# Parameterization: Kernelization

AiC FME, UPC

Fall 2023

# Kernelization

- Kernelization is a technique to obtain FPT algorithms for a parameterized problem $(L, \kappa)$.

# Kernelization

- Kernelization is a technique to obtain FPT algorithms for a parameterized problem $(L, \kappa)$.
- Based in auto-reductions

# Kernelization

- Kernelization is a technique to obtain FPT algorithms for a parameterized problem $(L, \kappa)$.
- Based in auto-reductions
- We look for a polynomial time algorithm that transforms an instance $x$ in another instance $x'$ of the problem (the kernel). So that

# Kernelization

- Kernelization is a technique to obtain FPT algorithms for a parameterized problem $(L, \kappa)$.
- Based in auto-reductions
- We look for a polynomial time algorithm that transforms an instance $x$ in another instance $x'$ of the problem (the kernel). So that
  - $x'$ is a yes instance iff $x$ is a yes instance.

# Kernelization

- Kernelization is a technique to obtain FPT algorithms for a parameterized problem $(L, \kappa)$.
- Based in auto-reductions
- We look for a polynomial time algorithm that transforms an instance $x$ in another instance $x'$ of the problem (the kernel). So that
  - $x'$ is a yes instance iff $x$ is a yes instance.
    $x$ and $x'$ are equivalent instances

# Kernelization

- Kernelization is a technique to obtain FPT algorithms for a parameterized problem $(L, \kappa)$.
- Based in auto-reductions
- We look for a polynomial time algorithm that transforms an instance $x$ in another instance $x'$ of the problem (the kernel). So that
  - $x'$ is a yes instance iff $x$ is a yes instance.
    $x$ and $x'$ are equivalent instances
  - the size of $x'$ is upperbounded by $f(\kappa(x))$, for some computable function $f$.

# Kernelization

- Kernelization is a technique to obtain FPT algorithms for a parameterized problem $(L, \kappa)$.
- Based in auto-reductions
- We look for a polynomial time algorithm that transforms an instance $x$ in another instance $x'$ of the problem (the kernel). So that
  - $x'$ is a yes instance iff $x$ is a yes instance.
    $x$ and $x'$ are equivalent instances
  - the size of $x'$ is upperbounded by $f(\kappa(x))$, for some computable function $f$.
- An algorithm that computes $x'$ and solves by brute force this instance has cost

# Kernelization

- Kernelization is a technique to obtain FPT algorithms for a parameterized problem $(L, \kappa)$.
- Based in auto-reductions
- We look for a polynomial time algorithm that transforms an instance $x$ in another instance $x'$ of the problem (the kernel). So that
  - $x'$ is a yes instance iff $x$ is a yes instance.
    $x$ and $x'$ are equivalent instances
  - the size of $x'$ is upperbounded by $f(\kappa(x))$, for some computable function $f$.
- An algorithm that computes $x'$ and solves by brute force this instance has cost
  $O(p(|x|) + g(f(\kappa(x)))$

# Kernelization

- Kernelization is a technique to obtain FPT algorithms for a parameterized problem $(L, \kappa)$.
- Based in auto-reductions
- We look for a polynomial time algorithm that transforms an instance $x$ in another instance $x'$ of the problem (the kernel). So that
  - $x'$ is a yes instance iff $x$ is a yes instance.
    $x$ and $x'$ are equivalent instances
  - the size of $x'$ is upperbounded by $f(\kappa(x))$, for some computable function $f$.
- An algorithm that computes $x'$ and solves by brute force this instance has cost
  $O(p(|x|) + g(f(\kappa(x)))$
  So, it is an FPT algorithm provided the problem is decidible.

# k-Vertex Cover: reduction rules?

- Often a kernelization is defined through reduction rules that, either allow us to produce an smaller equivalent instance or to show that, the original instance is a NO instance.

# k-Vertex Cover: reduction rules?

- Often a kernelization is defined through reduction rules that, either allow us to produce an smaller equivalent instance or to show that, the original instance is a NO instance.
- Technically, we could produce a NO instance of constant size, however we often see the construction as a preprocesing step that has the possibility of saying NO, and will do that as soon as possible.

