Kernelization++
- Turing Kernelization
- Lossy Kernelization

Lecture #14

Roohani Sharma
February 08, 2021
Kernelization++
- Turing Kernelization
- Lossy Kernelization

Lecture #14
Roohani Sharma
February 08, 2021
Kernelization
- Efficient pre-processing with guarantees for parameterized problems

Runs in polynomial time

\[ \Pi \text{ admits a kernel of size } g(k). \]

If \( g(k) \) is a polynomial/exponential function, then \( \Pi \) admits a polynomial/exponential kernel.
Let $L$ be a parameterized problem.

**OR-composition for $L$**

**Input**: $(x_1, k), \ldots, (x_t, k)$ such that $x_i \in \Sigma^*$ and $k$ is a non-negative integer.

**Output**: $(y, k^*)$ such that

- $(y, k^*) \in L$ if and only if $(x_i, k) \in L$ for some $i$.
- $k^* = \text{poly}(k)$.

**Time**: polynomial in the input, that is $\text{poly}(\sum_{i=1}^{t} |x_i| + k)$. 

Instances of language $L$

- $x_1$
- $x_2$
- $x_3$
- $x_4$
- $\ldots$
- $x_{t-1}$
- $x_t$

Instance of language $L$

- $y$

- $\text{poly}(k)$
Eg. k-path, Steiner Tree, Max leaf Subgraph (Exercise #03) do not admit poly kernel.

**Max Leaf Subtree**

**Input:** A graph $G$, a positive integer $k$

**Parameter:** $k$

**Question:** Does $G$ have a subtree with at least $k$ leaves?
Eg. k-path, Steiner Tree, Max leaf Subgraph (Exercise #03) do not admit poly kernel.

**Max Leaf Subtree**

**Input:** A graph G, a positive integer k  
**Parameter:** k  
**Question:** Does G have a subtree with at least k leaves?

Has a subtree with 4 leaves, no subtree with 5 leaves
Eg. k-path, Steiner Tree, Max leaf Subgraph (Exercise #03) do not admit poly kernel.

Max Leaf Subtree

**Input:** A graph G, a positive integer k  
**Parameter:** k  
**Question:** Does G have a subtree with at least k leaves?

Has a subtree with 4 leaves, no subtree with 5 leaves  
Has a subtree with 6 leaves
Eg. k-path, Steiner Tree, Max Leaf Subgraph (Exercise #03) do not admit poly kernel.

Max Leaf Subtree

**Input**: A graph $G$, a positive integer $k$

**Parameter**: $k$

**Question**: Does $G$ have a subtree with at least $k$ leaves?

Has a subtree with 4 leaves, no subtree with 5 leaves

Has a subtree with 6 leaves

Observe: It is not a coincidence that each solution subtree is a spanning
What next?
What next?

• In essence, a kernelization algorithm returns the “hard part” of the input with a guarantee that the hard part is small.
What next?

• In essence, a kernelization algorithm returns the “hard part” of the input with a guarantee that the hard part is small.

• There are many NP-hard problems which do not admit polynomial kernels, and for which the hardness do lie in a small part of the input, but may not be in one small part but in multiple small parts.
What next?

• In essence, a kernelization algorithm returns the "hard part" of the input with a guarantee that the hard part is small.

• There are many NP-hard problems which do not admit polynomial kernels, and for which the hardness do lie in a small part of the input, but may not be in one small part but in multiple small parts.

• For example, consider the Maximum Clique problem parameterized by the vertex cover size of the input. This problem is denoted by $\text{CLIQUE}/\text{VC}$. One can show that this does not admit a polynomial kernel using OR-composition.
What next?

• In essence, a kernelization algorithm returns the “hard part” of the input with a guarantee that the hard part is small.

• There are many NP-hard problems which do not admit polynomial kernels, and for which the hardness do lie in a small part of the input, but may not be in one small part but in multiple small parts.

