NP-Completeness reductions

NP-hardness & NP-completeness

Definition: A problem $B$ is NP-hard iff every problem $A \in NP$ satisfies $A \leq_p B$

Definition: A problem $B$ is NP-complete iff $B$ is NP-hard and $B \in NP$

Even though we seem to have lots of hard problems in $NP$ it is not obvious that such super-hard problems even exist!

Reductions by Simple Equivalence

Show: Independent-Set $\leq_p$ Clique

Independent-Set:
- Given a graph $G = (V, E)$ and an integer $k$, is there a subset $U$ of $V$ with $|U| \geq k$ such that no two vertices in $U$ are joined by an edge.

Clique:
- Given a graph $G = (V, E)$ and an integer $k$, is there a subset $U$ of $V$ with $|U| \geq k$ such that every pair of vertices in $U$ is joined by an edge.

Independent-Set $\leq_p$ Clique

- Given $(G, k)$ as input to Independent-Set where $G = (V, E)$
- Transform to $(G', k)$ where $G' = (V, E')$ has the same vertices as $G$ but $E'$ consists of precisely those edges that are not edges of $G$
- $U$ is an independent set in $G$ $\iff$ $U$ is a clique in $G'$

Satisfiability

- Boolean variables $x_1, \ldots, x_n$, taking values in $\{0, 1\}$: $0$ = false, $1$ = true
- Literals
  - $x_i$ or $\neg x_i$ for $i = 1, \ldots, n$
- Clause
  - a logical OR of one or more literals
    - e.g. $(x_1 \lor \neg x_3 \lor x_7 \lor x_{12})$
- CNF formula
  - a logical AND of a bunch of clauses
Satisfiability
- CNF formula example
  \( (x_1 \lor \neg x_3 \lor x_7 \lor x_{12}) \land (x_2 \lor \neg x_4 \lor x_7 \lor x_9) \)
  - If there is some assignment of 0’s and 1’s to the variables that makes it true then we say the formula is \textit{satisfiable}
    - the one above is, the following isn’t
    - \( x_1 \land (\neg x_1 \lor x_4) \land (\neg x_2 \lor x_5) \land \neg x_3 \)
- \( \text{SAT:} \) Given a formula \( F \), is it satisfiable?

Cook-Levin Theorem
- Theorem (Cook-Levin 1971):
  \( \text{SAT} \subseteq \text{P} \iff \text{P} = \text{NP} \)
  - Follows by showing that \( \text{SAT} \) is \textit{NP}-complete

Recall this useful property of polynomial-time reductions
- Theorem: If \( A \leq_p B \) and \( B \leq_p C \) then \( A \leq_p C \)

Cook-Levin Theorem & Implications
- Theorem: \( \text{SAT} \) is \textit{NP}-complete
- Corollary: \( C \) is \textit{NP}-hard \( \iff \) \( \text{SAT} \leq_p C \)
  - (or \( B \leq_p C \) for any \textit{NP}-complete problem \( B \))
- Proof:
  - If \( B \) is \textit{NP}-hard then every problem in \textit{NP} polynomial-time reduces to \( B \), in particular \( \text{SAT} \) does since it is in \textit{NP}
  - For any problem \( A \) in \textit{NP}, \( A \leq_p \text{SAT} \) and so if \( \text{SAT} \leq_p C \) we have \( A \leq_p C \).
  - therefore \( C \) is \textit{NP}-hard if \( \text{SAT} \leq_p C \)

Steps to Proving Problem \( B \) is \textit{NP}-complete
- Show \( B \) is \textit{NP}-hard:
  - State: Reduction is from \textit{NP}-hard Problem \( A \)
  - Show what the map \( f \) is
    - Argue that \( f \) is polynomial time
    - Argue correctness: \textit{two directions} Yes for \( A \) implies Yes for \( B \) and vice versa.
- Show \( B \) is in \textit{NP}
  - State what certificate is and why it works
  - Argue that it is polynomial-time to check.

