AI2615 算法设计与分析 2020-2021 春季学期

Lecture 15 – More Proofs for NP-hardness Problems

2021 年 6 月 4 日

Lecturer: 张驰豪 *Scribe:* 陶表帅

1 Reviewing Previous Lecture

Informally, P is the set of the problems that can be *solved* in a polynomial time. NP is the set of the problems that can be *verified* in a polynomial time. Problems in both P and NP are decision problems: $f : \Sigma^* \to \{0,1\}$. An instance $x \in \Sigma^*$ satisfying $f(x) = 1$ is called a *yes* instance, and an instance $x \in \Sigma^*$ satisfying $f(x) = 0$ is called a *no* instance.

Consider an instance of an NP problem. If it is a yes instance, there exists a *certificate* such that the verifier, by taking the instance and the certificate as inputs, outputs 1. Otherwise, if it is a no instance, the verifier will always output 0 for every certificate. A typical NP problem example is SAT. The verifier takes a CNF formula (the instance) and an assignment (the certificate) as inputs, and output 1 if and only if the assignment makes the formula evaluate to true. A yes instance, which is a satisfiable CNF formula, can always be verified in a polynomial time given a satisfying assignment as a certificate. On the other hand, the verifier will always output 0 in a polynomial time if the formula is not satisfiable, for any input assignment.

NP-complete problems are those hardest problems in NP. We have seen many NP-complete problems in the previous lectures, including SAT. We have $P \subseteq NP$, and whether this containment is proper is a central open problem in computer science. If a NP-complete problem admits a polynomial time algorithm, then $P = NP$.

As a remark, we have formulated SAT as a decision problem, while the search problem (find a satisfying assignment if the formula is satisfiable, or announce that the formula is not satisfiable) is somehow no harder. If we can solve the decision problem in a polynomial time, we can solve the search problem in a polynomial time as follows. If a CNF formula ϕ is not satisfiable, the algorithm for the decision problem already outputs 0, and the search problem is solved as well. If ϕ is satisfiable, we try both $x_1 = \textbf{true}$ and $x_1 = \textbf{false}$, substitute it into ϕ so that we obtain a CNF formula ϕ' with $n-1$ variables, and use the algorithm to decide if ϕ' is satisfiable. At least one of $x_1 = \textbf{true}$ and $x_1 = \textbf{false}$ will make ϕ' satisfiable, and we can finalize the value for x_1. We can iteratively find out the values for $x_2, \ldots, x_n$. We have applied the algorithm for the decision problem for at most $2n$ times, so the overall time complexity for solving the search problem is still a polynomial in n. In fact, most NP problems have this "search-to-decision reduction" property.

To formally prove that one problem is no easier than the other, we use the technique "reduction". A reduction from a problem f to a problem g, denoted by $f \le_k g$ (the subscript k is for Karp reduction), is a polynomial time computable mapping $\mathcal{R}$ such that $f(x) = 1$ if and only if $g(\mathcal{R}(x)) = 1$. This implies that a polynomial time algorithm to solve g can be used to solve f, which further implies that g is no easier than f.

Since we have proved SAT is NP-complete (SAT is a hardest problem in NP), to show another NP problem is NP-complete, we only need to reduce it from SAT (implying this problem is no easier than SAT; since SAT is already the hardest NP problem, this problem is also the hardest). The "web of reductions" in Fig. 1 presents those NP-complete problems identified by reductions (compared with Fig. 2 in the previous lecture, we have added the two problems: 3-Coloring and Max-Cut). In particular, we have presented the reductions for 3-SAT, Independent Set, Vertex Cover (exercise), Clique (exercise), 3-SAT ($\Delta \le 3$), and 3D-Matching in the previous lecture. In this lecture, we will present the reduction for ZOE, Subset-Sum, Hamiltonian Circuit, TSP, Max-Cut and 3-Coloring.