• For example, consider the Maximum Clique problem parameterized by the vertex cover size of the input. This problem is denoted by CLIQUE/VC. One can show that this does not admit a polynomial kernel using OR-composition.

• **CLIQUE/VC**: Given a graph $G$ and a vertex cover $X$ of $G$ of size at most $k$, find the size of a maximum clique in $G$. The parameter is $k$. 
What next?

• In essence, a kernelization algorithm returns the “hard part” of the input with a guarantee that the hard part is small.

• There are many NP-hard problems which do not admit polynomial kernels, and for which the hardness do lie in a small part of the input, but may not be in one small part but in multiple small parts.

• For example, consider the Maximum Clique problem parameterized by the vertex cover size of the input. This problem is denoted by CLIQUE/VC. One can show that this does not admit a polynomial kernel using OR-composition.

• **CLIQUE/VC**: Given a graph $G$ and a vertex cover $X$ of $G$ of size at most $k$, find the size of a maximum clique in $G$. The parameter is $k$.

$$|X| \leq k$$

$G - X$ is an independent set.
What next?

• In essence, a kernelization algorithm returns the "hard part" of the input with a guarantee that the hard part is small.

• There are many NP-hard problems which do not admit polynomial kernels, and for which the hardness do lie in a small part of the input, but may not be in one small part but in multiple small parts.

• For example, consider the Maximum Clique problem parameterized by the vertex cover size of the input. This problem is denoted by \textsc{Clique/VC}. One can show that this does not admit a polynomial kernel using OR-composition.

• \textsc{Clique/VC}: Given a graph $G$ and a vertex cover $X$ of $G$ of size at most $k$, find the size of a maximum clique in $G$. The parameter is $k$.

\begin{center}
\begin{tikzpicture}
  \node at (0,0) {$|X| \leq k$};
\end{tikzpicture}
\end{center}

\begin{center}
\begin{tikzpicture}
  \node at (0,0) {G- $X$ is an independent set. Any clique uses at most 1 vertex of G-$X$.};
\end{tikzpicture}
\end{center}
Turing Kernelization

Definition (Turing Kernel)

Let $Q$ be a parameterized problem, and let $f : \mathbb{N} \rightarrow \mathbb{N}$ be a computable function. A Turing Kernel for $Q$ of size $f$ is an algorithm that can decide if an instance of the problem is a YES instance in polynomial time, given access to an Oracle that solves instance of size $f(k)$ in unit time.
Turing Kernelization

Definition (Turing Kernel)

Let $Q$ be a parameterized problem, and let $f : \mathbb{N} \rightarrow \mathbb{N}$ be a computable function. A Turing Kernel for $Q$ of size $f$ is an algorithm that can decide if an instance of the problem is a YES instance in polynomial time, given access to an Oracle that solves instance of size $f(k)$ in unit time.
Turing Kernelization

Definition (Turing Kernel)
Let $Q$ be a parameterized problem, and let $f : \mathbb{N} \rightarrow \mathbb{N}$ be a computable function. A Turing Kernel for $Q$ of size $f$ is an algorithm that can decide if an instance of the problem is a YES instance in polynomial time, given access to an Oracle that solves instance of size $f(k)$ in unit time.

- We say $Q$ has a polynomial Turing kernel if $f(k)$ is a polynomial function.
Turing Kernelization

Definition (Turing Kernel)
Let $Q$ be a parameterized problem, and let $f : \mathbb{N} \rightarrow \mathbb{N}$ be a computable function. A Turing Kernel for $Q$ of size $f$ is an algorithm that can decide if an instance of the problem is a YES instance in polynomial time, given access to an Oracle that solves instance of size $f(k)$ in unit time.

