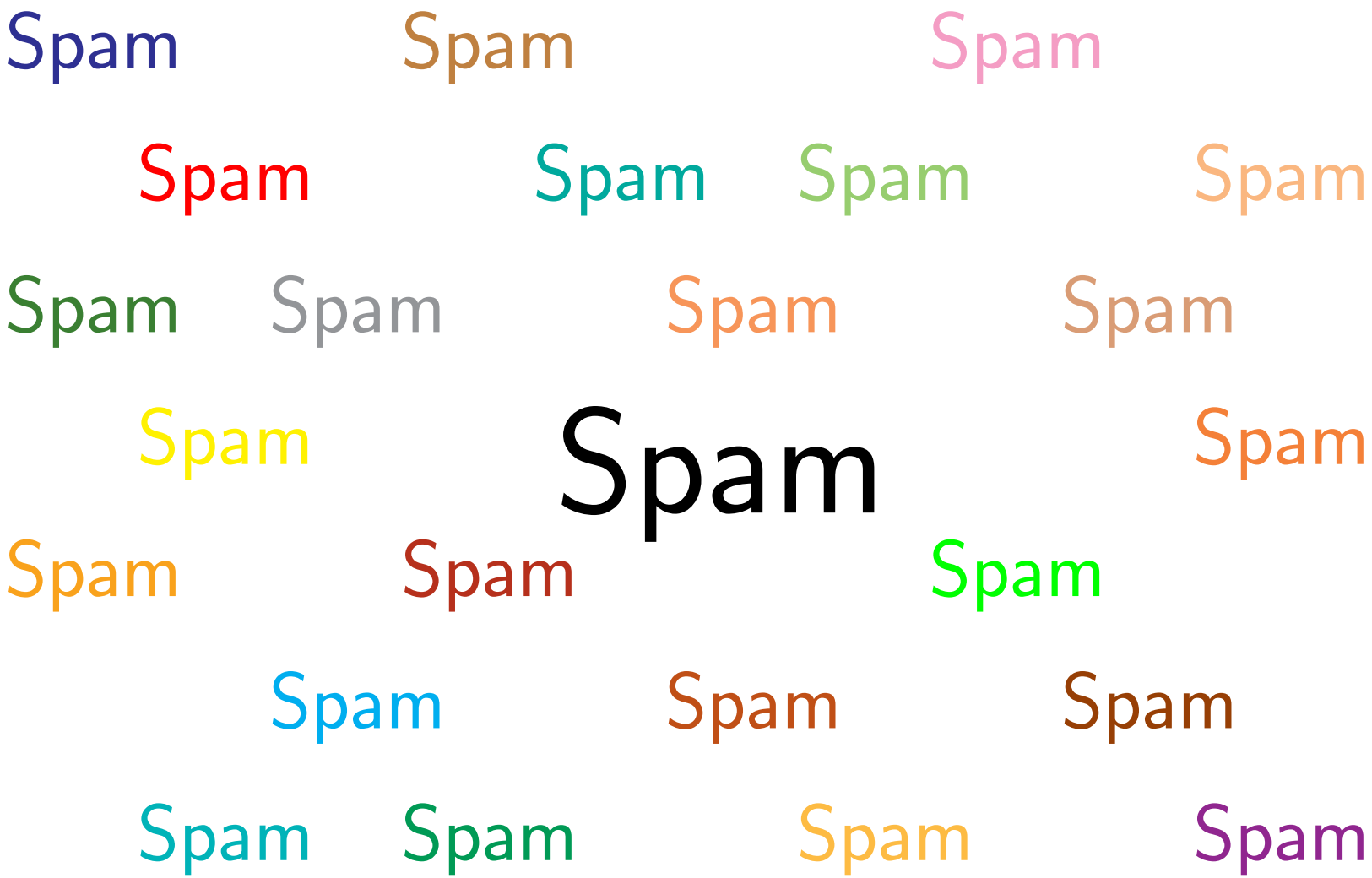
Parameterized Aspects of Triangle Enumeration

Matthias Bentert Till Fluschnik André Nichterlein
Rolf Niedermeier

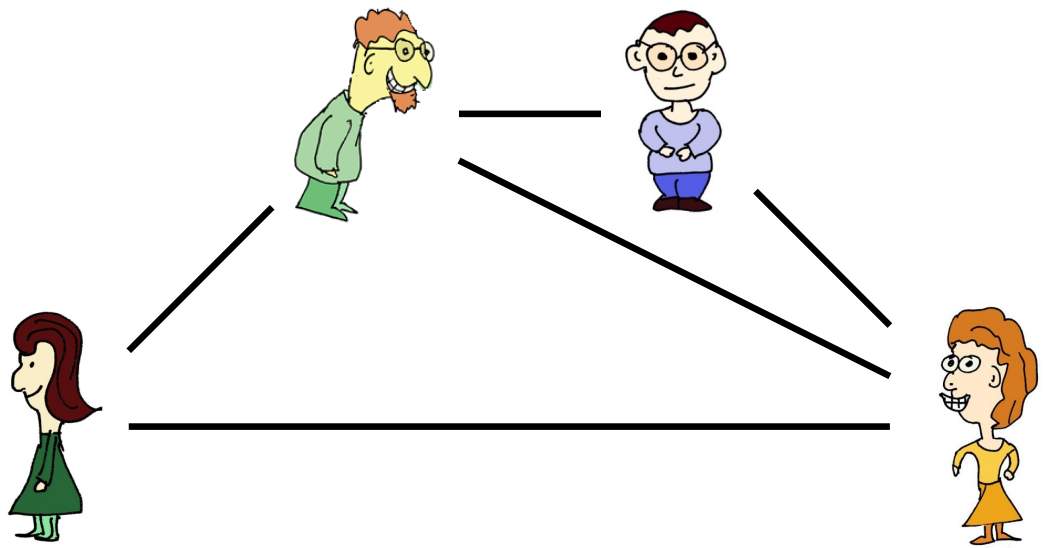
Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany

21st International Symposium on Fundamentals of Computation Theory, Bordeaux, September 12th, 2017

Spam Detection

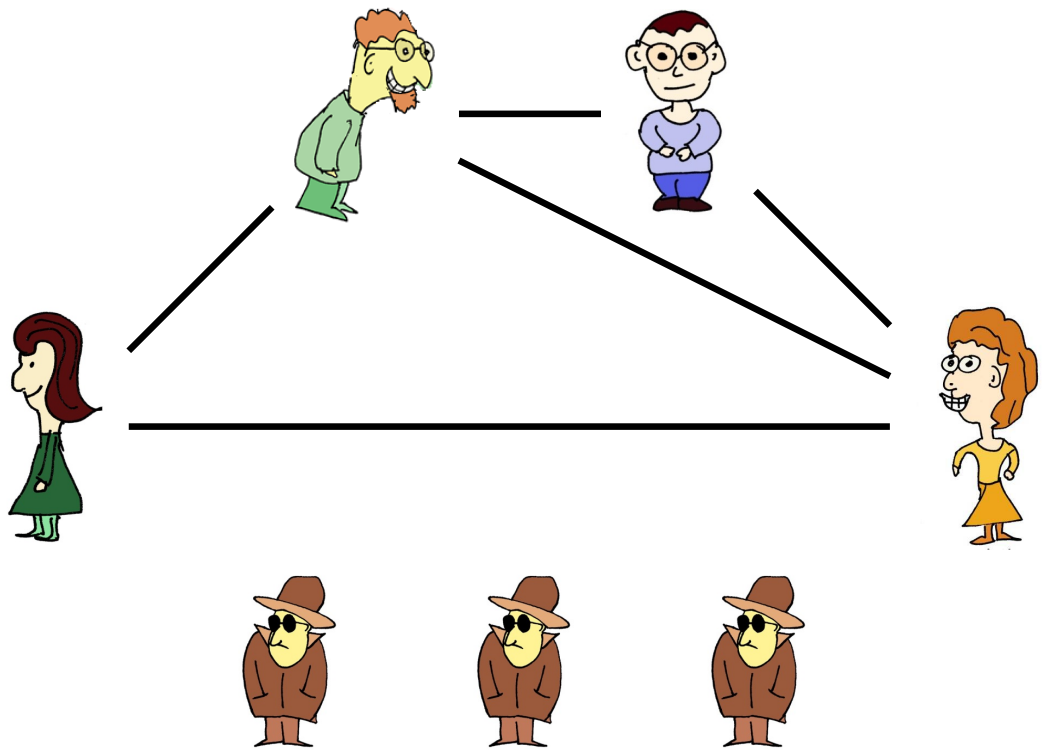


Spam Detection II



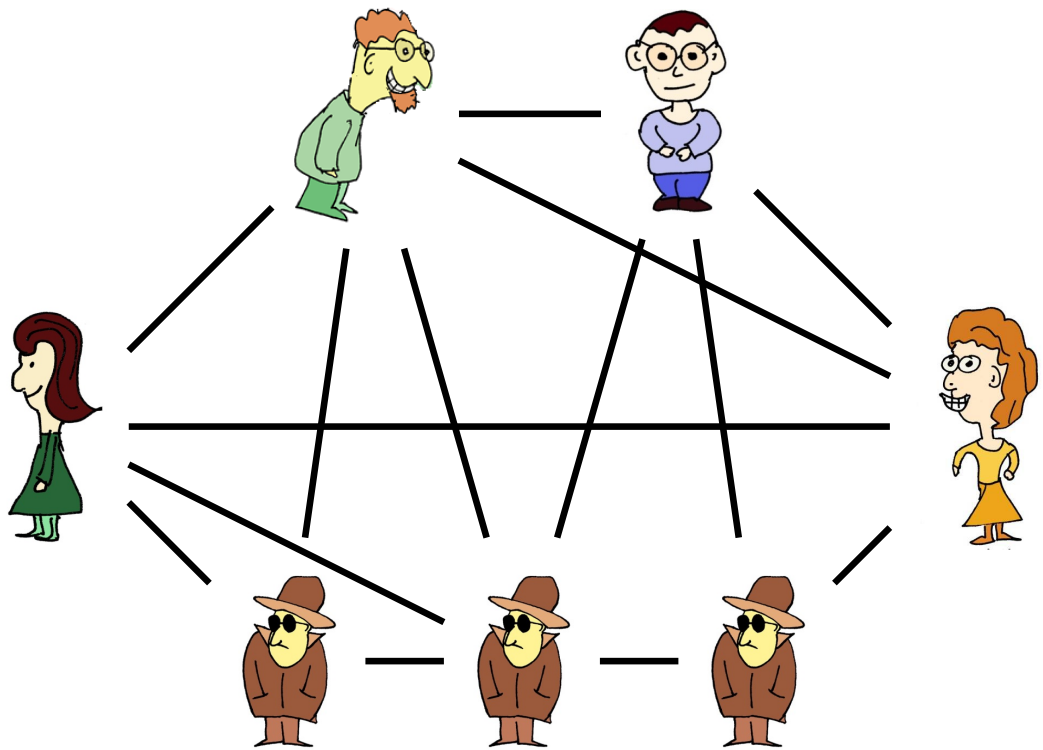
[Drawn by Piotr Faliszewski]

Spam Detection II



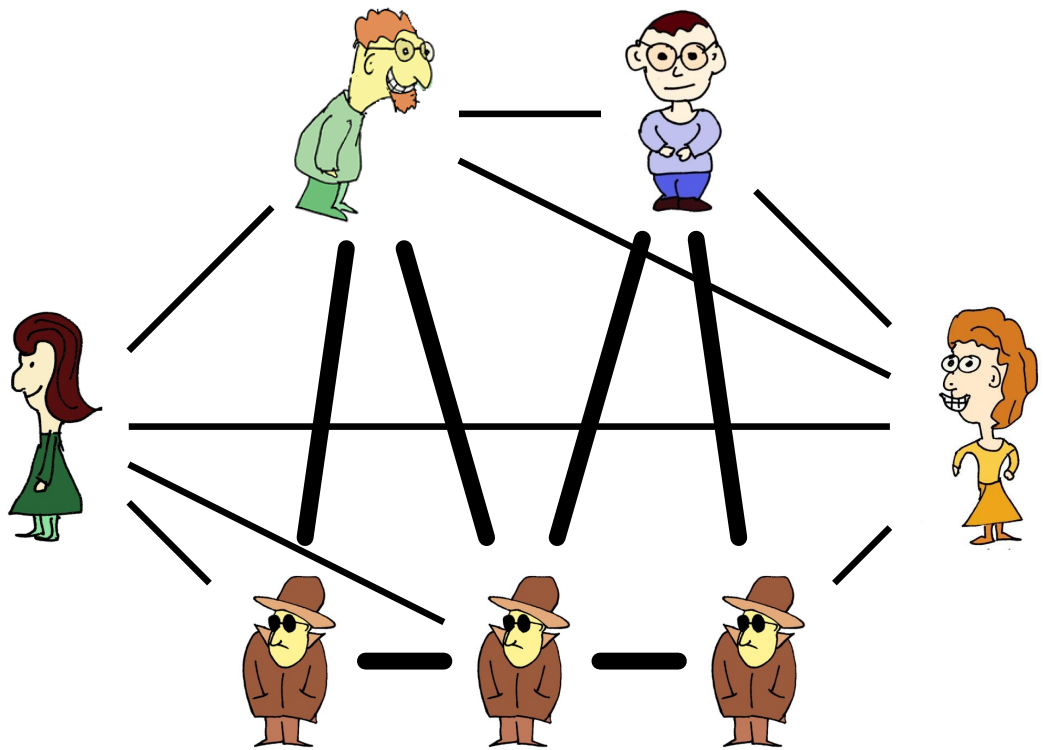
[Drawn by Piotr Faliszewski]

Spam Detection II



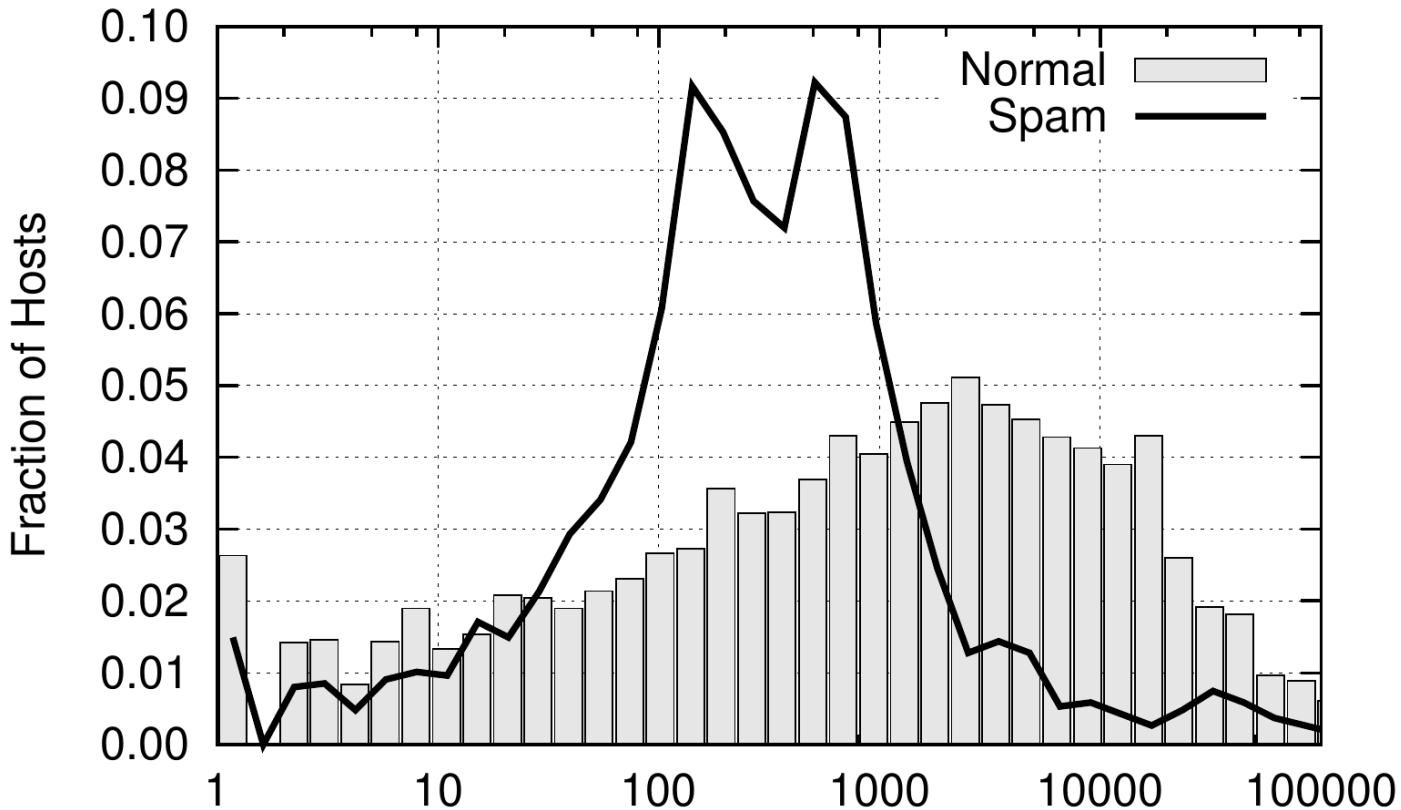
[Drawn by Piotr Faliszewski]

Spam Detection II



[Drawn by Piotr Faliszewski]

Spam Detection III



[Becchetti, L., Boldi, P., Castillo, C., Gionis, A.: Efficient Semi-Streaming Algorithms for Local Triangle Counting in Massive Graphs. ACM SIGKDD, 2008]

Further Applications

- Motif detection (computational biology)

[Yook, Oltvai, Barabasi: Proteomics, 2004]

Further Applications

- Motif detection (computational biology)

  [Yook, Oltvai, Barabasi: Proteomics, 2004]

- Identifying roles in online discussion forums

  [Wesler et al.: Journal of Social Structure, 2007]

Further Applications

- Motif detection (computational biology)

  [Yook, Oltvai, Barabasi: Proteomics, 2004]

- Identifying roles in online discussion forums

  [Wesler et al.: Journal of Social Structure, 2007]

- Social network analysis

  [Berry et al.: Physical Review E, 2011]

- Predicting the evolution of social networks

  [Leskovec et al.: ACM SIGKDD, 2008]

Problem Definition

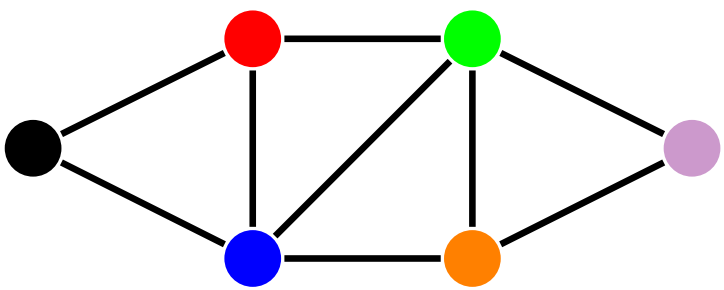
Problem (Triangle Enumeration)

Input: An undirected unweighted graph $G = (V, E)$.
Task: List all triangles in G.

