KAIST, School of Computing, Spring 2024
Algorithms for NP-hard problems (CS492)
Lecture: Eunjung KIM

Scribed By: Eunjung KIM
Introduction & Branch algorithms
Week 1: 27, 29 February 2024

Contents

1 Introduction and Parameterized algorithm

Coping with intractability. Most computational problems are NP-hard. Some[1] even say that "there are only six polynomial-time solvable problems, essentially. All other problems are NP-hard." Then what do we do with these *intractable* problems? There are a range of algorithmic paradigms designed to cope with the prevalent intractability of computational problems, each emphasizing different aspects of an *efficient* algorithm. In this lecture, we focus on the paradigms of parameterized algorithms, (fast) exact algorithms and approximation algorithms.

Computational Problem and formalism. Let us clarify what we mean by computational problems when we want to treat them in a mathematically rigorous manner. It depends on the context. When you talk about a *decision problem*, a computational problem is considered as a *language over some alphabet* Σ, i.e. a subset $L \subseteq \Sigma^*$ of all (finite-length, possibly zero) strings over Σ. Here, the alphabet Σ is a finite set of symbols (e.g. consisting of 0 and 1) that we use to encode[2] an input of the problem as a string. In the context of a decision problem, the language L is viewed as the set of (the encodings of) YES-instances to a problem at hand. In fact, the set of YES-instances as a language is THE VERY DEFINITION of a decision problem. For example, VERTEX COVER is a problem which takes as an input G and a non-negative integer k, and the answer is YES if G has a vertex cover of size at most k and is NO otherwise. This is a human way of describing the problem VERTEX COVER. On the other hand, a machine (formal) way of describing VERTEX COVER is to give the set $\{(G, k) : k$is a non-negative integer and G is a graph having a vertex cover of size at most $k\}$, with a suitable encoding using some alphabet Σ. With the former description, we are looking for an algorithm to decide whether the input instance is YES-instance or not. With the latter description, we are looking for an algorithm to test a *membership* of a string $x \in \Sigma^*$ in a language L, i.e. $x \in L$ or not. We usually use the human description, but it is useful to be aware of the formal description.

A *Turing machine* is nothing but an algorithm which executes a membership of an input string $x \in \sigma^*$ for a language L, where the algorithm is described not in a human way but in a machine way[3]. When you look into what happens in a Turing machine testing if $x \in L$, there is something called a *witness*. A witness $w \in \Sigma^*$ is an additional sequence of symbols, on top of the input $x \in \Sigma^*$, which describes an 'path' for the input string x from the initial state to an accepting state. A vertex cover of size at most k of a graph G (given as an input string) can be seen as such a witness as it witnesses that (G, k) is a YES-instance; such a vertex cover can be converted into a string which assists a Turing machine which solves the problem VERTEX COVER to reach an accepting state. Therefore one can rewrite a decision problem $L \subseteq \Sigma^*$ in the form $\{x \in \Sigma^* :$ exists $y \in \Sigma^*$ such that $R(x, y) = 1\}$. Here, $R(x, y) = 1$ is the indicator function of a relation $R \subseteq \Sigma^* \times \Sigma^*$, i.e. $R(x, y) = 1$ if and only if $(x, y) \in R$.

An *optimization problem* is formally described by a set of relations $R \subseteq \Sigma^* \times \Sigma^*$, a *cost function*[4] $c : R \to \mathbb{N}_0$, and $\mathsf{goal} \in \{\min, \max\}$. For example, an optimization version of the problem VERTEX COVER can be described with R as the set of all (x, y) where x is (the encoding of) a graph G, and y is (the encoding of) a vertex cover of G, the cost function c on (x, y) equals the size of the vertex cover y, and $\mathsf{goal} = \min$.

[1] I heard this assertion a few times, but could not trace down the source. But let me unashamedly posit it here for a dramatic effect :)

[2] It is presumed that there is a predetermined rule of how to encode an input to the problem at hand.

[3] To learn how to rewrite an algorithm in a machine-like way is the subject of automata theory. Notice that this is a must in order to implement our 'algorithmic idea' at the hardware level, and that is why Turing machine is called the mathematical model of a computer.