# k-Vertex Cover: reduction rules?

- Often a kernelization is defined through reduction rules that, either allow us to produce an smaller equivalent instance or to show that, the original instance is a NO instance.
- Technically, we could produce a NO instance of constant size, however we often see the construction as a preprocesing step that has the possibility of saying NO, and will do that as soon as possible.
- Let's look at a first kernelization for $p$-VC.

# k-Vertex Cover: reduction rules?

- Often a kernelization is defined through reduction rules that, either allow us to produce an smaller equivalent instance or to show that, the original instance is a NO instance.
- Technically, we could produce a NO instance of constant size, however we often see the construction as a preprocesing step that has the possibility of saying NO, and will do that as soon as possible.
- Let's look at a first kernelization for $p$-VC.

$p$-VERTEX COVER
Input: a graph $G$ and an integer $k$,
Parameter: $k$
Question: $\exists S \subseteq V(G) \mid |S| = k$ and $\forall \{u, v\} \in E(G) |\{u, v\} \cap S| \geq 1$?

# k-Vertex Cover: reduction rules?

- Let $(G, k)$ be a $k$-$\mathrm{VC}$ instance.
- recall: Two instances $x_1$ and $x_2$ of decision probem $\mathrm{P}$ are equivalent when "$x_1 \in \mathrm{P}$ iff $x_2 \in \mathrm{P}$".

# k-Vertex Cover: reduction rules?

- Let $(G, k)$ be a $k$-$\mathrm{VC}$ instance.
- recall: Two instances $x_1$ and $x_2$ of decision probem $\mathrm{P}$ are equivalent when "$x_1 \in \mathrm{P}$ iff $x_2 \in \mathrm{P}$".
- An isolated vertex has degree zero. Therefore it does not cover any edge!

# k-Vertex Cover: reduction rules?

- Let $(G, k)$ be a $k$-$\mathrm{VC}$ instance.
- recall: Two instances $x_1$ and $x_2$ of decision probem $\mathrm{P}$ are equivalent when "$x_1 \in \mathrm{P}$ iff $x_2 \in \mathrm{P}$".
- An isolated vertex has degree zero. Therefore it does not cover any edge!

**Obs 1**

If $v$ is an isolated vertex, $(G, k)$ and $(G - v, k)$ are equivalent.

# k-Vertex Cover: reduction rules?

- Let $(G, k)$ be a $k$-$\mathrm{VC}$ instance.
- recall: Two instances $x_1$ and $x_2$ of decision probem $\mathrm{P}$ are equivalent when "$x_1 \in \mathrm{P}$ iff $x_2 \in \mathrm{P}$".
- An isolated vertex has degree zero. Therefore it does not cover any edge!

**Obs 1**

If $v$ is an isolated vertex, $(G, k)$ and $(G - v, k)$ are equivalent.

- A vertex with degree $\geq k + 1$

# k-Vertex Cover: reduction rules?

- Let $(G, k)$ be a $k$-$\mathrm{VC}$ instance.
- recall: Two instances $x_1$ and $x_2$ of decision probem $\mathrm{P}$ are equivalent when "$x_1 \in \mathrm{P}$ iff $x_2 \in \mathrm{P}$".
- An isolated vertex has degree zero. Therefore it does not cover any edge!

**Obs 1**

If $v$ is an isolated vertex, $(G, k)$ and $(G - v, k)$ are equivalent.

- A vertex with degree $\geq k + 1$ must be part of a vertex cover of size $\leq k$.

# k-Vertex Cover: reduction rules?

- Let $(G, k)$ be a $k$-$\mathrm{VC}$ instance.
- recall: Two instances $x_1$ and $x_2$ of decision probem $\mathrm{P}$ are equivalent when "$x_1 \in \mathrm{P}$ iff $x_2 \in \mathrm{P}$".
- An isolated vertex has degree zero. Therefore it does not cover any edge!

**Obs 1**

If $v$ is an isolated vertex, $(G, k)$ and $(G - v, k)$ are equivalent.

- A vertex with degree $\geq k + 1$ must be part of a vertex cover of size $\leq k$.

**Obs 2**

If $v$ has degree $\geq k + 1$, $(G, k)$ and $(G - v, k - 1)$ are equivalent.