- We say $Q$ has a polynomial Turing kernel if $f(k)$ is a polynomial function.
- For CLIQUE/VC, we produced $O(n)$ instances, each of size $k+1$, such that each of them can be solved independently so give an output of the input instance.
Turing Kernelization

Definition (Turing Kernel)

Let $Q$ be a parameterized problem, and let $f : \mathbb{N} \rightarrow \mathbb{N}$ be a computable function. A **Turing Kernel** for $Q$ of size $f$ is an algorithm that can decide if an instance of the problem is a YES instance in polynomial time, given access to an Oracle that solves instance of size $f(k)$ in unit time.

- We say $Q$ has a **polynomial Turing kernel** if $f(k)$ is a polynomial function.
- For **Clique/VC**, we produced $O(n)$ instances, each of size $k+1$, such that each of them can be solved **independently** so give an output of the input instance.
- Generally speaking, one can produce instances such that the $i$-th instance depends on the Oracle’s answer to the previous $(i-1)$ instances. Such kind of Turing kernels are known for **k-Path** on certain graph classes.
Max leaf subgraph (MLS) do not admit poly kernel (Exercise #03). MLS OR-composes to itself.
**Max Leaf Subgraph (MLS)** do not admit poly kernel (Exercise #03). MLS OR-composes to itself.
**Max Leaf Subgraph (MLS)** do not admit poly kernel (Exercise #03). MLS OR-composes to itself.

The reduction essentially shows that **MLS** do not admit a polynomial kernel on disconnected graphs with a “lot” of connected components.
**Max leaf Subgraph (MLS)** do not admit poly kernel (Exercise #03). MLS OR-composes to itself.

The reduction essentially shows that **MLS** do not admit a polynomial kernel on **disconnected graphs** with a “lot” of connected components.

What happens to **MLS** on connected graphs?
**Max leaf Subgraph (MLS)** do not admit poly kernel (Exercise #03).

MLS OR-composes to itself.

The reduction essentially shows that **MLS** do not admit a polynomial kernel on disconnected graphs with a "lot" of connected components.

What happens to **MLS** on connected graphs?

**MLS** on connected graphs admit a polynomial kernel!
**Max Leaf Subgraph (MLS)** do not admit poly kernel (Exercise #03). **MLS** OR-composes to itself.

The reduction essentially shows that **MLS** do not admit a polynomial kernel on disconnected graphs with a “lot” of connected components.

What happens to **MLS** on connected graphs?

**MLS** on connected graphs admit a polynomial kernel!

**MLS** admits a polynomial Turing kernel!
MLS on connected graphs admit a polynomial kernel - Proof

Basic reduction rules:
If there exists a vertex \( v \) such that \(|N(v)| \geq k\), then it is a Yes-instance.
**MLS** on connected graphs admit a polynomial kernel - Proof

**Basic reduction rules:**
If there exists a vertex $v$ such that $|N(v)| \geq k$, then it is a **Yes**-instance.

If there exists a vertex $v$ such that $|N^2(v)| \geq k$, then it is a **Yes**-instance.
**MLS** on connected graphs admit a polynomial kernel - Proof

**Basic reduction rules:**

If there exists a vertex $v$ such that $|N(v)| \geq k$, then it is a *Yes*-instance.

If there exists a vertex $v$ such that $|N^2(v)| \geq k$, then it is a *Yes*-instance.

If there exists a vertex $v$ such that $|N^3(v)| \geq k$, then it is a *Yes*-instance.
**MLS** on connected graphs admit a polynomial kernel - Proof

**Basic reduction rules:**

If there exists a vertex $v$ such that $|N(v)| \geq k$, then it is a Yes-instance.

If there exists a vertex $v$ such that $|N^2(v)| \geq k$, then it is a Yes-instance.

If there exists a vertex $v$ such that $|N^3(v)| \geq k$, then it is a Yes-instance.

Therefore, we know that for each vertex $v$, $N^d(v) < k$ for each $d$. 

---

**Proof:**

We start by considering the given graph $G$ and its properties. The basic reduction rules indicate that if a vertex $v$ in the graph satisfies certain conditions on its neighborhood, the problem instance is a Yes-instance.