[4] For simplicity, we assume that the cost is a non-negative integer, but it can be over the reals, etc

There are other forms of computational problems (search problem, promise problem, function problem, counting problem, etc) and often one can be cast in the form of the other. For example, a decision problem can be seen as a special form of an optimization problem while an optimization problem can be seen as a decision problem as well. Usually which formalism of computational problem shall be chosen depends on the algorithmic framework as well as the core difficulty of the problem. That is, for parameterized problem we mostly presume decision problems and for fast exact algorithms and approximation algorithms, we presume an optimization problem. For PRIMALITY TESTING, in which we decide whether a given number N is a prime or not, it hardly makes sense to consider it as an optimization problem.

Parameterized problems. Consider the problem VERTEX COVER, i.e. as a language

$$L := \{(G, k) : k\text{is a non-negative integer and } G \text{ is a graph having a vertex cover of size at most } k\}.$$

One can parameterized VERTEX COVER by the *solution size*.

Formally, a *parameterization* of Σ^* is a mapping $\kappa : \Sigma^* \to \mathbb{N}_0$. A *parameterized problem* is a pair (L, κ), where $L \subseteq \Sigma^*$ and κ is a parameterization of Σ^*. We say that (L, κ) is the problem L parameterized by κ. For instance, consider the parameterization of VERTEX COVER (as a decision problem) by the solution size (budget). In this case, the parameterization κ at hand shall map an instance (G, k) of VERTEX COVER to k (a string $x \in \Sigma^*$ which is not well-formed will be mapped to an arbitrary non-negative integer, say 0). The parameterization κ does not necessarily map an instance $x \in L$ to an integer already explicit in the input x. For example, we may parameterize VERTEX COVER by $k - \mathtt{lp}^*(G)$, that is, an instance (G, k) of VERTEX COVER is mapped to $k - \mathtt{lp}^*(G)$ by κ. Such a mapping κ is well-defined for a given instance (G, k).

Definition 1. *A parameterized problem (L, κ) is* fixed-parameter tractable *if there is an algorithm which, for every $x \in \Sigma^*$, decides if $x \in L$ in time $f(\kappa(x)) \cdot |x|^{O(1)}$.*

is a language $P \subseteq \Sigma^* \times \mathbb{N}$,

There are many different ways of parameterizing a decision problem, and some might make more sense than others. One may even consider a parameterization by multiple parameters.

A remark: instead of defining a parameterized problem as a pair of a decision problem (a set of strings) and a mapping κ, one can define a parameterized problem as a decision problem itself, i.e. a set of strings. In this perspective, we say that $P \subseteq \Sigma^* \times \mathbb{N}_0$ is a parameterized problem, and an instance of a parameterized problem is of the form (x, k). Note that this is a string; one can encode the second component k with a special unary alphabet $\mathbf{1}$ and (x, k) is a string over $\Sigma \cup \{\mathbf{1}\}$. Then the parameterized problem (L, κ) defined above can be rewritten as a subset of $\Sigma^* \times \mathbb{N}_0$ by taking

$$\{(x, \kappa(x)) : x \in L\}.$$

Often, the parameterized problem is defined as a decision problem in this way, and an instance of a parameterized problem is written as $(x, k) \in \Sigma^* \times \mathbb{N}_0$. In this case, the second component of the instance (x, k) is called the parameter. According to this version, an instance to VERTEX COVER parameterized by the solution size is of the form $((G, k), k)$.

2 $\mathcal{O}^*(2^k)$-time algorithm for VERTEX COVER

The procedure **VC** in Algorithm 1 takes as an input instance a graph G and a non-negative integer k, and outputs YES or NO correctly.

Algorithm 1 Algorithm for VERTEX COVER

```
1: procedure VC(G, k)
2:     if G has no edge then return YES.
3:     else if k = 0 then return NO.
4:     else                                          ▷ G has an edge and k > 0
5:         Pick an edge uv.
6:         return VC(G − u, k − 1) or VC(G − v, k − 1).
```

Below, we prove the correctness and the running time of the algorithm **VC**[5].