# Reduction rules

- The previous observations suggest a preprocessing of the input:

# Reduction rules

- The previous observations suggest a preprocessing of the input: Iteratively remove isolated vertices and vertices with degree at least $k+1$, decreasing the parameter by one in the second case.

# Reduction rules

- The previous observations suggest a preprocessing of the input: Iteratively remove isolated vertices and vertices with degree at least $k+1$, decreasing the parameter by one in the second case.
- By Obs 1 and 2, the resulting instance $(G', k')$ is equivalent to the original instance.

# Reduction rules

- The previous observations suggest a preprocessing of the input: Iteratively remove isolated vertices and vertices with degree at least $k+1$, decreasing the parameter by one in the second case.
- By Obs 1 and 2, the resulting instance $(G', k')$ is equivalent to the original instance.
- Furthermore, it can be computed in polynomial time.

# Reduction rules

- The previous observations suggest a preprocessing of the input: Iteratively remove isolated vertices and vertices with degree at least $k+1$, decreasing the parameter by one in the second case.
- By Obs 1 and 2, the resulting instance $(G', k')$ is equivalent to the original instance.
- Furthermore, it can be computed in polynomial time.
- How big is $(G', k')$?

# Reduced instance

- Iteratively remove isolated vertices and vertices with degree at least $k+1$, decreasing the parameter by one in the second case.

# Reduced instance

- Iteratively remove isolated vertices and vertices with degree at least $k+1$, decreasing the parameter by one in the second case.
- In $(G', k')$ all the vertices have degree $\leq k$.

# Reduced instance

- Iteratively remove isolated vertices and vertices with degree at least $k+1$, decreasing the parameter by one in the second case.
- In $(G', k')$ all the vertices have degree $\leq k$.

**Obs 3**

If $G$ has a vertex cover with $\leq k$ vertices and all the vertices have degree $\leq k$,

# Reduced instance

- Iteratively remove isolated vertices and vertices with degree at least $k+1$, decreasing the parameter by one in the second case.
- In $(G', k')$ all the vertices have degree $\leq k$.

**Obs 3**

If $G$ has a vertex cover with $\leq k$ vertices and all the vertices have degree $\leq k$, $|E(G')| \leq k^2$.

# Reduced instance

- Iteratively remove isolated vertices and vertices with degree at least $k+1$, decreasing the parameter by one in the second case.
- In $(G', k')$ all the vertices have degree $\leq k$.

**Obs 3**

If $G$ has a vertex cover with $\leq k$ vertices and all the vertices have degree $\leq k$, $|E(G')| \leq k^2$.

- So, we can filter as NO instances those leading to reduced instances with a high number of edges!

# Reduced instance

- Iteratively remove isolated vertices and vertices with degree at least $k+1$, decreasing the parameter by one in the second case.
- In $(G', k')$ all the vertices have degree $\leq k$.

### Obs 3

If $G$ has a vertex cover with $\leq k$ vertices and all the vertices have degree $\leq k$, $|E(G')| \leq k^2$.

- So, we can filter as NO instances those leading to reduced instances with a high number of edges!
- By Obs 3, if $|E(G')| > k^2$, we replace $(G', k')$ by a trivial small NO-instance, which is again equivalent.

# Kernel

### Theorem

*Let $(G, k)$ be an instance to* $\textsc{p-VC}$. *In polynomial time we can obtain an equivalent* $\textsc{p-VC}$ *instance $(G', k')$ with $|V(G')|, |E(G')| \leq O(k^2)$.*

# Kernel

**Theorem**

*Let $(G, k)$ be an instance to* $\textsc{p-VC}$*. In polynomial time we can obtain an equivalent* $\textsc{p-VC}$ *instance* $(G', k')$ *with* $|V(G')|, |E(G')| \leq O(k^2)$*.*

- Such an instance is called a kernel.