Another \textit{NP}-complete problem:
\textit{Satisfiability} \( \leq_p \text{Independent-Set} \)
- A Tricky Reduction:
  - mapping \textit{CNF} formula \( F \) to a pair \( <G,k> \)
  - Let \( m \) be the number of clauses of \( F \)
  - Create a vertex in \( G \) for each literal in \( F \)
  - Join two vertices \( u \), \( v \) in \( G \) by an edge if
    - \( u \) and \( v \) correspond to literals in the same clause of \( F \), \textit{(green edges)}
    - \( u \) and \( v \) correspond to literals \( x \) and \( \neg x \) (or vice versa) for some variable \( x \). \textit{(red edges)}.
  - Set \( k = m \)
  - Clearly polynomial-time
Satisfiability \(\leq_p\) Independent-Set

\[ F: (x_1 \lor \neg x_3 \lor x_4) \land (x_2 \lor \neg x_4 \lor x_3) \land (x_2 \lor \neg x_1 \lor x_3) \]

Correctness:

- If \(F\) is satisfiable then there is some assignment that satisfies at least one literal in each clause.

- Consider the set \(U\) in \(G\) corresponding to the first satisfied literal in each clause.

- Since \(U\) has only one vertex per clause, no two vertices in \(U\) are joined by green edges.

- Since a truth assignment never satisfies both \(x\) and \(\neg x\), \(U\) doesn’t contain vertices labeled both \(x\) and \(\neg x\) and so no vertices in \(U\) are joined by red edges.

- Therefore \(G\) has an independent set, \(U\), of size at least \(m\).

- Therefore \((G, m)\) is a YES for independent set.

Given assignment \(x_1 = x_2 = x_3 = x_4 = 1\), \(U\) is as circled

Given \(U\), satisfying assignment is \(x_1 = x_2 = x_3 = 0\) or \(x_2 = 0\) or 1

Independent-Set is NP-complete

- We just showed that Independent-Set is NP-hard and we already knew Independent-Set is in NP.

- Corollary: Clique is NP-complete

  - We showed already that Independent-Set \(\leq_p\) Clique and Clique is in NP.
Reductions from a Special Case to a General Case

Show: Vertex-Cover \(\leq P\) Set-Cover

- **Vertex-Cover:**
  - Given an undirected graph \(G=(V,E)\) and an integer \(k\) is there a subset \(W\) of \(V\) of size at most \(k\) such that every edge of \(G\) has at least one endpoint in \(W\)? (i.e. \(W\) covers all edges of \(G\)).

- **Set-Cover:**
  - Given a set \(U\) of \(n\) elements, a collection \(S_1, \ldots, S_m\) of subsets of \(U\), and an integer \(k\), does there exist a collection of at most \(k\) sets whose union is equal to \(U\)?

The Simple Reduction

Transformation \(f\) maps \(\langle G=(V,E),k \rangle\) to \(\langle U,S_1, \ldots, S_m,k' \rangle\)

- \(U \leftarrow E\)
  - For each vertex \(v \in V\) create a set \(S_v\) containing all edges that touch \(v\)
- \(k' \leftarrow k\)
  - Reduction \(f\) is clearly polynomial-time to compute
  - We need to prove that the resulting algorithm gives the right answer.

Proof of Correctness

Two directions:

- If the answer to Vertex-Cover on \((G,k)\) is YES then the answer for Set-Cover on \((T(G),k)\) is YES
  - If a set \(W\) of \(k\) vertices covers all edges then the collection \(\{S_v \mid v \in W\}\) of \(k\) sets covers all of \(U\)
- If the answer to Set-Cover on \((T(G),k)\) is YES then the answer for Vertex-Cover on \((G,k)\) is YES
  - If a subcollection \(S_{v_1}, \ldots, S_{v_k}\) covers all of \(U\) then the set \(\{v_1, \ldots, v_k\}\) is a vertex cover in \(G\).

More Reductions

Show: Independent Set \(\leq P\) Vertex-Cover

- **Vertex-Cover:**
  - Given an undirected graph \(G=(V,E)\) and an integer \(k\) is there a subset \(W\) of \(V\) of size at most \(k\) such that every edge of \(G\) has at least one endpoint in \(W\)? (i.e. \(W\) covers all edges of \(G\)).

- **Independent-Set:**
  - Given a graph \(G=(V,E)\) and an integer \(k\), is there a subset \(U\) of \(V\) with \(|U| \geq k\) such that no two vertices in \(U\) are joined by an edge.

