1 Kernelization Algorithms

In the previous lecture, we saw two FPT algorithms for vertex cover. In one of these, given an instance (G, k) we repeatedly removed vertices of degree $> k$ and isolated vertices to produce an instance (G', k') with at most $2k^2$ vertices that we could then solve via brute force.

This method, in which the input is mapped to a smaller instance whose size depends only on k is an algorithmic technique called **kernelization**, which we will soon see is closely related to FPT.

Definition 2.1. A **kernelization algorithm** for a language $L \subseteq \Sigma^* \times \mathbb{N}$ is a function $\phi : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ that maps an instance (x, k) of L to an instance (x', k') of L (called the **kernel**) such that the following hold:

(i) (x, k) is in L if and only if (x', k') is in L

(ii) the size of (x', k') is independent of the size of x: $|x'| + k' \leq g(k)$ for some computable $g : \mathbb{N} \to \mathbb{N}$

(iii) ϕ is computable in time $poly(|x|, k)$

Notice that the aforementioned vertex cover algorithm satisfies all three of these conditions.

It seems natural that problems with kerenelization algorithms are in FPT: solving the output of the kernelization algorithm via brute force will run in FPT time. However, what may be more surprising is that the reverse is also true: any problem in FPT also admits a kernelization algorithm.

Theorem 2.2. *A parameterized language L is in FPT if and only if it admits a kernelization algorithm.*

Proof.

($\Leftarrow$) Given a kernelization algorithm ϕ, we can show L is in FPT via the following algorithm for deciding L. Given an instance (x, k) in L, we can first compute $\phi(x, k)$ in time $poly(|x|, k)$, and then determine whether it's in L via brute force. This is within the desired $f(k)poly(|x|, k)$ bound, because the size of $\phi(x, k)$ is bounded by a function of k.

($\Rightarrow$)

Given an FPT algorithm A for L that runs in $f(k)n^c$ time for some f and c, we will construct the following kernelization algorithm:

On input $(|x|, k)$:

- run A for $(|x| + k)^{c+1}$ steps
- if it halts, return a trivial yes or no instance based on A's output.
- otherwise, output $(|x|, k)$ itself

It is clear that the output of this algorithm will be in L if and only if the input is in L, and the run-time is explicitly polynomial in $|x|$ and k, so to show that this is a kernelization algorithm, it suffices to show that the output size is bounded by a function of k.

This is clear in the first case, so let's look at the second case. If A didn't halt in $(|x|+k)^{c+1}$ steps, $(|x|+k)^{c+1} < f(k)n^c$. Since $(|x|+k)^{c+1} \geq n^{c+1}$, we have $n^{c+1} < f(k)n^c$, which means $n < f(k)$. Therefore, in this case, $|x|$ is bounded by $f(k)$, and therefore the output size is still bounded by a function of k.

□

2 LP Kernel for Vertex Cover

Now that we're equipped with the basic tools for working with kernelization, we'll return to the Vertex Cover problem and proceed with a case study of a specific kernelization algorithm.

Observe that the Vertex Cover problem on a graph $G = (V, E)$ can be expressed as an integer program in the following way:

Variables: $x_u, \forall u \in V$

Constraints:

- $x_u \in 0, 1\ \forall u \in V$
- $x_u + x_v \geq 1\ \forall (u, v) \in E$

Although solving integer programs is also a NP-hard problem, this formulation is useful because linear programs can be solved in polynomial time. Relaxing the constraints in the above integer program (instead of restricting the x_u's to be 0 or 1, allow $0 \leq x_u \leq 1$) produces a linear program that yields a classic polynomial time 2-approximation algorithm for vertex cover: get the optimal solution to the LP, then place all $x_u \geq \frac{1}{2}$ in your vertex cover.

We will now see that this linear program also provides a kernelization algorithm: we can solve the LP and then choose all vertices $x_u \geq \frac{1}{2}$ to be in our kernel.

To argue that this algorithm is in fact a kernelization, we first need an additional tool, the following theorem.

Theorem 2.3. *(Nemhauser/Trotter [NL75]) The Vertex Cover L.P. always has an half-integral optimal solution. Equivalently, there always exists an optimal solution such that for all u, $x_u \in \{0, \frac{1}{2}, 1\}$.*

Proof. Let Z be an optimal LP solution. Our approach will be to iteratively modify the solution to give at least one more vertex a value in $\{0, \frac{1}{2}, 1\}$ while preserving optimality.

