Aalto University

Lecture Notes - Week VII

NP Problems and Polynomial Transformation

Fernando Dias, Philine Schiewe and Piyalee Pattanaki

February 19, 2024

CHAPTER 1

NP-Completeness

In the problem analyses over the last two lectures, it was hinted that a problem in the class NP can be expressed as another problem in the same class. In this lecture, we are formalizing this process:

$$\begin{tikzcd}
\boxed{(X_1, Y_1)} \arrow[r, "p_1"] \arrow[d, "\mathrm{Alg}_1"] & \boxed{(X_2, Y_2)} \arrow[d, "\mathrm{Alg}_2"] \\
\boxed{f(x_1)} & \boxed{f(x_2)} \arrow[l, "p_2"]
\end{tikzcd}$$

To sum up, (X_1, Y_1) **polynomially transforms** to $(X_2 Y_2)$ if there exists polynomial function $p_1 : X_1 \to X_2$ such that:

$$p_1(x_1) \in Y_2 \quad \text{for all } x_1 \in Y_1 \text{ and}$$
$$p_1(x_1) \in X_2 \setminus Y_2 \quad \text{for all } x_1 \in X_1 \setminus Y_1$$

Hence, yes-instances are mapped to yes-instances, no-instances are mapped to no-instances. Also, (X_1, Y_1) is at most as hard as (X_2, Y_2) and for general polynomial function p_2: **polynomial reduction**.

Using the definition of NP-Completeness allows us to claim that if $(X, Y) \in NP$, such problem is called NP-complete if all other problems in NP **polynomially** transform to (X, Y).

NP-complete problems are the "hardest" problems in NP. However, if one NP-complete problem is solvable in polynomial time, all are ($P = NP$). This leads to the final question:

*Do NP-complete problems **actually** exist?*

The basic problem for NP-Complete is the Satisfiability Problem (SAT), defined as:

literal: a binary variable, e.g. x, or its negation, e.g. $\neg x$

clause: a disjunction of literals, e.g.

$$x_1 \vee \neg x_2$$

CNF: conjunctive normal form, a conjunction of disjunction, e.g.

$$(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge \neg x_4$$

SAT: satisfiability problem: Can a boolean formula, given as CNF, be satisfied?

This problem was initially fully described and categorized as an NP by **Stephen Cook** in **1971**.

Proof 1 (SAT)

- *show that for any **nondeterministic algoritm** an equivalent SAT instance can be constructed in polynomial time*

- *Need "narrow" definition of algorithms;*

Since then, many (**thousand** of) problems have been shown to be NP-complete. **Karp's original 21 NP-complete problems** is a list of several NP-problems which was curated by Karp, R.M. (1975), in *On the complexity of combinatorial problems.* Networks 5 (1975), 45–68.

As an alternative to solve such problems, integer linear programming is used. However, it is important to note that **ILP is NP-complete**.

Proof 2 (ILP)

- *idea: $(X, Y) \rightsquigarrow$ SAT $\rightsquigarrow$ integer linear programming*
- *check given solution in **polynomial** time $\Rightarrow$ integer linear programming is in NP*
- *let F be a formula in CNF, **construct** ILP P*
  - *for each variable x_i of F **construct** a binary variable y_i for P*
  - *for each clause C **introduce one constraint** to P:*
    $\sum_{i:\, x_i \in C} y_i + \sum_{i:\, \neg x_i \in C} (1 - y_i) \geq 1$
- *P is feasible $\iff$ F is satisfiable*

Remark: A problem that can be formulated as an integer linear program is not automatically NP-complete.

Recalling the Knapsack problem, considered S as a multiset of positive integers. Can you partition S into S_1, S_2 such that:

$$\sum_{s \in S_1} s = \sum_{s \in S_2} s?$$

This version is known to be **NP-complete**.

The decision version of knapsack problem is where items I with weight w_i, value c_i, $i \in I$, maximum weight B, minimum value C and the goal is to find $I' \subset I$ with:

$$\sum_{i \in I} w_i \leq B$$
$$\sum_{i \in I} c_i \geq C?$$

How can you **show** that knapsack is NP-complete?