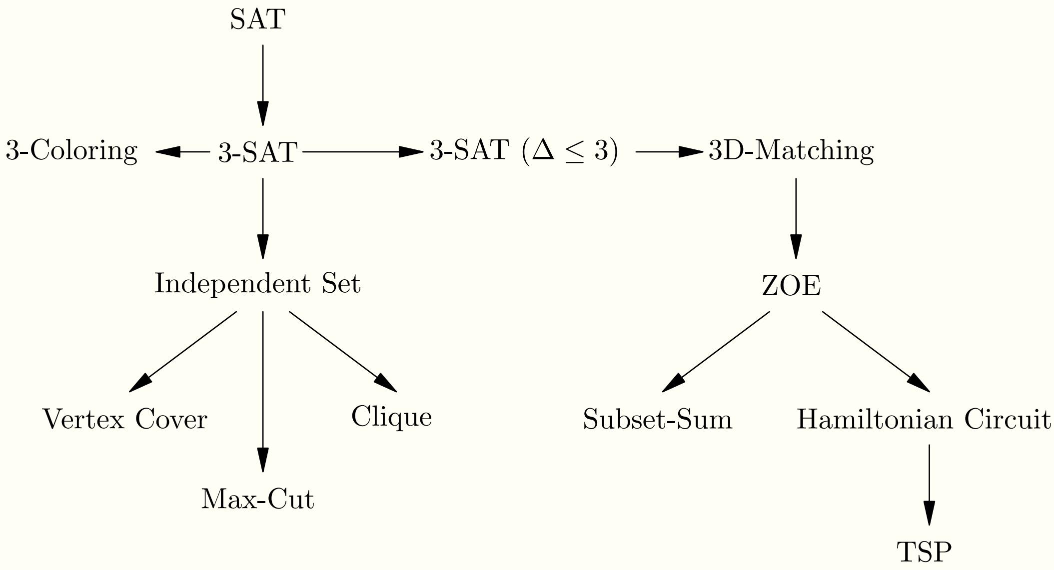

Figure 1: The web of reductions.

2 ZOE

Problem 1 (ZOE). Given an $m \times n$ zero-one matrix $A \in \{0,1\}^{m\times n}$, decide if there exists a zero-one vector $\mathbf{x} \in \{0,1\}^n$ such that $A\mathbf{x} = \mathbf{1}$.

The ZOE problem can be formulated as follows. We have m constraints for n zero-one variables $x_1, \ldots, x_n \in \{0,1\}$ such that each constraint takes the form

$$x_{i_1} + x_{i_2} + \cdots + x_{i_k} = 1.$$

Since ZOE is obviously in NP and we have seen 3D-matching is NP-complete, the following theorem shows that ZOE is NP-complete.

Theorem 2. *3D-matching* $\leq_k$ *ZOE.*

Proof. A 3D-matching instance can be described by a set of hyper-edges $\{e_i = (a_{i_1}, b_{i_2}, c_{i_3})\}$. For each edge e_i, we construct a variable $x_i \in \{0,1\}$ for the ZOE instance. The value of x_i represents if edge e_i is selected in the matching. To ensure the set of selected edges forms a matching, we need to make sure, for each vertex u in $A \cup B \cup C$, the sum of all those x_i's such that $u \in e_i$ equals to exactly 1. This gives us a constraint in ZOE. Therefore, we can use a set of ZOE constraints to describe the requirement that the selected edges form a matching. □

3 Subset-Sum

Problem 3 (Subset-Sum). Given a collection of positive integers S and an integer $k \in \mathbb{Z}^+$, decide if S contains a subset whose sum is exactly k.

Again, it is easy to see that subset-sum is in NP, and a simple reduction can show that subset-sum is NP-complete.

Theorem 4. *ZOE* $\leq_k$ *Subset-Sum*

Proof. A ZOE instance A can be rewritten as follows:

$$\sum_{i=1}^{n} x_i \mathbf{a}_i = \mathbf{1},$$

where $x_i \in \{0,1\}$ is the i-th entry of the vector $\mathbf{x}$ and $\mathbf{a}_i$ is the i-th column of A. Equivalently, we are asked if we can select a subset of n vectors with sum exactly an all-one vector. A natural way to view it as a subset-sum instance is to view each vector a binary encoding of an integer. However, what we have is that, for each $t = 1, \ldots, m$, the t-th bits of those select integers add up to 1; what we want is that the sum of those select integers is exactly the number represented by a m-bit all-one string. These two statements are not equivalent, as there may be carries in the additions. This can be easily fixed: instead of viewing the vector as a binary representation of a number, we can view it as a $(n+1)$-ary representation of a number, so that a carry in the addition will never happen. □

4 Hamiltonian Circuit

To show that Hamiltonian circuit is NP-complete, we first show that the following intermediate problem is NP-complete.