1. **Case $|N(v)| \geq k$:**
   - If the degree of a vertex $v$ exceeds $k$, it means that $v$ has at least $k$ neighbors. This suggests that $v$ is connected to a significant portion of the graph, making it a potential core of a large clique or a region of high interest.
   - Therefore, for each such vertex $v$, we can conclude $N^d(v) < k$ for each $d$.

2. **Case $|N^2(v)| \geq k$:**
   - This case is analogous to the first but considers the degree of a vertex's neighbors. If a vertex's neighbors have a combined degree of at least $k$, it implies a dense subgraph around $v$.
   - For each such vertex $v$, $N^d(v) < k$ holds.

3. **Case $|N^3(v)| \geq k$:**
   - Similar to the above cases, this indicates a high connectivity or density of a vertex's neighbor's neighbors.
   - Thus, $N^d(v) < k$ for each $d$.

By applying these basic reduction rules, we can systematically reduce the size of the problem instance while preserving its essential characteristics. This reduction process is crucial for establishing the polynomial kernel of the problem. The proof continues by demonstrating how these rules can be used to efficiently simplify the input graph, thereby proving the existence of a polynomial kernel for MLS on connected graphs.
**MLS** on connected graphs admit a polynomial kernel - Proof

**Basic reduction rules:**
If there exists a vertex \( v \) such that \( |N(v)| \geq k \), then it is a Yes-instance.

If there exists a vertex \( v \) such that \( |N^2(v)| \geq k \), then it is a Yes-instance.

If there exists a vertex \( v \) such that \( |N^3(v)| \geq k \), then it is a Yes-instance.

Therefore, we know that for each vertex \( v \), \( N^d(v) < k \) for each \( d \).

**Reduction rule for long degree-2 paths:**
If there exists a path \( v_1-v_2-v_3 \) such that degree of each \( v_i \) is exactly 2 in \( G \), then contract the edge \( v_1-v_2 \).
**Greedy:** Let us try to greedily build a solution and see where we fail.
MLS on connected graphs admit a polynomial kernel - Proof

**Greedy:** Let us try to greedily build a solution and see where we fail.
**MLS** on connected graphs admit a polynomial kernel - Proof

**Greedy:** Let us try to greedily build a solution and see where we fail.

**Procedure:** Let us try to construct vertex disjoint stars in $G$.
Greedy: Let us try to greedily build a solution and see where we fail.

Procedure: Let us try to construct vertex disjoint stars in G.

- Let $v$ be a vertex of degree at least 3.
- Let $S_v$ be a star with $v$ and its neighbours in (the original graph $G$).
- Remove $N^2(v)$ from $G$ and repeat (as long as there is a vertex of degree at least 3 in the resulting graph).
**Greedy:** Let us try to greedily build a solution and see where we fail.

**Procedure:** Let us try to construct vertex disjoint stars in $G$.

- Let $v$ be a vertex of degree at least 3.
- Let $S_v$ be a star with $v$ and its neighbours in (the original graph $G$).
- Remove $N^2(v)$ from $G$ and repeat (as long as there is a vertex of degree at least 3 in the resulting graph).

**Reduction Rule:** Suppose the above procedure runs for at least $k$ steps, then report Yes-instance.

\[ r \geq k \]
**Greedy:** Let us try to greedily build a solution and see where we fail.

**Procedure:** Let us try to construct vertex disjoint stars in $G$.

1. Let $v$ be a vertex of degree at least 3.
2. Let $S_v$ be a star with $v$ and its neighbours in (the original graph $G$).
3. Remove $N^2(v)$ from $G$ and repeat (as long as there is a vertex of degree at least 3 in the resulting graph).

**Reduction Rule:** Suppose the above procedure runs for at least $k$ steps, then report Yes-instance.

$$r \geq k$$
**MLS** on connected graphs admit a polynomial kernel - Proof

**Greedy:** Let us try to greedily build a solution and see where we fail.

**Procedure:** Let us try to construct vertex disjoint stars in $G$.

- Let $v$ be a vertex of degree at least 3.
- Let $S_v$ be a star with $v$ and its neighbours in (the original graph $G$).
- Remove $N^2(v)$ from $G$ and repeat (as long as there is a vertex of degree at least 3 in the resulting graph).

