ as well.

If we add a gadget as we did above for every clause in our 3-SAT problem, then the constructed graph G is 3-colorable if and only if the 3-SAT instance is satisfiable. Thus we have given a polynomial time reduction from 3SAT to 3COLOR, proving that 3COLOR is NP-complete.

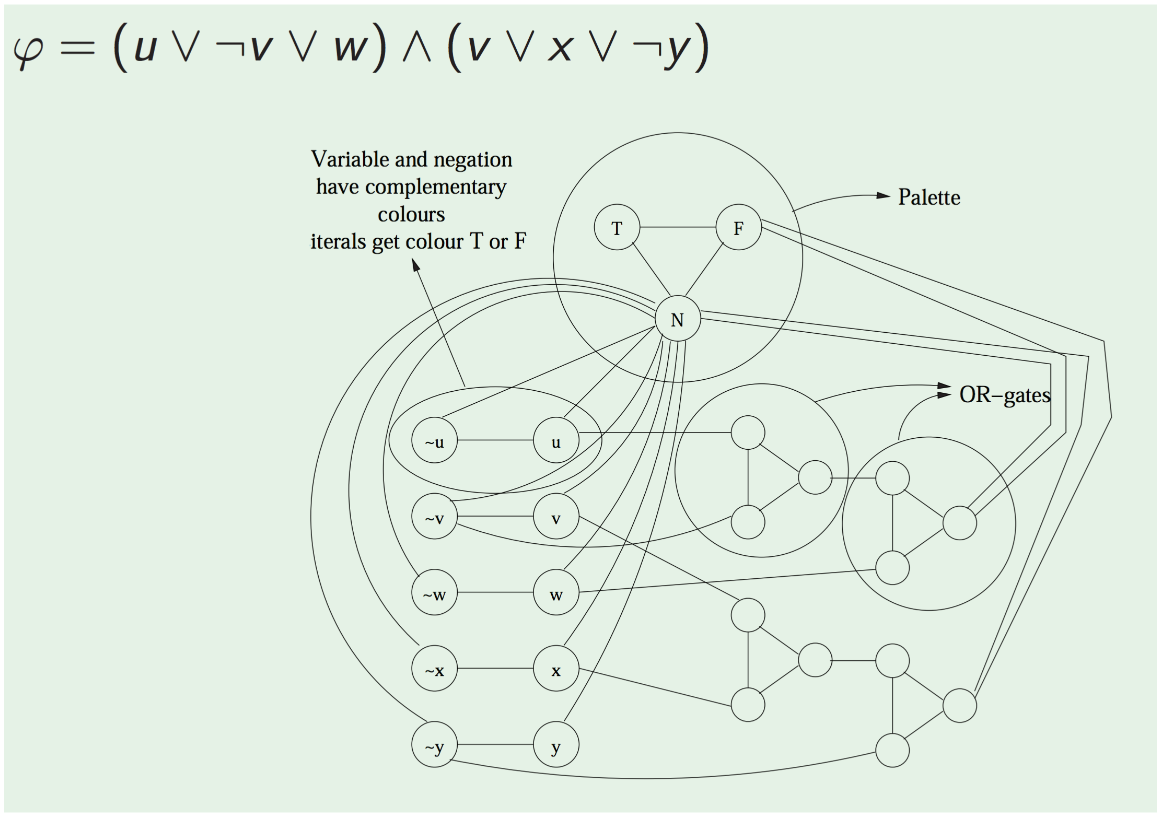
$\varphi = (u \vee \neg v \vee w) \wedge (v \vee x \vee \neg y)$
Variable and negation have complementary colours iterals get colour T or F
Palette
OR−gates
T
F
N
~u
u
~v
v
~w
w
~x
x
~y
y

Image taken from: `https://cgi.csc.liv.ac.uk/~igor/COMP309/3CP.pdf`