Polynomial transformations and reductions

Let \( X_1 = (D_1, F_1) \) and \( X_2 = (D_2, F_2) \) be two feasibility problems.

Assume \( \exists \) function \( g : D_1 \rightarrow D_2 \) such that \( \forall d \in D_1, g(d) \in D_2 \).

If \( g \) is computable in time that is polynomial in the length of the encoding of \( d \), then \( X_1 \) is polynomially reducible to \( X_2 \).

**Proposition**

If \( X_1 \) is polynomially reducible to \( X_2 \), and \( X_2 \in \text{P} \), then \( X_1 \in \text{P} \).

**Proof**

Let \( \text{Alg} \) be for \( X_1 \).

1. Compute \( g \).
2. Apply \( \text{Alg} \) for \( X_2 \). //

**E.g.**

- \( X_1 \) is perfect matching feasibility problem on a bipartite graph, with end node of bipartite having same number of sides.
- \( X_2 \) is max flow problem and has been solved.

**Reduction**

Let \( X = (D, F) \) be a feasibility problem.

1. \( X_1 \) is given as example (i),
2. \( X_2 \) is perfect matching feasibility problem.

Then \( g \) is identity map.
X \times X_1$ feasibilty problem

$X_i$ is polynomially reducible to $X_2$ if there exists an algorithm for $X_2$ which is subroutines, and the edge $A_i$ runs in poly time under the assumption that edge $A_i$ takes each call of the subroutine $x_i$ once.

Ex) Any transformation of subroutine is only once, on the transformed data $g(d_i)$.

Pro: If $X_i$ is polynomially reducible to $X_2$ and $X_2 \in P$ then $X_i \in P$.

Short complexity of edge is when $X_i$ is at most

$p(i) \cdot p_2(p(i))$,

where $p(i)$ is complexity of $A_i$ with assumption that each call of subroutine takes, and the $p_2(i)$ is the required by $p_2$.

We make the at most $p(i)$ times, each call of $A_i$ has input of length at most $p(i)$ since that we have to write the output clean and clear in an upper bound $i$ how much we can write $i$. 
**Definition:** A feasibility problem \( X \in \text{NP} \) is said to be \( \text{NP-}\text{complete} \) if all other problems in \( \text{NP} \) polynomially reduce to \( X \).

(Not direction)

**Theorem:** If \( X \) is \( \text{NP-}\text{complete} \) and \( X \in \text{P} \) then \( \text{P}=\text{NP} \).

**Proof:** Any problem in \( \text{NP} \) is polynomially reducible to \( X \), and \( X \in \text{P} \), so \( \text{NP}\subseteq \text{P} \).
The problem SATISFIABILITY (SAT).

A Boolean variable \( x \) is a variable that can assume only the values true, false. Boolean variables can be combined using \( \lor \) (denoted by \( + \)), and \( \land \) (denoted by \( \cdot \)) and \( \neg \) (denoted by \( \overline{\cdot} \)) to form Boolean formulas:

\[
\overline{x}_3 \cdot (x_1 + \overline{x}_0 + x_3)
\]

For this to be true, \( \overline{x}_3 \) must be true and \((x_1 + \overline{x}_0 + x_3)\) must be true.

If \( x_1 = \text{true} \), \( x_0 = \text{true} \), \( x_3 = \text{false} \):

- expression has value true

- \( x_1 = \text{false} \), \( x_0 = \text{false} \), \( x_3 = \text{true} \):

- expression has value false.

This formula is satisfiable since 3 assignments of variables and the expression is true.

Consider \( x_1 \cdot \overline{x}_1 \). This is not satisfiable.

Clauses are subformulas of the expression containing only \( \lor \) and \( \neg \):

- \( \overline{x}_3 \) is a clause, \( x_1 + \overline{x}_0 + x_3 \) is a clause.

Satisfiability problem:

Given \( n \) clauses \( C_1, \ldots, C_m \) involving \( n \) variables \( x_1, \ldots, x_n \), is the formula \( C_1 \lor C_2 \lor \cdots \lor C_m \) satisfiable?
Theorem (Cook)

Schrödinger is NP-complete.

Idea of proof: Build a Turing machine that solves all NP problems in NP.
Then, poly-time reduce Turing machine to SAT.

Theorem. If $X_1 \in \text{NP-complete}$ and $X_2$ is poly-reducible to $X_1 \in \text{NP}$, then $X_2 \in \text{NP-complete}$.

Fact. Observe, since poly-time reducibility is a transitive property.

Theorem. The 0-1 integer programming feasibility problem is NP-complete.

Let we reduce SAT to 0-1 IP form. Show how exact value of SAT

is equivalent to a 0-1 IP.
Let $3$-$SAT$ be the problem where each clause is a satisfiability problem with each clause containing exactly 3 literals.

Thus $3$-$SAT$ is $NP$-complete (Leave part ii as exercise).

Proof. $3$-$SAT$ is in $NP$ since it is a special case of SAT.