# Kernel

**Theorem**

*Let $(G, k)$ be an instance to* p-VC. *In polynomial time we can obtain an equivalent* p-VC *instance $(G', k')$ with $|V(G')|, |E(G')| \leq O(k^2)$.*

- Such an instance is called a kernel.
- A kernel

# Kernel

**Theorem**

*Let $(G, k)$ be an instance to* $\textsc{p-VC}$*. In polynomial time we can obtain an equivalent* $\textsc{p-VC}$ *instance $(G', k')$ with $|V(G')|, |E(G')| \leq O(k^2)$.*

- Such an instance is called a kernel.
- A kernel
  - is an equivalent instance,

# Kernel

**Theorem**

*Let $(G, k)$ be an instance to* $\text{P-VC}$*. In polynomial time we can obtain an equivalent* $\text{P-VC}$ *instance* $(G', k')$ *with* $|V(G')|, |E(G')| \leq O(k^2)$*.*

- Such an instance is called a kernel.
- A kernel
  - is an equivalent instance,
  - can be computed in polynomial time, and

# Kernel

**Theorem**

*Let $(G, k)$ be an instance to $\textsc{p-VC}$. In polynomial time we can obtain an equivalent $\textsc{p-VC}$ instance $(G', k')$ with $|V(G')|, |E(G')| \leq O(k^2)$.*

- Such an instance is called a kernel.
- A kernel
  - is an equivalent instance,
  - can be computed in polynomial time, and
  - has size bounded by a function of the parameter

# Kernelization algorithm

# Kernelization algorithm

- Assume that Ker-P is a polynomial time algorithm computing a kernel for a given instance of problem P and that Alg-P is an exact (exponential time) algorithm for P.

# Kernelization algorithm

- Assume that $\textsc{Ker-P}$ is a polynomial time algorithm computing a kernel for a given instance of problem $\mathrm{P}$ and that $\textsc{Alg-P}$ is an exact (exponential time) algorithm for $\mathrm{P}$.

```
function AlgKernel-P(x)
    z = Ker-P(x)
    return (Alg-P(z))
```

# Kernelization algorithm

- Assume that KER-P is a polynomial time algorithm computing a kernel for a given instance of problem P and that ALG-P is an exact (exponential time) algorithm for P.

  **function** ALGKERNEL-P($x$)
  $z =$KER-P($x$)
  **return** (ALG-P($z$))

- ALGKER-P-VC is an FPT algorithm for P.

# A kernelization algorithm for p-VC

**function** ALGKERNEL-P-VC$(G, k)$
$(G', k') =$ Iteratively remove isolated vertices and vertices with degree at least $k+1$, decreasing the parameter by one in the second case.
**if** $|E(G')| > k^2$ **then return** NO
**for** each $S \subseteq V'$ with $|S| = k'$ **do**
  **if** $S$ is a vertex cover **then return** SI
**return** NO

# A kernelization algorithm for p-VC

**function** $\textsc{AlgKernel-p-VC}(G, k)$
$(G', k') =$ Iteratively remove isolated vertices and vertices with degree at least $k+1$, decreasing the parameter by one in the second case.
**if** $|E(G')| > k^2$ **then return** $\textsc{no}$
**for** each $S \subseteq V'$ with $|S| = k'$ **do**
  **if** $S$ is a vertex cover **then return** $\textsc{si}$
**return** $\textsc{no}$

$\textsc{AlgKernel-p-VC}$ runs in $O(n^c + k^{2k}k^2) = O(n^c) + O(k^{2k+2})$

# p-MaxSat

P-MAXSAT
Input: a Boolean CNF formula $F$ and an integer $k$.
Parameter: $k$.
Question: Is there a variable assignment satisfying at least $k$ clauses?

# p-MaxSat

P-MAXSAT
Input: a Boolean CNF formula $F$ and an integer $k$.
Parameter: $k$.
Question: Is there a variable assignment satisfying at least $k$ clauses?

Recall that the size of a CNF formula is the sum of clause lengths (# literals); we ignore as usual log-factors.

# p-MaxSat: Reduction rules

# p-MaxSat: Reduction rules

- A clause in $F$ is trivial if it contains both a positive and negative literal in the same variable.

# p-MaxSat: Reduction rules

- A clause in $F$ is trivial if it contains both a positive and negative literal in the same variable.

**Obs 1**

Let $F'$ be obtained from formula $F$ by removing all $t$ trivial clauses. $(F', k - t)$ and $(F, k)$ are equivalent.