Problem 5 (Hamiltonian Circuit with Paired Edges). Given an undirected multi-graph $G = (V, E)$ (a multi-graph is a graph where more than one edge is allowed for a pair of vertices) and a set of edge-pairs $F = \{(e_1, e_1'), \dots (e_k, e_k')\}$, decide if G contains a Hamiltonian circuit that uses exactly one edge from each pair in F.

Theorem 6. *Hamiltonian circuit with paired edges is NP-complete.*

Proof. It is clear that the problem is in NP, as the encoding of a valid circuit can be served as a certificate. To show it is NP-complete, we present a reduction from the ZOE problem.

Given a ZOE instance $A \in \{0, 1\}^{m \times n}$, we construct an instance $(G = (V, E), F)$ for Hamiltonian circuit with paired edges as follows. The graph G contains $m + n$ vertices: $u_1, \dots, u_n, v_1, \dots, v_m$. For each $i = 1, \dots, n$, construct two edges $e_i = \{u_i, u_{i+1}\}$ and $\bar{e}_i = \{u_i, u_{i+1}\}$, and add the pair $(e_i, \bar{e}_i)$ to F. Here, we let $u_{n+1} = v_1$. For each $j = 1, \dots, m$, let $a_{j1}, \dots, a_{jn} \in \{0, 1\}$ be the j-th row of A. If $a_{jk} = 1$, construct an edge $f_{jk} = \{v_j, v_{j+1}\}$, and add the pair $(f_{jk}, \bar{e}_k)$ to F. In particular, the number of edges between v_j and v_{j+1} is exactly the number of ones in the j-th row of A. Here, we let $v_{m+1} = u_1$. This finishes the description of the construction. The construction can clearly be done in a polynomial time.

It is easy to see that a Hamiltonian path must visit all the vertices in the following order (or a rotation of the following order):

$$u_1 \to u_2 \to \dots \to u_n \to v_1 \to v_2 \to \dots \to v_m \to u_1.$$

In addition, for any pair of two adjacent vertices above, exactly one edge must be selected to be in the circuit. In particular, for each $i = 1, \dots, n$, exactly one of $e_i, \bar{e}_i$ must be selected; for each $j = 1, \dots, m$, exactly one of those f_{jk}'s must be selected. Finally, if $\bar{e}_i$ is selected, then f_{ji} must not be selected for all j; if e_i is selected, then we know $\bar{e}_i$ cannot be selected, and f_{ji} must be selected for all j.

These observations imply that the instance we constructed exactly simulates the ZOE instance. e_i being selected represents $x_i = 1$, while $\bar{e}_i$ being selected represents $x_i = 0$. The above observations then imply f_{ji} is selected if and only if $x_i = 1$. Since the edge f_{jk} exists if and only if $a_{jk} = 1$, the observation that exactly one edge from v_j to v_{j+1} must be selected implies that $\sum_{k=1}^{n} a_{jk} x_k = 1$. By considering this equation for all j, we have $A\mathbf{x} = \mathbf{1}$.

Since the constructed instance exactly simulates the ZOE instance, the constructed instance is a yes instance if and only if the ZOE instance is a yes instance. □

Now we are ready to prove that Hamiltonian circuit is NP-complete. This can be done by a reduction from the Hamiltonian circuit with paired edges problem.

Theorem 7. *Hamiltonian circuit is NP-complete.*

Proof. It is clear that the Hamiltonian circuit problem is in NP, as the encoding of a valid circuit can be served as a certificate. To show it is NP-complete, we present a reduction from the Hamiltonian circuit with paired edges problem.

Given a instance (G, F) of the Hamiltonian circuit with paired edges problem, we construct a Hamiltonian circuit instance G' as follows. Starting from G, for each pair $\{e = \{u, v\}, e' = \{u', v'\}\} \in F$, we replace the two edges e, e' with the gadget shown in Fig. 2. This finishes the construction of G', and this construction can clearly be done in a polynomial time.

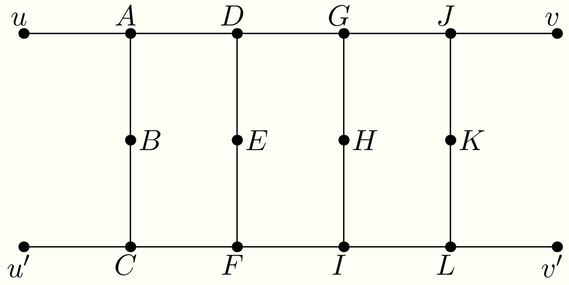

Figure 2: The gadget for the pair $\{e = \{u, v\}, e' = \{u', v'\}\}$.

