More NP-Complete Problems

Lecture 19–20

Binghui Peng

2-SAT vs. 3-SAT

$$\text{3-SAT} = \{\varphi : \varphi \text{ is a satisfiable CNF formula with clauses of size 3}\}.$$

$$\text{2-SAT} = \{\varphi : \varphi \text{ is a satisfiable CNF formula with clauses of size 2}\}.$$

Linear programming vs. Integer programming

Linear Programming:

$$\mathrm{LP} = \left\{ A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m : \exists x \in \mathbb{R}^n \text{ such that } Ax \leq b \right\}.$$

Integer Programming:

$$\mathrm{IP} = \left\{ A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m : \exists x \in \mathbb{Z}^n \text{ such that } Ax \leq b \right\}.$$

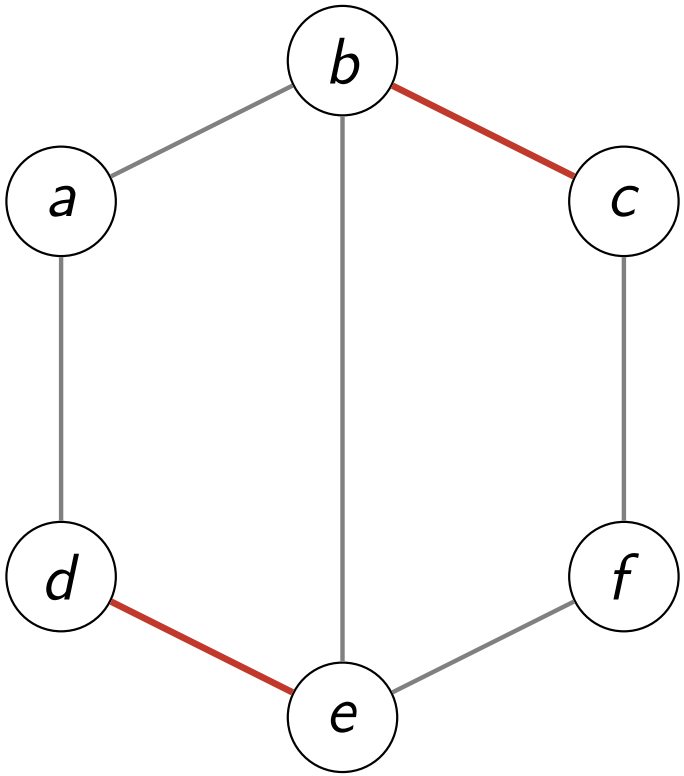
Matching

(2D) Matching:

$$\text{MATCHING} = \{\langle G, k\rangle : G \text{ has a matching of size at least } k\}.$$

A matching is a set of edges with no shared endpoints.

Example: Ordinary Matching



The highlighted edges $\{(b, c), (d, e)\}$ form a matching because they share no endpoints.

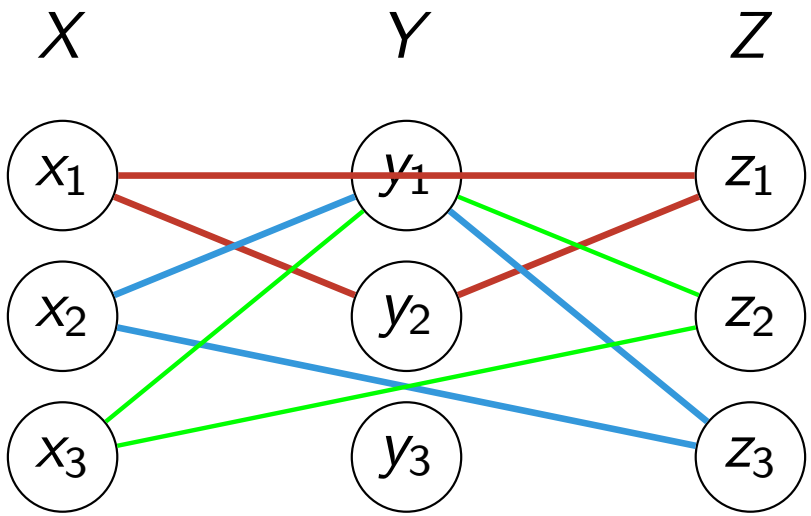
3-Dimensional Matching

3-dimensional matching:

$$\text{3D-MATCHING} = \Big\{ \langle X, Y, Z, \mathcal{T}, k \rangle \;\Big|\; \exists M \subseteq \mathcal{T} \text{ such that } |M| = k \text{ and no two triples in } M \text{ have overlaps} \Big\}.$$

Here $\mathcal{T} \subseteq X \times Y \times Z$ and $|X| = |Y| = |Z|$.