Problem Definition

Problem (Triangle Enumeration)

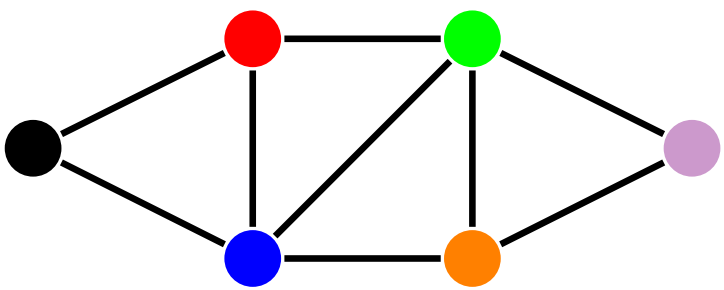
Input: An undirected unweighted graph $G = (V, E)$.
Task: List all triangles in G.

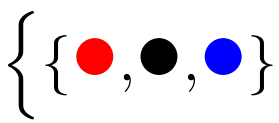


Problem Definition

Problem (Triangle Enumeration)

Input: An undirected unweighted graph $G = (V, E)$.
Task: List all triangles in G.

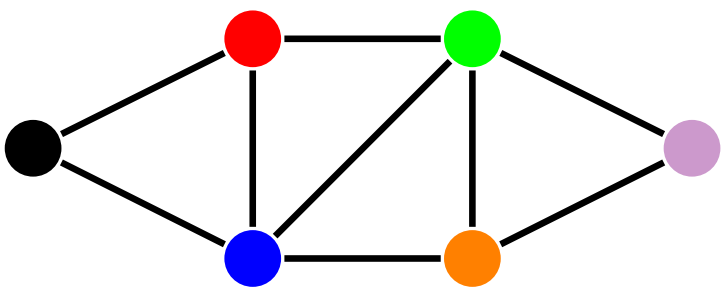


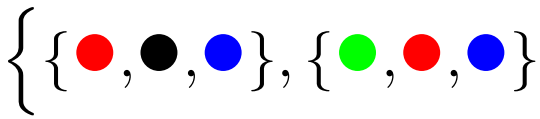


Problem Definition

Problem (Triangle Enumeration)

Input: An undirected unweighted graph $G = (V, E)$.
Task: List all triangles in G.

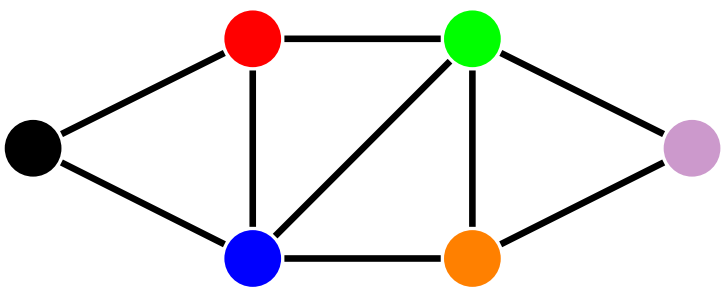


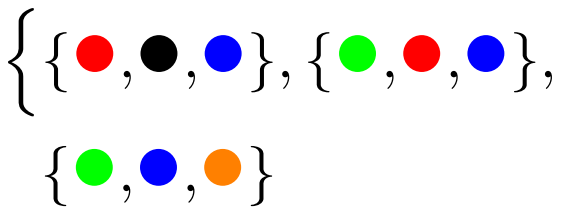


Problem Definition

Problem (Triangle Enumeration)

Input: An undirected unweighted graph $G = (V, E)$.
Task: List all triangles in G.

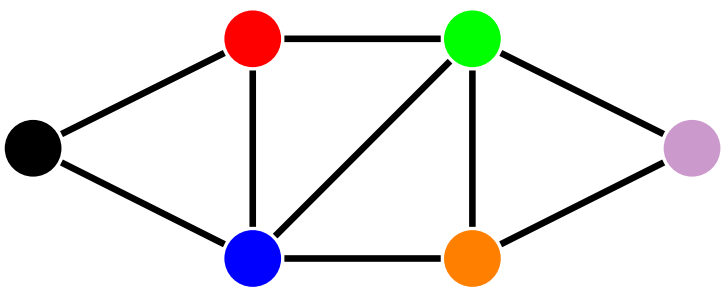


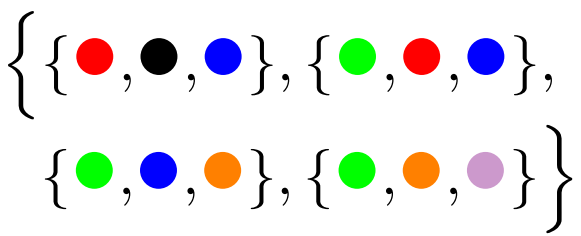


Problem Definition

Problem (Triangle Enumeration)

Input: An undirected unweighted graph $G = (V, E)$.
Task: List all triangles in G.





Related Work

1. Triangle Detection and Triangle Counting in non-standard computation models:
   - Streaming Model

     [Becchetti et al.: ACM SIGKDD, 2008]
   - Quantum Computing

     [Lee, Magniez, Santha: Algorithmica, 2017]
   - MapReduce

     [Park et al.: CIKM, 2014]

Related Work

1. Triangle Detection and Triangle Counting in non-standard computation models:
   - Streaming Model

     [Becchetti et al.: ACM SIGKDD, 2008]
   - Quantum Computing

     [Lee, Magniez, Santha: Algorithmica, 2017]
   - MapReduce

     [Park et al.: CIKM, 2014]
2. Best algorithms: $O(n^3)$ and $O(m^{1.5})$

Related Work

1. Triangle Detection and Triangle Counting in non-standard computation models:
   - Streaming Model

     [Becchetti et al.: ACM SIGKDD, 2008]
   - Quantum Computing

     [Lee, Magniez, Santha: Algorithmica, 2017]
   - MapReduce

     [Park et al.: CIKM, 2014]
2. Best algorithms: $O(n^3)$ and $O(m^{1.5})$
3. FPT in P (e. g. $O(n^3)$ vs. $O(k^2 \cdot n^2)$)

   [Giannopoulou, Mertzios, Niedermeier: TCS, 2015]

Related Work

1. Triangle Detection and Triangle Counting in non-standard computation models:
   - Streaming Model

     [Becchetti et al.: ACM SIGKDD, 2008]
   - Quantum Computing

     [Lee, Magniez, Santha: Algorithmica, 2017]
   - MapReduce

     [Park et al.: CIKM, 2014]
2. Best algorithms: $O(n^3)$ and $O(m^{1.5})$
3. FPT in P (e. g. $O(n^3)$ vs. $O(k^2 \cdot n^2)$)

   [Giannopoulou, Mertzios, Niedermeier: TCS, 2015]
   - Parameterized by degeneracy d in $O(m \cdot d)$ time.

     [Chiba and Nishizeki: SIAM Journal on Computing, 1985]

Related Work

1. Triangle Detection and Triangle Counting in non-standard computation models:
   - Streaming Model

     [Becchetti et al.: ACM SIGKDD, 2008]
   - Quantum Computing

     [Lee, Magniez, Santha: Algorithmica, 2017]
   - MapReduce

     [Park et al.: CIKM, 2014]
2. Best algorithms: $O(n^3)$ and $O(m^{1.5})$
3. FPT in P (e. g. $O(n^3)$ vs. $O(k^2 \cdot n^2)$)

   [Giannopoulou, Mertzios, Niedermeier: TCS, 2015]
   - Parameterized by degeneracy d in $O(m \cdot d)$ time.

     [Chiba and Nishizeki: SIAM Journal on Computing, 1985]
   - Parameterized by vertex cover K and the maximum degree Δ_K in K in $O(K \cdot \Delta_k^2)$ time.

     [Green and Bader: IEEE Social Computing, 2013]

Our Results

Table: $\#\mathsf{T}$: number of triangles; Δ: maximum degree

parameter k	result

Our Results

Table: #T : number of triangles; Δ: maximum degree

	parameter k	result
solving		
kernel		
hard		

Our Results

Table: #T : number of triangles; Δ: maximum degree

	parameter k	result
solving	distance to cographs	$O(\#\mathsf{T} + n + m \cdot k)$ time
	distance to d-degenerate + maximum degree Δ	$O(k \cdot \Delta^2 + n \cdot d^2)$ time
	clique-width	$O(n^2 + n \cdot k^2 + \# T)$ time
kernel		
hard		

Our Results

Table: #T : number of triangles; Δ: maximum degree

	parameter k	result
solving	distance to cographs	$O(\#\mathsf{T} + n + m \cdot k)$ time
	distance to d-degenerate + maximum degree Δ	$O(k \cdot \Delta^2 + n \cdot d^2)$ time
	clique-width	$O(n^2 + n \cdot k^2 + \# T)$ time
kernel	feedback edge number	linear size and linear time
	distance to d-degenerate	$O(k \cdot 2^k)$ size and $O(n \cdot d \cdot (k + 2^k))$ time
hard		

Our Results

Table: #T : number of triangles; Δ: maximum degree

	parameter k	result
solving	distance to cographs	$O(\#\mathsf{T} + n + m \cdot k)$ time
	distance to d-degenerate + maximum degree Δ	$O(k \cdot \Delta^2 + n \cdot d^2)$ time
	clique-width	$O(n^2 + n \cdot k^2 + \# T)$ time
kernel	feedback edge number	linear size and linear time
	distance to d-degenerate	$O(k \cdot 2^k)$ size and $O(n \cdot d \cdot (k + 2^k))$ time
hard	domination number, chromatic number, and diameter	for $k \geq 9$ as hard as in general graphs

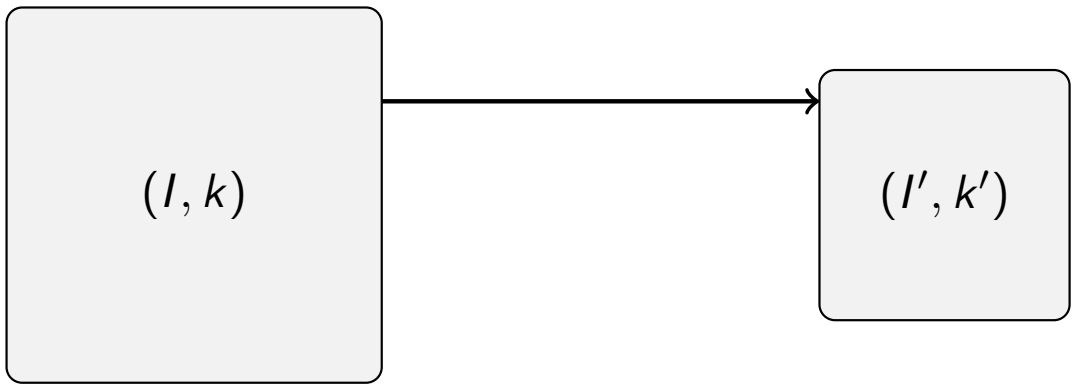
Our Results

Table: #T : number of triangles; Δ: maximum degree

	parameter k	result
solving	distance to cographs	$O(\#\mathsf{T} + n + m \cdot k)$ time
	distance to d-degenerate + maximum degree Δ	$O(k \cdot \Delta^2 + n \cdot d^2)$ time
	clique-width	$O(n^2 + n \cdot k^2 + \# T)$ time
kernel	feedback edge number	linear size and linear time
	distance to d-degenerate	$O(k \cdot 2^k)$ size and $O(n \cdot d \cdot (k + 2^k))$ time
hard	domination number, chromatic number, and diameter	for $k \geq 9$ as hard as in general graphs

Kernelization

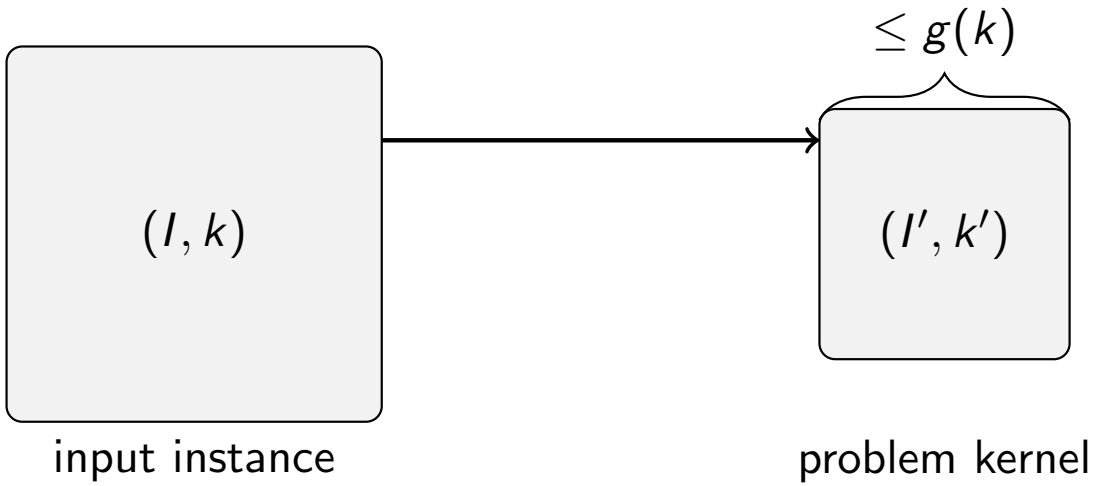
Kernels (for Decision Problems)



Definition (kernel)

A **reduction to a problem kernel** is a mapping of instances (I, k) to instances (I', k') such that

Kernels (for Decision Problems)

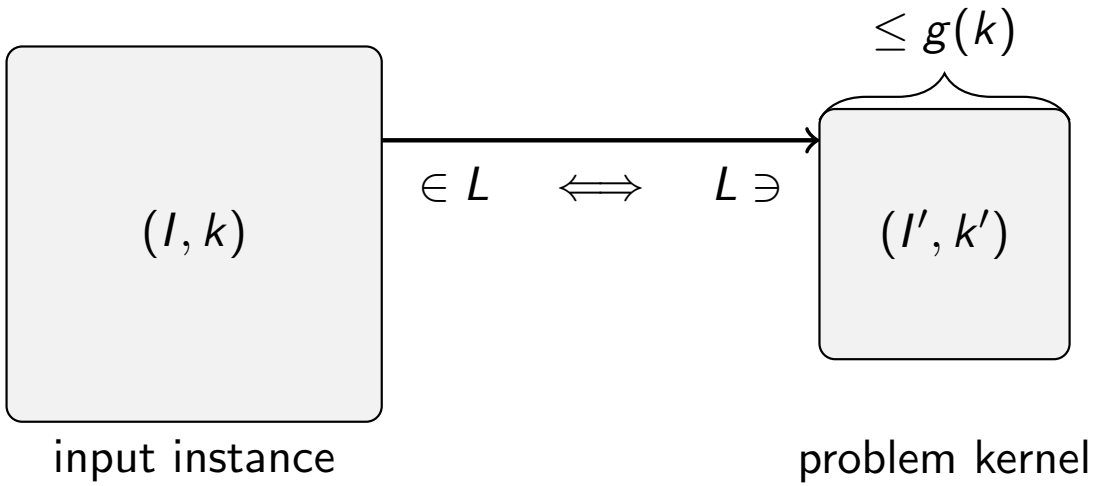


Definition (kernel)

A **reduction to a problem kernel** is a mapping of instances (I, k) to instances (I', k') such that

- $|I'| + k' \leq g(k)$,

Kernels (for Decision Problems)

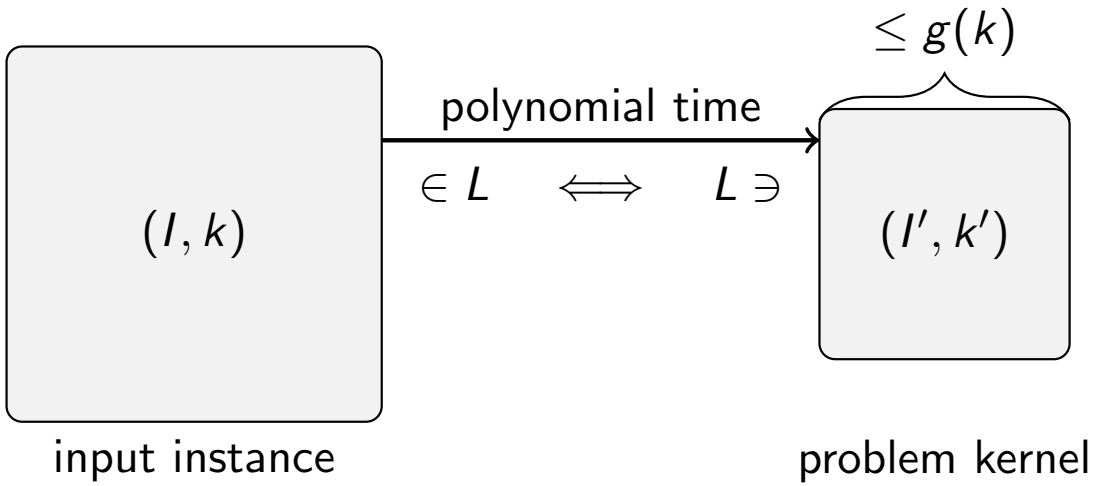


Definition (kernel)

A **reduction to a problem kernel** is a mapping of instances (I, k) to instances (I', k') such that

- $|I'| + k' \leq g(k)$,
- $(I, k) \in L \Leftrightarrow (I', k') \in L$, and

Kernels (for Decision Problems)

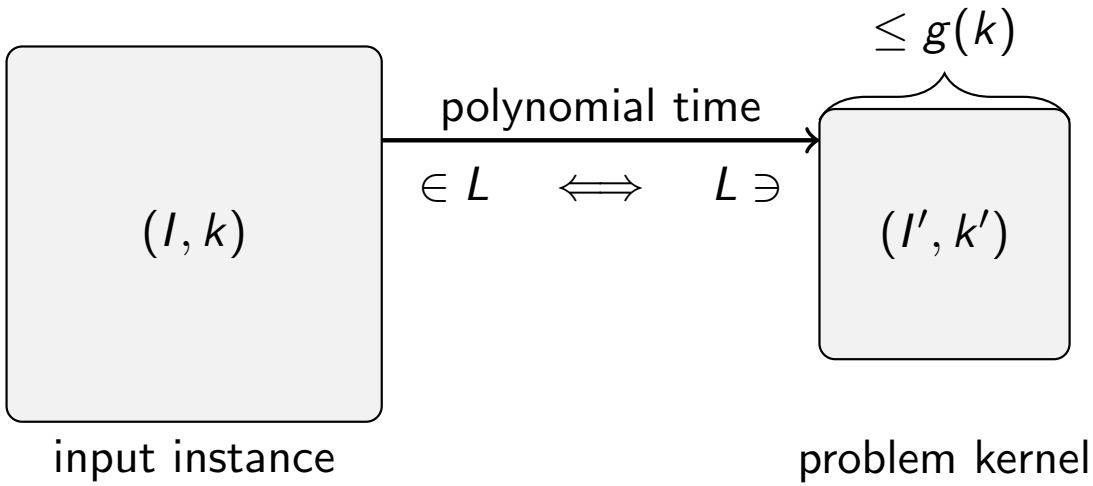


Definition (kernel)

A **reduction to a problem kernel** is a mapping of instances (I, k) to instances (I', k') such that

- $|I'| + k' \leq g(k)$,
- $(I, k) \in L \Leftrightarrow (I', k') \in L$, and
- it can be computed in polynomial time.