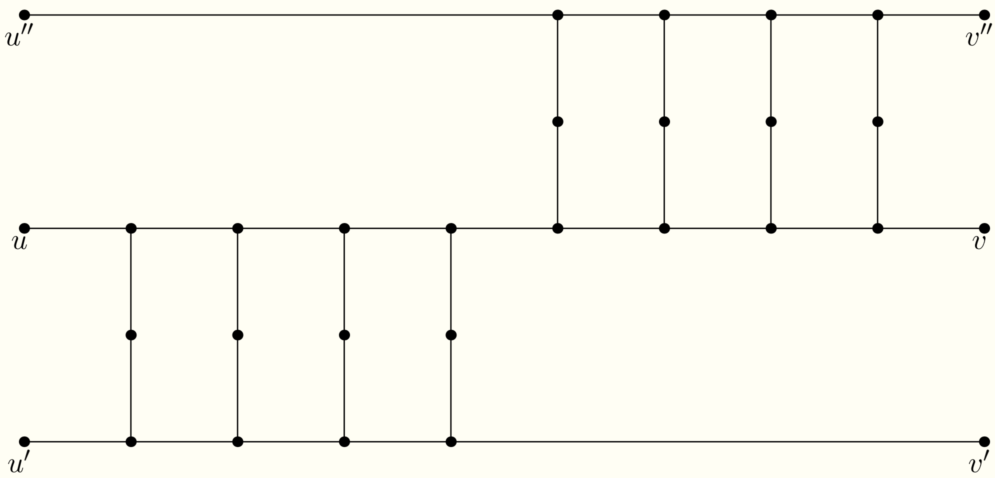

Figure 3: The gadget for the two pairs $\{e = \{u, v\}, e' = \{u', v'\}\}$ and $\{e = \{u, v\}, e'' = \{u'', v''\}\}$.

By some observations, it is easy to see that there are only two possible ways to visit each of the 12 intermediate vertices $A, B, C, D, E, F, G, H, I, J, K, L$ exactly once. For the first possibility, we can visit the 12 vertices by the following order $u \to A \to B \to C \to F \to E \to D \to G \to H \to I \to L \to K \to J \to v$, or the reverse of this order. For the second possibility, we can visit the 12 vertices by the following order $u' \to C \to B \to A \to D \to E \to F \to I \to H \to G \to J \to K \to L \to v'$, or the reverse of this order. As a result, if we choose to go from u, we can only reach v, and if we chose to go from v, we can only reach u. In either case, we can no longer go from u' to v' or from v' to u', as the 12 vertices are already used. In G, this says that if we choose the edge e, we cannot use the edge e'. Similar analysis shows that if we choose the edge e', we cannot use the edge e. Therefore, the gadget enforces the requirement that exactly one of e and e' must be chosen.

If an edge $e = (u, v)$ is contained in two pairs $\{e = \{u, v\}, e' = \{u', v'\}\}$ and $\{e = \{u, v\}, e'' = \{u'', v''\}\}$, we can use the gadget in Fig. 3. The case where e is contained in more than two pairs can be handled similarly. The remaining part of the proof is straightforward. □

5 Travelling Salesman Problem (TSP)

Problem 8 (TSP, decision version). Given an undirected complete weighted graph $G = (V, E, w)$ and a positive number k, decide if there is a tour that visits each vertex exactly once such that the length of the tour is at most k.

Theorem 9. *TSP is NP-complete.*

Proof. Firstly, TSP is clearly in NP, as a tour can be used as a certificate. To show it is NP-complete, we present a reduction from Hamiltonian circuit.

Given a Hamiltonian circuit instance G, we construct a TSP instance (G', k) as follows. Firstly, G' and G share the same vertex set. Then, for each pair of vertices u, v, if $\{u, v\}$ is an edge in G, let the weight of $\{u, v\}$ be 1 in G'; if $\{u, v\}$ is not an edge in G, let the weight of $\{u, v\}$ be $1 + \alpha$ in G', where α is a positive number. Finally, let k be the number of vertices in G (or G'). The construction can clearly be computed in a polynomial time.

If G contains a Hamiltonian circuit, the same circuit is a valid TSP tour, which has length exactly $|V| = k$. Thus, a yes instance for the Hamiltonian circuit is mapped to a yes instance for the TSP problem.