Another category is NP-Hardness. A problem $\mathcal{P}$ is called *NP-hard* if all problems in NP **polynomially** reduce to $\mathcal{P}$.

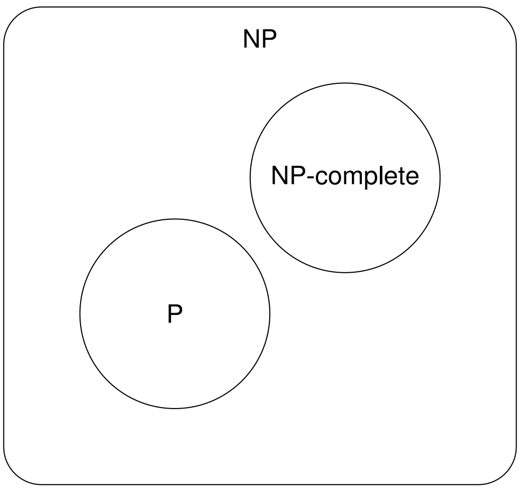
Remark: $\mathcal{P}$ not necessarily in NP.

Some examples are:

- (optimization version of) knapsack
- multi-commodity flows
- travelling salesperson problem
- uncapacitated facility location

Back again, the main question is: $P \neq NP$?

Showing whether $P = NP$ or $P \neq NP$ is one of the **Millennium Prize Problems** and many researchers have spent their lives working on this question. Most of them believe that $P \neq NP$. However, if $P = NP$, this would have a large influence on the **(cyber) security** of **cryptography**.



CHAPTER 2
NP-Complete Proof

Based on the discussion from the previous lecture, the "recipe" to categorize a **novel problem** as **NP-complete** is as follows:

- Show it is in NP:
  Verify that if a **candidate solution** is valid in polynomial time;
- Show it is NP-Hard:
  Reduce to a known NP-Complete problem $\longrightarrow$ **polynomial reduction**.

With these two steps, **a novel problem can be considered a NP-Complete problem**.

The most challenging task is finding an efficient way to reduce one problem to another. A few examples are displayed below.

2.1 CLIQUE AND INDEPENDENT SET

Knowing that both **clique** and **independent set** are NP-Complete, there is a simple transformation between them:

- For a graph $G = (V, E)$, build a complementary graph G';

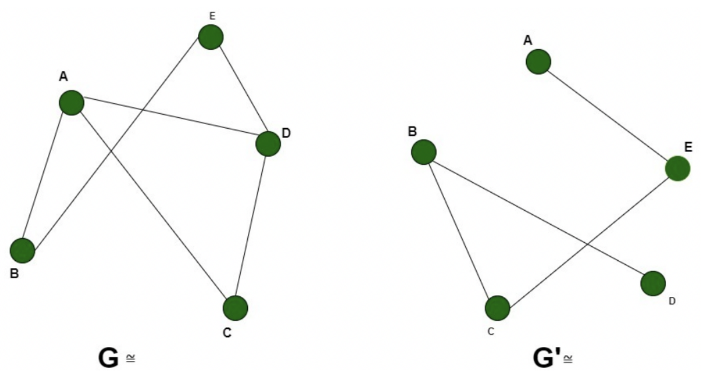


Figure 2.1: G and complimentary G'

- For every $v \in V$, it creates another set of nodes $v \in V'$;

- Add an edge in G' for every edge not in G.

Remark: Complimentary graph can be calculated in polynomial time.

If **there is** an independent set of size k in the complement graph G', **no two nodes share** an edge in G'. Hence, **all of those edges share an edge** in G forming a **clique of size** k.

If **there is** a clique of size k in the graph G, all nodes share an edge in G, implying that there is **no two nodes share** an edge in G'. Hence, **all of those edges share an edge** in G', forming an **independent set of size** k.

