# Complexity Theory

1. We have defined a relationship $\leq_p$ among languages. This relationship is reflexive, (i.e $L \leq_p L'$ for all languages) and transitive (i.e, if $L \leq_p L'$, and $L' \leq_p L''$, then $L \leq_p L''$). Why is it not symmetric, namely, why is it that if $L \leq_p L'$, it is not necessarily true that $L' \leq_p L$?

   **Solution:** Consider two languages $L = \emptyset$ and $L' = \{a\}$ over a non-empty language $\Sigma$, and $a \in \Sigma$.

   1. $L \leq_p L'$: We transform any input into the string "$aa$", which can be done in polytime. Since $L$ is empty, and doesn't contain any string, it always returns 'No'. Thus, the reduction maps any input to "$aa$", which determines that the reduction passes or not. Thus, $L \leq_p L'$.
   2. $L' \not\leq_p L$: This is trivial, since $L = \emptyset$ has no strings, thus we cannot construct a reduction such that '$a$' in $L'$ is mapped to any string in $L$, since we basically have to map '$a$' to nothing. Thus, $L' \not\leq_p L$.

   Hence, we have given a counterexample to the symmetry of the relationship $\leq_p$, hence, it is not symmetric.

2. Show that NP is closed under union, intersection, concatenation, and Kleene star.

   **Solution:** Consider any two arbitrary languages $L_1$ and $L_2$, which have non-deterministic turing machines $M_1$ and $M_2$ respectively. To prove the above, we basically a construct a non-deterministic turing machines $M$ such that:

   1. **Union:**

      M = "On input $w$:

      1. Run $M_1$ on $w$. If $M_1$ accepts, **accept**.
      2. Run $M_2$ on $w$. If $M_2$ accepts, **accept**.
      3. **reject** otherwise. "

      Clearly, the longest branch on an input $w$ of length $n$ is $\mathcal{O}(n^{max\{k,l\}})$. Thus, $M$ is a non-deterministic turing machine that accepts $L_1 \cup L_2$, and hence, $NP$ is closed under union.

2. **Intersection:**

   M = "On input $w$:

   1. Run $M_1$ on $w$. If $M_1$ rejects, **reject**.
   2. Else run $M_2$ on $w$. If $M_2$ rejects, **rejects**.
   3. **accept** otherwise. "

   Clearly, the longest branch on an input $w$ of length $n$ is $\mathcal{O}(n^{max\{k,l\}})$. Thus, $M$ is a non-deterministic turing machine that accepts $L_1 \cap L_2$, and hence, $NP$ is closed under intersection.

3. **Concatenation:**

   M = "On input $w$:

   1. Non-deterministically split $w$ into two strings $x$ and $y$ such that $w = xy$.
   2. For all $x$ and $y$, run $M_1$ on $x$ and $M_2$ on $y$. If both accept, **accept**.
   3. **reject** otherwise. "

   Clearly, the longest branch on an input $w$ of length $n$ is $\mathcal{O}(n^{max\{k+l\}})$. Thus, $M$ is a non-deterministic turing machine that accepts $L_1 \cdot L_2$, and hence, $NP$ is closed under concatenation.

4. **Kleene Star:**

   M = "On input $w$:

   1. If $w = \epsilon$, **accept**.
   2. Non-deterministically select a number $m$ such that $1 \leq m \leq |w|$
   3. Non-deterministically split $w$ into $m$ strings such that $w_1, w_2, \ldots, w_m$.
   4. $\forall i, 1 \leq i \leq m$ : run $M_1$ on $w_i$. If $M_1$ rejects, **reject**.
   5. **accept** otherwise ($M_1$ accepted al $w_i, 1 \leq i \leq m$). "

   Steps 1 and 2 take $\mathcal{O}(n)$ time, since the size of the input is bounded by $n$. Step 3 is also doable in polytime using non-determinism. Step 4 takes polynomial time since a loop would have to be run $m$ times, hence, $M$ runs in polynomial time.

3. Show that if P = NP, then NP $\subset$ EXP, where $\subset$ denotes the proper subset relation.

**Solution:** By definition, EXP includes all problems that can be solved by a deterministic TM in exponential time; $2^{p(n)}$, where $p(n)$ is a polynomial in the input size $n$.

If P = NP, then every problem in NP can be solved in polynomial time by a deterministic TM, which is much faster than EXP time. Since polytime algorithms are a subset of exponential-time algorithms, we have NP $\subset$ P $\subset$ EXP. Thus, if P = NP, then all problems in NP can be solved in exponential time, and hence, NP $\subset$ EXP.