To show $3$-$SAT$ is $NP$-complete, we reduce SAT to $3$-$SAT$.

Consider a clause $C_i = \overline{\lambda}_1 + \lambda_2 + \ldots + \lambda_k$, and let $\lambda_k$ be a literal $x_i$ or $\overline{x}_i$ for some $i$.

Assume $k \geq 3$.

Consider the clauses $C_i$ and $\overline{C}_i$.

\begin{align*}
\lambda_1 + \lambda_2 + x_1 \\
\overline{x}_1 + \lambda_2 + x_3 \\
\vdots \\
x_k + \lambda_{k-1} + \lambda_k
\end{align*}

\begin{align*}
\overline{x}_1 + \lambda_2 + x_3 \\
\vdots \\
x_k + \lambda_{k-1} + \lambda_k
\end{align*}

Consider the clauses $C_i$ and $\overline{C}_i$.

\begin{align*}
\lambda_1 + \lambda_2 + x_1 & \quad \text{if } \lambda_k = x_i, \text{ and let } x_i \text{ be true for some } i. \\
\overline{x}_1 + \lambda_2 + x_3 & \quad \text{if } \lambda_k = \overline{x}_i, \text{ and let } \overline{x}_i \text{ be true for some } i.
\end{align*}

If $k = 3$, use old clause in new formulation.

If $k > 3$, replace $C_i$ by $\lambda_1 + \lambda_2 + \lambda_3$.

If $k = 2$, replace $C_i$ by $\lambda_1 + \lambda_2 + \lambda_3$.

If $k = 1$, replace $C_i$ by $\lambda_1 + \lambda_2 + \lambda_3$.

For each literal, we need to have $y$ and $z$ so be false, so add clauses:

\begin{align*}
\overline{x} + \lambda_1 + \lambda_2 \\
\overline{x} + \lambda_2 + \lambda_3 \\
x_1 + \overline{x} + \lambda_2 \\
\overline{x} + \overline{x} + \lambda_3
\end{align*}
Reducing 3-SAT to Hamiltonian Circuit

\[ F = (x_1 + \overline{x}_3 + x_3)(\overline{x}_1 + x_2 + \overline{x}_3)(\overline{x}_1 + \overline{x}_2 + x_3) \]

The Hamilton circuit shown corresponds to:
- \( t(x_1) = \text{true} \)
- \( t(x_2) = \text{false} \)
- \( t(x_3) = \text{false} \)

Use subgraph \( B \) for each clause.

Connect a literal in a clause to the literal on the variable side, using the A subgraph.
Hamiltonian circuit

Given a graph $G$, is there a circuit visiting each node exactly once?

The Hamiltonian circuit is $NP$-complete.

Proof: Clearly in $NP$.

To show problem is $NP$-complete:

Transform $3$-SAT to Hamiltonian circuit:

So given an instance of $3$-SAT, we can create a graph such that there is a Hamiltonian tour in the graph iff the instance of $3$-SAT is feasible.

Consider graph:

\[
\begin{array}{cccc}
\text{u} & 2_1 & 2_2 & 2_3 & 2_4 \\
A: & 2_1 & 2_2 & 2_3 & 2_4 \\
\text{v} & \text{u}' & \text{v}', \text{u}' & \text{v}' & \text{u}'
\end{array}
\]

Any tour of $A$ has to traverse either node $2_i$ or not.

or a self loop $\text{u}'$ or $\text{u}'$ but not both.

Write graph as:

\[
\begin{array}{c}
\text{u} \\
A - \text{onetower}
\end{array}
\]

\[
\begin{array}{c}
\text{v} \\
\text{u}'
\end{array}
\]
Consider graph $G$:

![Graph Diagram]

Outside Edges only connect at $u_i$ and $u_j$.

Note that any Hamiltonian circuit cannot reverse all these edges $(u_i, u_j), (u_j, u_k), (u_k, u_i)$, but it can reverse any combination of them.

With $x_i$:

![Graph Diagram]

So, think of reverse edge $u_j u_i$, regard linked $x_j$ as not holding. Since $x_j$'s reverse on edge, we have clause holding.

Have $n$ varying clauses, is put a copy of $T_i$ series

Class $n$:

![Graph Diagram]

Connect tops and bottom

$Ax$: A component in clause containing $x$, always take $x$, row (or line) per each clause.

Connect $u_j$ to edge in clause in a left part of $v_i$, if $x_j$ is $X_i$, to right part it is $X_i$.

Clauses:

![Graph Diagram]

Form of edge $x_j$:

- (false) - (true)
\[ C_1 = x_1 + x_2 + x_3 \quad C_2 = x_3 + x_4 + x_5 \]

Clauses

Variables
The TSP is NP-complete.

To show NP-complete.

Reduce Hamiltonian circuit to TSP.

Let G = (V, E) be a complete graph on |V| vertices.

If (i, j) ∈ E then d_{ij} = 1

(., .) ∉ E then d_{ij} = 2.

Find \textbf{\sum}_{i=1}^{n} d_{ii} \leq |V|