Lemma 1. *Given an input instance (G, k) of* VERTEX COVER*, the procedure* **VC** *correctly decides whether an input instance (G, k) is* YES *or* NO *in time $\mathcal{O}(2^k \cdot poly(n))$.*

Proof: First, let us show the correctness of **VC**; that is, **VC**(G,k) returns YES if and only if (G, k) is a YES-instance. We prove this by induction on k. Notice that when $k = 0$, the procedure **VC** always returns some output (and do not branch). Therefore, in any recursive call during the execution of the procedure, the parameter k remains non-negative. If $k = 0$ or $|E| = \emptyset$, then it is trivial to verify that Lines 5 and 3 respectively returns a correct output. Therefore, we consider the case when $|E| > 0$ and $k > 0$.

If (G, k) is a YES-instance, that is G has a vertex cover X of size at most k, then at least one of u and v in Line 5 is included in X. Observe that at least one of the instances $(G - u, k - 1)$ and $(G - v, k - 1)$ is a YES-instance because $X \setminus u$ (respectively $X \setminus v$) is a vertex cover of $G - u$ (respectively $G - v$). By induction hypothesis, **VC** returns a correct output upon each of $(G - u, k - 1)$ and $(G - v, k - 1)$. This means that, by the fact that one of them is a YES-instance, the returned output [6] at Line 6 will be YES and thus the output of **VC**(G, k) is correct.

Now suppose that (G, k) is a NO-instance. Then the instance $(G - u, k - 1)$ (likewise $(G - v, k - 1)$) is a NO-instance. Indeed, if $G - u$ has a vertex cover X' of size at most $k - 1$ then $X' \cup \{u\}$ is also a vertex cover of G and we have $|X' \cup \{u\}| \leq k$, a contradiction. Therefore, by induction hypothesis **VC** outputs NO to both instances called in Line 6. Therefore, Line 6 returns NO. This proves the correctness of the algorithm.

To see the running time, observe that the search tree depicting the relations of the recursive calls has depth at most k. This is because each instance created during the recursive calls have $k \geq 0$ and every branching decreases the parameter value by 1. Therefore, there are at most 2^k leaves and $\mathcal{O}(2^k)$ nodes in the search tree. As the algorithms spends $poly(n)$ steps at each node, the running time follows. □

[5] A detailed proof is given for this algorithm to illustrate what a correctness proof in full should be like. For other algorithms that we'll encounter later, not so much details might be delineated.

[6] Here, we construe 'or' as the boolean operation OR by equating YES to True and NO to False.

3 $\mathcal{O}^*((3k)^k)$-time algorithm for FEEDBACK VERTEX SET

Lemma 2. *For any feedback vertex set X of $G = (V, E)$, it holds that*

$$\sum_{v \in X} (d(v) - 1) \geq |E| - |V| + 1.$$

Proof: All the edges of G are either counted as part of the forest $G - X$, or incident with X. Now,

$$\begin{aligned} |E| &= \text{\# edges with both endpoints in } V - X + \text{\# edges incident with } X \\ &\leq (|V - X| - 1) + \sum_{v \in X} deg(v). \end{aligned}$$

Here, we use the fact that the number of edges in an n-vertex forest is at most $n - 1$. Note also that $\sum_{v \in X} deg(v)$ (over)counts the number of edges incident with X. This yields the claimed inequality. □

Lemma 3. *Let $G = (V, E)$ be a graph of degree at least three. Let $\{v_1, v_2, \ldots, v_n\}$ be the vertex set of G such that*

$$deg(v_1) \geq deg(v_2) \geq \cdots \geq deg(v_n).$$

Then for any feedback vertex set X, we have $X \cap \{v_i : i \leq 3k\} \neq \emptyset$.

Proof: Suppose not. Observe

$$\sum_{i \leq 3k} (deg(v_i) - 1) \geq 3 \sum_{v \in X} (deg(v) - 1) \geq 3\big(|E| - |V| + 1\big),$$

because each vertex of X has degree at most $deg(v_{3k})$. Also, due to $X \subseteq \{v_{3k+1}, \ldots, n\}$

$$\sum_{i > 3k} (deg(v_i) - 1) \geq \sum_{v \in X} (deg(v) - 1) \geq |E| - |V| + 1.$$