4. ExactOne3SAT is NP-Complete

**Solution:** ExactOne3SAT: A variation of 3SAT where each clause has exactly three literals, and the goal is to determine if there exists a truth assignment such that exactly one literal in each clause is true.

ExactOne3SAT is in NP, since we can verify in polynomial time that a given truth assignment satisfies the conditions of the problem.

To show that ExactOne3SAT is NP-Hard, we reduce 3SAT to ExactOne3SAT. We show that 3SAT $\leq_p$ ExactOne3SAT.

Given an boolean formaula $\phi$ of 3SAT, let's assume it consists of clauses $C_1, C_2, ..., C_m$, where each clause has exactly three literals. We construct a new boolean formula $\phi'$ of ExactOne3SAT in such a way that $\phi'$ has a satisfying assignment iff the original formula $\phi$ has a satisfying assignment.

Clause Transformation: For each clause $C_i = (l_1 \vee l_2 \vee l_3)$ in $\phi$, we introduce two new clauses for the ExactOne3SAT formula $\phi'$. [Here $l_i$ is a literal, and $\vee$ is the logical OR operator.]

We introduce new auxillary variables $y_i$ for each clause $C_i$. The idea is to ensure that exactly one of the original literals in the clause is true, thus we do this by adding auxillary variables as so:

- For each clause $C_i = (l_1 \vee l_2 \vee l_3)$, we construct the following ExactOne3SAT clauses:
  1. $(l_1 \vee l_2 \vee y_i)$
  2. $\neg y_i \vee l_3 \vee z_i$

  The auxillary variable $y_j$ ensures that exactly one of the literals in the original clause is true, if neither or both are true, $y_i$ controls which of them is allowed to be true. The second clause ensures that the third literal $l_3$ behaves similarly using the auxillary variable $z_i$.

If a clause in 3SAT has at least one true literal, the transformation ensures that exactly one literal will be true in each new clause of the ExactOne3SAT instance.

Conversely, if there is a solution to the ExactOne3SAT instance, it corresponds to a valid solution to the original 3SAT instance.

The transformation can be done in polynomial time for each clause in the 3SAT instance, as we only add a small number of new variables and clauses. Thus, we have shown that 3SAT $\leq_p$ ExactOne3SAT, and since 3SAT is NP-Complete, ExactOne3SAT is NP-Complete.

5. Dominating Set is NP-Complete

**Solution:** Dominating Set is a subset $D$ of vertices $V$ such that every vertex in $V$ is either in $D$ or adjacent to at least one vertex in $D$. Thus, we need to find if there exists a Dominating Set of size $k$.

We can easily build a verifier $V$ that verifies a Dominating Set in polynomial time. Given a certificate $c$ which is a subset of $k$ vertices, we can verify in polynomial time that the subset forms a Dominating Set. This would take $\mathcal{O}(n^2)$ time, since we would have to check each vertex and its neighbors.

We reduce VC to DS to show that DS is NP-Hard.

Given an instance of the Vertex Cover problem $(G, k)$, we construct an instance of the Dominating Set problem $(G', k)$ as follows:

- Take the original graph $G = (V, E)$
- Construct a new graph $G'$ by adding a new vertex $v_e$ for each edge $e = (u, v) \in E$ in $G$.
- In $G'$, connect each $v_e$ to the two endpoints $u$ and $v$ of the edge $e$ from $G$.

In the new graph $G'$ each new vertex $v_e$, needs to be dominated by either $u$ or $v$, where $u$ and $v$ are the endpoints of the edge $e = (u, v)$ in the original graph. A vertex cover in the original graph $G$ of size $k$ selects $k$ vertices such that every edge is covered. Similarly, in the graph $G'$, a dominating set of size k must include a set of vertices that dominate all the vertices $v_e$. This happens exactly when the dominating set corresponds to a vertex cover in the original graph G, because the vertices in the vertex cover can dominate all the new vertices $v_e$.

Thus, $G$ has a vertex cover of size $k$ iff $G'$ has a dominating set of size $k$. Since Vertex Cover is NP-Complete, and Dominating Set is in NP, Dominating Set is NP-Complete.