Kernels (for Decision Problems)

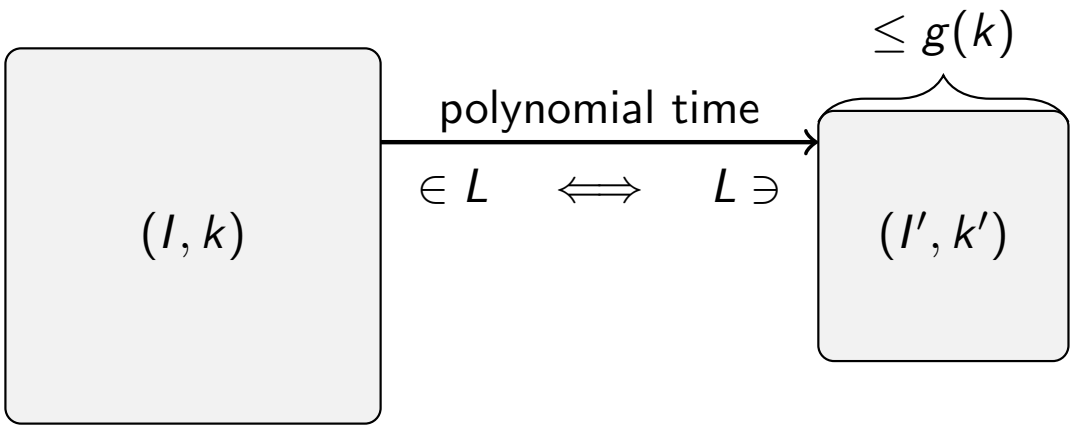


Definition (kernel)

A **reduction to a problem kernel** is a mapping of instances (I, k) to instances (I', k') such that

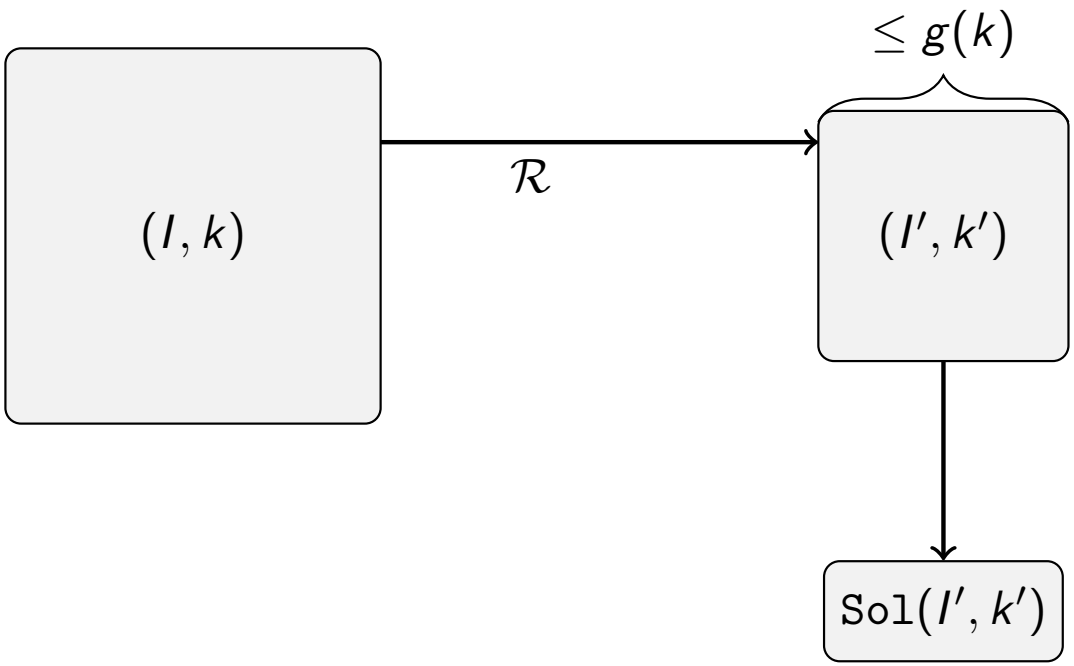
- $|I'| + k' \leq g(k)$,
- $(I, k) \in L \Leftrightarrow (I', k') \in L$, and
- it can be computed in polynomial time.

Enum-Kernels (for Enumeration Tasks)



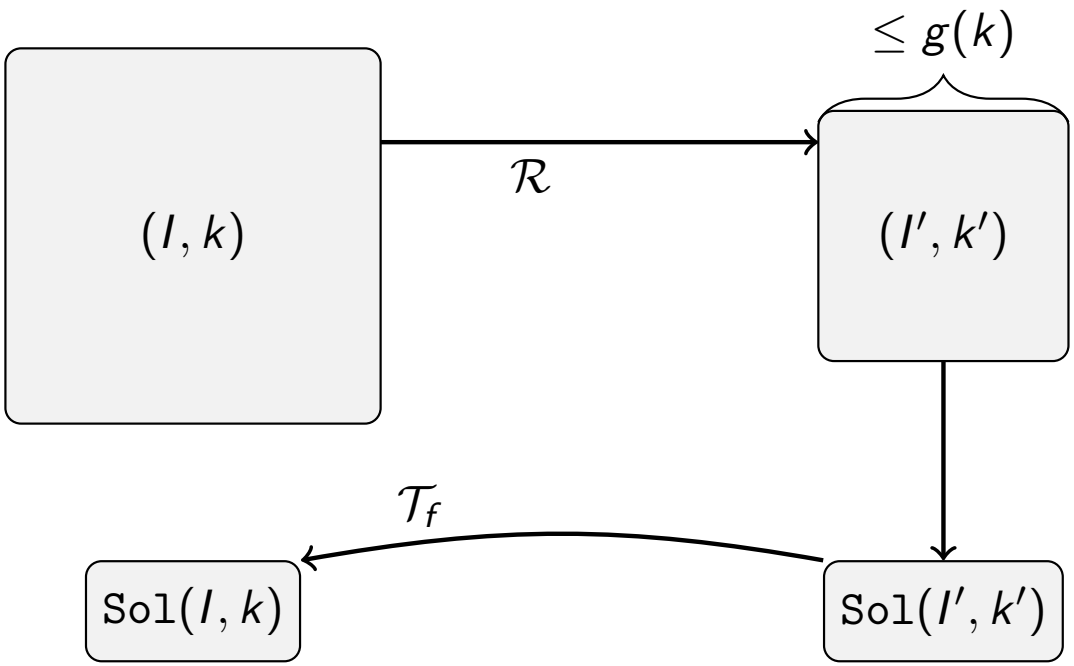
(I, k)	input
(I', k')	kernel

Enum-Kernels (for Enumeration Tasks)



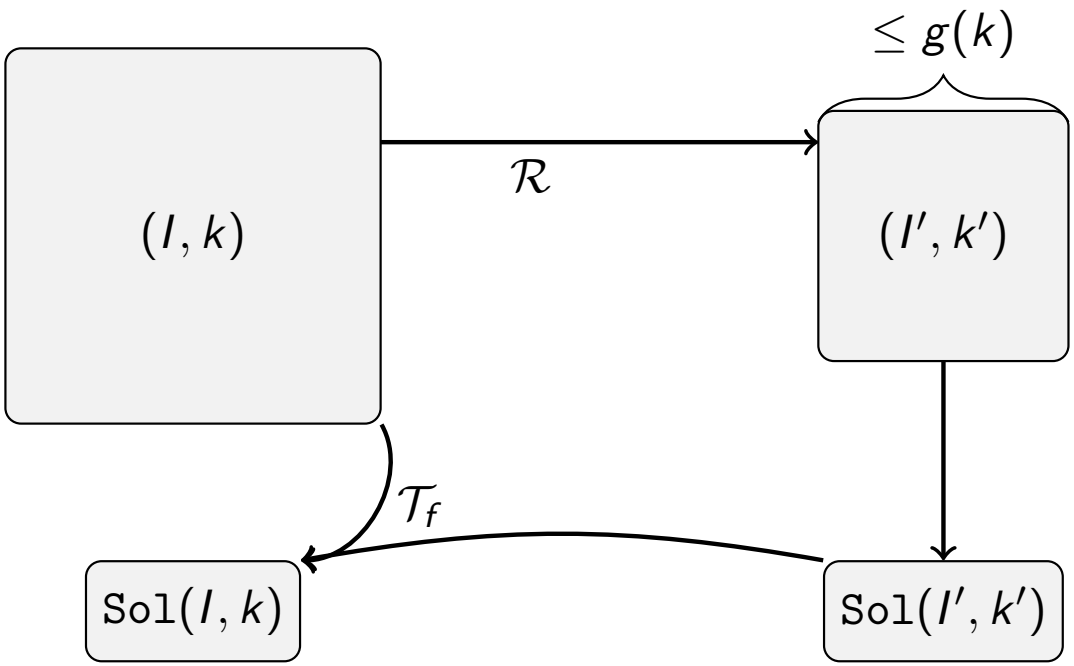
(I, k)	input
(I', k')	kernel
$\mathcal{R}$	kernelization
$\texttt{Sol}$	(all) solutions

Enum-Kernels (for Enumeration Tasks)



(I, k)	input
(I', k')	kernel
$\mathcal{R}$	kernelization
$\texttt{Sol}$	(all) solutions
f	lift function
$\mathcal{T}_f$	algorithm for f

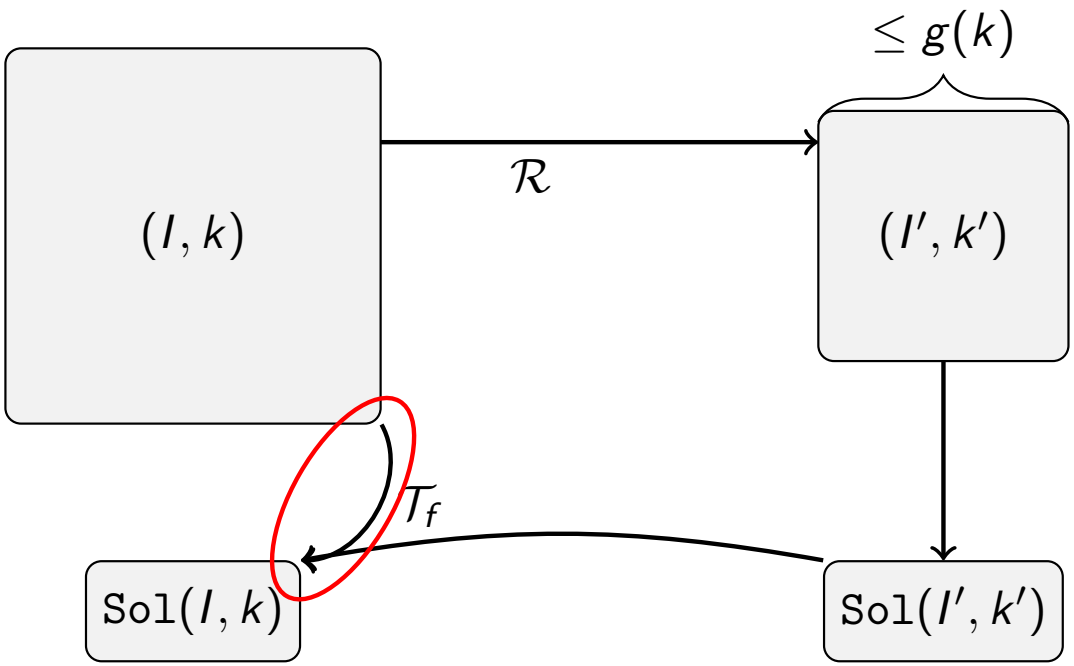
Enum-Kernels (for Enumeration Tasks)



(I, k)	input
(I', k')	kernel
$\mathcal{R}$	kernelization
$\mathtt{Sol}$	(all) solutions
f	lift function
$\mathcal{T}_f$	algorithm for f

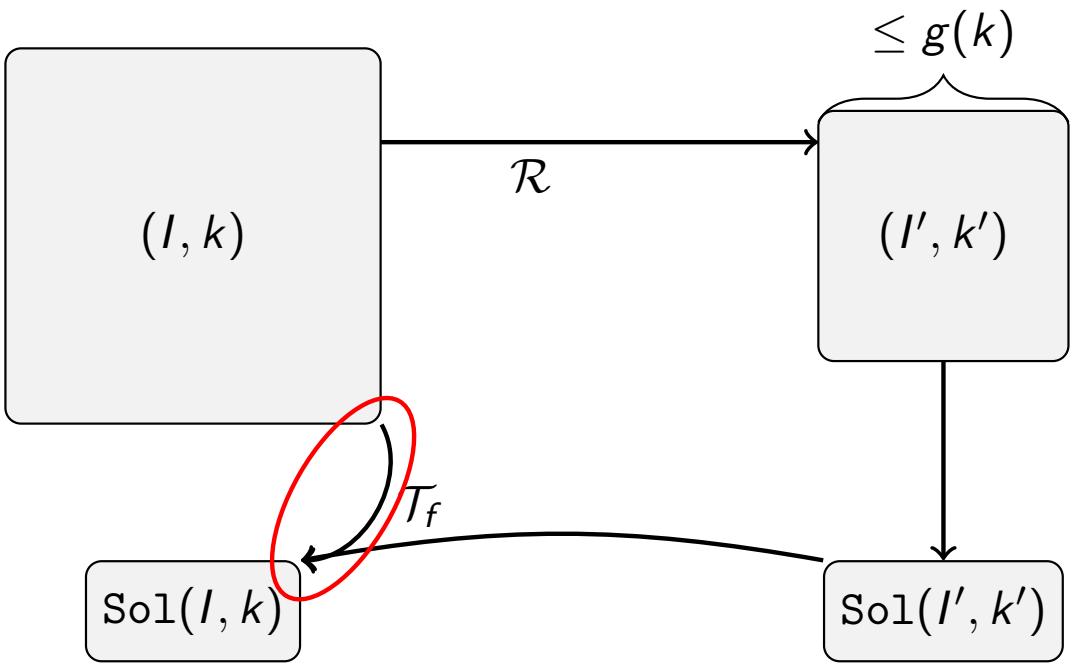
[Creignou et al.: Theory of Computing Systems, 2017]

Disadvantage of Enum-Kernels



(I, k)	input
(I', k')	kernel
$\mathcal{R}$	kernelization
$\texttt{Sol}$	(all) solutions
f	lift function
$\mathcal{T}_f$	algorithm for f

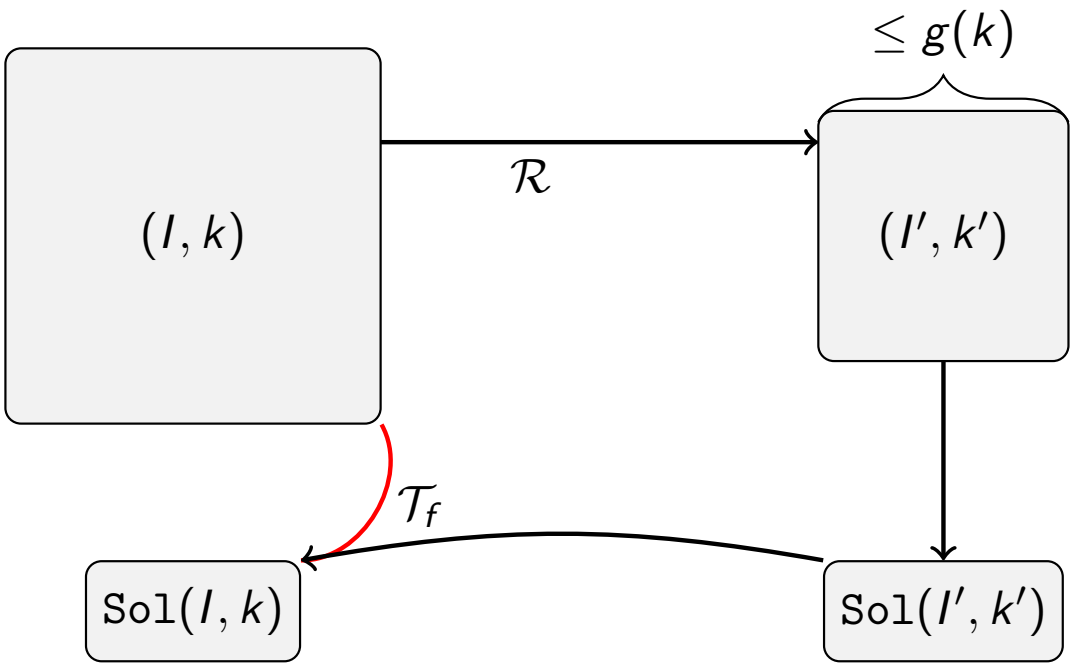
Disadvantage of Enum-Kernels



(I, k)	input
(I', k')	kernel
$\mathcal{R}$	kernelization
$\texttt{Sol}$	(all) solutions
f	lift function
$\mathcal{T}_f$	algorithm for f

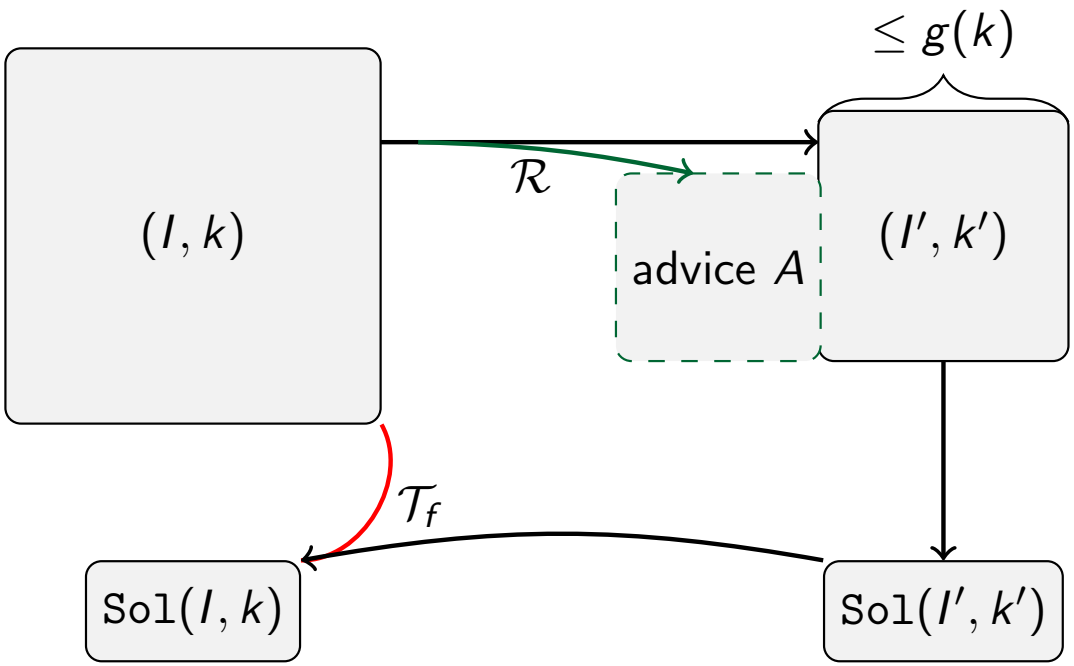
Disadvantage: Kernel instance is insufficient!
(input instance is still needed)

