Problem 1

Consider the following language:

\[ \text{DOUBLE-SAT} = \{ \langle \phi \rangle \mid \phi \text{ is a boolean formula that has at least 2 different satisfying assignments} \} \]

Prove that DOUBLE-SAT is NP-Complete.

Solution:

First, let’s look at a simple boolean formula that is in DOUBLE-SAT:

\[ \phi_1 = x \lor y \]

If we set \( x = \text{True} \) and \( y = \text{False} \), \( \phi_1 \) will evaluate to True.

If we set \( x = \text{True} \) and \( y = \text{True} \), \( \phi_1 \) will evaluate to True.

Hence \( \phi_1 \) has at least 2 different satisfying assignments and thus \( \phi_1 \in \text{DOUBLE-SAT} \).

*Note it has a third satisfying assignment as well, but at this point we don’t care.

Onto the proof:

**Part 1:** DOUBLE-SAT is in NP aka a candidate assignment can be verified in polynomial time

The verifier for DOUBLE-SAT is as follows.

On input \( \langle \phi, c_1, c_2 \rangle \):

1. Check that \( \phi \) is a valid boolean formula and that \( c_1 \) and \( c_2 \) are both variable-value pairs.
2. Check that \( c_1 \) and \( c_2 \) each contain all of the variables in \( \phi \) and that there is only one assigned value for each variable.
3. Check that the assignments specified by \( c_1 \) and \( c_2 \) are different.
4. Check that \( \phi \) evaluates to True when the values of \( c_1 \) are plugged in.
5. Check that \( \phi \) evaluates to True when the values of \( c_2 \) are plugged in.
6. If checks 1 - 5 pass, ACCEPT; otherwise, REJECT.

Let \( n_1 \) be the number of variables in \( \phi \) and \( n_2 \) be the length of \( \phi \). Step 1 takes \( O(n_2 + 2n_1) \approx O(n) \) time. Step 2 takes \( O(n_1 * n_2) \approx O(n^2) \) time to check every variable in \( \phi \) against the list of values. Step 3 takes \( O(n_1^2) \) time to compare the candidate solutions. Steps 4 and 5 each take \( O((n_1 * n_2) + n_2) \approx O(n^2) \) time plug in the candidate solutions and evaluate \( \phi \). Step 6 takes constant time. Overall, this algorithm runs in polynomial time.

Thus, DOUBLE-SAT is in NP.
**Part 1 (alt):** DOUBLE-SAT is in NP aka a boolean formula can be decided in nondetermin-istic polynomial time

The decider for DOUBLE-SAT is as follows.

On input $\langle \phi \rangle$:

1. Check that $\phi$ is a valid boolean formula.
2. Nondeterministically generate a candidate solution to $\phi$, $c_1$.
3. Nondeterministically generate a second candidate solution, $c_2$, that is different from $c_1$.
4. Check that $\phi$ evaluates to True when the values of $c_1$ are plugged in.
5. Check that $\phi$ evaluates to True when the values of $c_2$ are plugged in.
6. If checks 1 - 5 pass, ACCEPT; otherwise, REJECT.

Let $n_1$ be the number of variables in $\phi$ and $n_2$ be the length of $\phi$. Step 1 takes $O(n_2)$ time. Steps 2 and 3 each take $O(n_1)$ to generate candidate solutions. Steps 4 and 5 each take $O((n_1 * n_2) + n_2) \approx O(n^2)$ time plug in the candidate solutions and evaluate $\phi$. Step 6 takes constant time. Overall, this algorithm runs in polynomial time.

Thus, DOUBLE-SAT is in NP.
Part 2: DOUBLE-SAT is NP-HARD via reduction from SAT.

We’ll define the function \( f(\phi) = \psi \) such that:

\[
\psi = \phi \land (x \lor \overline{x}) \quad \text{where } x \text{ is a variable that does not appear in } \phi.
\]

If \( n \) is the length of \( \phi \), it will take \( O(n) \) time to find a variable name that is not in \( \phi \) and constant time to append \( ñ(x \lor \overline{x}) ñ \) to \( \phi \). Thus this reduction can be computed in polynomial time.

If \( \phi \in SAT \), then \( \psi \in DOUBLE-SAT \). If \( \phi \in SAT \), then we know that the left side of \( \psi \) is satisfiable. We can then set our new variable \( x \) to True, which will satisfy \( \psi \). Alternatively, we can then set our new variable \( x \) to False, which will satisfy \( \psi \). Thus there are at least 2 different satisfying assignments and \( \psi \in DOUBLE-SAT \).

If \( \phi \notin SAT \), then \( \psi \notin DOUBLE-SAT \). If \( \phi \notin SAT \), then there is no way to satisfy the left side of \( \psi \). Because of the \( \land \) operator, this leaves us with no way to satisfy \( \psi \) overall. If \( \psi \) cannot be satisfied then it certainly cannot have 2 satisfying assignments and \( \psi \notin DOUBLE-SAT \).

Thus, DOUBLE-SAT is in NP-HARD. Since it is in NP as well, DOUBLE-SAT is NP-Complete.

