More NP-Completeness: Optimization problems

Hsiang-Hsuan (Alison) Liu

1 Optimization problems and decision problems

According to the Turing machine model, the classes P and NP are defined on deciding if an instance is a yes-instance or a no-instance of a problem, by accepting or rejecting this instance, respectively. It is natural to consider the *decision problems* that are problems with yes/no answers. For example, in the Partition problem, we are given a set of numbers, and the answer is *"Yes, the set can be partitioned into two equal-sum subsets"* or *"No, the set cannot be partitioned into two equal-sum subsets"*.

However, we are also curious about the hardness of optimization problems. For example, how difficult it is to find the *largest* subset of vertices in the given graph such that no two vertices in the subset are adjacent? Or, how difficult it is to allocate the *minimum* number of the CCTV we need to cover all narrow alleys in the city?

To identify the difficulty of optimization problems and fit in the Turing machine model, we need a systematic method to transform the optimization to a (not-too-much-easier) decision version. The common method is introducing an extra parameter into the optimization problem:

- **Maximization problems.**
  - Optimization version: Given the input $\langle I \rangle$, what is the satisfying solution with the largest objective?
  - Decision version: Given the input $\langle I \rangle$, is there a satisfying solution with objective at least k?
- **Minimization problems.**
  - Optimization version: Given the input $\langle I \rangle$, what is the satisfying solution with the smallest objective?
  - Decision version: Given the input $\langle I \rangle$, is there a satisfying solution with objective at most k?

For example, the MaximumClique problem is an optimization problem, which is defined as: Given a graph G, what is the size of the maximum subset of vertices such that any pair of vertices in the subset are adjacent? The decision version of the MaximumClique problem is that *Given a graph G, is there a subset of at least k vertices such that any pair of vertices in the subset are adjacent?*

Optimization problem, decision version, and hardness. Note that the decision version of an optimization problem is not harder than the optimization version. Intuitively, if one can solve the optimization problem, they can solve the corresponding decision version. For example, if there is an algorithm that can find the maximum clique in the given graph, it can give the correct answer to the problem *"If there exists a clique with size at least k"* by running the maximum clique algorithm. If the algorithm returns a clique (which is a maximum one), and the clique is with size greater than or equal to k, then the answer to the decision version is "*yes*". However, we cannot use the decision version to solve the optimization problem directly. Therefore, given an optimization problem, we can argue that the optimization problem is not easier than its decision version.

Since the optimization version is not easier than the decision version, if we show that for an optimization problem, its decision version is NP-hard, then the optimization is at least NP-hard.

2 NP-Completeness of Graph Problems

In this section, we introduce two NP-complete problems, VertexCover and FeedbackVertexSet.

Vertex Cover. Given graph $G = (V, E)$, a *vertex cover* is a subset of vertices $C \subseteq V$ such that for all edge $(u, v) \in E$, u or v is in C. The minimum vertex cover problem asks about finding the minimum vertex cover in the given graph.

There are many interesting properties of a vertex cover in the graph. By definition, every edge in the graph is "covered" by at least one of its endpoints, which is in the vertex cover. Another property is that if we remove all the vertices in the vertex cover and the edges incident to the vertices, the remaining graph is an *independent set*. That is, a set of vertices without any edge. This can be proven by contradiction: assume from the contrary that there is still an edge (u, v) in the remaining graph, which means that neither u nor v is in the vertex cover. Then, edge (u, v) is not covered, which contradicts to the vertex cover definition.

The decision version of the minimum vertex cover is defined as follows:

VertexCover $= \{\langle G\rangle \mid G$ has a vertex cover with size at most $k\}$.

Now, we show the NP-hardness of VertexCover can be proven by reduction from 3SAT.