Enum-Advice Kernels



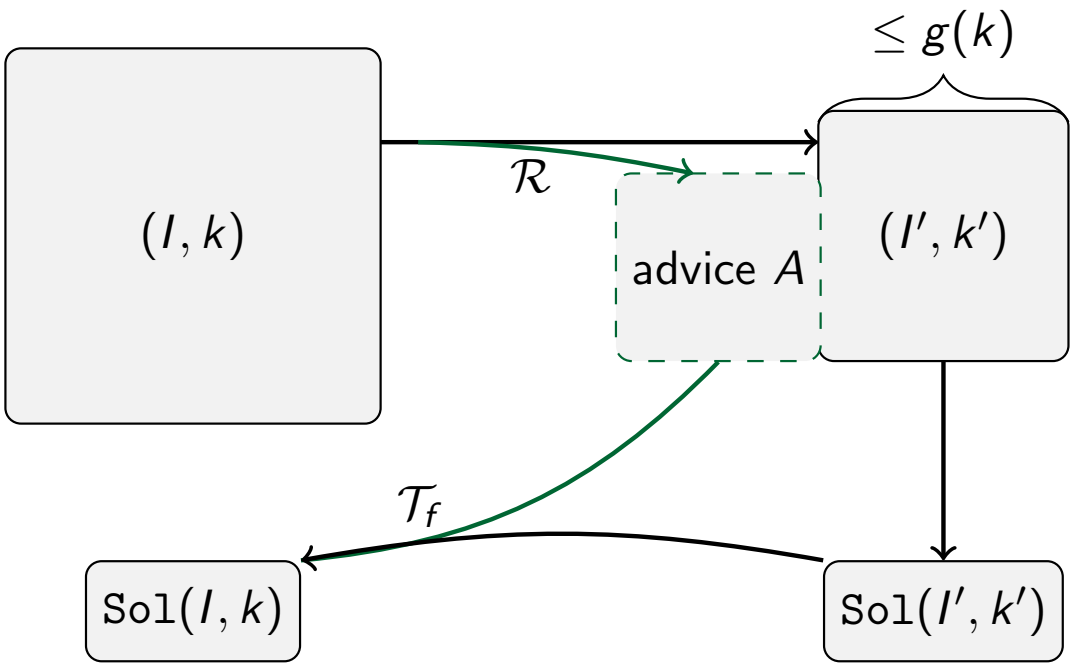
(I, k)	input
(I', k')	kernel
$\mathcal{R}$	kernelization
$\texttt{Sol}$	(all) solutions
f	lift function
$\mathcal{T}_f$	algorithm for f

Enum-Advice Kernels



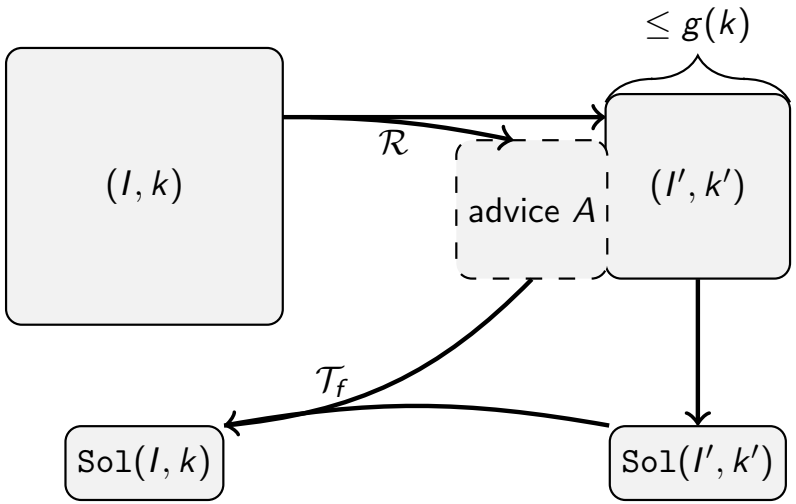
(I, k)	input
(I', k')	kernel
$\mathcal{R}$	kernelization
$\texttt{Sol}$	(all) solutions
f	lift function
$\mathcal{T}_f$	algorithm for f

Enum-Advice Kernels



(I, k)	input
(I', k')	kernel
$\mathcal{R}$	kernelization
$\texttt{Sol}$	(all) solutions
f	lift function
$\mathcal{T}_f$	algorithm for f

Enum-Advice Kernels

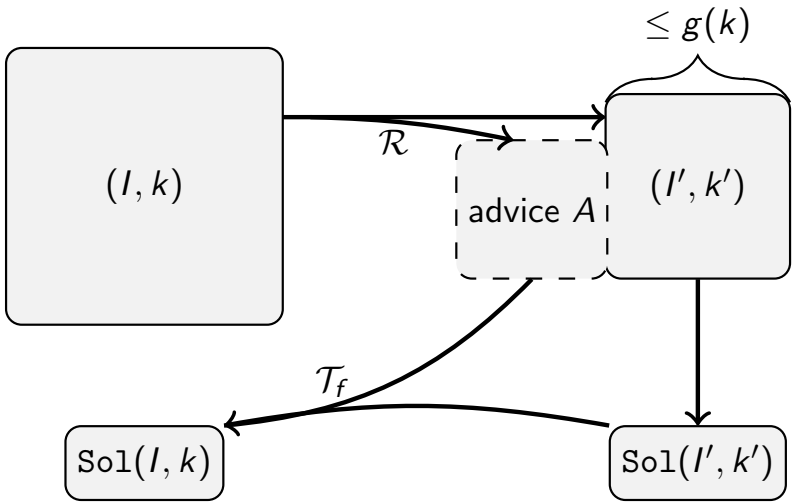


(I, k)	input
(I', k')	kernel
$\mathcal{R}$	kernelization
$\texttt{Sol}$	(all) solutions
f	lift function
$\mathcal{T}_f$	algorithm for f

Definition (enum-advice kernel)

A reduction to an **enum-advice kernel** is a mapping of instances (I, k) to instances (I', k', A) such that

Enum-Advice Kernels



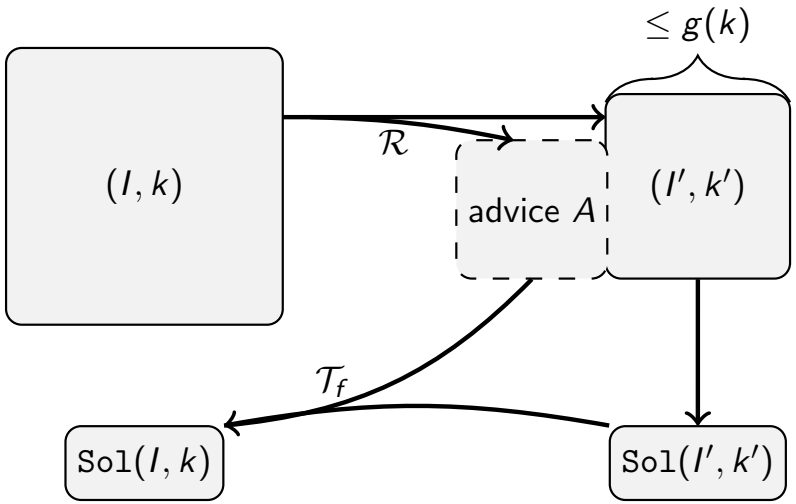
(I, k)	input
(I', k')	kernel
$\mathcal{R}$	kernelization
$\texttt{Sol}$	(all) solutions
f	lift function
$\mathcal{T}_f$	algorithm for f

Definition (enum-advice kernel)

A reduction to an **enum-advice kernel** is a mapping of instances (I, k) to instances (I', k', A) such that

- $|I'|, k \leq g(k)$,

Enum-Advice Kernels



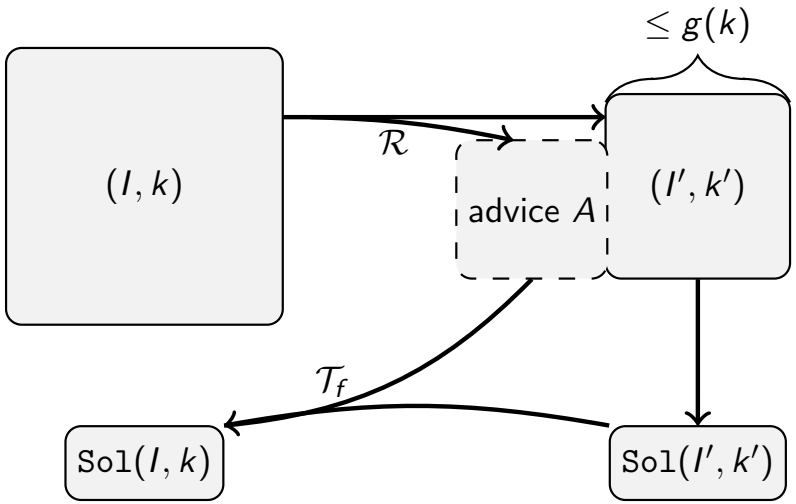
(I, k)	input
(I', k')	kernel
$\mathcal{R}$	kernelization
$\texttt{Sol}$	(all) solutions
f	lift function
$\mathcal{T}_f$	algorithm for f

Definition (enum-advice kernel)

A reduction to an **enum-advice kernel** is a mapping of instances (I, k) to instances (I', k', A) such that

- $|I'|, k \leq g(k)$,
- every solution is listed exactly once, and

Enum-Advice Kernels



(I, k)	input
(I', k')	kernel
$\mathcal{R}$	kernelization
$\texttt{Sol}$	(all) solutions
f	lift function
$\mathcal{T}_f$	algorithm for f

Definition (enum-advice kernel)

A reduction to an **enum-advice kernel** is a mapping of instances (I, k) to instances (I', k', A) such that

- $|I'|, k \leq g(k)$,
- every solution is listed exactly once, and
- $\mathcal{R}$ and $\mathcal{T}_f$ can be computed fast enough.

Distance to d-Degenerate Graphs

Definition (degeneracy)

A graph has degeneracy d if every subgraph has a vertex of degree at most d.

Distance to d-Degenerate Graphs

Definition (degeneracy)

A graph has degeneracy d if every subgraph has a vertex of degree at most d.

Definition (distance to d-degenerate graphs)

The distance to d-degenerate graphs is the least number of vertices that have to deleted from a graph in order to reduce its degeneracy to at most d.

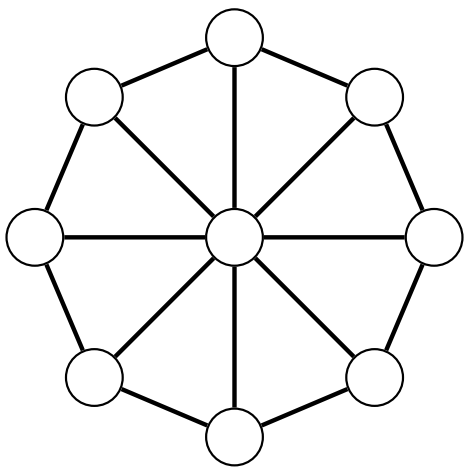
Distance to d-Degenerate Graphs

Definition (degeneracy)

A graph has degeneracy d if every subgraph has a vertex of degree at most d.

Definition (distance to d-degenerate graphs)

The distance to d-degenerate graphs is the least number of vertices that have to deleted from a graph in order to reduce its degeneracy to at most d.



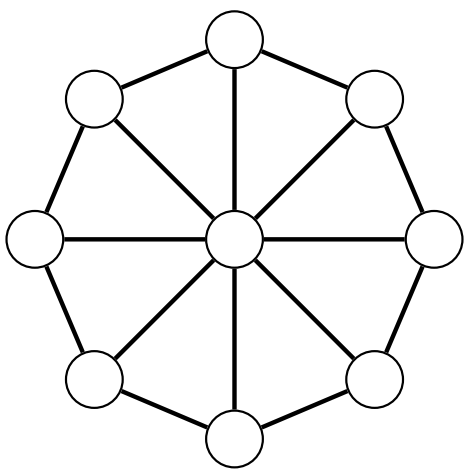
Distance to d-Degenerate Graphs

Definition (degeneracy)

A graph has degeneracy d if every subgraph has a vertex of degree at most d.

Definition (distance to d-degenerate graphs)

The distance to d-degenerate graphs is the least number of vertices that have to deleted from a graph in order to reduce its degeneracy to at most d.



$d = 1$?

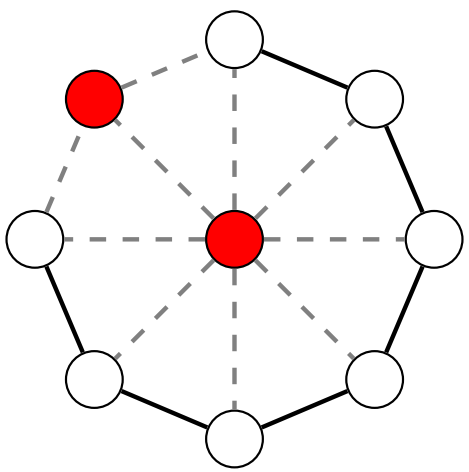
Distance to d-Degenerate Graphs

Definition (degeneracy)

A graph has degeneracy d if every subgraph has a vertex of degree at most d.

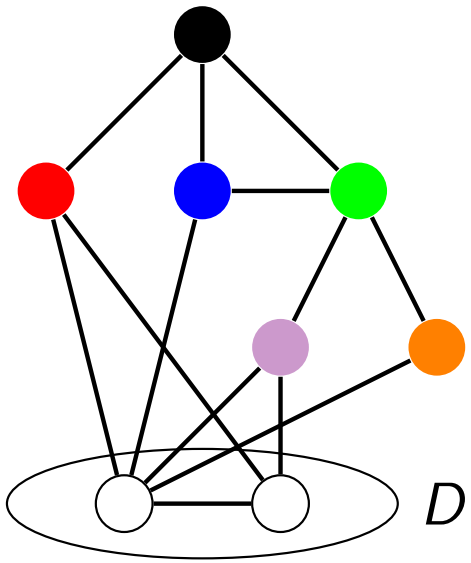
Definition (distance to d-degenerate graphs)

The distance to d-degenerate graphs is the least number of vertices that have to deleted from a graph in order to reduce its degeneracy to at most d.



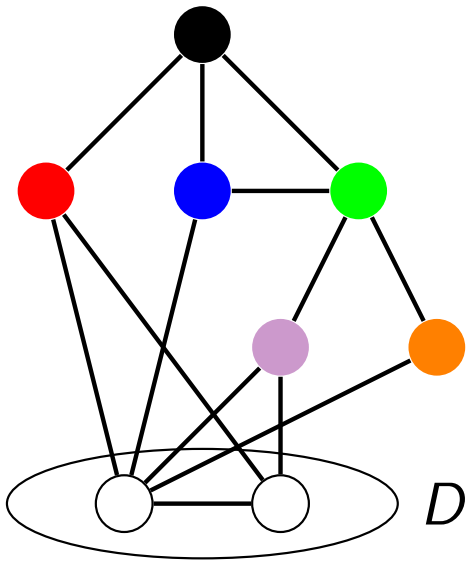
$d = 1$

Enum-Advice Kernel for almost d-Degenerate Graphs



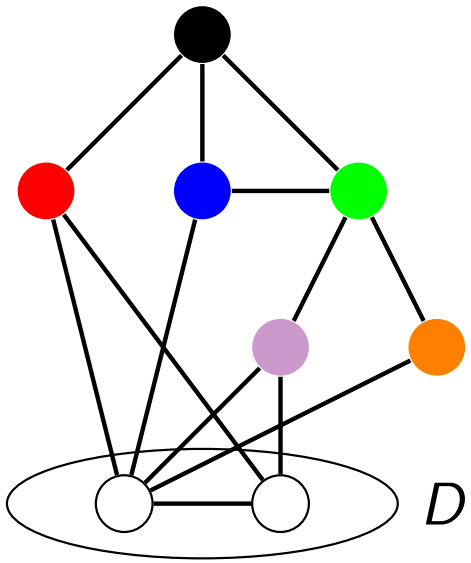
Enum-Advice Kernel for almost d-Degenerate Graphs

- Compute the set T' of triangles with at most one vertex in D.



Enum-Advice Kernel for almost d-Degenerate Graphs

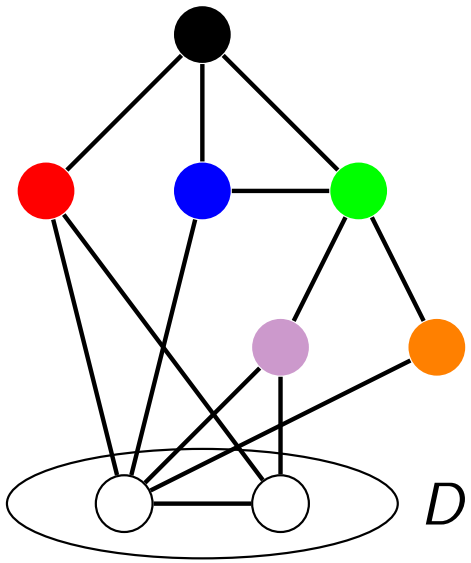
- Compute the set T' of triangles with at most one vertex in D.



$T' = \{\{$ ● ● ● $\}\}$

Enum-Advice Kernel for almost d-Degenerate Graphs

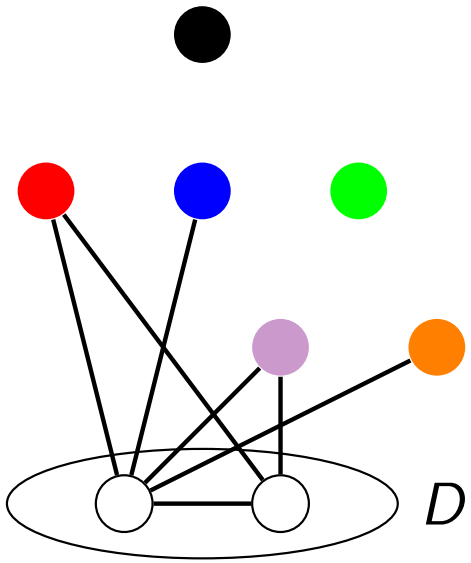
- Compute the set T' of triangles with at most one vertex in D.
- Delete all edges which have no endpoint in D.



$T' = \{\{$ ● ● ● $\}\}$

Enum-Advice Kernel for almost d-Degenerate Graphs

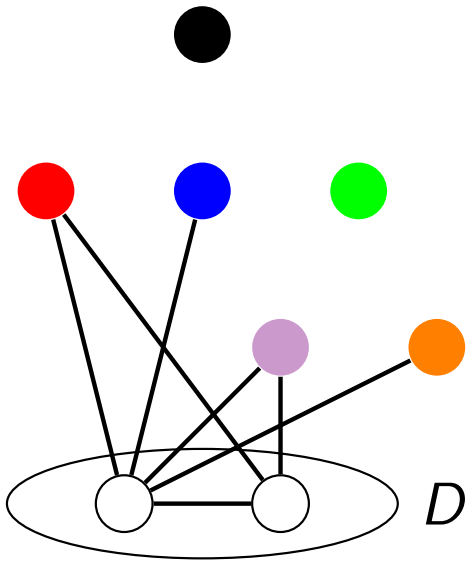
- Compute the set T' of triangles with at most one vertex in D.
- Delete all edges which have no endpoint in D.



$T' = \{\{$ ● ● ● $\}\}$

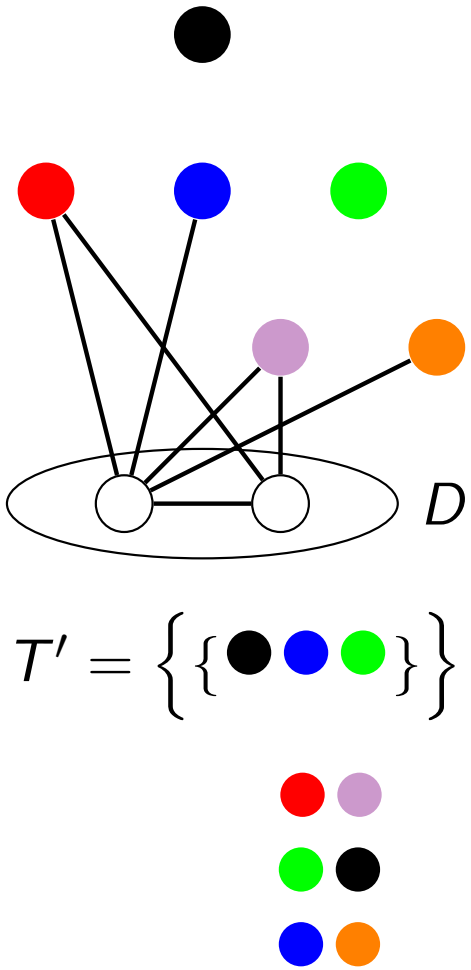
Enum-Advice Kernel for almost d-Degenerate Graphs

- Compute the set T' of triangles with at most one vertex in D.
- Delete all edges which have no endpoint in D.
- Compute all twins (vertices with exactly the same neighbors).