Summing the two equalities, we get

$$\sum_{i=1}^{n} (deg(v_i) - 1) = 2|E| - |V| \geq 4\big(|E| - |V| + 1\big),$$

and thus

$$3|V| > 2|E| = \sum_{v \in V} deg(v).$$

This contradicts the assumption that each v has degree at least three. □

4 $\mathcal{O}^*(4^{k - \mathtt{lp}^*(G)})$-time algorithm for VERTEX COVER

Here, we consider the problem VERTEX COVER parameterized above LP. An input instance and the question is the same as in VERTEX COVER, but the parameter under consideration is $k - \mathtt{lp}^*(G)$. Here k is the budget for vertex cover in the input, and $\mathtt{lp}^*(G)$ is the value of an optimal fractional solution to the following linear program LPVC for VERTEX COVER.

$$
\begin{aligned}
[\mathsf{LPVC}] \qquad \min & \sum_{u \in V(G)} x_u & \\
& x_u + x_v \geq 1 & \forall (u,v) \in E(G) \\
& x_u \geq 0 & \forall u \in V(G)
\end{aligned}
$$

Observe that if an optimal solution to the above LP is integral, then it corresponds to an optimal vertex cover. In general, an optimal solution x^* to LPVC is not necessarily integral.

The basis of this parameterization is the following fact:

$$\text{the size of an optimal vertex cover of } G \geq \mathtt{lp}^*(G).$$

Therefore, the parameter $k - \mathtt{lp}^*(G)$ can be assumed to be non-negative; otherwise, it holds that $k < \mathtt{lp}^*(G)$, which means that any vertex cover of G has size strictly larger than k. Especially G does not have a vertex cover of size at most k. We can correctly declare (G, k) as a NO-instance in this case.

Lemma 4. *If $k < \mathtt{lp}^*(G)$, then the input instance (G, k) is a* NO*-instance.*

Half-integrality of LP for VERTEX COVER

Now we establish the half-integrality of LPVC, that is, the property that there is always an optimal fractional solution x^* to LPVC such that $x^*_v \in \{0, \frac{1}{2}, 1\}$ for every $v \in V$.

Let x^* be an optimal fractional solution fo LPVC, which is not necessarily half-integral. We partition[7] $V(G)$ according to their values in x^*.

- $H_0 := \{u \in V(G) : x^*_u > 0,5\}$
- $C_0 := \{u \in V(G) : x^*_u < 0,5\}$
- $R_0 := \{u \in V(G) : x^*_u = 0.5\}$

It is known that LP can be solved in polynomial time, hence the partition (R_0, H_0, C_0) can be found in polynomial time. We shall see that x^* can be converted into a half-integral solution, i.e. $x_u \in \{0, 0.5, 1\}$ while maintaining the optimality, and thus showing that there exists a *half-integral* optimal solution to LP formulation of VERTEX COVER.

Observation 1. *Any edge incident with C_0 is incident with H_0. In particular, C_0 is an independent set of G.*

Lemma 5. *For every subset $H'_0 \subseteq H_0$, we have $|H'_0| \leq |N(H'_0) \cap C_0|$.*

Proof: Suppose not, i.e. there exists $H'_0 \subseteq H_0$ such that $|H'_0| > |N(H'_0) \cap C_0|$. Take $\epsilon = \min\{x^*_u - 0.5 : u \in H'_0\}$ and note that $\epsilon > 0$. Consider a solution x' defined as:

$$x'_u = \begin{cases} x^*_u - \epsilon & \text{if } u \in H'_0 \\ x^*_u + \epsilon & \text{if } u \in N(H'_0) \cap C_0 \\ x^*_u & \text{otherwise} \end{cases}$$

[7] H, C and R represent 'Head', 'Crown' and 'Rest'.

It is easy to verify that x' is a feasible LP solution. The objective value of x' equals

$$\sum_{u \in V(G)} x^*_u + \epsilon(-|H'_0| + |N(H'_0) \cap C_0)|) < \sum_{u \in V(G)} x^*_u.$$

This contradict the optimality of x^*, thus our claim follows. □

Lemma 6. *The following solution $\tilde{x}$ is optimal to* LPVC*:*

$$\tilde{x}_u = \begin{cases} 0.5 & \text{if } u \in R_0 \\ 1 & \text{if } u \in H_0 \\ 0 & \text{if } u \in C_0 \end{cases}$$

Proof: Let H'_0 be the set of vertices in H_0 with $x^*_u < 1$ and C'_0 be the set of vertices in C_0 with $x^*_u > 0$. Among all optimal solutions to LP yielding the partition (R_0, H_0, C_0), choose x^* so as to minimize the set $H'_0 \cup C'_0$. We shall show that $H'_0 \cup C'_0 = \emptyset$, which establishes $x^* = \tilde{x}$ and thus the statement.