Consider all vertices in Z with values not in $\{0, \frac{1}{2}, 1\}$. Let ε be the minimum over all distances from vertices in this set to the closest value in $\{0, \frac{1}{2}, 1\}$.

$$\varepsilon = \min_{u | z_u \notin \{0, \frac{1}{2}, 1\}} \left(z_u, 1 - z_u, \left| \frac{1}{2} - z_u \right| \right)$$

We will now use epsilon to define two new solutions, Z^+ and Z^-. On a high level, Z^+ will shift the incorrectly valued vertices towards $\frac{1}{2}$, while Z^- will shift them towards 0 and 1.

$$Z^+ = \left\{ z_u^+ \;\middle|\; z_u^+ = \begin{cases} z_u + \varepsilon & z_u < \frac{1}{2} \\ z_u - \varepsilon & z_u > \frac{1}{2} \end{cases} \right\}$$

$$Z^- = \left\{ z_u^- \;\middle|\; z_u^- = \begin{cases} z_u - \varepsilon & z_u < \frac{1}{2} \\ z_u + \varepsilon & z_u > \frac{1}{2} \end{cases} \right\}$$

Claim 2.4. *Both Z^+ and Z^- are optimal LP solutions.*

Proof. First, we need to show that Z^+ and Z^- are both valid solutions. To do this, we will show that for all edges (u, v), $z_u^+ + z_v^+ \geq 1$ and $z_u^- + +z_v^- \geq 1$

Consider an arbitrary edge (u, v). We will case on the values of z_u and z_v:

- Case 1: $z_u, z_v \in (\frac{1}{2}, 1)$
  Because ε was the minimum distance to 0, 1, or $\frac{1}{2}$ over all vertices, $z_u - \varepsilon \geq \frac{1}{2}$. Thus, $z_u^+, z_v^+, z_u^-, z_v^- \geq \frac{1}{2}$. Thus, in both Z^- and Z^+, both endpoints of this edge are at least $\frac{1}{2}$, so this constraint is satisfied.

- Case 2: $z_u < \frac{1}{2}$ and $z_v > \frac{1}{2}$
  In both Z^+ and Z^-, one of the vertices is increased by ε, and the other is decreased by the same amount. Therefore, the sum remains the same as it was in Z, and so must still be at least 1.

- Case 3: $z_u, z_v \in (0, \frac{1}{2})$
  This case is impossible: z_u and z_v cannot sum to 1 and we know Z was a valid solution.

Now that we know Z^+ and Z^- are solutions, we must show that they are optimal.

Observe that the value of the objective function of Z is the average of the objective functions of $Z+$ and Z^-:

$$\sum z_u = \frac{\sum z_u^+ + \sum z_u^-}{2}$$

If either Z^+ or Z^- is suboptimal, it must have a lower objective function value than Z. However, this means the other must have a higher objective function value than Z. This is impossible, since it contradicts optimality of Z, so Z^+ and Z^- must also be optimal. □

Therefore, we can find a half-integral solution via picking either Z^+ or Z^-, whichever decreases the number of incorrectly valued vertices, and then repeating the process using this new optimal solution. Because the number of incorrect vertices decreases by at least 1 each time, this process must terminate, producing the desired half-integral solution.

□

Now, we return to the kernelization algorithm.

Given an input (G, k), $G = (V, E)$, we first solve the LP, and then convert our optimal solution into a half-integral one as above.

Now, we can separate the vertices by their value in the optimal solution: let $V_0 = \{u | x_u = 0\}$, $V_1 = \{u | x_u = 1\}$, $V_{1/2} = \{u | x_u = \frac{1}{2}\}$.

We will output $(G[V_{1/2}], k - |V_1|)$ as our kernel, where $G[V_{1/2}]$ is the subgraph induced by $V_{1/2}$. Intuitively, we fix all vertices in V_1 to be in the vertex cover, all the vertices in V_0 to be outside the vertex cover, and we check to see whether we can cover the remaining edges with the correct number of vertices from $V_{1/2}$.

Observe that $|V_{1/2}| \leq 2k$ ($\sum x_u$ is at most k, because otherwise we know immediately that there is no k vertex cover, and can output a trivial no instance, so no more than $2k$ vertices can have value $\frac{1}{2}$). Therefore, the size of $G[V_{1/2}]$ depends only on k, and thus our kernel satisfies the size property.