If G does not contain a Hamiltonian circuit, then any TSP tour must use at least one edge with weight $1 + \alpha$. As a result, the length of the tour is at least $|V| + \alpha > k$. Thus, a no instance for the Hamiltonian circuit is mapped to a no instance for the TSP problem. □

In the reduction above, we have seen that it is NP-complete to decide if the minimum length TSP tour is at most $|V|$ or at least $|V| + \alpha$. The only requirement for α is $\alpha > 0$. In fact, we can let α be a very large number. In this case, it is NP-complete to decide if the minimum length TSP tour is reasonably short or very long. This implies we do not have any polynomial time approximation algorithm for TSP if assuming $\mathrm{P} \neq \mathrm{NP}$. In particular, we cannot even have a polynomial time approximation algorithm with the approximation ratio as worse as an exponential factor.

Theorem 10. *If we have a polynomial time F-approximation algorithm for TSP for some F that can depend on the parameters of the TSP instance, then $P = NP$*

Proof. Suppose we have a polynomial time F-approximation algorithm $\mathcal{A}$ for TSP. We show that we can use $\mathcal{A}$ to solve the Hamiltonian circuit problem in a polynomial time. Since Hamiltonian circuit is NP-complete, this implies $\mathrm{P} = \mathrm{NP}$.

Given a Hamiltonian circuit instance G, we use the same construction for (G', k) in the previous proof, with α set to $\alpha = F \cdot |V|$. Then we run algorithm $\mathcal{A}$ for this constructed instance. We discuss two cases: 1) $\mathcal{A}$ outputs a tour with length at most $F \cdot |V|$; 2) $\mathcal{A}$ outputs a tour with length more than $F \cdot |V|$.

For the first case, we know that G contains a Hamiltonian circuit. Otherwise, the TSP tour in G' must have length at least $|V| + \alpha = (F+1)|V| > F \cdot |V|$, which contradicts to our assumption.

For the second case, we know that G does not contain a Hamiltonian circuit. Since $\mathcal{A}$ is a F-approximation algorithm, the optimal TSP tour must have length more than $F \cdot |V|/F = |V|$. On the other hand, if G contains a Hamiltonian circuit, then G' must have a TSP tour with length exactly $|V|$, which is a contradiction. Thus, G must not contain a Hamiltonian circuit.

We have described above how to use $\mathcal{A}$ to solve the Hamiltonian circuit problem in a polynomial time. $\square$

We say that TSP is NP-hard to approximate to within any finite factor.

6 Max-Cut

In this section, we will show that Max-Cut is NP-complete by a reduction from Independent Set. Notice that we have seen in the previous lecture that Independent Set is NP-complete.

Problem 11 (Max-Cut, decision version). Given an undirected graph $G = (V, E)$ and an integer k, decide if V can be partitioned to A, B such that $|E(A, B)| \geq k$.

Problem 12 (Independent Set, decision version). Given an undirected graph $G = (V, E)$ and an integer k, decide if G contains an independent set of size (at least) k.

Theorem 13. *Max-Cut is NP-complete.*

Proof. Max-Cut is clearly in NP, as the partition $\{A, B\}$ can be served as a certificate (then we can count the number of edges between A and B to see if it is at least k). To show Max-Cut is NP-complete, we reduce it from the independent set problem.

Given an independent set instance $(G = (V, E), k)$, we construct a max-cut instance $(G' = (V', E'), k')$ as follows. Firstly, V' contains all the vertices in V. We slightly abuse the notation and let V to denote both vertices in the original graph G and vertices in G' that correspond to V in the original graph. Secondly, for each edge $e = \{u, v\} \in E$ in the original graph, we construct two vertices u_e, v_e and three edges $\{u, u_e\}, \{u_e, v_e\}$ and $\{v_e, v\}$. Thirdly, we construct a vertex x and create an edge between x and each of the remaining vertices. We have constructed a total of $|V| + 2|E| + 1$ vertices and $|V| + 5|E|$ edges. Lastly, set $k' = k + 4|E|$. Let E'_V be the set of the edges $\{\{x, v\} \mid v \in V\}$ that connect the vertex x to those vertices in $V \subseteq V'$. Let $E'_E = E' \setminus E'_V$. Figure 4 shows the gadget corresponding to each edge $e = \{u, v\}$ in the original graph. Notice that, in the gadget, $\{u, x\}$ and $\{v, x\}$ belong to E'_V, and the remaining five edges belong to E'_E.