Proof. For any instance of 3SAT, $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_k$, we generate an instance of VertexCover, $G = (V, E)$ and k', as follows: For each Boolean variable x_i where $i \in [1, \ell]$, we set two vertices v_i and $\overline{v_i}$ in G. For each clause C_i containing three literals, ℓ_{i_1}, ℓ_{i_2}, and ℓ_{i_3}, there are three vertices in V, which form a triangle (3-clique), that are corresponding to the three literals. If x_i is in C_j, there is an edge between the variable vertex v_i and the clause vertex corresponding to the literal x_i in C_j. If $\overline{x_i}$ is in C_j, an edge exists between the variable vertex v_i and the clause vertex corresponding to the literal $\overline{x_i}$ in C_j. Finally, $k' = \ell + 2k$ (where there are ℓ Boolean variables and k clauses.

The transformation takes polynomial time $O(k\ell)$

Now, we show that the reduction works by showing that there is a satisfying assignment to ϕ if and only if there is a size-$(\ell + 2k)$ vertex cover in G. Suppose that ϕ has a satisfying assignment. We construct a size-$(\ell + 2k)$ vertex cover by selecting one of the variable vertices corresponding to the truth assignment. Since ϕ is satisfiable, there must be one of such a literal in every pair of v_i and $\overline{v_i}$. As the satisfying assignment feasibly makes every clause have at least one TRUE literal, for each clause, at most two of its three clause-variable edges remain uncovered now. We pick the endpoints of these uncovered edges that are in the clause triangles and cover the edges in the clause triangle. (If there are less than two such uncovered edges, we pick arbitrarily two vertices in the clause triangle.) The total number of picked vertices is $\ell + 2k$.

Suppose that G has a clique C of size $(\ell+2k)$. Since every variable-variable edge need to be covered by one of its endpoints, there are at least ℓ variable vertices in C. Since we need at least two vertices in a triangle to cover all its edges, there are at least $2k$ clause vertices in C. As $|C| \leq \ell + 2k$, there are exactly ℓ variable vertices in C, and each clause triangle has exactly 2 vertices in C. We assign the truth value to the variables in ϕ according to the picked variable vertices in C; if v_i is in C, we assign x_i as TRUE, else, we assign x_i as FALSE. The assignment satisfies ϕ because at least one of the three variable-clause vertices of a clause is covered by a variable vertex, which is corresponding to a true literal. Hence, each clause has one literal which is assigned TRUE and the formula ϕ is satisfied.

□

Feedback Vertex Set. Given graph $G = (V, E)$, a *feedback vertex set* is a subset of vertices $F \subseteq V$ such that removing all vertices and the incident edges in F make the remaining graph cycle-free. The minimum feedback vertex set problem is to find a minimum feedback vertex set to make the given graph cycle-free. The decision version of the minimum feedback vertex set is defined as follows:

FVS $= \{\langle G\rangle \mid G$ has a feedback vertex set with size at most $k\}$.

Now, we prove the NP-hardness of FVS by reducing from VertexCover.

Theorem 1. *FVS is NP-complete.*

Proof. To prove that FVS is in NP, we use a size-k feedback vertex set F as the certificate. The verifier should check F it is a proper subset of the vertices in V, and if G is cycle-free after removing all edges incident to the vertices in F. The later can be done by running a breadth-first-search on the resulting graph. The checking time is in polynomial of the size of G.

To prove the NP-hardness, we show that VertexCover $\leq_P$ FVS. For any instance of VertexCover, $G = (V, E)$ and k, we construct an instance of FVS, $G' = (V', E')$ and k' as follows. For each vertex $v_i \in V$, there is a corresponding vertex $v'_i \in V'$. More over, for each edge (v_i, v_j), there is a corresponding vertex $v'_{i,j} \in V'$. For each edge $(v_i, v_j) \in E$, we construct three edges in E': (v'_i, v'_j), $(v'_i, v'_{i,j})$, and $(v'_j, v'_{i,j})$. (See Figure 1.) Finally, we set $k' = k$. The construction takes constant time to each element in V or E and can be done in polynomial-time.