**Reduction Rule:** Suppose the above procedure runs for at least $k$ steps, then report Yes-instance.

$$r \geq k$$
**Greedy:** Let us try to greedily build a solution and see where we fail.

**Procedure:** Let us try to construct vertex disjoint stars in $G$.

Let $v$ be a vertex of degree at least 3.
Let $S_v$ be a star with $v$ and its neighbours in (the original graph $G$).
Remove $N^2(v)$ from $G$ and repeat (as long as there is a vertex of degree at least 3 in the resulting graph).

**Reduction Rule:** Suppose the above procedure runs for at least $k$ steps, then report Yes-instance.

$r \geq k$
**Greedy:** Let us try to greedily build a solution and see where we fail.

**Procedure:** Let us try to construct vertex disjoint stars in $G$.

- Let $v$ be a vertex of degree at least 3.
- Let $S_v$ be a star with $v$ and its neighbours in (the original graph $G$).
- Remove $N^2(v)$ from $G$ and repeat (as long as there is a vertex of degree at least 3 in the resulting graph).

**Reduction Rule:** Suppose the above procedure runs for at least $k$ steps, then report Yes-instance.

$$r \geq k$$
**MLS** on connected graphs admit a polynomial kernel - Proof

**Greedy:** Let us try to greedily build a solution and see where we fail.

**Procedure:** Let us try to construct vertex disjoint stars in G.

Let \( v \) be a vertex of degree at least 3.
Let \( S_v \) be a star with \( v \) and its neighbours in (the original graph \( G \)).
Remove \( N^2(v) \) from \( G \) and repeat (as long as there is a vertex of degree at least 3 in the resulting graph).

**Reduction Rule:** Suppose the above procedure runs for at least \( k \) steps, then report Yes-instance.

\[
r \geq k
\]
**MLS on connected graphs admit a polynomial kernel - Proof**

**Greedy:** Let us try to greedily build a solution and see where we fail.

**Procedure:** Let us try to construct vertex disjoint stars in G.

Let \( v \) be a vertex of degree at least 3.
Let \( S_v \) be a star with \( v \) and its neighbours in (the original graph \( G \)).
Remove \( N^2(v) \) from \( G \) and repeat (as long as there is a vertex of degree at least 3 in the resulting graph).

**Reduction Rule:** Suppose the above procedure runs for at least \( k \) steps, then report Yes-instance.

\[ r \geq k \]
**Greedy:** Let us try to greedily build a solution and see where we fail.

**Procedure:** Let us try to construct vertex disjoint stars in $G$.

- Let $v$ be a vertex of degree at least 3.
- Let $S_v$ be a star with $v$ and its neighbours in (the original graph $G$).
- Remove $N^2(v)$ from $G$ and repeat (as long as there is a vertex of degree at least 3 in the resulting graph).

**Reduction Rule:** Suppose the above procedure runs for at least $k$ steps, then report Yes-instance.

$r \geq k$
MLS on connected graphs admit a polynomial kernel - Proof

Greedy: Let us try to greedily build a solution and see where we fail.

Procedure: Let us try to construct vertex disjoint stars in $G$.

1. Let $v$ be a vertex of degree at least 3.
2. Let $S_v$ be a star with $v$ and its neighbours in (the original graph $G$).
3. Remove $N^2(v)$ from $G$ and repeat (as long as there is a vertex of degree at least 3 in the resulting graph).

Reduction Rule: Suppose the above procedure runs for at least $k$ steps, then report Yes-instance.

Claim: The stars in the red boxes are disjoint.
**Greedy:** Let us try to greedily build a solution and see where we fail.

**Procedure:** Let us try to construct vertex disjoint stars in $G$.

Let $v$ be a vertex of degree at least 3.