(Note that while $G[V_{1/2}]$ has $O(k)$ vertices, it may have $O(k^2)$ edges. In fact, we can prove that no kernel with a number of edges subquadratic in k can exist under standard complexity assumptions. [FR08])

Observe additionally that this kernel was produced in polynomial time: LPs can be solved in polytime.

Therefore, it remains only to justify correctness.

We need to show that G has a vertex cover of size k if and only if $G[V_{1/2}]$ has a vertex cover of size $k - |V_1|$. To do this, we need the following lemma.

Lemma 2.5. *Given an LP solution with V_1, V_0, and $V_{1/2}$ as defined above, there exists an optimal vertex cover $S \subseteq V$ of G such that $V_1 \subseteq S$ and $S \cap V_0 = \emptyset$.*

Proof. Take an arbitrary optimal vertex cover S of G.

Consider $S' = (S \cup V_1) \setminus (S \cap V_0)$, the set resulting from adding V_1 to S and removing any elements of V_0.

S' is still a vertex cover: all edges with an endpoint in V_0 must have had their other endpoint in V_1, so any edges that would have been left uncovered by removing V_0 must have been covered by adding V_1.

We want to show that S' is still an optimal vertex cover: $|S'| \leq |S|$. To show this, we will show that we removed at least as many vertices from S as we added in.

Claim 2.6. $|S \cap V_0| \geq |V_1 \setminus S|$

Proof. Construct a new *LP* solution where all vertices in $S \cap V_0$ and $V_1 \setminus S$ are set to $\frac{1}{2}$. This is still a valid solution: the only concern is that we are decreasing the values of $V_1 \setminus S$, so if there was an edge to $V_0 \setminus S$, its other endpoint would not be increased to compensate for this decrease. However, we know no such edges can exist, because S is a vertex cover, and such an edge would have neither endpoint in S. This new solution's objective function differs from that of the original by $\frac{1}{2}|S \cap V_0| - \frac{1}{2}|V_1 \setminus S|$. Since the original solution was optimal, this difference cannot be negative. Thus, we have

$$\frac{1}{2}|S \cap V_0| - \frac{1}{2}|V_1 \setminus S| \geq 0$$

$$|S \cap V_0| \geq |V_1 \setminus S|$$

□

This finishes the proof of the lemma.

□

Now, we can finally conclude by arguing correctness of our kernel.

First, suppose S' is a vertex cover of $G[V_{1/2}]$ of size $k - |V_1|$. We claim $S' \cup V_1$ is a vertex cover of size k in G. The only potentially uncovered edges would be from V_0 to $V_{1/2}$; however, no such edge can exist, because it would violate the LP's constraint. Thus, we have a valid vertex cover.

Now, suppose S is a vertex cover of G of size k. By the Lemma 2.5 above, assume S contains V_1 and does not contain any elements of V_0. Then, $S \cap V_{1/2}$ is the desired $k - |V_1|$ vertex cover of $G[V_{1/2}]$.

3 d-Hitting Set

We'll now briefly introduce a related problem, to be further discussed in future lectures. **d-Hitting set** is a generalization of the vertex cover problem to d-hypergraphs.

More precisely, given an instance $H = (V, F)$, a d-hypergraph (meaning F contains subsets of vertices, each of size at most d), we ask the following question: *is there a hitting set of size k, i.e. does there a exist a subset of the vertices of size at k that nontrivially intersects with every S in F?*

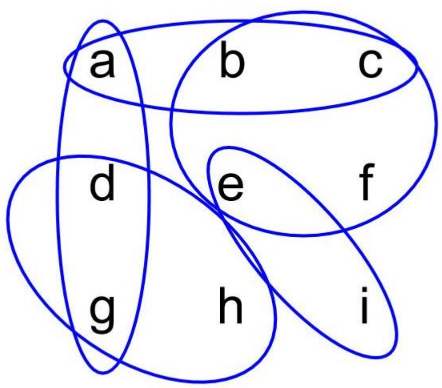

Figure 2.1: An instance of d-hitting set for $d = 4$. $\{a, e, g\}$ is a hitting set

Exercise. Show that d-Hitting set is in FPT by providing a $d^k poly(n)$ algorithm. (Hint: recall the decision tree algorithm for vertex cover discussed in Lecture #1)

References

[FR08] L. Fortnow and Santhanam R. Infeasibility of instance compression and succinct PCPs for NP. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 133–142, 2008. 2

[NL75] G.L. Nemhauser and Trotter L.E. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975. 2.3