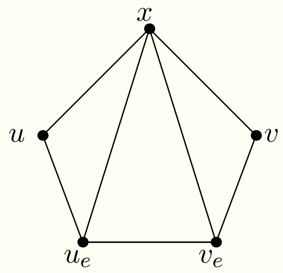

Figure 4: The edge gadget.

Suppose the independent set instance is a yes instance. Let I be an independent set in G with $|I| = k$. We aim to show that G' contains a cut $\{A, B\}$ with size $|E(A, B)| \geq k' = k + 4|E|$. The vertex set A is defined as follows. Firstly, A contains all the vertices in I. Secondly, for each edge $e = \{u, v\}$ in the original graph,

- if $u \in I$ and $v \notin I$, then include v_e into A;
- if $u \notin I$ and $v \in I$, then include u_e into A;
- if $u, v \notin I$, then include both u_e and v_e into A;
- notice that we cannot have $u, v \in I$ as I is an independent set.

Then, B is the set of the remaining vertices: $B = V' \setminus A$. In particular, $x \in B$.

Now, let us calculate $|E(A, B)| = |E(A, B) \cap E'_V| + |E(A, B) \cap E'_E|$, and we will calculate each of $|E(A, B) \cap E'_V|$ and $|E(A, B) \cap E'_E|$. Firstly, $|E(A, B) \cap E'_V| = k$, as I is a subset of A containing k vertices and $x \in B$. Secondly, we show that $|E(A, B) \cap E'_E| = 4|E|$. For each edge $e = \{u, v\}$ in the original graph, we have three possibilities for the five vertices in the gadget: 1) $u, v_e \in A$ and $u_e, v, x \in B$, 2) $v, u_e \in A$ and $u, v_e, x \in B$, and 3) $u_e, v_e \in A$ and $u, v, x \in B$. It can be easily checked that the gadget contains exactly 4 edges in E'_E in any of those three cases. Therefore, since we have $|E|$ edges in the original graph, we have $|E(A, B) \cap E'_E| = 4|E|$. Putting these together, we have $|E(A, B)| = k + 4|E| = k'$, which implies that the max-cut instance is a yes instance.

Suppose the independent set instance is a no instance, and we aim to show that the max-cut instance is also a no instance. We prove the contra-positive of this. Suppose the max-cut instance is a yes instance: there exist A, B such that $|E(A, B)| \geq k' = k + 4|E|$. We aim to show that G contains an independent set with k vertices. Without loss of generality, we assume $x \in B$ (if $x \in A$, we can just rename the two sets A and B). Let $I = V \cap A$. Let E_I be the set of edges $\{u, v\} \in E$ in the original graph such that $u, v \in I$. We will find an upper bound for $|E(A, B)| = |E(A, B) \cap E'_V| + |E(A, B) \cap E'_E|$ by finding an upper bound for each of $|E(A, B) \cap E'_V|$ and $|E(A, B) \cap E'_E|$.

Firstly, it is clear that $|E(A, B) \cap E'_V| = |I|$. Secondly, to find an upper bound for $|E(A, B) \cap E'_E|$, we consider each edge gadget. For each edge $e = \{u, v\}$ in the original graph, if $e = \{u, v\} \in E_I$ (i.e., we have $u, v \in I$), then the number of edges in $E(A, B) \cap E'_E$ is at most 3 in the gadget:

- if $u, v, u_e, v_e \in A$, we have $\{u_e, x\}, \{v_e, x\} \in E(A, B) \cap E'_E$;
- if $u, v, u_e \in A$ and $v_e \in B$, we have $\{u_e, x\}, \{u_e, v_e\}, \{v_e, v\} \in E(A, B) \cap E'_E$;

- if $u, v, v_e \in A$ and $u_e \in B$, we have $\{u, u_e\}, \{u_e, v_e\}, \{v_e, x\} \in E(A,B) \cap E'_E$;
- if $u, v \in A$ and $u_e, v_e \in B$, we have $\{u, u_e\}, \{v, v_e\} \in E(A,B) \cap E'_E$.