2.2 TSP AND HAMILTONIAN CYCLE

Another example involves TSP and Hamiltonian cycle, in which both problems are related to finding a **cycle**. A quick transformation between them is:

- For a graph $G = (V, E)$, build a complementary graph G';

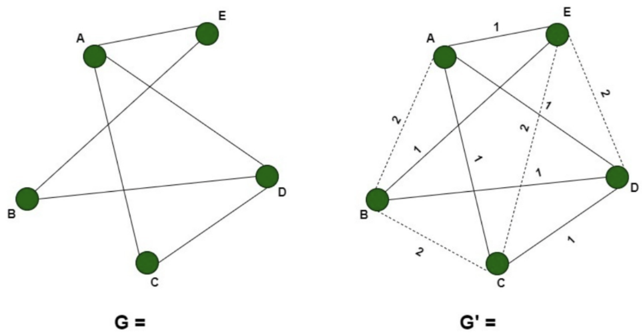


Figure 2.2: G and complimentary G'

- For every pair of nodes (u, v) without an edge in G, add an edge in G'.
- If edge (u, v) exists in G, set the weight to zero; otherwise, assign weight equal to one.

The graph G has a Hamiltonian cycle if **there is** a cycle in G' passing through **all nodes only once with combined weight equal to zero**.

If the cycle passes through all nodes and the combined weight is zero, it means that the cycle **only contains edges present** in G. Hence, a **Hamiltonian cycle exists** in G.

If there is a Hamiltonian cycle in G, it also forms a **cycle** in G' with a combined weight equal to zero. Hence, a **solution for TSP** exists in G'.

2.3 INDEPENDENT SET AND VERTEX COVER

In this example, both problems can be traced to **covering** problems. In addition, if a graph G has an **independent set** S, it also has a **vertex cover** $V - S$.

If S is an independent set, there is no edge $(u, v) \in G$, such that both v and u are in S. Therefore, either v or u **has to be in** $V - S$.

Suppose $V - S$ is a vertex cover between any pair of nodes $u, v \in S$, the edge connecting them **would not exist** in $V - S$. Otherwise, it violates the definition of such vertex cover. Hence, a single edge can reach no pair in S, creating an independent set.

Remark: Independent Set of size k corresponds to a Vertex Cover of size $V - |k|$.

2.4 3-SAT TO CLIQUE

A 3-SAT (as a particular case of SAT) comprises three literal clauses. The goal is to reduce a clique of size k in a group of k clauses ϕ. The following steps assist in this reduction:

- Building a graph G of k clusters with a **maximum** of 3 nodes in each cluster;
- Each cluster corresponds to a **clause** in ϕ;
- Each node in a cluster is **labeled with a literal from the clause**;
- An edge is put between all pairs of nodes in different cluster **except for pairs of the form** $(x, \bar{x})$;
- **No edge is put between any pair of nodes in the same cluster**.

Given the following clause, as an example:

$$\phi = (x_2 + x_1 + \bar{x}_3)(\bar{x}_1 + \bar{x}_2 + x_4)(x_2 + \bar{x}_4 + x_3)$$

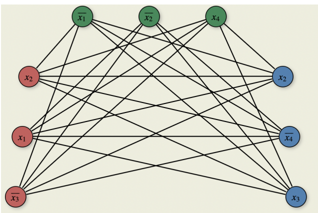


Figure 2.3: 3-SAT to clique

If **two nodes are connected**, it means that the literal can be simultaneously *true*.

If **two literals**, not in the same clause, can be assigned *true* simultaneously, the nodes are also connected.

Using the polynomial reduction, we can prove that if G has k-size clique, ϕ is satisfiable.

Proof 3
If G has a clique of size k, the clique has ***exactly one node*** *in from each cluster. Hence, all corresponding literals can be assigned* true *with each literal belonging to an* ***individual*** *k* ***clauses****. Then, ϕ is* ***satisfiable****.*

*If ϕ is **satisfiable**, a combination of nodes corresponds to it. Let the set of nodes be A. Some literals are* true *from each clause and are also in A. Remembering that **two literals cannot be from the same clause**, a clique can be formed by connecting a single node from each clause, forming a **clique**.*