Example: 3-Dimensional Matching



Edges with the same color represent one triple. The red triple (x_1, y_2, z_1) and the blue triple (x_2, y_1, z_3) form a 3-dimensional matching because no element is used twice.

Poll

We have discuss 2SAT, 3SAT, 2D matching, 3D matching, linear programming, integer programming.

Poll

We have discuss 2SAT, 3SAT, 2D matching, 3D matching, linear programming, integer programming.

Which problem is in P? Which one is NP-complete?

Poll

We have discuss 2SAT, 3SAT, 2D matching, 3D matching, linear programming, integer programming.

Which problem is in P? Which one is NP-complete?

Poll in piazza.

Linear Programming

Linear Programming asks whether a system of linear inequalities has a solution.

$$\mathrm{LP} = \{\langle A, b\rangle : \exists x \in \mathbb{R}^n \text{ such that } Ax \leq b\}.$$

Linear programming is a famous problem in P.

Integer Programming

Integer Programming asks whether a system of linear inequalities has a solution in **integers**.

$$\mathrm{IP} = \{\langle A, b\rangle : \exists x \in \mathbb{Z}^n \text{ such that } Ax \leq b\}.$$

This looks almost identical to LP, except that the variables must be integral.

Integer Programming

Theorem. Integer Programming is NP-complete.

Observation: adding integrality constraints changes a polynomial-time decidable problem into an NP-complete one.

Integer Programming $\in$ NP

Certificate: an integer vector x.

Verifier:

1. check that each entry of x is an integer,
2. compute Ax,
3. check that every inequality in $Ax \leq b$ holds.

All of these checks can be done in polynomial time in the input size.
Therefore Integer Programming $\in$ NP.

Reduction $3\mathrm{SAT} \leq_m^p \mathrm{IP}$

We now prove that IP is NP-hard by reducing from $3\mathrm{SAT}$.

Input: a CNF formula φ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.

Variables of the IP:

1. for each Boolean variable x_i, introduce an integer variable, also called x_i,
2. enforce $0 \leq x_i \leq 1$ for every i.

Thus feasible solutions force each x_i to behave like a Boolean variable.

Encoding the Clauses

For each clause, write one linear inequality. **Examples:**

1. $(x_i \vee x_j \vee x_k)$ becomes

$$x_i + x_j + x_k \geq 1.$$

2. $(\neg x_i \vee x_j \vee \neg x_k)$ becomes

$$(1 - x_i) + x_j + (1 - x_k) \geq 1.$$

More generally, each literal contributes either x_i or $(1 - x_i)$.

The clause is satisfied exactly when the corresponding inequality holds.

Why the Reduction Works

Let S be the system of inequalities constructed from φ.

If φ is satisfiable: Take a satisfying assignment and set each IP variable $x_i \in \{0, 1\}$ accordingly.

Then every bound $0 \leq x_i \leq 1$ holds, and every clause inequality holds because each clause has at least one true literal.

If S has an integer solution: Since $0 \leq x_i \leq 1$ and x_i is integral, each x_i is either 0 or 1.

Interpret these values as a truth assignment. Every clause inequality then implies that the corresponding clause has at least one satisfied literal.

Ordinary Matching

Recall the graph matching problem:

$$\text{MATCHING} = \{\langle G, k\rangle : G \text{ has a matching of size at least } k\}.$$

A matching is a set of edges with no shared endpoints.

Fact: this problem is in P.

Now we generalize from edges to triples.

3-Dimensional Matching

Let X, Y, Z be disjoint sets of equal size, and let

$$\mathcal{T} \subseteq X \times Y \times Z$$

be a collection of triples.

A **3-dimensional matching** is a subset $M \subseteq \mathcal{T}$ such that no two triples in M has overlaps.

$$\begin{aligned} \text{3D-MATCHING} = \{&\langle X, Y, Z, \mathcal{T}, k\rangle \text{ such that} \\ &\text{there exists a matching of size } k\}. \end{aligned}$$

3D-MATCHING is NP-complete

Theorem. 3D-MATCHING is NP-complete.

Observation: Extending matching from pairs to triples changes a polynomial-time problem into an NP-complete one.