Reduction Idea

Claim: In a graph \(G=(V,E)\), \(S\) is an independent set iff \(V-S\) is a vertex cover

- \(\Rightarrow\) Let \(S\) be an independent set in \(G\)
  - Then \(S\) contains at most one endpoint of each edge of \(G\)
  - At least one endpoint must be in \(V-S\)
  - \(V-S\) is a vertex cover
- \(\Leftarrow\) Let \(W=V-S\) be a vertex cover of \(G\)
  - Then \(S\) does not contain both endpoints of any edge (else \(W\) would miss that edge)
  - \(S\) is an independent set

Reduction

Map \(\langle G,k \rangle\) to \(\langle G,n-k \rangle\)

- Previous lemma proves correctness
- Clearly polynomial time
- We also get that Vertex-Cover \(\leq P\) Independent Set
Problems we already know are NP-complete

- Satisfiability
- Independent-Set
- Clique
- Vertex-Cover

There are 1000’s of practical problems that are NP-complete, e.g. scheduling, optimal VLSI layout etc.

A particularly useful problem for proving NP-completeness

3-SAT: Given a CNF formula $F$ having precisely 3 variables per clause (i.e., in 3-CNF), is $F$ satisfiable?

Theorem: 3-SAT is NP-complete

Alternate Proof based on CNFSAT:

3-SAT $\in$ NP

Certificate is a satisfying assignment

Just like SAT it is polynomial-time to check the certificate

CNFSAT $\leq_p$ 3-SAT

Reduction:

- map CNF formula $F$ to another CNF formula $G$ that has precisely 3 variables per clause.
- $G$ has one or more clauses for each clause of $F$
- $G$ will have extra variables that don’t appear in $F$
- for each clause $C$ of $F$ there will be a different set of variables that are used only in the clauses of $G$ that correspond to $C$

CNFSAT $\leq_p$ 3-SAT

Goal:

- An assignment $a$ to the original variables makes clause $C$ true in $F$ if
  - there is an assignment to the extra variables that together with the assignment $a$ will make all new clauses corresponding to $C$ true.
- Define the reduction clause-by-clause
  - We’ll use variable names $z_j$ to denote the extra variables related to a single clause $C$ to simplify notation
  - in reality, two different original clauses will not share $z_j$

For each clause $C$ in $F$:

- If $C$ has 3 variables:
  - Put $C$ in $G$ as is
- If $C$ has 2 variables, e.g. $C = (x_1 \lor x_2) \land (x_3 \lor \neg x_1) \land (x_4 \lor \neg x_2)$
  - Use a new variable $z$ and put two clauses in $G$
    - $(x_1 \lor \neg x_2 \lor z) \land (x_1 \lor \neg x_2 \lor \neg z)$
    - If original $C$ is true under assignment $a$ then both new clauses will be true under $a$
    - If new clauses are both true under some assignment $b$ then the value of $z$ doesn’t help in one of the two clauses so $C$ must be true under $b$
- If $C$ has 1 variable: e.g. $C = x_1$
  - Use two new variables $z_1, z_2$ and put 4 new clauses in $G$
    - $(x_1 \lor \neg z_1 \lor z_2) \land (x_1 \lor \neg z_1 \lor \neg z_2) \land (x_1 \lor z_1 \lor \neg z_2) \land (x_1 \lor z_1 \lor z_2)$
  - If original $C$ is true under assignment $a$ then all new clauses will be true under $a$
  - If new clauses are all true under some assignment $b$ then the values of $z_1$ and $z_2$ don’t help in one of the 4 clauses so $C$ must be true under $b$
CNFSAT ≤\text{P} 3-SAT
- If $C$ has $k \geq 4$ variables: e.g. $C = (x_1 \lor \ldots \lor x_k)$
  - Use $k-3$ new variables $z_2, \ldots, z_{k-2}$ and put $k-2$ new clauses in $G$
    
    $(x_1 \lor x_2 \lor z_2) \land (\neg x_2 \lor x_3 \lor z_3) \land (\neg x_3 \lor x_4 \lor z_4) \land \ldots$
    