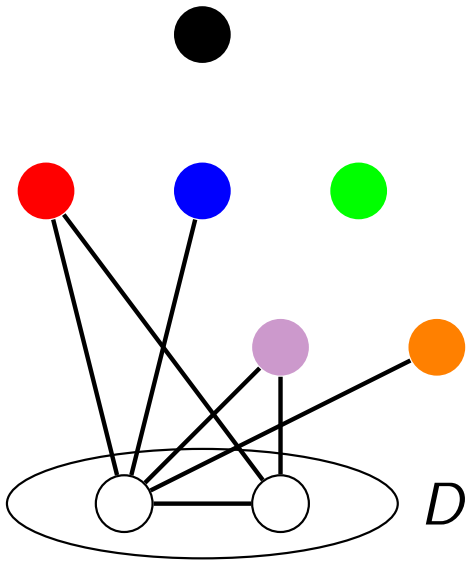
Enum-Advice Kernel for almost d-Degenerate Graphs

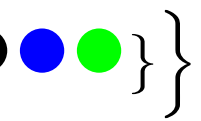
- Compute the set T' of triangles with at most one vertex in D.
- Delete all edges which have no endpoint in D.
- Compute all twins (vertices with exactly the same neighbors).

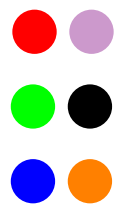


Enum-Advice Kernel for almost d-Degenerate Graphs

- Compute the set T' of triangles with at most one vertex in D.
- Delete all edges which have no endpoint in D.
- Compute all twins (vertices with exactly the same neighbors).
- Delete all but one twin.

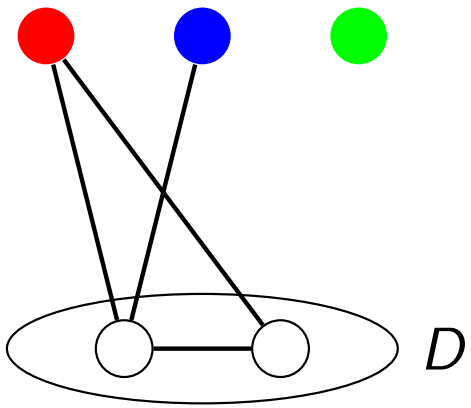


$T' =$ 



Enum-Advice Kernel for almost d-Degenerate Graphs

- Compute the set T' of triangles with at most one vertex in D.
- Delete all edges which have no endpoint in D.
- Compute all twins (vertices with exactly the same neighbors).
- Delete all but one twin.

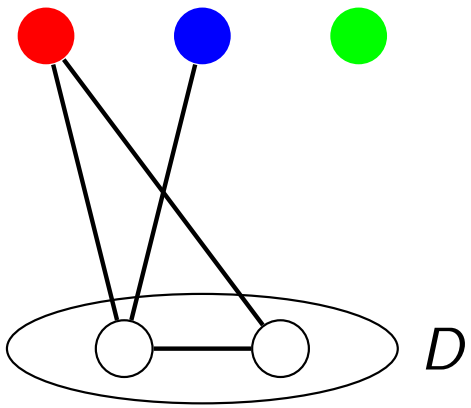


$T' = \{\{$ ● ● ● $\}\}$

M : ● ← ● ●
● ← ● ●
● ← ● ●

Enum-Advice Kernel for almost d-Degenerate Graphs

- Compute the set T' of triangles with at most one vertex in D.
- Delete all edges which have no endpoint in D.
- Compute all twins (vertices with exactly the same neighbors).
- Delete all but one twin.
- Set parameter $k' = k$ and advice $A = (T', M)$.



$T' = \{\{$ ● ● ● $\}\}$

M : ● ← ● ●
● ← ● ●
● ← ● ●

Computational Hardness

Hardness in P?

Can **all** parameters be used to design $f(k) \cdot (n + m)$ time algorithms?

Hardness in P?

Can **all** parameters be used to design $f(k) \cdot (n + m)$ time algorithms?

How can we prove that certain parameters are unsuited?

Hardness in P?

Can **all** parameters be used to design $f(k) \cdot (n + m)$ time algorithms?

How can we prove that certain parameters are unsuited?

Use **reductions!**

Hardness in P?

Can **all** parameters be used to design $f(k) \cdot (n + m)$ time algorithms?

How can we prove that certain parameters are unsuited?

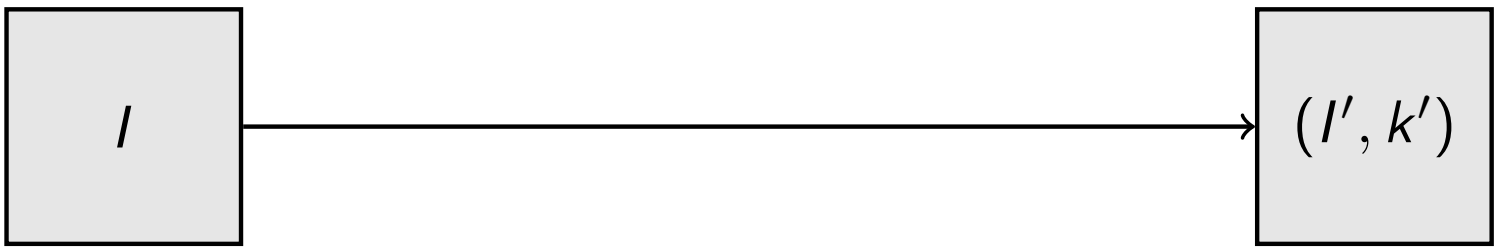
Use **reductions!**
($\rightsquigarrow$ Triangle Detection)

General-Problem-Hardness

Definition (General-Problem-hardness)

We call a problem L c-**General-Problem-hard**(f) if there exists an algorithm $\mathcal{A}$ transforming any input instance I of L into a new instance (I', k') of a parameterized version of L such that

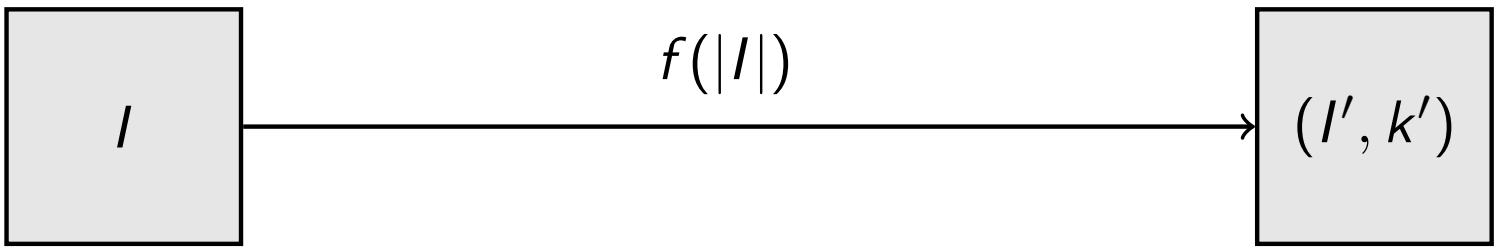
General-Problem-Hardness



Definition (General-Problem-hardness)

We call a problem L c-**General-Problem-hard**(f) if there exists an algorithm $\mathcal{A}$ transforming any input instance I of L into a new instance (I', k') of a parameterized version of L such that

General-Problem-Hardness

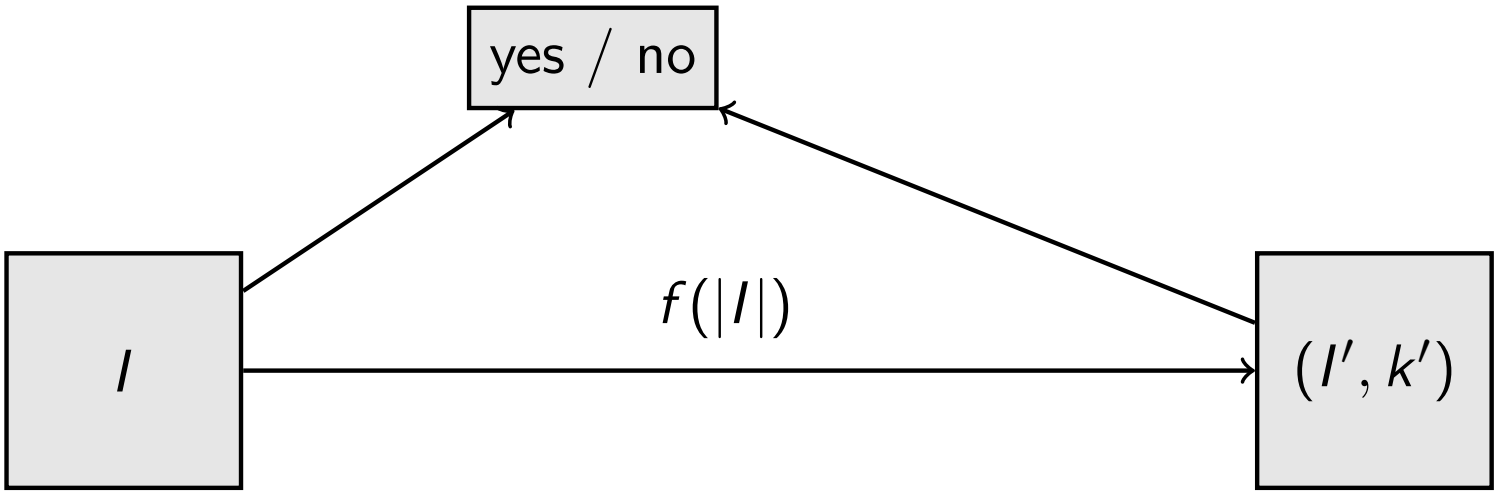


Definition (General-Problem-hardness)

We call a problem L c-**General-Problem-hard**(f) if there exists an algorithm $\mathcal{A}$ transforming any input instance I of L into a new instance (I', k') of a parameterized version of L such that

- $\mathcal{A}$ runs in $f(|I|)$ time,

General-Problem-Hardness

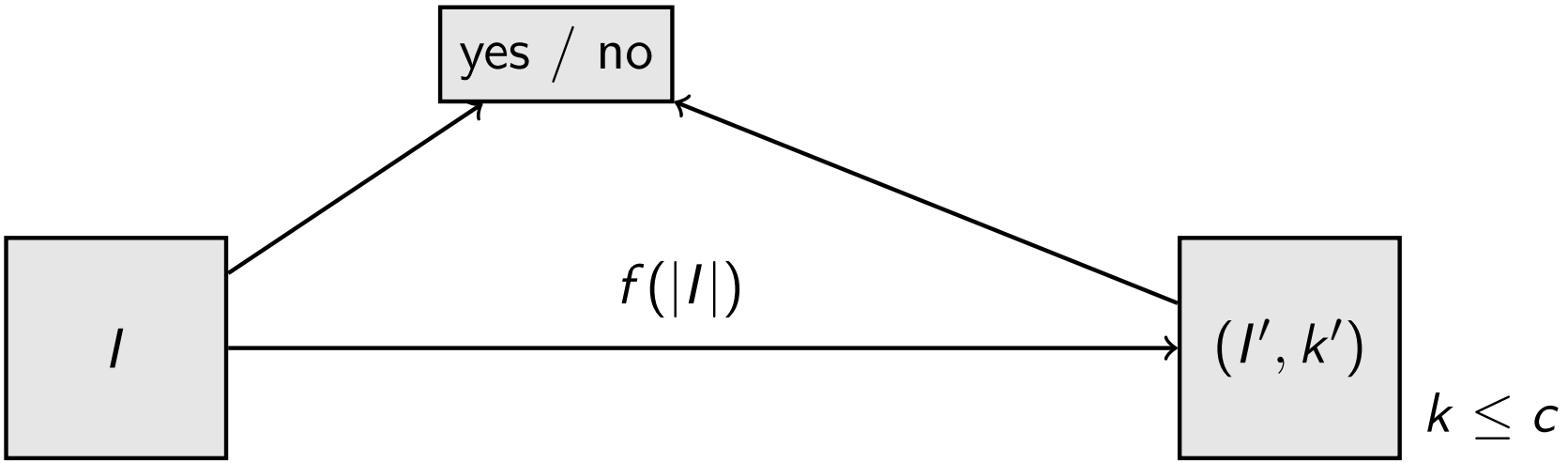


Definition (General-Problem-hardness)

We call a problem L c-**General-Problem-hard**(f) if there exists an algorithm $\mathcal{A}$ transforming any input instance I of L into a new instance (I', k') of a parameterized version of L such that

- $\mathcal{A}$ runs in $f(|I|)$ time,
- $I \in L \Leftrightarrow I' \in L$,

General-Problem-Hardness

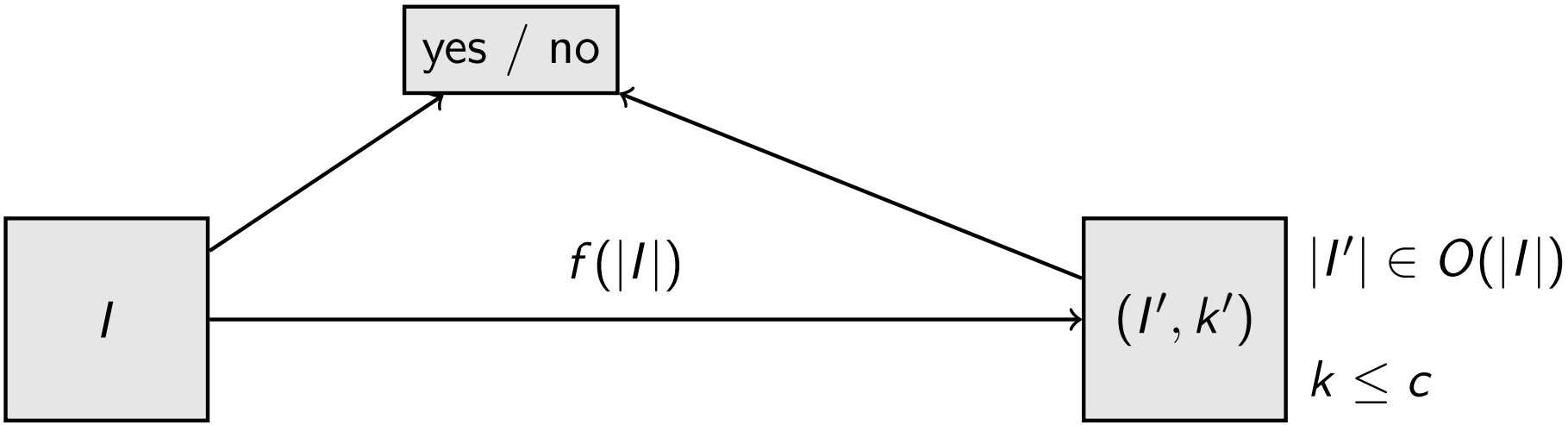


Definition (General-Problem-hardness)

We call a problem L c-**General-Problem-hard**(f) if there exists an algorithm $\mathcal{A}$ transforming any input instance I of L into a new instance (I', k') of a parameterized version of L such that

- $\mathcal{A}$ runs in $f(|I|)$ time,
- $k' \leq c$, and
- $I \in L \Leftrightarrow I' \in L$,

General-Problem-Hardness

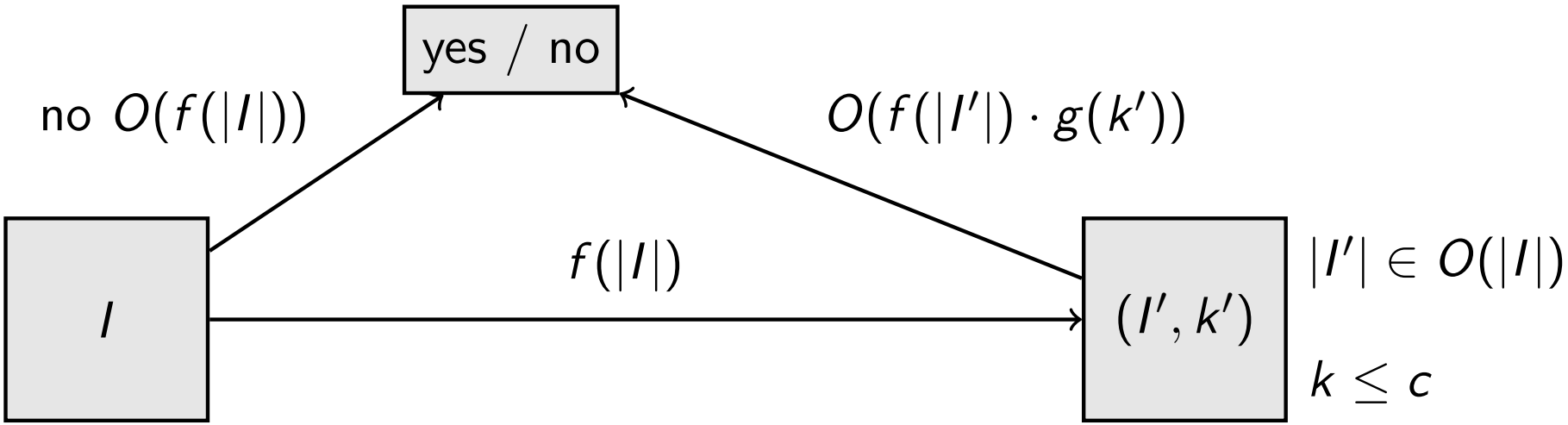


Definition (General-Problem-hardness)

We call a problem L c-**General-Problem-hard**(f) if there exists an algorithm $\mathcal{A}$ transforming any input instance I of L into a new instance (I', k') of a parameterized version of L such that

- $\mathcal{A}$ runs in $f(|I|)$ time,
- $k' \leq c$, and
- $I \in L \Leftrightarrow I' \in L$,
- $|I'| \in O(|I|)$.