Reduction 3-SAT $\leq_m^p$ 3D-MATCHING

Given a 3-CNF formula.

$$\varphi = C_1 \wedge \cdots \wedge C_m$$

We build disjoint sets X, Y, Z and triples $\mathcal{T}$ with

$$|X| = |Y| = |Z| = 6m, \quad k = 6m$$

The construction has three parts:

1. a clause gadget for each clause,
2. a variable gadget for each variable,
3. garbage-collection gadgets for leftover literal occurrences.

Clause Gadgets

For each clause C_j: Say $C_j = (x_1 \vee \neg x_2 \vee x_3)$

- Create 6 nodes $x_1, \neg x_1, x_2, \neg x_2, x_3, \neg x_3 \in X$
- Create a node $c_j \in Y$ and a node $d_j \in Z$.
- Add the following three triples (edges)

$$(x_1, c_j, d_j), \quad (\neg x_2, c_j, d_j), \quad (x_3, c_j, d_j).$$

Comment: Because c_j and d_j occur nowhere else, any perfect matching must choose exactly one of these three clause triples.

Variable Gadgets

Fix a variable x_i, suppose it appears (n_i) times in the clause

- Create n_i nodes in Y and n_i nodes in Z

$$y_i^1, \ldots, y_i^{n_i} \in Y \qquad \text{and} \qquad z_i^1, \ldots, z_i^{n_i} \in Z.$$

- Add the $2n_i$ triples (edges)
  - (x_i^t, y_i^t, z_i^t) for $t = 1, \ldots, n_i$,
  - $(\neg x_i^t, y_i^{t+1}, z_i^t)$ for $t = 1, \ldots, n_i$, where $y_i^{t+1} := y_i^1$

Observation: In any perfect matching, the gadget can be covered in only two ways:

1. choose all "positive" triples (x_i^t, y_i^t, z_i^t), or
2. choose all "negative" triples $(\bar{x}_i^t, y_i^{t+1}, z_i^t)$.

This binary choice encodes the truth value of x_i.

Garbage-Collection Gadgets

After the variable gadgets encodes the truth table and the clause gadgets choose one literal per clause, some occurrence elements of X remain uncovered.

There are exactly $2m$ such leftover elements.

To this end, we create $2m$ nodes in Y_i and $2m$ nodes Z_i, they are connected with every nodes in X

Observation These garbage triples absorb whichever literal occurrences are not used by the clause gadgets or the chosen side of a variable gadget.

Why the Reduction Works

If φ is satisfiable:

1. for each variable, choose the variable-gadget side consistent with the assignment,
2. for each clause, choose a clause triple corresponding to one true literal,
3. use garbage triples to cover all remaining occurrence elements.

If there is a perfect 3-dimensional matching:

1. each variable gadget forces one of its two consistent global choices,
2. interpret that choice as the truth value of the variable.
3. each clause gadget forces exactly one chosen literal occurrence,
4. The chosen clause triple for each clause must use an occurrence compatible with that variable choice, so every clause is satisfied.

2-SAT is in P

Theorem 2-SAT $\in \mathrm{P}$.

2-SAT: Implication Graph

Given a 2-CNF formula φ, build a **directed** graph G_φ as follows.

Nodes: one node for every literal x_i and $\neg x_i$.

Edges: for each clause $(x_1 \vee x_2)$, add the two implications

$$\neg x_1 \to x_2 \qquad \text{and} \qquad \neg x_2 \to x_1 .$$

Intuition: $(x_1 \vee x_2)$ implies $(\neg x_1 \Rightarrow x_2)$ and $(\neg x_2 \Rightarrow x_1)$.

An edge $u \to v$ means: "if u is TRUE then v must be TRUE."

2-SAT: Algorithm

Algorithm (runs in $O(V + E)$ time):

1. Construct the implication graph G_φ.
2. Compute all **strongly connected components** (SCCs) of G_φ.
3. **If** any variable x_i has x_i and $\neg x_i$ in the **same** SCC, output reject.
4. **Otherwise**, output accept

Correctness of the algorithm

Unsatisfiability: If x_i and $\neg x_i$ lie in the same SCC, there are paths $x_i \rightsquigarrow \neg x_i$ and $\neg x_i \rightsquigarrow x_i$. Any assignment forces a chain of implications leading to a contradiction.

Satisfiability: We construct the satisfiable assignment as follow: We pick an arbitrary SCC, an arbitrary variable in the SCC, and make it True. Then cascade the value to all other variables in the same SCC. We prove by induction that this generates a smaller 2SAT instance (with variables in the SCC fixed) that is also satisfiable (so we can derive the assignment recursively).