Notice that we do not count $\{u, x\}$ and $\{v, x\}$, as they have been counted in $|E(A,B) \cap E'_V|$. For each edge $e = (u, v) \notin E_I$, the number of edges in $E(A,B) \cap E'_E$ is at most 4 in the gadget. To see this, there are 5 edges in $E(A,B) \cap E'_E$ in the gadget. In addition, at least one edge will be lost: if one of u_e and v_e is in B, then we lost the edge $\{u_e, x\}$ or $\{v_e, x\}$; if we have $u_e, v_e \in A$, then the edge $\{u_e, v_e\}$ is lost. Putting these together, we have $|E(A,B) \cap E'_E| \leq 3|E_I| + 4|E \setminus E_I| = 4|E| - |E_I|$, and

$$|E(A,B)| = |E(A,B) \cap E'_V| + |E(A,B) \cap E'_E| \leq |I| + 4|E| - |E_I|.$$

On the other hand, since $|E(A,B)| \geq k' = k + 4|E|$, we have

$$|I| + 4|E| - |E_I| \geq k + 4|E|,$$

which implies

$$|I| \geq k + |E_I|.$$

Finally, for each $\{u, v\} \in E_I$, we remove one of u and v from I. After removing $|E_I|$ vertices from I, we obtain an independent set of size at least k, which implies the independent set instance is a yes instance. □

7 3-Coloring

Definition 14 (3-Coloring). Given an undirected graph $G = (V, E)$ and a set of 3 colors, decide if we can assign a color from the set to each vertex such that no two adjacent vertices have the same color.

In general, we can define k-coloring, which uses a set of k colors instead of 3 colors. 1-coloring is trivial, as the instance is a yes instance if and only if there is no edge at all: $E = \emptyset$. For 2-coloring, the problem becomes deciding if a graph is a bipartite graph, which can be easily solved in a polynomial time. We will see that k-coloring is NP-complete for any $k \geq 3$.

Theorem 15. *3-coloring is NP-complete.*

Proof. 3-coloring is clearly in NP, as a color assignment of all vertices is a certificate for a yes instance. To show it is NP-complete, we reduce it from 3-SAT.

Given a 3-SAT instance ϕ, we construct a 3-coloring instance $G = (V, E)$ as follows. We will name the three colors T, F, N which stand for "true", "false", "neutral". The reason for this naming will be apparent soon. Firstly, we construct three vertices named t, f, n and three edges $\{t, f\}, \{f, n\}, \{n, t\}$. It is easy to see these 3 vertices, forming a triangle, must be assigned different colors. Without loss of generality, we assume $c(t) = T, c(f) = F, c(n) = N$, where $c : V \to \{T, F, N\}$ is the color assignment function. We call this triangle the "palette": in the remaining part of the graph, whenever we want to enforce that a vertex cannot be assigned a particular color, we connect it to one of the three vertices in the palette.

For each variable x_i in ϕ, we construct two vertices $x_i, \bar{x}_i$, and three edges $\{x_i, \bar{x}_i\}, \{x_i, n\}, \{\bar{x}_i, n\}$. This ensures that it must be either $c(x_i) = T, c(\bar{x}_i) = F$ or $c(x_i) = F, c(\bar{x}_i) = T$. The former case corresponds to assigning **true** to variable x_i, and the latter case corresponds to assigning **false** to variable x_i.

After we have constructed the gadgets simulating the Boolean assignment for variables, we need to create a gadget to simulate each clause m. We use $m = (x_i \vee \bar{x}_j \vee x_k)$ as an example to illustrate the reduction. Firstly, we create a vertex m and connect it to the two vertices n, f, so that we can only have $c(m) = T$ (i.e., the clause must be evaluated to **true**). Then, we need to create a gadget that connects the three vertices x_i, $\bar{x}_j$ and x_k (constructed in the previous step) as "inputs", and connects vertex m at the other end as "output". The gadget must simulate the logical OR operation.

The gadget shown on the left-hand side in Fig. 5 simulates the logical OR operation for two inputs: if both two input vertices are assigned F, then the output vertex cannot be assigned T, for otherwise there is no valid way to assign colors for vertices A and B; if at least one input vertex is assigned T, say, Input 1 is assigned T, then it is possible to assign the output vertex T, since it is always valid to assign F to A and N to B. The gadget for the clause $m = (x_i \vee \bar{x}_j \vee x_k)$ can then be constructed by applying two OR gadgets, shown on the right-hand side of Fig. 5. We do this for all the clauses.