General-Problem-Hardness

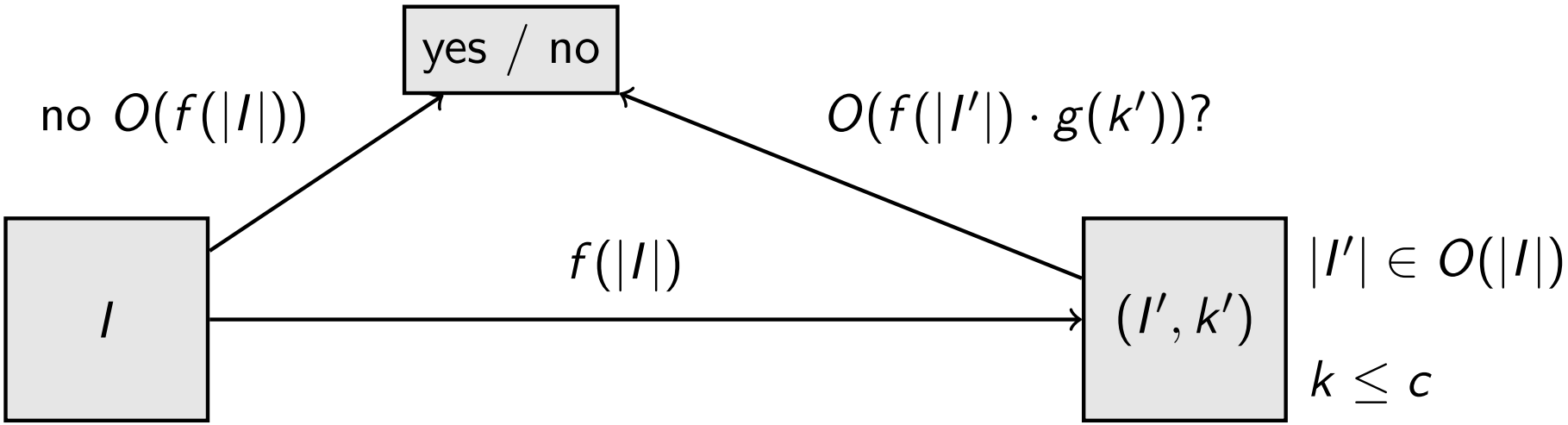


Definition (General-Problem-hardness)

We call a problem L c-**General-Problem-hard**(f) if there exists an algorithm $\mathcal{A}$ transforming any input instance I of L into a new instance (I', k') of a parameterized version of L such that

- $\mathcal{A}$ runs in $f(|I|)$ time,
- $k' \leq c$, and
- $I \in L \Leftrightarrow I' \in L$,
- $|I'| \in O(|I|)$.

General-Problem-Hardness

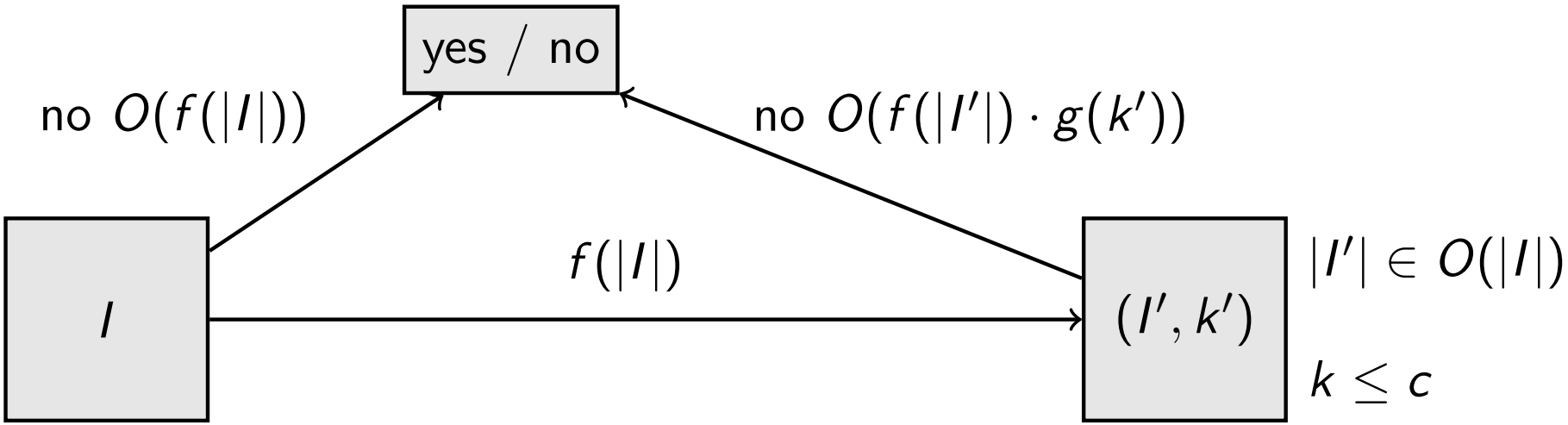


Definition (General-Problem-hardness)

We call a problem L c-**General-Problem-hard**(f) if there exists an algorithm $\mathcal{A}$ transforming any input instance I of L into a new instance (I', k') of a parameterized version of L such that

- $\mathcal{A}$ runs in $f(|I|)$ time,
- $k' \leq c$, and
- $I \in L \Leftrightarrow I' \in L$,
- $|I'| \in O(|I|)$.

General-Problem-Hardness

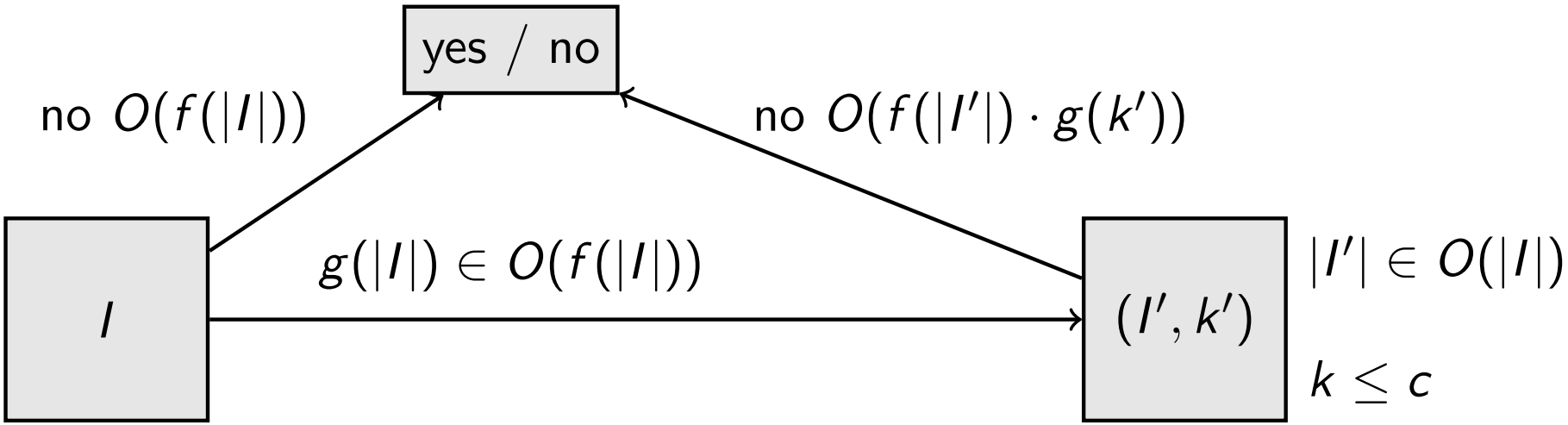


Definition (General-Problem-hardness)

We call a problem L c-**General-Problem-hard**(f) if there exists an algorithm $\mathcal{A}$ transforming any input instance I of L into a new instance (I', k') of a parameterized version of L such that

- $\mathcal{A}$ runs in $f(|I|)$ time,
- $k' \leq c$, and
- $I \in L \Leftrightarrow I' \in L$,
- $|I'| \in O(|I|)$.

General-Problem-Hardness



Definition (General-Problem-hardness)

We call a problem L c-**General-Problem-hard**(f) if there exists an algorithm $\mathcal{A}$ transforming any input instance I of L into a new instance (I', k') of a parameterized version of L such that

- $\mathcal{A}$ runs in $f(|I|)$ time,
- $k' \leq c$, and
- $I \in L \Leftrightarrow I' \in L$,
- $|I'| \in O(|I|)$.

"Useless" Parameters

- domination number

Definition (domination number)

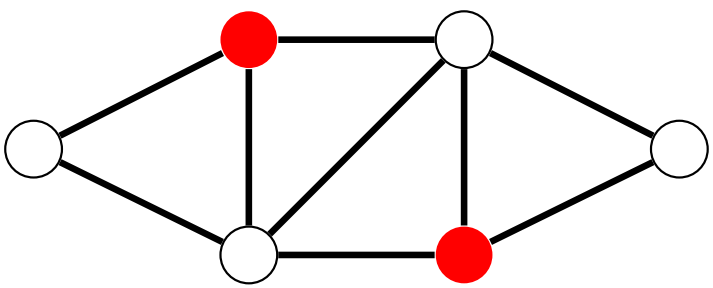
The **domination number** of a graph is the cardinality of a smallest set $\mathcal{D}$ of vertices such that $N[v] \cap \mathcal{D} \neq \emptyset$ for each vertex v.

"Useless" Parameters

- domination number

Definition (domination number)

The **domination number** of a graph is the cardinality of a smallest set $\mathcal{D}$ of vertices such that $N[v] \cap \mathcal{D} \neq \emptyset$ for each vertex v.



"Useless" Parameters

- domination number
- chromatic number

Definition (chromatic number)

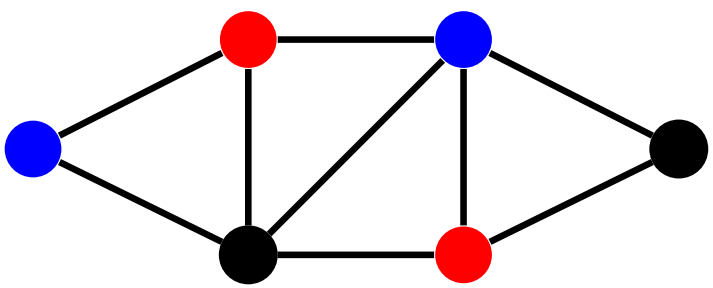
The **chromatic number** of a graph is the least number of colors needed to color each vertex with one color such that the endpoints of an edge have different colors.

"Useless" Parameters

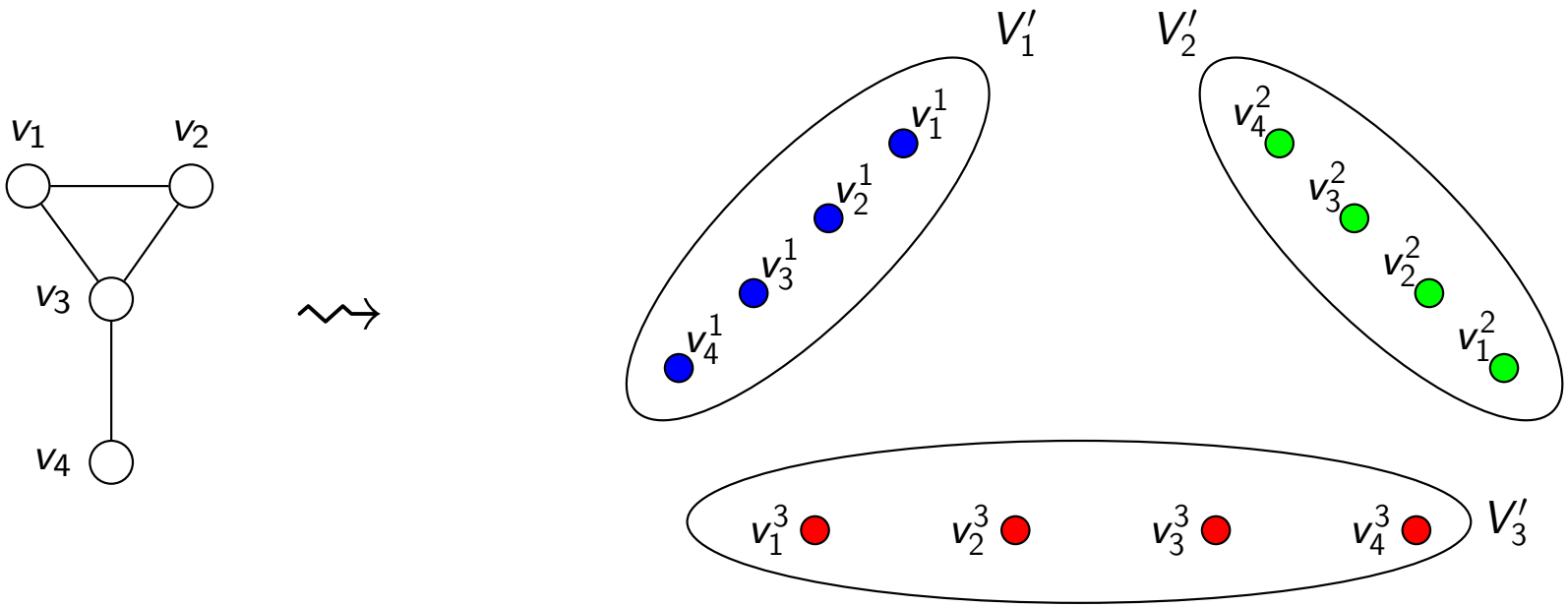
- domination number
- chromatic number

Definition (chromatic number)

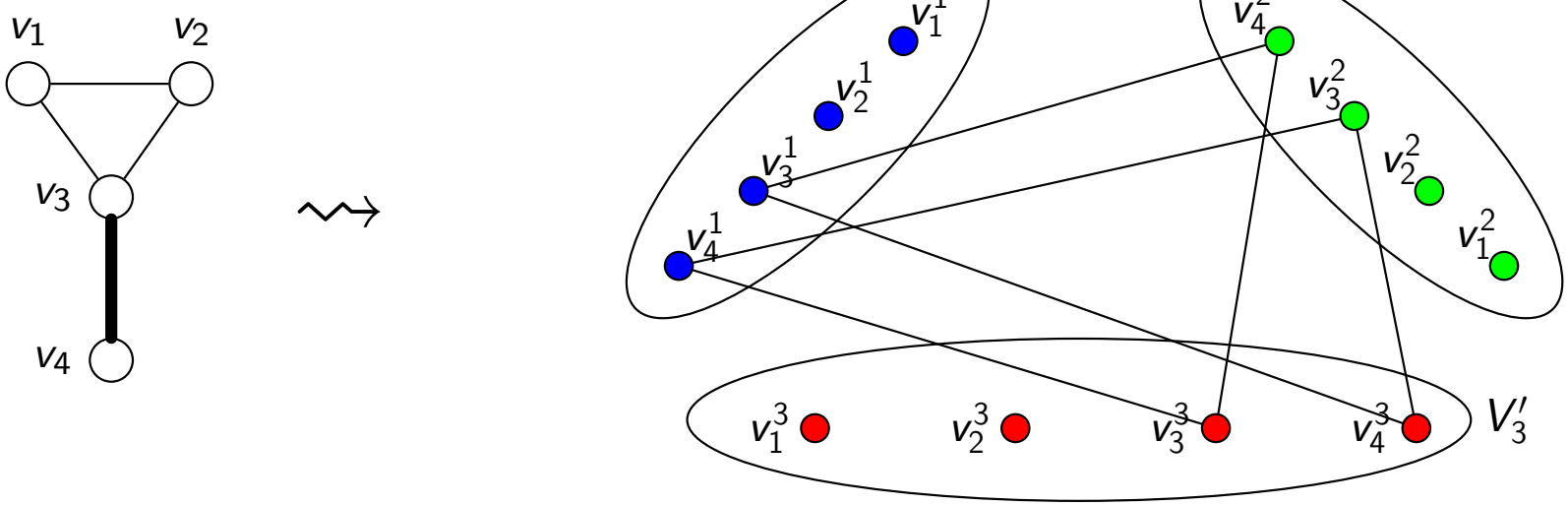
The **chromatic number** of a graph is the least number of colors needed to color each vertex with one color such that the endpoints of an edge have different colors.