6. 3 Dimensional Matching

**Solution:**

## 3 Dimensional Matching (3DM)

**Definition 3.** ***3DM:*** *Given disjoint sets $X$, $Y$, and $Z$, each of $n$ elements and triples $T \subseteq X \times Y \times Z$ is there a subset $S \subseteq T$ such that each element $\in X \cup Y \cup Z$ is in exactly one $s \in S$?*

3DM is NP. Given a certificate, which lists a candidate list of triples, a verifier can check that each triple belongs to $T$ and every element of $X \cup Y \cup Z$ is in one triple.

3DM is also NP-complete, via a reduction from 3SAT. We build gadgets for the variables and clauses.

The variable gadget for variable $x_i$ is displayed in the picture. Only red or blue triangles can be chosen, where the red triangles correspond to the true literal, while the blue triangles correspond to the false literal. Otherwise, there will be overlap, or some inner elements will not be covered. There is an $2n_{x_i}$ "wheel" for each variable gadget, where $n_{x_i}$ corresponds to the number of occurrences of $x_i$ in the formula.

The clause gadget for the clause $x_i \wedge \bar{x_j} \wedge x_k$ is displayed in the picture. The dot that is unshared in each variable's triangle within the clause gadget is also the single dot within the variable gadget.

Then, if we set $x_i$ to true, we take all of the red false triangles in the variable gadget, leaving a blue true triangle available to cover the clause gadget. However, this still potentially leaves $\bar{x_j}$ and $x_k$ uncovered, so we need a garbage collection





gadget, pictured below. There are $\sum_x n_x$ clauses of these gadgets, because there are $n_x$ unnecessary elements of the variable gadget that won't be covered. However, of the remaining elements, one of these per clause will be covered, so the remaining need to be covered by the garbage collection clause.

Thus, if a solution exists to the 3SAT formula, we can find the solution to the 3DM problem by choosing the points in 3DM corresponding to the variable values. If we have a 3DM solution, we can map this back to the satisfying assignment for 3SAT. Thus, our reduction is complete and 3DM is NP-hard.

7. SUBSET SUM is NP-Complete

**Solution:** SUBSET SUM = $\{(S, t) \mid S$ is a multiset of positive integers, and some subset of $S$ sums to $t\}$

**PROOF** The following is a verifier $V$ for *SUBSET-SUM*.

$V$ = "On input $\langle\langle S, t\rangle, c\rangle$:

1. Test whether $c$ is a collection of numbers that sum to $t$.
2. Test whether $S$ contains all the numbers in $c$.
3. If both pass, *accept*; otherwise, *reject*."

**ALTERNATIVE PROOF** We can also prove this theorem by giving a nondeterministic polynomial time Turing machine for *SUBSET-SUM* as follows.

$N$ = "On input $\langle S, t\rangle$:

1. Nondeterministically select a subset $c$ of the numbers in $S$.
2. Test whether $c$ is a collection of numbers that sum to $t$.
3. If the test passes, *accept*; otherwise, *reject*."

We reduce 3SAT to SUBSET SUM to show that SUBSET SUM is NP-Hard.

Using a 3CNF formula $\phi$, we construct an instance of SUBSET SUM which contains a subcollection whose summation target is $t$.

Then let $\phi$ be a boolean formula in 3CNF with $n$ variables $x_1, x_2, ..., x_n$ and $m$ clauses $C_1, C_2, ..., C_m$. We reduce $\phi$ to SUBSET SUM as follows:

1. For each variable $x_i$, we construct two integers $y_i$ and $z_i$ which represent the truth values of $x_i$ in the subcollection ($y$ can be true and $z$ can be false)
2. We construct a table with $k + 1$ columns, and $2(k + 1)$ rows.
3. For every $y, z$ in the $i - th$ row, set the $i - th$ column value as 1 and the rest be 0.
4. For every $y, z$ in the $j - th$ column, set y = 1 for the jth column if x is true, else set z as true. All else to be set as 0.
5. For every $g, h$ in the (j+1)-th row, set the jth column as 1 and set everything to 0
6. Set the first 1 columns of final row as 1 and the remaining k columns as 3.