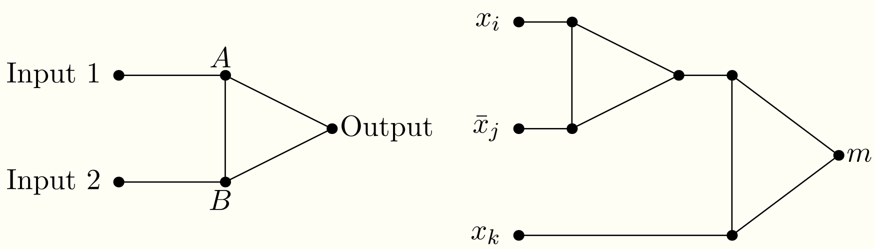

Figure 5: The OR gadget (left-hand side) and the gadget for the clause $m = (x_i \vee \bar{x}_j \vee x_k)$ (right-hand side).

The remaining part of the proof is straightforward. We have shown that G simulate assignments to ϕ. Thus, ϕ is satisfiable if and only if there is a valid color assignment to vertices in G. Finally, it is easy to verify that the construction of G can be done in a polynomial time. □

Exercise 16. Show that k-coloring is NP-complete for any $k \geq 3$.

7.1 3-Coloring on Planner Graph

In this section, we consider the coloring problem on planner graphs. We have seen that 1-coloring and 2-coloring are polynomial time solvable even for general graphs. Thus, these two problems for planner graphs are also in P. On the other hand, the famous *four color theorem* states that every planner graph is

4-colorable. Thus, every instance of the k-coloring problem on planner graphs is a yes instance for $k \geq 4$. The only remaining problem is the 3-coloring problem on planner graphs. In fact, $k = 3$ is the only value for k making the problem NP-complete.

Theorem 17. *3-coloring is NP-complete for planner graphs.*

Proof. Again, the problem is clearly in NP. To show it is NP-complete, we reduce it from the 3-coloring problem for general graphs. Given a 3-coloring problem instance G, we aim to construct a planner graph G' so that G is 3-colorable if and only if G' is. The idea is that, we embed G onto a plane. Then, whenever we have two crossing edges $\{x, x'\}$ and $\{y, y'\}$, we create a gadget that simulates the two edges while making the graph planner.

Firstly, we create a gadget in Fig. 6 which is called a "crossover gadget", and we will denote it by H. We can verify that it satisfies the following two properties:

1. for any valid color assignment c of H, we must have $c(x) = c(x')$ and $c(y) = c(y')$;
2. there exist two valid color assignments c_1, c_2 of H such that $c_1(x) = c_1(x') = c_1(y) = c_1(y')$ and $c_2(x) = c_2(x') \neq c_2(y) = c_2(y')$.

The gadget H ensures that the color of x and x' must be the same, while we want any two adjacent vertices to have different colors. To apply this gadget on an edge $\{u, v\}$ that intersect another edge $\{u', v'\}$, we only need to let u, u' be x, x' respectively, and let v, v' be *connected to* y, y' respectively. The first property of the gadget then ensures u and v must have different colors, and the same holds for u' and v'. The second property ensures that, the color assignments for u, v do not place any restriction on the color assignments for u', v'.

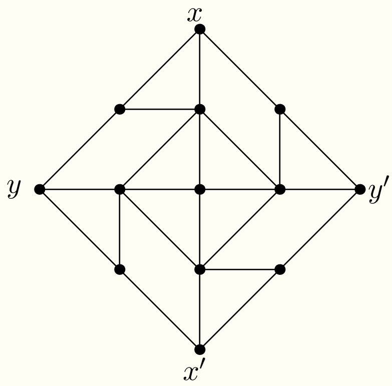

Figure 6: The crossover gadget H.

The case where an edge $\{u, v\}$ intersects multiple edges can be handled similarly. Fig. 7 illustrates what we can do for an edge intersecting four other edges. In Fig. 7, u and v' must be assigned the same color by the first property of the gadget, so u and v must be assigned different colors.

We have finished describing the construction. The remaining part of the proof is straightforward. $\square$

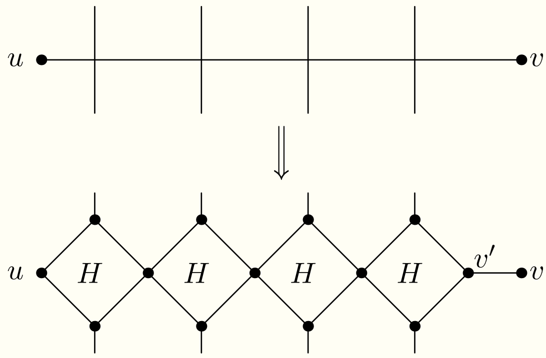

Figure 7: Replacing an edge $\{u, v\}$ with four crossing by four crossover gadgets.