Part 2 (alt): DOUBLE-SAT is NP-HARD via reduction from SAT.

We’ll define the function \( f(\phi) = \psi \) such that:

\[
\psi = \phi \land (x \lor y) \quad \text{where } x \text{ and } y \text{ are variables that do not appear in } \phi.
\]

If \( n \) is the length of \( \phi \), it will take \( O(n) \) time to find two variable names that are not in \( \phi \) and constant time to append \( ñ(x \lor y) ñ \) to \( \phi \). Thus this reduction can be computed in polynomial time.

If \( \phi \in SAT \), then \( \psi \in DOUBLE-SAT \). If \( \phi \in SAT \), then we know that the left side of \( \psi \) is satisfiable. We can then set our new variables \( x \) to True and \( y \) to False, which will satisfy \( \psi \). Alternatively, we can then set our new variable \( x \) to False and \( y \) to True, which will satisfy \( \psi \). Thus there are at least 2 different satisfying assignments and \( \psi \in DOUBLE-SAT \).

If \( \phi \notin SAT \), then \( \psi \notin DOUBLE-SAT \). If \( \phi \notin SAT \), then there is no way to satisfy the left side of \( \psi \). Because of the \( \land \) operator, this leaves us with no way to satisfy \( \psi \) overall. If \( \psi \) cannot be satisfied then it certainly cannot have 2 satisfying assignments and \( \psi \notin DOUBLE-SAT \).

Thus, DOUBLE-SAT is in NP-HARD. Since it is in NP as well, DOUBLE-SAT is NP-Complete.
Part 2 (fail): A common mistake on this problem was to define the following reduction from SAT or 3SAT:

\[ f(\phi) = \phi \lor x \] where \( x \) is a variable that does not appear in \( \phi \).

It is easy to show, as above, that this reduction can be accomplished in polynomial time.

If \( \phi \in \text{SAT} \), then \( \psi \in \text{DOUBLE-SAT} \). This case checks out fine. If \( \phi \in \text{SAT} \), then we can set our new variable \( x \) to True or False, both of which will satisfy \( \psi \). Thus there are at least 2 different satisfying assignments and \( \psi \in \text{DOUBLE-SAT} \).

If \( \phi \notin \text{SAT} \), then \( \psi \notin \text{DOUBLE-SAT} \). This case does NOT check out. If \( \phi \notin \text{SAT} \), then we can still satisfy \( \psi \) by setting \( x \) to True. However, there is almost certainly another way to assign the variables of \( \phi \). So long as \( x = \text{True} \), \( \psi \) will be satisfied. Thus \( \psi \) could potentially still be in \( \text{DOUBLE-SAT} \), even though \( \phi \notin \text{SAT} \).

FAIL.
Part 2 (alt): DOUBLE-SAT is NP-HARD via reduction from SAT.

We'll define the function \( f(\phi) = \psi \) such that, on input \( \phi \):

1. Copy \( \phi \) to a new formula \( \phi' \).
2. Change every variable name in \( \phi' \) to a name that does not appear in \( \phi \).
3. Output \( \phi \lor \phi' \)

If \( n \) is the length of \( \phi \), it will take \( O(n) \) time to copy the formula, and \( O(n^2) \) time to change the variable names. Finally, it will take another \( O(n) \) to output the final formula. Thus this reduction can be computed in polynomial time.

If \( \phi \in \text{SAT} \), then \( \phi \lor \phi' \in \text{DOUBLE-SAT} \). If \( \phi \in \text{SAT} \), then \( \phi \lor \phi' \) can be satisfied with a satisfying assignment to either the right or the left side of the formula. Thus there are at least 2 different satisfying assignments and \( \phi \lor \phi' \in \text{DOUBLE-SAT} \).

If \( \phi \lor \phi' \in \text{DOUBLE-SAT} \), then \( \phi \in \text{SAT} \). If \( \phi \lor \phi' \in \text{DOUBLE-SAT} \), then at least one side of the formula had to have been satisfiable. Since both sides are copies of \( \phi \) then it must have been the case that \( \phi \in \text{SAT} \).

Thus, DOUBLE-SAT is in NP-HARD. Since it is in NP as well, DOUBLE-SAT is NP-Complete.

Part 2 (alt): DOUBLE-SAT is NP-HARD via reduction from SAT.

We'll define the function \( f(\phi) = \psi \) such that, on input \( \phi \):

1. Copy \( \phi \) to a new formula \( \overline{\phi} \).
2. Negate every variable in \( \overline{\phi} \) (i.e. \( x \rightarrow \overline{x} \) and \( \overline{x} \rightarrow x \)).
3. Output \( \phi \lor \overline{\phi} \)

If \( n \) is the length of \( \phi \), it will take \( O(n) \) time to copy the formula, and \( O(n) \) time to negate the variable names. Finally, it will take another \( O(n) \) to output the final formula. Thus this reduction can be computed in polynomial time.