The table looks as so:

| | 1 | 2 | 3 | 4 | $\cdots$ | $l$ | $c_1$ | $c_2$ | $\cdots$ | $c_k$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $y_1$ | 1 | 0 | 0 | 0 | $\cdots$ | 0 | 1 | 0 | $\cdots$ | 0 |
| $z_1$ | 1 | 0 | 0 | 0 | $\cdots$ | 0 | 0 | 0 | $\cdots$ | 0 |
| $y_2$ | | 1 | 0 | 0 | $\cdots$ | 0 | 0 | 1 | $\cdots$ | 0 |
| $z_2$ | | 1 | 0 | 0 | $\cdots$ | 0 | 1 | 0 | $\cdots$ | 0 |
| $y_3$ | | | 1 | 0 | $\cdots$ | 0 | 1 | 1 | $\cdots$ | 0 |
| $z_3$ | | | 1 | 0 | $\cdots$ | 0 | 0 | 0 | $\cdots$ | 1 |
| $\vdots$ | | | | | $\ddots$ | $\vdots$ | $\vdots$ | | $\vdots$ | $\vdots$ |
| $y_l$ | | | | | | 1 | 0 | 0 | $\cdots$ | 0 |
| $z_l$ | | | | | | 1 | 0 | 0 | $\cdots$ | 0 |
| $g_1$ | | | | | | | 1 | 0 | $\cdots$ | 0 |
| $h_1$ | | | | | | | 1 | 0 | $\cdots$ | 0 |
| $g_2$ | | | | | | | | 1 | $\cdots$ | 0 |
| $h_2$ | | | | | | | | 1 | $\cdots$ | 0 |
| $\vdots$ | | | | | | | | | $\ddots$ | $\vdots$ |
| $g_k$ | | | | | | | | | | 1 |
| $h_k$ | | | | | | | | | | 1 |
| $t$ | 1 | 1 | 1 | 1 | $\cdots$ | 1 | 3 | 3 | $\cdots$ | 3 |

Now considering a sample boolean expression, take $y_i$ if $x_i$ is true, else take $z_i$. If the number of literals in $c_j$ is at most 2, take $g_i$, else if the number is 1 take $h_j$.

The reduction takes polynomial time, and the boolean expression has a satisfying assignment iff the SUBSET SUM instance has a subcollection that sums to $t$. Thus, SUBSET SUM is NP-Complete.

**Solution:** A better solution is as follows:

We sketch the proof. SubSum is in **NP**, since there is an obvious polynomial time computable verifier for the problem. The witness is a subset $S$, and the verifier simply checks that $\sum_{i \in S} a_i = t$, which can be done in polynomial time.

To show that SubSum is **NP**-hard, we shall show that

$$\text{3SAT} \leq_P \text{SubSum}.$$

We describe the polynomial time reduction next. Given a 3-sat formula $\phi$, our algorithm needs to output numbers $a_1, \ldots, a_k$ and $t$ such that $\text{SubSum}(a_1, \ldots, a_k, t) = 1$ if and only if $\phi$ is satisfiable.

Suppose $\phi$ has $n$ variables and $m$ clauses. Then, we will have $k = 2n + 2m$, and all of the numbers $a_1, \ldots, a_k$ and $t$ will be $n + m$ digit numbers, written in base 10. Moreover, all the digits of $a_1, \ldots, a_k$ will be either 0 or 1, and the numbers will be chosen in such a way that adding any subset of $a_1, \ldots, a_k$ will never produce a carry.

For each variable $x_i$ of the formula $\phi$, we shall have two numbers: $t_i$ and $f_i$. The $i$'th digit of $t_i$ and $f_i$ will be set to 1 and all of the remaining $n - 1$ digits in the first $n$ digits will be set to 0. Meanwhile, in the target number $t$, all of the first $n$ digits will be set to 1. This choice ensures that choosing any subset of $t_1, f_1, \ldots, t_n, f_n$ that sums to $t$ corresponds to choosing either $t_i$ or $f_i$ to be included in the set, for each $i$. In other words, a subset of these numbers that sums to $t$ corresponds to a truth assignment to the variables $x_1, \ldots, x_n$.

Next, we need to add more digits to ensure that this truth assignment satisfies all the clauses. For every $i, j$, if $x_i$ occurs in the $j$'th clause, we make the $n + j$'th digit of $t_i$ 1. If $\neg x_i$ occurs in the $j$'th clause, we make the $n + j$'th digit of $f_i$ 1. All other digits (upto the $n + m$'th digit) of $t_i, f_i$ are set to 0. This choice ensures that if the subset chosen satisfies the $j$'th clause, then the $j$'th digit of the sum will be either $1, 2$ or 3. Finally, we add two numbers $b_j, c_j$, which are 0 in all digits, except for the $j$'th digit. The $j$'th digit of both numbers is 1. This ensures that if the $j$'th clause is satisfied by the assignment, then one can pick $0, 1$ or 2 elements of $\{b_j, c_j\}$ to add to the subset, so that the sum of the $j$'th digits is 3.