    $(\neg x_{k-3} \lor x_{k-2} \lor z_{k-2}) \land (\neg z_{k-2} \lor x_{k-1} \lor x_k)$
- If original $C$ is true under assignment $a$ then some $x_i$ is true for $i \leq k$. By setting $z_j$ true for all $j < i$ and false for all $j \geq i$, we can extend $a$ to make all new clauses true.
- If new clauses are all true under some assignment $b$ then some $x_i$ must be true for $i \leq k$ because $z_2 \land (\neg z_2 \lor z_3) \land \ldots \land (\neg z_{k-3} \lor z_{k-2}) \land \neg z_{k-2}$ is not satisfiable

Graph Colorability
- **Defn:** Given a graph $G=(V,E)$, and an integer $k$, a $k$-coloring of $G$ is
  - an assignment of up to $k$ different colors to the vertices of $G$ so that the endpoints of each edge have different colors.
- **3-Color:** Given a graph $G=(V,E)$, does $G$ have a 3-coloring?
- **Claim:** 3-Color is NP-complete
- **Proof:** 3-Color is in NP:
  - Hint is an assignment of red, green, blue to the vertices of $G$
  - Easy to check that each edge is colored correctly

3-SAT ≤\text{P} 3-Color
- **Reduction:**
  - We want to map a 3-CNF formula $(F)$ to a graph $(G)$ so that
    - $G$ is 3-colorable iff $F$ is satisfiable

3-SAT ≤\text{P} 3-Color
- **Variable Part:** in 3-coloring, variable colors correspond to some truth assignment (same color as T or F)

3-SAT ≤\text{P} 3-Color
- **Clause Part:** Add one 6 vertex gadget per clause connecting its 'outer vertices' to the literals in the clause
Any truth assignment satisfying the formula can be extended to a 3-coloring of the graph.

Any 3-coloring of the graph colors each gadget triangle using each color.

Any 3-coloring of the graph has an F opposite the O color in the triangle of each gadget.

Any 3-coloring of the graph has T at the other end of the blue edge connected to the F.

Any 3-coloring of the graph yields a satisfying assignment to the formula.

More NP-completeness

Subset-Sum problem

Given n integers \(w_1, \ldots, w_n\) and integer \(W\)

Is there a subset of the \(n\) input integers that adds up to exactly \(W\)?

O(nW) solution from dynamic programming but if \(W\) and each \(w_i\) can be \(n\) bits long then this is exponential time.
### 3-SAT ≤ₚ Subset-Sum

Given a 3-CNF formula with \( m \) clauses and \( n \) variables.

- Will create \( 2m + 2n \) numbers that are \( m+n \) digits long.
  - Two numbers for each variable \( x_i \):
    - \( t_i \) and \( f_i \) (corresponding to \( x_i \) being true or \( x_i \) being false).
  - Two extra numbers for each clause:
    - \( u_j \) and \( v_j \) (filler variables to handle number of false literals in clause \( C_j \)).

<table>
<thead>
<tr>
<th>( i )</th>
<th>( j )</th>
<th>( C_j := (x_1 \lor \neg \ x_2 \lor \neg \ x_3 \lor \neg \ x_4) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_1 )</td>
<td>( 1000 \ldots \ 00010 \ldots \ 1 )</td>
<td></td>
</tr>
<tr>
<td>( f_1 )</td>
<td>( 1000 \ldots \ 01001 \ldots \ 0 )</td>
<td></td>
</tr>
<tr>
<td>( t_2 )</td>
<td>( 0100 \ldots \ 01000 \ldots \ 1 )</td>
<td></td>
</tr>
<tr>
<td>( f_2 )</td>
<td>( 0100 \ldots \ 00111 \ldots \ 0 )</td>
<td></td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( u_{i \neq j} )</td>
<td>( 0000 \ldots \ 01000 \ldots \ 0 )</td>
<td></td>
</tr>
<tr>
<td>( u_{i \neq j} )</td>
<td>( 0000 \ldots \ 00100 \ldots \ 0 )</td>
<td></td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( W )</td>
<td>( 1111 \ldots \ 13333 \ldots \ 3 )</td>
<td></td>
</tr>
</tbody>
</table>