Let $S_v$ be a star with $v$ and its neighbours in (the original graph $G$).

Remove $N^2(v)$ from $G$ and repeat (as long as there is a vertex of degree at least 3 in the resulting graph).

**Reduction Rule:** Suppose the above procedure runs for at least $k$ steps, then report Yes-instance.

**Claim:** The stars in the red boxes are disjoint.

**Constructing a subtree from these stars (with at least $k$ leaves):**

Join the red stars by adding arbitrary paths between the $v_i$ vertices.

The resulting connected graph has at least $r$ leaves.
**Procedure:** Let us try to construct vertex disjoint stars in $G$.

Let $v$ be a vertex of degree at least 3.
Let $S_v$ be a star with $v$ and its neighbours in (the original graph $G$).
Remove $N^2(v)$ from $G$ and repeat (as long as there is a vertex of degree at least 3 in the resulting graph).

Suppose the above procedure runs for $r < k$ steps.

$$|X| = O(k^2)$$
Procedure: Let us try to construct vertex disjoint stars in G.

Let $v$ be a vertex of degree at least 3.
Let $S_v$ be a star with $v$ and its neighbours in (the original graph $G$).
Remove $N^2(v)$ from $G$ and repeat (as long as there is a vertex of degree at least 3 in the resulting graph).

Suppose the above procedure runs for $r < k$ steps.

$|X| = O(k^2)$

Every vertex of $G-X$ has degree at most 2.
$G-X$ is a disjoint union of paths and cycles.
**Procedure:** Let us try to construct vertex disjoint stars in $G$.

Let $v$ be a vertex of degree at least 3. Let $S_v$ be a star with $v$ and its neighbours in (the original graph $G$). Remove $N^2(v)$ from $G$ and repeat (as long as there is a vertex of degree at least 3 in the resulting graph).

Suppose the above procedure runs for $r < k$ steps.

Every vertex of $G-X$ has degree at most 2. $G-X$ is a disjoint union of paths and cycles. The green vertices are neighbours of $X$. $|N(X)| = O(k^2)$
**Proof:** Let us try to construct vertex disjoint stars in $G$.

Let $v$ be a vertex of degree at least 3.
Let $S_v$ be a star with $v$ and its neighbours in (the original graph $G$).
Remove $N^2(v)$ from $G$ and repeat (as long as there is a vertex of degree at least 3 in the resulting graph).

Suppose the above procedure runs for $r < k$ steps.

$\left|X\right| = O(k^2)$

Every vertex of $G-X$ has degree at most 2.
$G-X$ is a disjoint union of paths and cycles.
The green vertices are neighbours of $X$.
$|N(X)| = O(k^2)$
The black vertices between two consecutive green vertices are degree 2 vertices in the entire graph.
**Procedure:** Let us try to construct vertex disjoint stars in $G$.

Let $v$ be a vertex of degree at least 3.
Let $S_v$ be a star with $v$ and its neighbours in (the original graph $G$).
Remove $N^2(v)$ from $G$ and repeat (as long as there is a vertex of degree at least 3 in the resulting graph).

Suppose the above procedure runs for $r < k$ steps.

$|X| = O(k^2)$

Every vertex of $G-X$ has degree at most 2.
$G-X$ is a disjoint union of paths and cycles.
The **green vertices** are neighbours of $X$.
$|N(X)| = O(k^2)$
The **black vertices** between two consecutive **green vertices** are degree 2 vertices in the entire graph.
Therefore, $|V(G) \setminus X| \leq (|N(X)| + 1) = O(k^2)$
Lower bound machinery for Turing kernels?

• How to show that a problem does not exhibit any Turing kernel?

• So far, no machinery exists that allows one to prove such statements.

• Rather, we developed some hardness theory based on conjectures like,

**Connected Vertex Cover** does not admit a Turing kernel, or

**Steiner Tree** does not admit a Turing kernel.