# p-MaxSat: Reduction rules

# p-MaxSat: Reduction rules

- A clause in $(F, k)$ is long if it contains at least $k$ literals, and short otherwise.

# p-MaxSat: Reduction rules

- A clause in $(F, k)$ is long if it contains at least $k$ literals, and short otherwise.
- If $F$ contains at least $k$ long clauses, $(F, k)$ is a YES instance of p-MaxSat.

# p-MaxSat: Reduction rules

- A clause in $(F, k)$ is long if it contains at least $k$ literals, and short otherwise.
- If $F$ contains at least $k$ long clauses, $(F, k)$ is a YES instance of p-MAXSAT.

**Obs 2**

Let $F_s$ be obtained from formula $F$ by removing all $\ell < k$ long clauses. $(F_s, k - \ell)$ and $(F, k)$ are equivalent.

# p-MaxSat: Reduction rules

# p-MaxSat: Reduction rules

**Obs 3**

If $F$ contains at least $2k$ clauses, $(F, k)$ is a YES instance of p-MaxSat.

# p-MaxSat: Reduction rules

**Obs 3**

If $F$ contains at least $2k$ clauses, $(F, k)$ is a YES instance of p-MaxSat.

**Proof.**

Take an arbitrary truth assignment $x$ and its complement $\overline{x}$ obtained by flipping all variables. Every clause of $F$ is satisfied by $x$ or by *overlinex* (or by both). The one that satisfies most clauses satisfies at least $k$ clauses. $\square$

# A kernelization algorithm for p-MaxSat

```
function AlgKernel-p-MaxSat(F, k)
    Remove from F all t trivial clauses and set k = k − t
    if F has at least k long clauses then return YES
    Remove from F all ℓ long clauses and set k = k − ℓ
    if F has at least 2k clauses then return YES
    for each set of k clauses do
        for each selection of one literal per clause in the set do
            if selection has a compatible truth assignment then
                return YES
    return NO
```

# A kernelization algorithm for p-MaxSat

1: **function** ALGKERNEL-P-MAXSAT$(F, k)$
2: Remove from $F$ all $t$ trivial clauses and set $k = k - t$
3: **if** $F$ has at least $k$ long clauses **then return** YES
4: Remove from $F$ all $\ell$ long clauses and set $k = k - \ell$
5: **if** $F$ has at least $2k$ clauses **then return** YES
6: **for** each set of $k$ clauses **do**
7: **for** each selection of one literal per clause in the set **do**
8: **if** selection has a compatible truth assignment **then**
9: **return** YES
10: **return** NO

After step 5, $F$ contains at most $2k'$ clauses with at most $k'$ literals, for $k' = k - t - \ell$.

# A kernelization algorithm for p-MaxSat

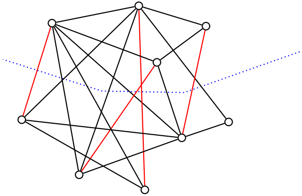
1: **function** ALGKERNEL-P-MAXSAT$(F, k)$
2: Remove from $F$ all $t$ trivial clauses and set $k = k - t$
3: **if** $F$ has at least $k$ long clauses **then return** YES
4: Remove from $F$ all $\ell$ long clauses and set $k = k - \ell$
5: **if** $F$ has at least $2k$ clauses **then return** YES
6: **for** each set of $k$ clauses **do**
7: **for** each selection of one literal per clause in the set **do**
8: **if** selection has a compatible truth assignment **then**
9: **return** YES
10: **return** NO

After step 5, $F$ contains at most $2k'$ clauses with at most $k'$ literals, for $k' = k - t - \ell$.

ALGKERNEL-P-MAXSAT is an FPT algorithm for P-MAXSAT.

- *Crown decomposition* is a general kernelization technique based on some results on matchings.

- *Crown decomposition* is a general kernelization technique based on some results on matchings.
- For disjoint vertex subsets $U, W$ of a graph $G$, $M$ is a matching of $U$ into $W$ if every edge of $M$ connects a vertex of $U$ and a vertex of $W$ and every vertex of $U$ is an endpoint of some edge of $M$.
  We also say that $M$ saturates $U$.