If \( \phi \in \text{SAT} \), then \( \phi \lor \overline{\phi} \in \text{DOUBLE-SAT} \). If \( \phi \in \text{SAT} \), then \( \phi \lor \overline{\phi} \) must also be satisfiable by negating the satisfying assignment to \( \phi \). Thus, \( \phi \lor \overline{\phi} \) can be satisfied with a satisfying assignment to either the right or the left side of the formula. Thus there are at least 2 different satisfying assignments and \( \phi \lor \overline{\phi} \in \text{DOUBLE-SAT} \).

If \( \phi \lor \overline{\phi} \in \text{DOUBLE-SAT} \), then \( \phi \in \text{SAT} \). If \( \phi \lor \overline{\phi} \in \text{DOUBLE-SAT} \), then at least one side of the formula had to have been satisfiable. Since both sides are essentially copies of \( \phi \) then it must have been the case that \( \phi \in \text{SAT} \).

Thus, DOUBLE-SAT is in NP-HARD. Since it is in NP as well, DOUBLE-SAT is NP-Complete.
**Part 2 (alt):** DOUBLE-SAT is NP-HARD via reduction from 3SAT.

We’ll define the function \( f(\phi) = \psi \) such that, on input \( \phi \):

1. Select one 3SAT clause of the form \( (x \lor y \lor z) \).
2. Replace this clause with two new clauses \( (x \lor y \lor z \lor a) \land (x \lor y \lor z \lor \overline{a}) \).
3. Output this new formula as \( \psi \)

It will take constant time to make this insertion and \( O(n) \) to output the final formula. Thus this reduction can be computed in polynomial time.

**If** \( \phi \in \text{3SAT} \), **then** \( \psi \in \text{DOUBLE-SAT} \). If \( \phi \in \text{3SAT} \), then every clause in \( \phi \) had at least one True variable. Thus both of the modified clauses in \( \psi \) will have at least one True variable, regardless of how \( a \) is assigned. \( \psi \) will thus be satisfiable with \( a \) assigned to either True or False, resulting in 2 different satisfying assignments to \( \psi \). Thus \( \psi \in \text{DOUBLE-SAT} \).

**If** \( \phi \notin \text{3SAT} \), **then** \( \psi \notin \text{DOUBLE-SAT} \). If \( \phi \notin \text{3SAT} \), then at least one clause could not be satisfied. The modified clauses in \( \psi \) will not be able to rectify this situation, since \( a \) can only contribute a True value to one of those clause. Thus \( \psi \) is not satisfiable and \( \psi \notin \text{DOUBLE-SAT} \).

Thus, DOUBLE-SAT is in NP-HARD. Since it is in NP as well, DOUBLE-SAT is NP-Complete.
Part 2: DOUBLE-SAT is NP-Hard via reduction from 3SAT - aka Nobody’s Perfect

In the “Nobody’s Perfect” version of 3SAT, our reduction takes a 3SAT problem and produces another 3SAT problem in which every clause not only needs one True variable to be satisfied, but must also have at least one False variable. After all, nobody’s perfect. GET IT?!?!

We’ll define the function $f(\phi) = \psi$ such that each clause of the form $(x_i \lor y_i \lor z_i)$ in $\phi$ becomes the clause pair $(x_i \lor y_i \lor a_i) \land (\overline{a_i} \lor z_i \lor x_F)$ in $\psi$ (a new $a_i$ will be needed for each new clause pair whereas $x_F$ is reused in every clause).

Since $f$ requires a single scan through $\phi$, we can construct $\psi$ in polynomial time.

If $\phi \in 3SAT$, then $\psi \in DOUBLE-SAT$. If $\phi \in 3SAT$, then at least one of $x_i, y_i, z_i$ must be true. If either $x_i$ or $y_i$ is true, we can set $a_i$ to false (guaranteeing that the first clause has both a true and a false) and set $x_F$ to false, which ensures the same for the second clause since $\overline{a_i}$ is true. If $x_i$ and $y_i$ are both false, then $z_i$ must be true. In this case we can set $\overline{a_i}$ to false, which will make $a_i$ true. Again, both clauses will have a true and a false value in them, and satisfies the Nobody’s Perfect condition. Knowing that this condition is satisfied, we also know that negating all of the variable assignments will also satisfy $\psi$ (each clause will still have at least on True variable and at least one False variable). Thus there are at least 2 different satisfying assignments and $\psi \in DOUBLE-SAT$.

If $\psi \in DOUBLE-SAT$, then $\phi \in 3SAT$. If $x_F$ is false, then we are guaranteed that $x_i, y_i, z_i$ must be true because either $\overline{a_i}$ or $z_i$ must then be true. But what if $x_F$ is true? Based on the definition of NPSAT, if some assignment satisfies NPSAT, then it’s inverse will too. That is, if an assignment satisfies an NPSAT equation, you can flip the truth value of every variable and arrive at an assignment that still satisfies the equation. So if you have a valid assignment for NPSAT, but $x_F$ is true, you can just flip all the variable assignments, which will put us in the previous condition where $x_F = false$ forces either $x_i, y_i, z_i$ to be true. Thus, $\phi$ is satisfiable.

Since NPSAT is in NP and NP-hard, we know it is NP-complete.