For the sake of contradiction, $H'_0 \cup C'_0 \neq \emptyset$. Take $\delta = \min\{1 - x^*_u : u \in H'_0\} \cup \{x^*_u : u \in C'_0\}$ and note that $\delta > 0$. Consider a solution x' defined as:

$$x'_u = \begin{cases} x^*_u + \delta & \text{if } u \in H'_0 \\ x^*_u - \delta & \text{if } u \in C'_0 \\ x^*_u & \text{otherwise.} \end{cases}$$

As we chose minimum δ, it holds $x' \geq 0$. It is straightforward to verify that x' is feasible.

Now we argue that x' is an optimal LP solution. From Lemma 5, we know $|H'_0| \leq |N(H'_0) \cap C_0|$. Also observe that $N(H'_0) \cap C_0 \subseteq C'_0$ since otherwise, there would be an edge (u, v) with $u \in H'_0$ and $v \in C_0 \setminus C'_0$ such that $x^*_u + x^*_v < 1 + 0 = 1$, contradicting the feasibility of x^*. We derive $|H'_0| \leq |C'_0|$. Then, the objective value of x' equals

$$\sum_{u \in V(G)} x^*_u + \delta(|H'_0| - |C'_0|),$$

which does not exceed $\sum_{u \in V(G)} x^*_u$. Therefore, x' is an optimal solution such that $\{u \in H_0 : x'_u < 1\} \cup \{u \in C_0 : x'_u > 0\} \subsetneq H'_0 \cap C'_0$, a contradiction. This complete the proof. □

We remark that the proof of Lemma 6 is constructive: one can use the procedure in the proof to transform an arbitrary optimal solution to a half-integral solution. Fom now on we assume that x^* is a half-integral solution as in Lemma 6.

Preprocessing: how to get all-$\frac{1}{2}$ optimal solution

From the previous discussion, we know that a half-integral optimal solution x^* for VERTEX COVER can be obtained in polynomial time. We define the set $V_a(x^*)$ as the set of all vertices v such that $x^*_v = a$ for $a \in \{0, .5, 1\}$. When x^* is obvious from the context, we drop it. From the half-integrality of x^*, it is obvious that $(V_0, V_1, V_{\frac{1}{2}})$ forms a partition of V.

Lemma 7. *For the graph $G[V_{\frac{1}{2}}]$, all-$\frac{1}{2}$ is an optimal fractional solution.*

Proof: If all-$\frac{1}{2}$ is not an optimal fractional solution to LPVC of $G[V_{\frac{1}{2}}]$, consider an optimal fractional solution z^* of $G[V_{\frac{1}{2}}]$ and observe

$$\sum_{v \in V_{\frac{1}{2}}} z^*_v < \sum_{v \in V_{\frac{1}{2}}} x^*_v.$$

We can obtain a new feasible solution to LPVC of G by replacing the value of x^*_v by z^*_v for all $v \in V_{\frac{1}{2}}$ (and keep other values unchanged). It is not difficult to verify that the new fractional solution is feasible and has strictly smaller objective value than x^*, contradicting the optimality of x^*. □

Lemma 8. *There exists an optimal vertex cover containing V_1.*

Proof: Let S be an optimal vertex cover of G and let $S_{\frac{1}{2}} S_1, S_0$ be its intersections with $V_{\frac{1}{2}}, V_1$ and V_0. We take a new set

$$S' := (S \setminus S_0) \cup V_1,$$

i.e. the set obtained from S by removing all of C_0 and adding all of V_1. This is a vertex cover of G because any edge incident previously covered by S_0, thus incident with V_0 must be also incident with V_1, see Observation 1. We claim that S' is again an optimal vertex cover. To see this, it suffices to show that the vertices newly added are not more than those removed, that is, $|V_1 - S_1| \leq |S_0|$.