If $M$ saturates $U$, $|U| \leq |W|$

# Crown decomposition: Definition

- A crown decomposition of a graph $G = (V, E)$ is a partitioning of $V$ into three parts $C$, $H$ and $R$, such that

# Crown decomposition: Definition

- A crown decomposition of a graph $G = (V, E)$ is a partitioning of $V$ into three parts $C$, $H$ and $R$, such that
  - $C \neq \emptyset$ is an independent set.

# Crown decomposition: Definition

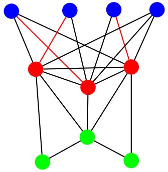
- A crown decomposition of a graph $G = (V, E)$ is a partitioning of $V$ into three parts $C$, $H$ and $R$, such that
  - $C \neq \emptyset$ is an independent set.
  - There are no edges between vertices of $C$ and $R$.
    Removing $H$ separates $C$ from $R$.

# Crown decomposition: Definition

- A crown decomposition of a graph $G = (V, E)$ is a partitioning of $V$ into three parts $C$, $H$ and $R$, such that
  - $C \neq \emptyset$ is an independent set.
  - There are no edges between vertices of $C$ and $R$.
    Removing $H$ separates $C$ from $R$.
  - Let $E'$ be the set of edges between vertices of $C$ and $H$. Then $E'$ contains a matching of $H$ into $C$.

# Crown decomposition: Definition

- A crown decomposition of a graph $G = (V, E)$ is a partitioning of $V$ into three parts $C$, $H$ and $R$, such that
  - $C \neq \emptyset$ is an independent set.
  - There are no edges between vertices of $C$ and $R$. Removing $H$ separates $C$ from $R$.
  - Let $E'$ be the set of edges between vertices of $C$ and $H$. Then $E'$ contains a matching of $H$ into $C$.



# Computing a crown decomposition

**Theorem (König's theorem)**

*In every undirected bipartite graph the size of a maximum matching is equal to the size of a minimum vertex cover.*

**Theorem (Hall's theorem)**

*Let $G = (V_1, V_2, E)$ be an undirected bipartite graph. $G$ has a matching saturating $V_1$ iff for all $X \subseteq V_1$, we have $|N(X)| \geq |X|$.*

Can you obtain a minimum vertex cover in a bipartite graph in polynomial time?

# Computing a crown decomposition

**Theorem (König's theorem)**

*In every undirected bipartite graph the size of a maximum matching is equal to the size of a minimum vertex cover.*

**Theorem (Hall's theorem)**

*Let $G = (V_1, V_2, E)$ be an undirected bipartite graph. $G$ has a matching saturating $V_1$ iff for all $X \subseteq V_1$, we have $|N(X)| \geq |X|$.*

Can you obtain a minimum vertex cover in a bipartite graph in polynomial time? YES!

# Computing a crown decomposition

**Theorem (Hopcroft-Karp, SIAM J. Computing 2, 225–231 (1973))**

*Let $G = (V_1, V_2, E)$ be an undirected bipartite graph on $n$ vertices and $m$ edges. Then we can find a maximum matching as well as a minimum vertex cover of $G$ in time $O(m\sqrt{n})$. Furthermore, in time $O(m\sqrt{n})$ either we can find a matching saturating $V_1$ or an inclusion-wise minimal set $X \subseteq V_1$ such that $|N(X)| < |X|$.*

# Crown lemma

**Lemma**

*Let $G = (V, E)$ be a graph without isolated vertices and with at least $3k + 1$ vertices. There is a polynomial-time algorithm that either*

- *finds a matching of size $k + 1$ in $G$; or*
- *finds a crown decomposition of $G$.*

**Proof**

We compute a maximal matching $M$ in $G$.
If $|M| \geq k + 1$, we are done.

Now, $1 \leq |M| \leq k+1$.

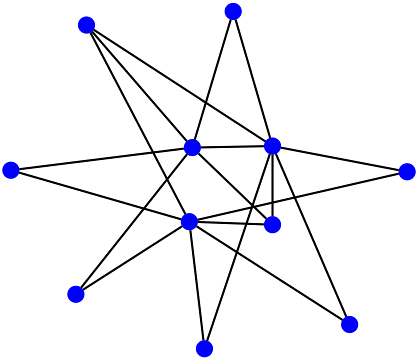
Let $V_M$ be the end points of $M$ and $I = V - V_M$.