Now we prove that the reduction works. Suppose that there is a size-k vertex cover C of G. First observe that there are two types of cycles in G', 1) cycles containing no $v'_{i,j}$ vertices, and 2) cycles containing at least one $v'_{i,j}$ vertex. Consider removing all vertices in C from V', there is no edge between any two vertices in V', v'_i and v'_j. Therefore, there are no type-1 cycles in the remaining graph. Furthermore, because every vertex $v'_{i,j}$ only adjacent to v'_i and v'_j, the degree of $v'_{i,j}$ is at most 1 after removing vertices in C. Thus, there are no type-2 cycles left. Hence, C is a size-k feedback vertex set of G', and $\langle G', k' \rangle$ is a yes-instance of FVS.

For the other direction, suppose that there is a size-k feedback vertex set F of G'. We make a feedback vertex set F' of k with size at most k as follows. If there is a vertex $v'_{i,j}$ in F, we replace it by v'_i or v'_j, which was not in F. If both v'_i and v'_j are already in F, we simply remove $v'_{i,j}$. Because $v'_{i,j}$ has degree 2, it can only breaks the cycle $(v'_i, v'_j, v'_{i,j})$, and this cycle can be broken by v'_i or v'_j. Since we only replace or remove $v'_{i,j}$ from F to get F', F' is a feasible feedback vertex set with size at most k'.

Now, we argue that the vertices in F' form a vertex cover in G. Since there is no vertex $v'_{i,j}$ in F', at least one of v_i or v_j is in F', removing all vertices in F' leaves no edge between any pair of v'_i and v'_j. Otherwise, there is a cycle $(v'_i, v'_j, v'_{i,j})$, and it contradicts to the fact that F' is a feedback vertex set. Thus, F' is a vertex cover in G. That is, $\langle G, k \rangle$ is a yes-instance of the VertexCover problem. □

Figure 1: Illustration of the instance transformation from VertexCover to FVS. The left hand side is the instance graph of VertexCover, and the right hand side is the instance graph of FVS, where the orange vertices are vertices $v'_{i,j}$.

3 From problems without parameter to with parameter

In this section, we will see a reduction that introduces new parameters.

Bin Packing. Recall the (offline) Bin Packing problem, where we aim to pack n items with size within $(0, 1]$ into the minimum number of capacity-1 bins. The decision version is defined as follows:

BinPacking $= \{\langle U, k \rangle \mid$ The items in U can be partitioned into at most k disjoint subsets such that the total size of the items in each subset is no more than $1\}$.

The Bin Packing problem is NP-complete. The NP-hardness can be shown by polynomial-time reduction from Partition.

Theorem 2. *BinPacking is NP-complete.*

Proof. To prove that BinPacking is in NP, we use a k-partition of items in U as the certificate. In details, the k-partition can be encoded in an $|U|$-array, where the i-th entry is the index of the part the i-th item belongs to. The certificate has total size $|U| \log k$, which is polynomial in the input $\langle U, k \rangle$. The verifier should check if this partition is a proper partition of U, and if each subset has sum of size no more than 1. The checking time is in polynomial of the number of elements in U.

To prove the NP-hardness, we show that Partition $\leq_P$ BinPacking. For any instance of Partition, S, we construct an instance of BinPacking, (U, k) as follows. For each element $a_i \in S$, there is

a corresponding item $u_i \in U$ and $size(u_i) = \frac{2/cdota_i}{X}$, where X is half of the sum of all elements in S. We set $k = 2$. The construction can be done in polynomial time.

Now, we prove that the reduction works. Suppose that there is a partition of S, S_1 and S_2. For all elements $a_i \in S_1$, the sum is X. The sum of corresponding u_i's is 1, so the corresponding items can be placed in one bin. It also holds for S_2. Hence, the items can be packed into 2 bins.