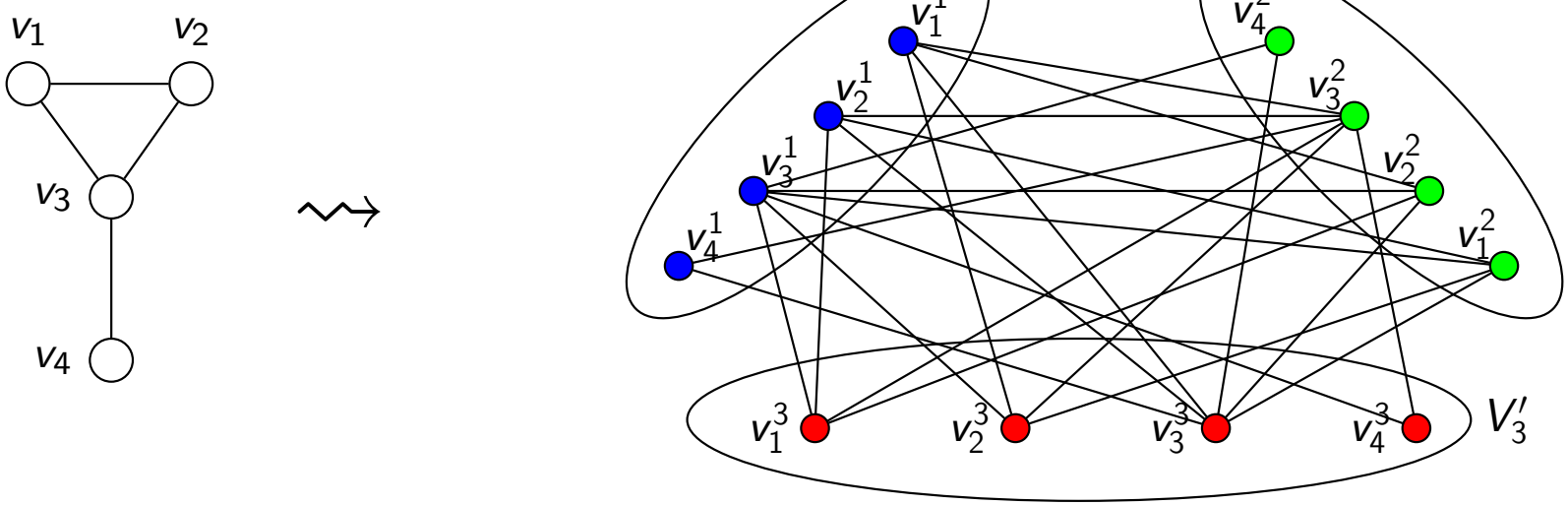
One Reduction to Rule Them All



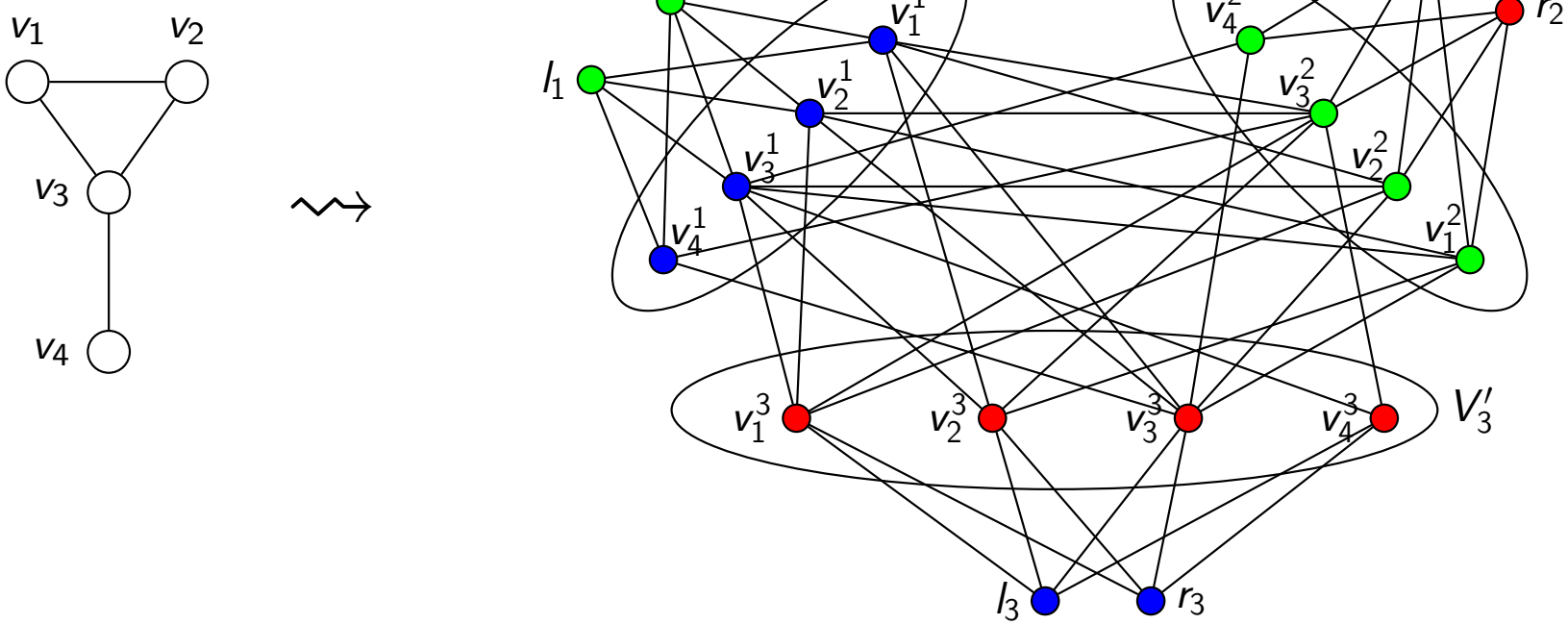
One Reduction to Rule Them All



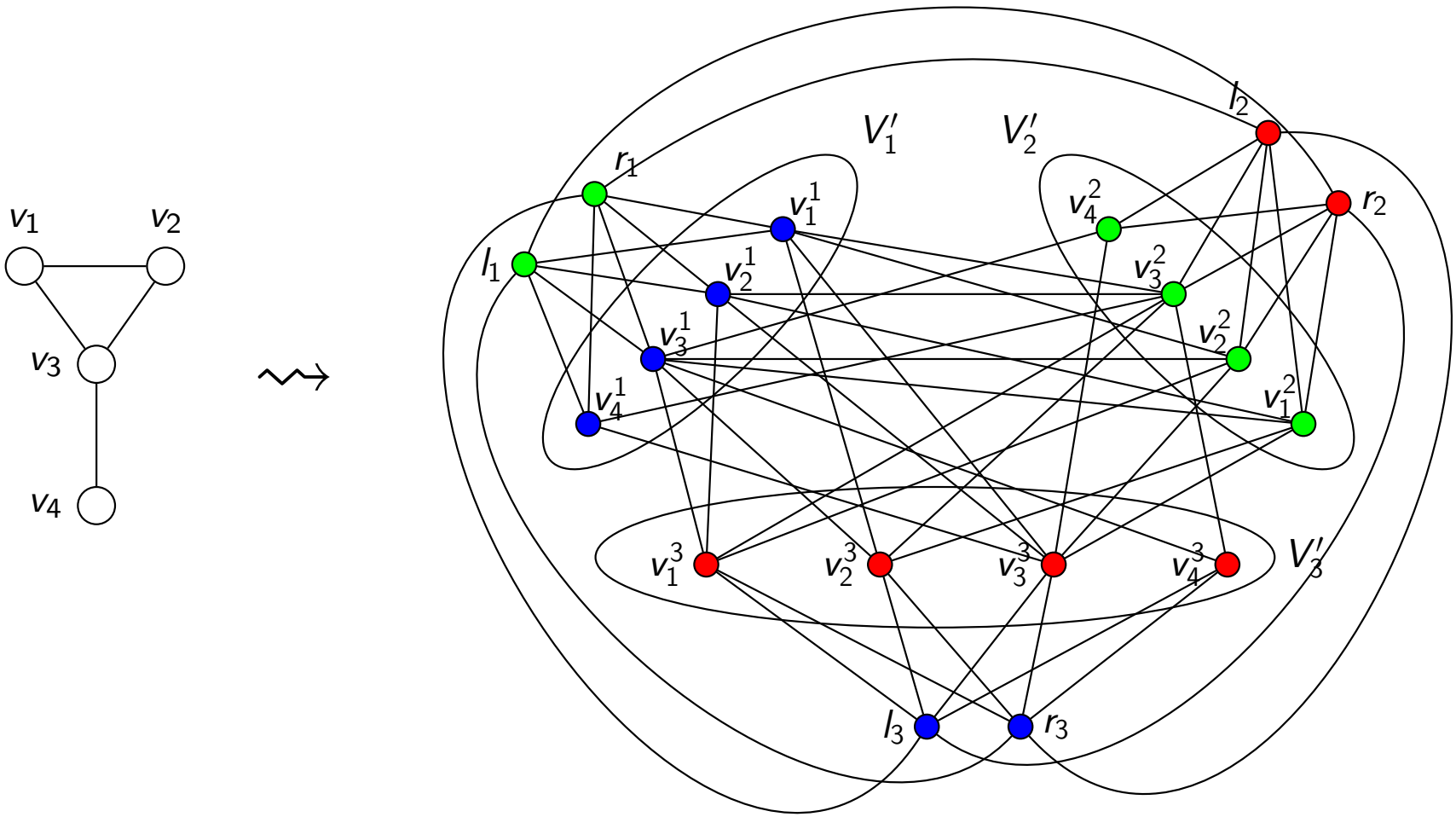
One Reduction to Rule Them All



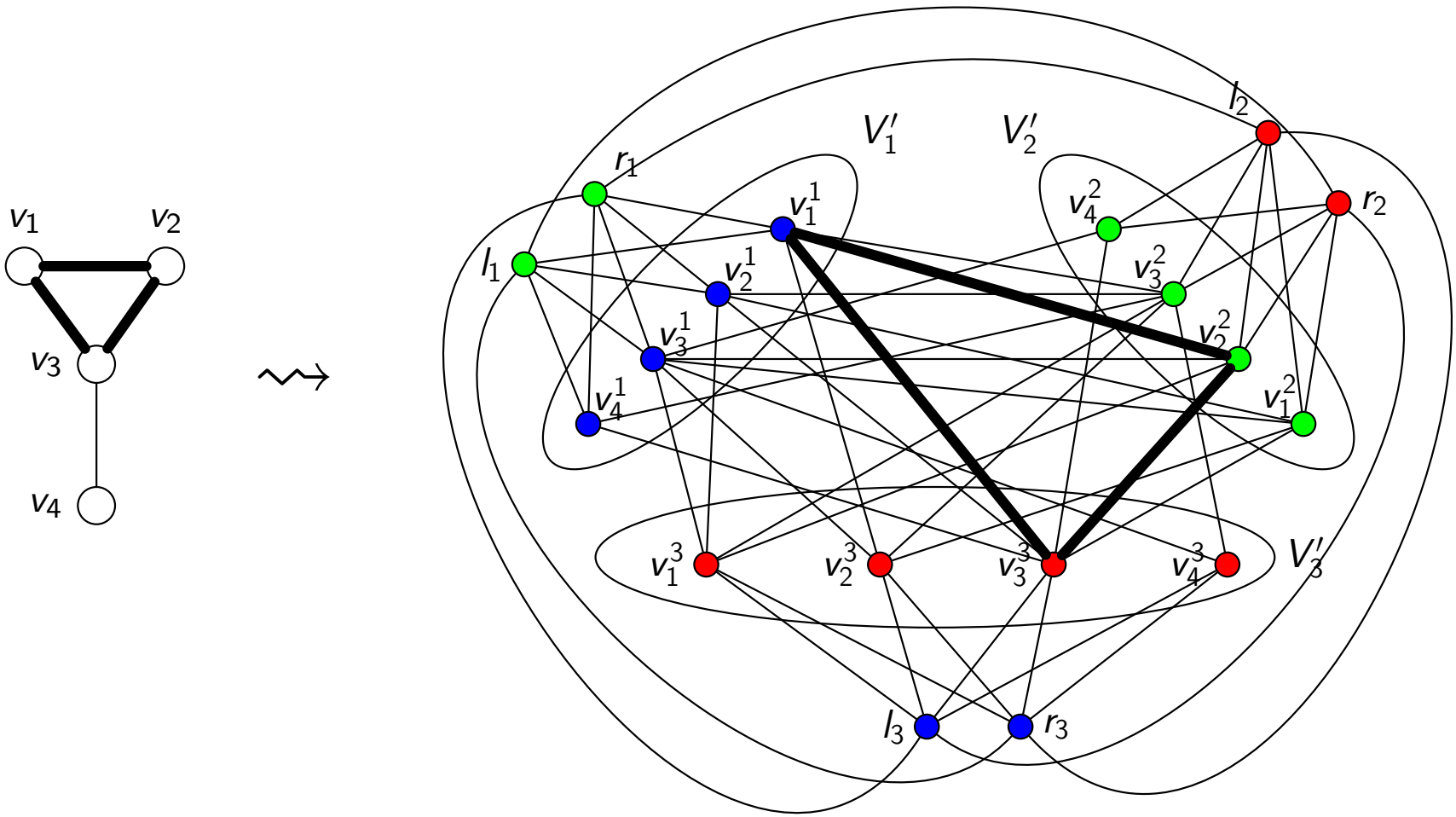
One Reduction to Rule Them All



One Reduction to Rule Them All



One Reduction to Rule Them All



Conclusion

1. Summary
   - New kernelization concept for enumeration tasks

Conclusion

1. Summary
   - New kernelization concept for enumeration tasks
   - New notion of hardness

Conclusion

1. Summary
   - New kernelization concept for enumeration tasks
   - New notion of hardness
   - Algorithms and hardness results for different parameters

Conclusion

1. Summary
   - New kernelization concept for enumeration tasks
   - New notion of hardness
   - Algorithms and hardness results for different parameters
2. Future work
   - Is there a polynomial-size kernel for distance to d degenerate graphs?

Conclusion

1. Summary
   - New kernelization concept for enumeration tasks
   - New notion of hardness
   - Algorithms and hardness results for different parameters
2. Future work
   - Is there a polynomial-size kernel for distance to d degenerate graphs?
   - Generalizing to larger cycles or cliques?

Conclusion

1. Summary
   - New kernelization concept for enumeration tasks
   - New notion of hardness
   - Algorithms and hardness results for different parameters
2. Future work
   - Is there a polynomial-size kernel for distance to d degenerate graphs?
   - Generalizing to larger cycles or cliques?

Thank you

Clique-Width I

The clique-width of a graph G is the minimum number k such that G can be constructed using a k-expression.

Clique-Width I

The clique-width of a graph G is the minimum number k such that G can be constructed using a k-expression.
A k-expression consists of four operations which use k labels:

- Creating a new vertex with some label i.
- Disjoint union of two labeled graphs.

Clique-Width I

The clique-width of a graph G is the minimum number k such that G can be constructed using a k-expression.
A k-expression consists of four operations which use k labels:

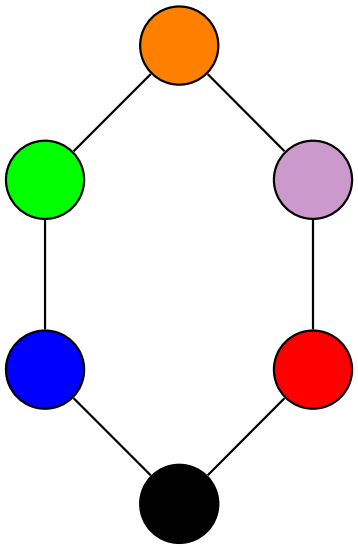
- Creating a new vertex with some label i.
- Disjoint union of two labeled graphs.
- Edge insertion between every vertex with label i and every vertex with label j.

Clique-Width I

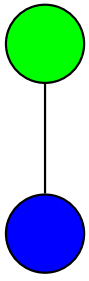
The clique-width of a graph G is the minimum number k such that G can be constructed using a k-expression.
A k-expression consists of four operations which use k labels:

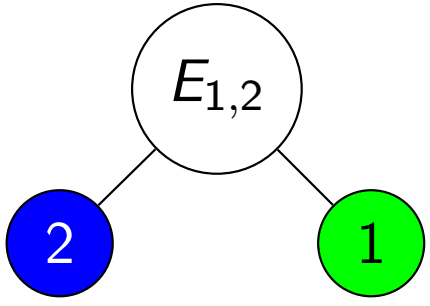
- Creating a new vertex with some label i.
- Disjoint union of two labeled graphs.
- Edge insertion between every vertex with label i and every vertex with label j.
- Renaming of label i to j.

Clique-Width I

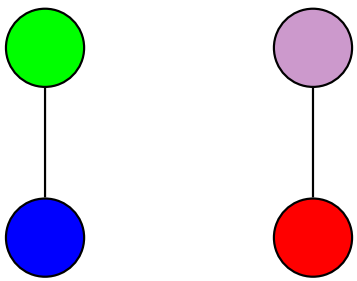


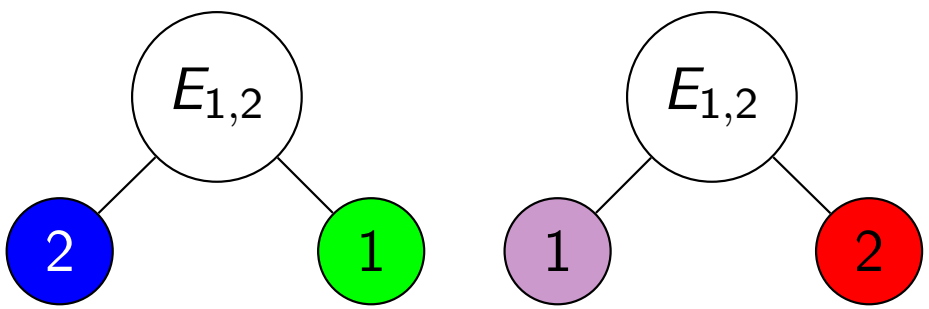
Clique-Width I



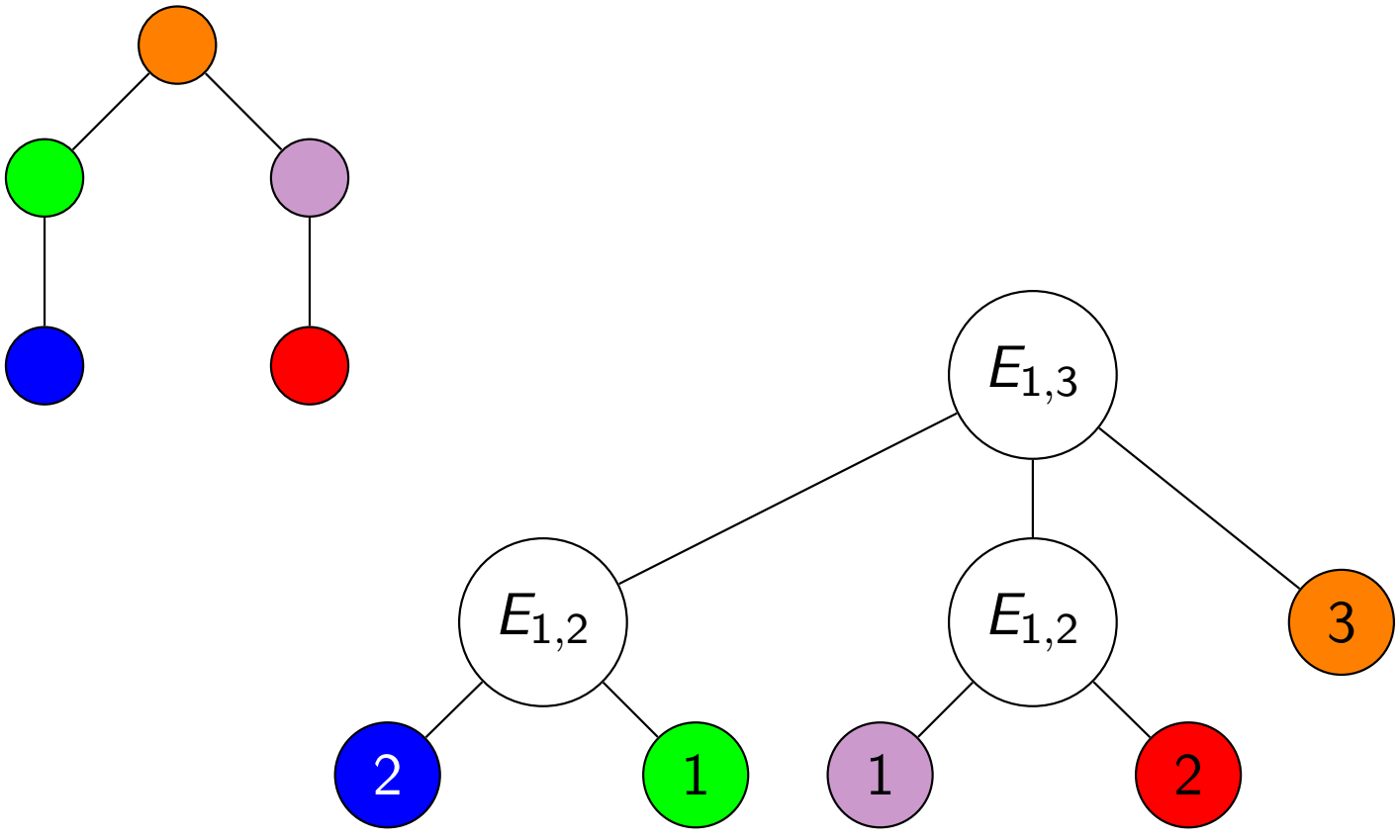


Clique-Width I

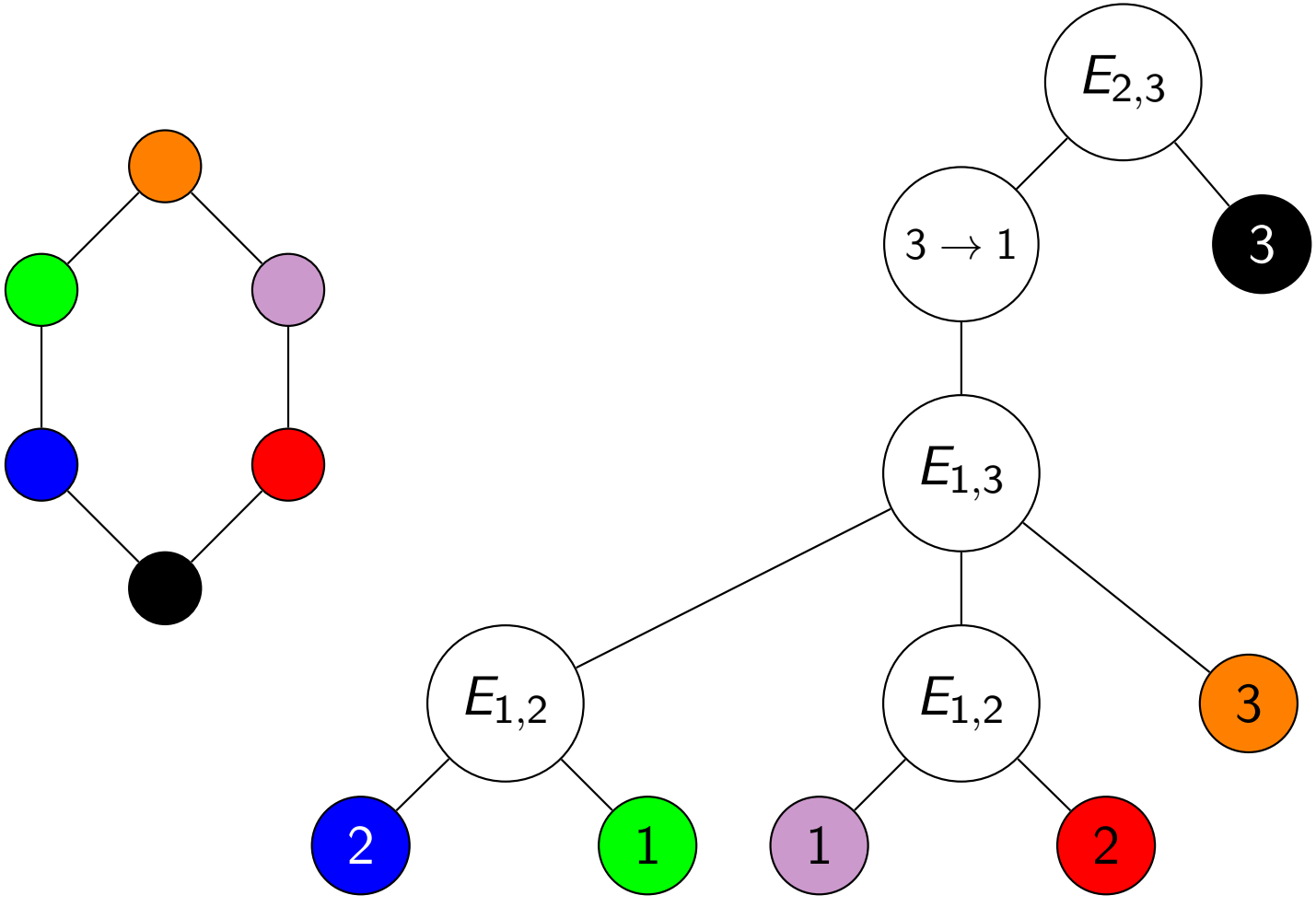




Clique-Width I



Clique-Width I



Clique-Width II

Theorem

Triangle Enumeration *is solvable in* $O(n^2 + n \cdot k^2 + \#\mathsf{T})$ *time, given a* k*-expression of the input graph.*