- $M$ is a maximal matching, so $I$ is an independent set.
- Let $G_{I,V_M}$ be the bipartite subgraph induced in $G$ by $I$ and $V_M$.
- In polynomial time, we compute a minimum size vertex cover $X$ and a maximum matching $M'$ in $G_{I,V_M}$.
- If $|M'| \geq k$, we are done. From now on, $|M'| \leq k$ and also $|X| \leq k$.
- If $X \cap V_M = \emptyset$, $X = I$. Then, $|I| = |X| \leq k$ and $|V| = |I| + |X| \leq k + 2k \leq 3k$!
- Then, $X \cap V_M \neq \emptyset$

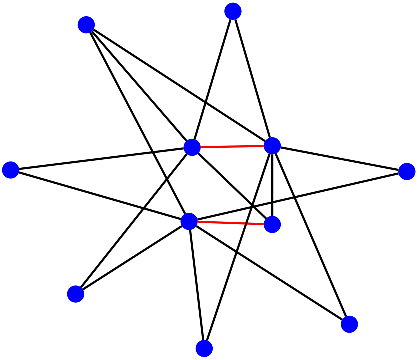
- We obtain a crown decomposition $(C, H, R)$ as follows.
- Since $|X| = |M'|$, every edge of the matching $M'$ has exactly one endpoint in $X$.
- Let $M^*$ be the subset of $M'$ such that every edge from $M^*$ has exactly one endpoint in $X \cap V_M$ and let $V_{M^*}$ denote the set of endpoints of edges in $M^*$.
- Set head $H = X \cap V_M = X \cap V_{M^*}$, crown $C = V_{M^*} \cap I$, and the remaining part is $R$.
- $C$ is an independent set and, by construction, $M^*$ is a matching of $H$ into $C$.
- Since $X$ is a vertex cover of $G_{I,V_M}$, every vertex of $C$ can be adjacent only to vertices of $H$.

End proof

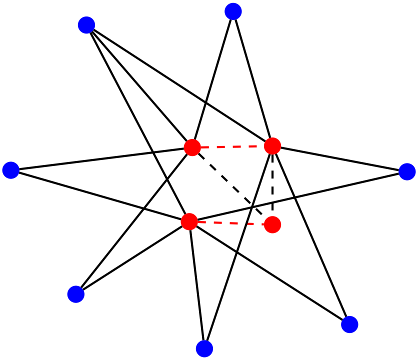
# An example with $k = 3$



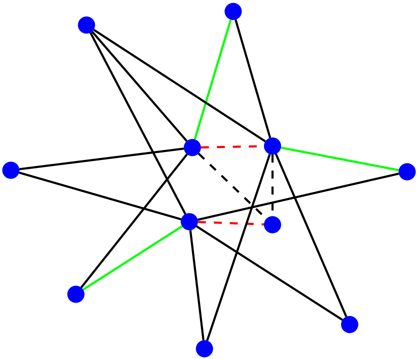
# An example with $k = 3$



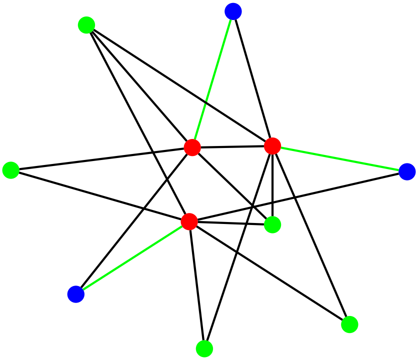
# An example with $k = 3$



# An example with $k = 3$



# An example with $k = 3$



# Crown decomposition: Vertex cover

Consider a Vertex Cover instance $(G, k)$.

- By an exhaustive application of the isolated vertex reduction rule, we may assume that $G$ has no isolated vertices.
- If $|V(G)| > 3k$, we use the crown lemma to get either
- a matching of size $k+1$, (so $(G, k)$ is a no-instance) or a crown decomposition $C$, $H$, $R$.

# Crown decomposition: Vertex cover

From the crown decomposition $C$, $H$, $R$ of $G$, let $M$ be a matching of $H$ into $C$.