For the other direction, suppose that the items in U can be packed in two bins. Each of the bin has total size 1 since the total size of all items in U is $\Sigma_i\ size(u_i) = \frac{\Sigma_i\ a_i}{X} = 2$. The corresponding two subsets of S has equal size and form a partition. □

General case, special case, and hardness. Note that in the NP-hardness proof, we reduce the PARTITION problem to a special case of BINPACKING problem, where the items can be packed into two bins. Therefore, be more precisely, by this reduction we show that the special case of BINPACKING is NP-hard. Then, how about the general case of BINPACKING, where the packed-in-two-bins restriction is relaxed? Intuitively, the general case is not easier than the special case (recall the "if one can solve the general case, they can use the algorithm for general case to solve the special case" argument). Therefore, the general case is at least NP-hard if the special case is NP-hard. This can be formally proved by reducing the special case to the general case in polynomial time (if you really want).

4 Strong and Weak NP-Completeness

The KNAPSACK problem, although it is NP-complete (see Exercise More NPC 2, Question 4), can be solved in $O(nB)$ time by dynamic programming, where n and B are the numbers of items and the bag capacity, respectively. The time complexity $O(nB)$ *looks like* polynomial while it is actually exponential in the input size, as the capacity B is represented in only $\log B$ bits.

The problems that are NP-complete when numbers are given in binary but polynomial-time solvable when numbers are given in unary encoding are called *weakly NP-complete* problems. There are known algorithms that solve these problems in time bounded by a polynomial in the numeric value of the input (instead of in the input length). The polynomial algorithms in the input's numeric value are *pseudo-polynomial time* algorithms.

Besides KNAPSACK, the PARTITION and SUBSET-SUM problems are also weakly NP-complete problems.

On the other hand, there are NP-complete problems like BINPACKING that do not have pseudo-polynomial time algorithms. These problems are called *strongly NP-complete*. SAT, 3SAT, VERTEX-COVER, FEEDBACKVERTEXSET,INDEPENDENTSET, CLIQUE, and INTEGERLINEARPROGRAMMING are also Strongly NP-complete problems. It is easy to see why SAT and 3SAT are strongly NP-complete: there are no even numerical parameters in the problem input. For graph problems like VERTEXCOVER, you can see that even if the parameter k is as small as $\log |V|$, the problem is still NP-hard. To show that a problem is strongly NP-hard, we need to 1) reduce a strongly NP-complete problem to it and 2) ensure that the reduction does not use a parameter with too large magnitude (compared to the input size $O(|V| + |E|)$).

5 NP and Co-NP

Recall that the definition of a problem in NP is that *for each yes-instance*, there exists a certificate that helps to verify. The class *co-NP* is defined as follows.

Definition 1. *A problem L is* **co-NP** *if for every no-instance w, there exists a certificate c such that there is a deterministic Turing machine can use c to verify that $w \notin L$ in polynomial time in the length of w.*

An alternative definition is:

Definition 2. *A language L is in* **co-NP** *if its complement language $\overline{L}$ is in* NP.

(Now you can see why we use the "polynomial-time verifiable" definition of class NP a lot: it is less confusing.)

Some examples of problem in co-NP are:

- NOT-HAMILTONIAN $= \{\langle G \rangle \mid G$ has no Hamiltonian cycle$\}$
- UNSATISIFIABLE $= \{\langle \phi \rangle \mid$ All truth assignments make ϕ false.$\}$
- INTEGERFACTORIZATION $= \{\langle n, k \rangle \mid n$ has a prime factor less than $k.\}$

By the definition of NP and co-NP, it follows that

$$\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{co\text{-}NP}.$$

That is, any problem in P is in both NP and co-NP. So far, it is unknown whether $\mathbf{P} = \mathbf{NP} \cap \mathbf{co\text{-}NP}$. It is also unknown whether $\mathbf{NP} = \mathbf{co\text{-}NP}$