Observe that $N(V_1 - S_1) \cap C_0 \subseteq S_0$ since otherwise S_0 fails to cover all edges incident with $V_1 - S_1$. Applying Lemma 5 for $V_1 - S_1$, we have

$$|V_1 - S_1| \leq |N(V_1 - S_1) \cap V_0| \leq |S_0|.$$

That is, the new vertex cover S' containing V_1 is not larger than S, thus is an optimal vertex cover. □

Lemma 9. *The instance $I' = (G', k')$ is equivalent to $I = (G, k)$, where $G' = G[V_{\frac{1}{2}}]$ and $k' = k - |V_1|$. Furthermore, it holds that $k' - \mathtt{lp}^*(G[V_{\frac{1}{2}}]) = k - \mathtt{lp}^*(G)$.*

Proof: The proof of the first statement follows easily from Lemma 8. The second statement is derived from the following.

$$\begin{aligned}
\mathtt{lp}^*(G) &= \sum_{v \in V} x^*_v = \sum_{v \in V_0} x^*_v + \sum_{v \in V_1} x^*_v + \sum_{v \in V_{\frac{1}{2}}} x^*_v \\
&= \frac{1}{2} \cdot |V_{\frac{1}{2}}| + |V_1| \\
&= \mathtt{lp}^*(G[V_{\frac{1}{2}}]) + |V_1|,
\end{aligned}$$

where the last equality holds because of Lemma 7. □

Notice that Lemma 7 guarantees that all-$\frac{1}{2}$ is an optimal fractional solution to $G[V_{\frac{1}{2}}]$, but it does not guaranteed that it's the *unique* optimal solution. So how can we achieve the uniqueness? We do it *greedily*, namely, try each vertex v and fix its LP value to one and see if it leads to yet another optimal fractional solution. To be precise, here is the reduction rule. We write $|y^*|$ to denote $\sum_{v \in V} y^*_v$.

Reduction Rule 1. *If there is a vertex $v \in V$ and a half-integral solution y^* to LPVC such that $y^*_v = 1$ and $|y^*| = \mathtt{lp}^*(G)$, then define the new instance $I' = (G', k')$ with $G' = G[V_{\frac{1}{2}}(y^*)]$ and $k' = k - |V_1(y^*)|$.*

The soundness of Reduction Rule 1 is derived immediately from Lemma 9. It is clear that after applying Reduction Rule 1 exhaustively, all-$\frac{1}{2}$ is the unique optimal fractional solution to LPVC of the resulting graph. We just state this observation as the next statement.

Observation 2. *If (G, k) is irreducible with respect to Reduction Rule 1, then $x^*_v = \frac{1}{2}$ for all $v \in V$ is the unique fractional optimal solution to* LPVC.

Branching + Preprocessing with runtime analysis

Algorithm 2 Algorithm for VERTEX COVERabove LP

```
1: procedure VC-above-LP(G, k)
2:     Apply Reduction Rule 1 until it can no longer be applied.
3:                          ▷ Now, (G, k) has all-1/2 as the unique fractional optimal sol.
4:     if k < lp*(G) then return NO.                                   ▷ Lemma 4
5:     else if k ≥ 2 · lp*(G) then return YES.                         ▷ Lemma 10
6:     else                                  ▷ G has an edge and k − lp*(G) ≥ 0
7:         Pick an edge uv.
8:         return VC-above-LP(G − u, k − 1) or VC-above-LP(G − v, k − 1).
```

The algorithm for the problem VERTEX COVERabove LP is presented as **VC-above-LP**. The correctness of Line 4 is due to Lemma 4. To see the correctness of Line 5, we need the next lemma [8].

Lemma 10. *For any graph G, one can find a vertex cover Z of size at most $2 \cdot \mathtt{lp}^*(G)$ in polynomial time. In particular, an instance (G, k) to* VERTEX COVER *is a* YES*-instance if $k \geq 2 \cdot \mathtt{lp}^*(G)$.*

Proof: First compute an optimal half-integral solution x^* to LPVC of G and consider the usual partition $(V_0, V_1, V_{\frac{1}{2}})$. We take $Z = V_1 \cup V_{\frac{1}{2}}$. Since V_0 is an independent set, Z is a vertex cover. Now

$$|Z| = |V_1| + |V_{\frac{1}{2}}| \leq 2(|V_1| + 0.5 \cdot |V_{\frac{1}{2}}|) = 2 \cdot \mathtt{lp}^*(G).$$