- The matching $M$ witnesses that, for every vertex cover $X$ of $G$, $X$ contains at least $|M| = |H|$ vertices of $H \cap C$ to cover the edges of $M$.
- $H$ covers all edges of $G$ that are incident to $H \cup C$.
- So, there exists a minimum vertex cover of $G$ that contains $H$, and we may reduce $(G, k)$ to $(G - H, k - |H|)$.
- Further, in $(G - H, k - |H|)$, $c \in C$ is isolated and can be eliminated.

# Crown decomposition: Vertex cover

As the crown lemma promises that $H \neq \emptyset$, we can always reduce the graph as long as $|V(G)| > 3k$.

**Lemma**

*Vertex Cover admits a kernel with at most $3k$ vertices.*

# Crown decomposition: Max SAT

**Lemma**

*Max SAT admits a kernel with at most $k$ variables and $2k$ clauses.*

# Kernelization: summary

# Kernelization: summary

- For parameterized problems, kernelization algorithms are a method to obtain FPT algorithms.

# Kernelization: summary

- For parameterized problems, kernelization algorithms are a method to obtain FPT algorithms.
- These are preprocessing algorithms that can add to any algorithmic method (e.g. approximation/exact algorithms).

# Kernelization: summary

- For parameterized problems, kernelization algorithms are a method to obtain FPT algorithms.
- These are preprocessing algorithms that can add to any algorithmic method (e.g. approximation/exact algorithms).
- Kernelization algorithms usually consist of reduction rules, which reduce simple local structures (degree 1 vertices / high degree vertices / long clauses, etc), and a bound $f(k)$ for irreducible instances $(X, k)$ that allows us to

# Kernelization: summary

- For parameterized problems, kernelization algorithms are a method to obtain FPT algorithms.
- These are preprocessing algorithms that can add to any algorithmic method (e.g. approximation/exact algorithms).
- Kernelization algorithms usually consist of reduction rules, which reduce simple local structures (degree 1 vertices / high degree vertices / long clauses, etc), and a bound $f(k)$ for irreducible instances $(X, k)$ that allows us to
  - return NO if $|X| > f(k)$, for minimization problems, or
  - return YES if $|X| > f(k)$, for maximization problems.

# Designing kernelization algorithms

# Designing kernelization algorithms

- What are the trivial substructures, where an optimal solution of a certain form can be guaranteed?

# Designing kernelization algorithms

- What are the trivial substructures, where an optimal solution of a certain form can be guaranteed?
- Is there a reduction rule reflecting this?

# Designing kernelization algorithms

- What are the trivial substructures, where an optimal solution of a certain form can be guaranteed?
- Is there a reduction rule reflecting this?
- Can a bound be proved for irreducible instances? If not, which structures are problematic? Etc...

# Designing kernelization algorithms

- What are the trivial substructures, where an optimal solution of a certain form can be guaranteed?
- Is there a reduction rule reflecting this?
- Can a bound be proved for irreducible instances? If not, which structures are problematic? Etc...
- Any problem in FPT admits a kernelization.

# Designing kernelization algorithms

- What are the trivial substructures, where an optimal solution of a certain form can be guaranteed?
- Is there a reduction rule reflecting this?
- Can a bound be proved for irreducible instances? If not, which structures are problematic? Etc...
- Any problem in FPT admits a kernelization.
- Hardness notion?

# Designing kernelization algorithms

- What are the trivial substructures, where an optimal solution of a certain form can be guaranteed?
- Is there a reduction rule reflecting this?
- Can a bound be proved for irreducible instances? If not, which structures are problematic? Etc...
- Any problem in FPT admits a kernelization.

- Hardness notion?
- We would like to get a kernel as small as possible.

# Designing kernelization algorithms

- What are the trivial substructures, where an optimal solution of a certain form can be guaranteed?
- Is there a reduction rule reflecting this?
- Can a bound be proved for irreducible instances? If not, which structures are problematic? Etc...
- Any problem in FPT admits a kernelization.

- Hardness notion?
- We would like to get a kernel as small as possible.
- Statements like: $(L, \kappa)$ does not admit a linear (quadratic) kernel unless some complexity assumption fails are the kind of results showing kernelization hardness.