The second statement follows immediately. □

The rest of the correctness proof is rather tedious. To analyze the running time of **VC-above-LP**, consider the measure

$$\mu(G, k) = k - \mathtt{lp}^*(G).$$

We want to argue that the measure decreases by at least $\frac{1}{2}$ in each branching of Line 8. The key observation is the following inequality (in fact, equality holds)

$$\mathtt{lp}^*(G - v) \geq \mathtt{lp}^*(G) - \frac{1}{2}.$$

Suppose this is not the case, that is, $\mathtt{lp}^*(G - v) \leq \mathtt{lp}^*(G) - 1$ and let y^* be an optimal fractional solution of $G - v$, i.e. $|y^*| = \mathtt{lp}^*(G - v)$. We can extend y^* to a fractional solution z^* of G by setting the value at v equal to 1. Then

$$|z^*| = |y^*| + 1 = \mathtt{lp}^*(G - v) + 1 \leq \mathtt{lp}^*(G),$$

[8] From $\mathtt{lp}^*(G) \leq$ the size of an optimal vertex cover, we deduce that the size of Z is at most twice the size of an optimal vertex cover. That is, Lemma 10 presents a 2-approximation algorithm for VERTEX COVER

which means that z^* is an optimal fractional solution of G which is not all-$\frac{1}{2}$. This contradicts that Reduction Rule 1 has been applied exhaustively in Line 2. Now

$$\mu(G-v,k-1) = (k-1) - \mathtt{lp}^*(G-v) \leq k-1-(\mathtt{lp}^*(G)-\frac{1}{2}) = \mu(G,k)-\frac{1}{2}.$$

By symmetry, the branching at the endpoint u also decreases the measure by $\frac{1}{2}$.

During the branching algorithm, the measure μ can drop down to -0.5 at most (in which case, we declare as NO-instance). Therefore, the length of a longest root-to-leaf path in the search tree is $2\mu(G,k) = 2(k-\mathtt{lp}^*(G))$. It follows that the number of leaves in the search tree is at most $4^{k-\mathtt{lp}^*(G)}$, and the running time is $\mathcal{O}^*(4^{k-\mathtt{lp}^*(G)})$.

Because of Line 5 (see Lemma 10 as well), we may assume that $k < 2 \cdot \mathtt{lp}^*(G)$ since otherwise, we can immediately declare the given instance as YES, in constant time. Therefore, $k - \mathtt{lp}^*(G) \leq \frac{k}{2}$. We summarize the result in the next statement.

Lemma 11. *The algorithm* **VC-above-LP** *solves, given an instance (G,k), the problem* VERTEX COVER*above LP in time $\mathcal{O}^*(4^{k-\mathtt{lp}^*(G)})$ as well as $\mathcal{O}^*(2^k)$-time* [9].

Problems List

VERTEX COVER
Instance: a graph $G=(V,E)$, a positive integer k.
Question: Does G have a vertex cover of size at most k, i.e. a set X of vertices such that $G-X$ is an independent set?

ODD CYCLE TRANSVERSAL
Instance: a graph $G=(V,E)$, a positive integer k.
Question: Does G have a vertex cover of size at most k, i.e. a set X of vertices such that $G-X$ is bipartite?

FEEDBACK VERTEX SET
Instance: a graph $G=(V,E)$, a positive integer k.
Question: Does G have a feedback vertex set (fvs) of size at most k, i.e. a set X of vertices such that $G-X$ is acyclic?

d-HITTING SET
Instance: a universe V, a family $\mathcal{E}$ of subsets of V each of which having size at most d, a positive integer k.
Question: Is there a hitting set of $\mathcal{E}$, i.e. a subset X of V such that $X \cap e \neq \emptyset$ for every $e \in \mathcal{E}$?

[9] Recall that we already have a simple branching algorithm for VERTEX COVER running in time $\mathcal{O}^*(2^k)$. Therefore, the second running time might seem unnecessary. However, almost identical algorithm work for problems such as Multiway Cut and others, which are generalizations of VERTEX COVER. For those problems, this LP-based branching approach is the only known way to achieve $\mathcal{O}^*(2^k)$-time algorithm.

k-PATH
Instance: a graph $G = (V, E)$, a positive integer k.
Question: Does G have a path on k vertices?