Clique-Width II

Given a k-expression tree B of G one can compute every node u in B the partition of V_u into its at most k twin-classes where V_u is the set of vertices corresponding to the leaves of the subtree of B rooted at u. [Bui-Xuan et al.: European Journal of Combinatorics, 2013]

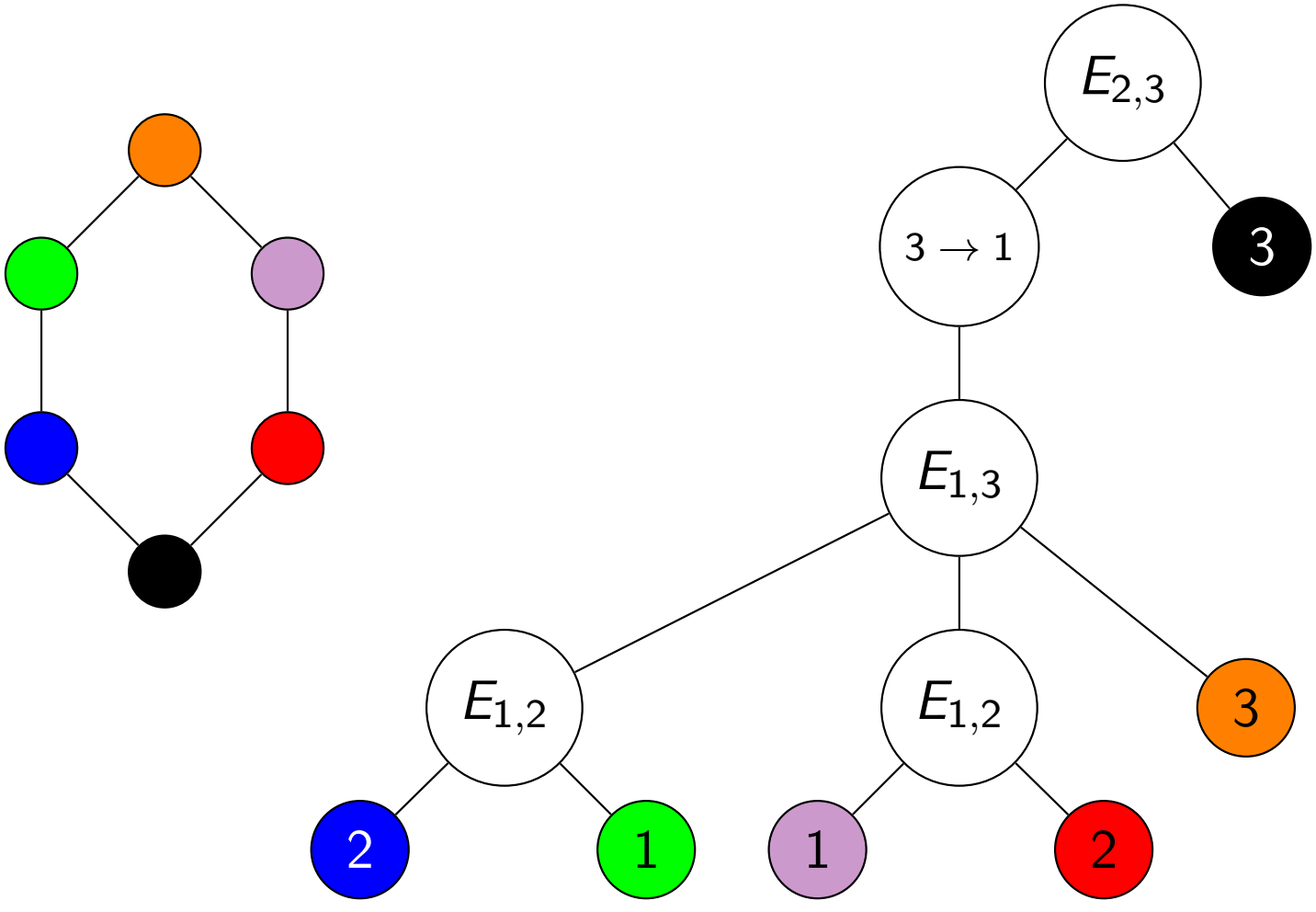
Clique-Width II

Given a k-expression tree B of G one can compute every node u in B the partition of V_u into its at most k twin-classes where V_u is the set of vertices corresponding to the leaves of the subtree of B rooted at u. [Bui-Xuan et al.: European Journal of Combinatorics, 2013]

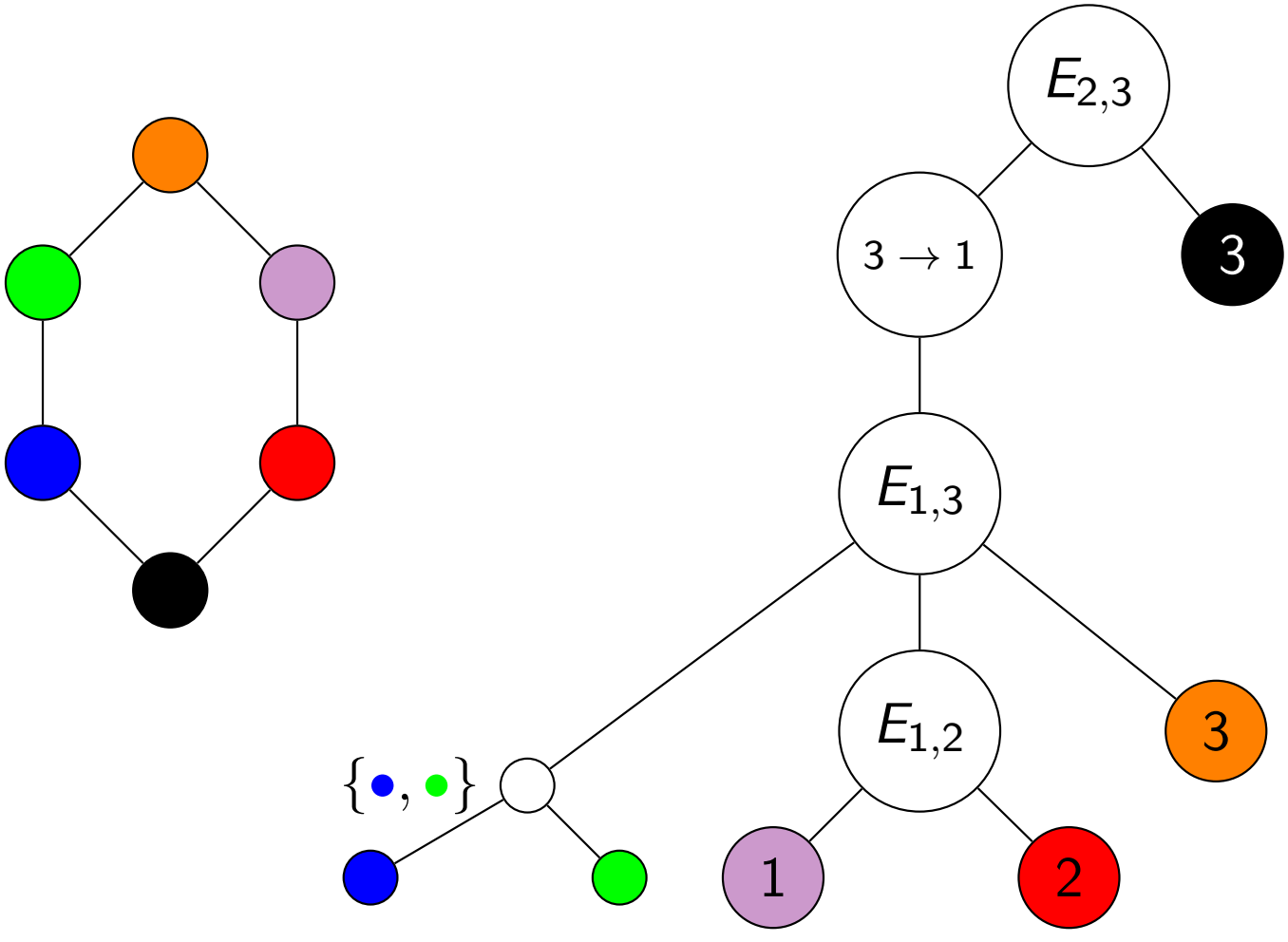
Definition (twin-class)

A **twin-class** in V_u is a set $V' \subseteq V_u$ of vertices such that every vertex in $V \setminus V_u$ either has all vertices in V' as its neighbors or none of them.

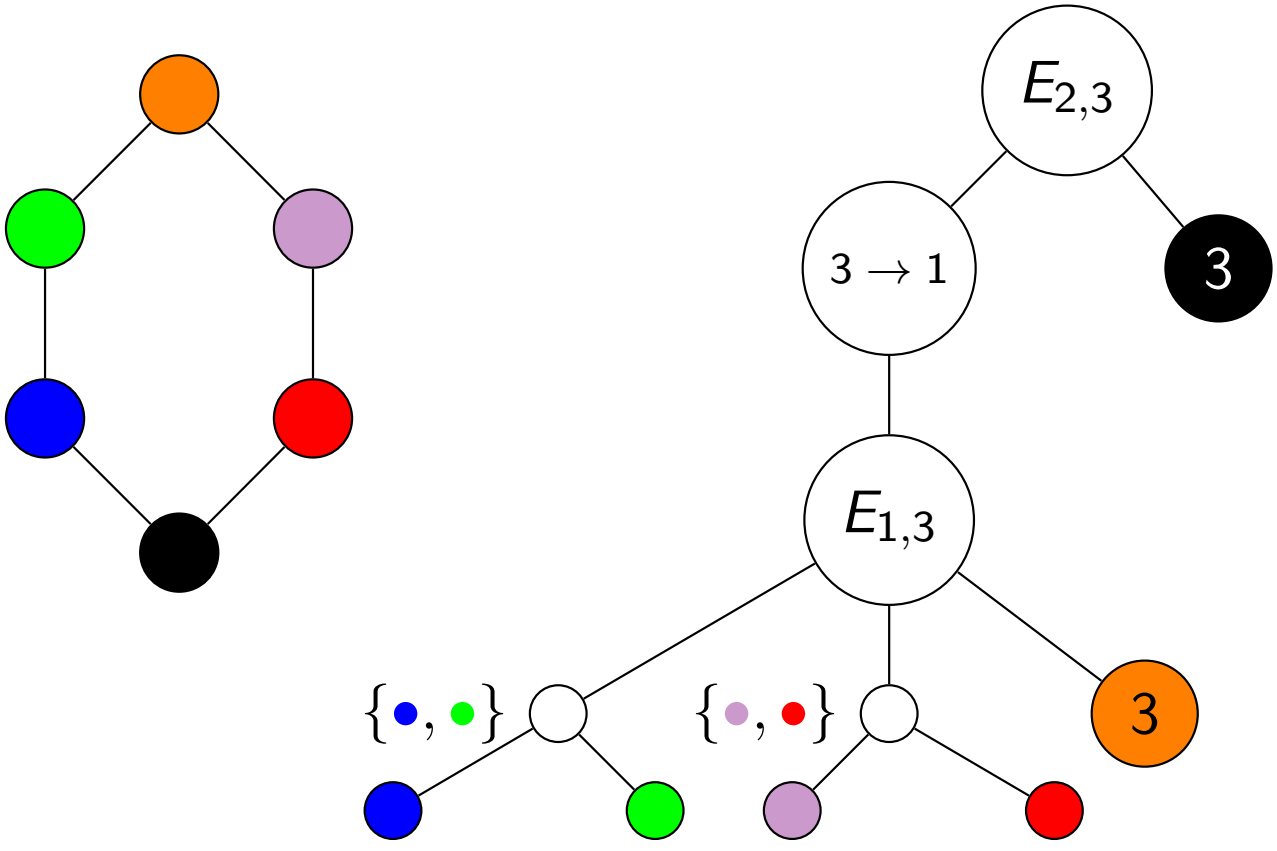
Clique-Width II



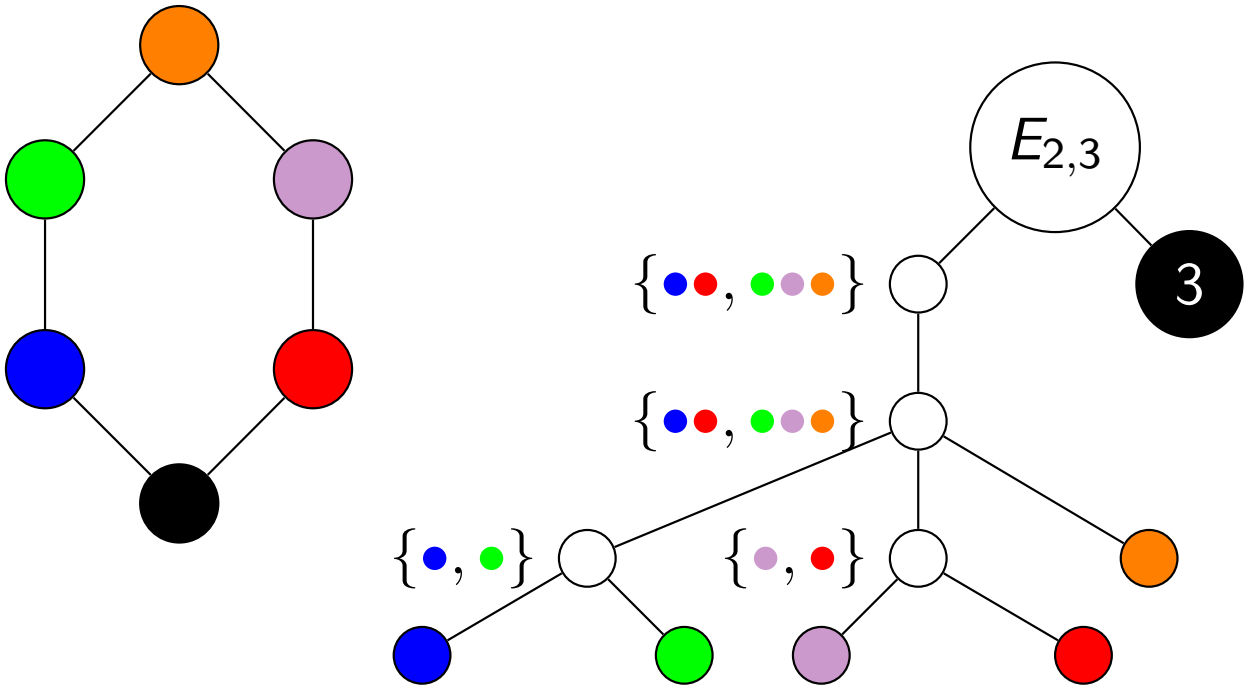
Clique-Width II



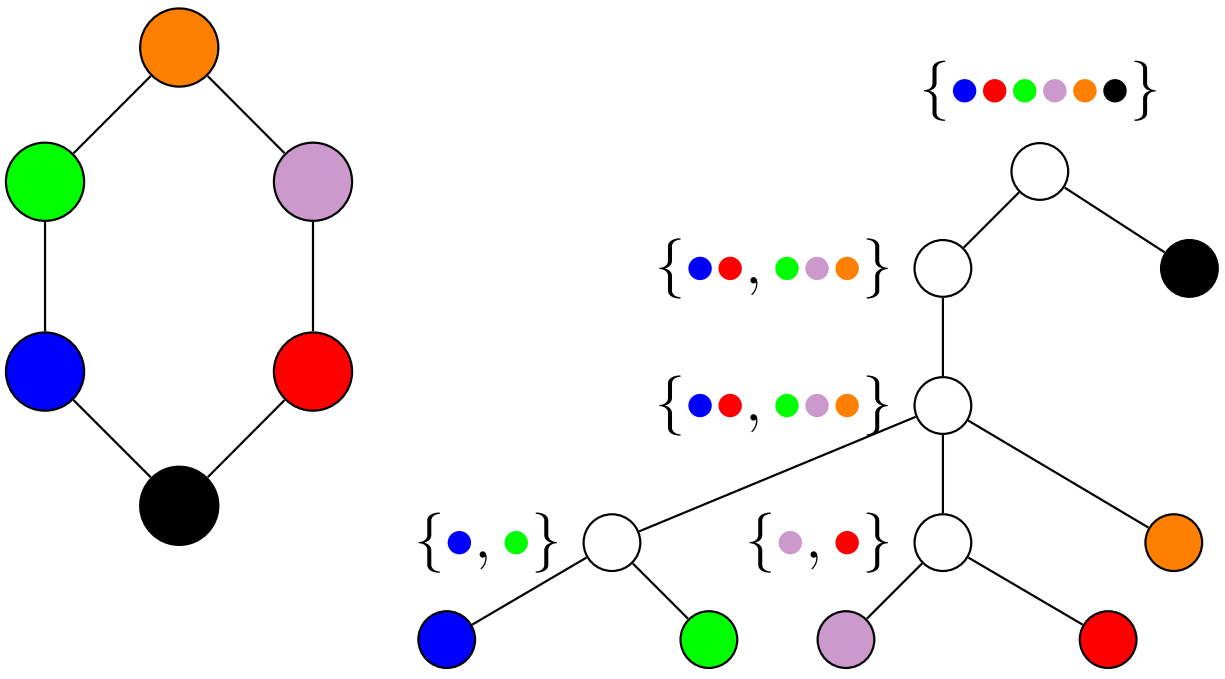
Clique-Width II



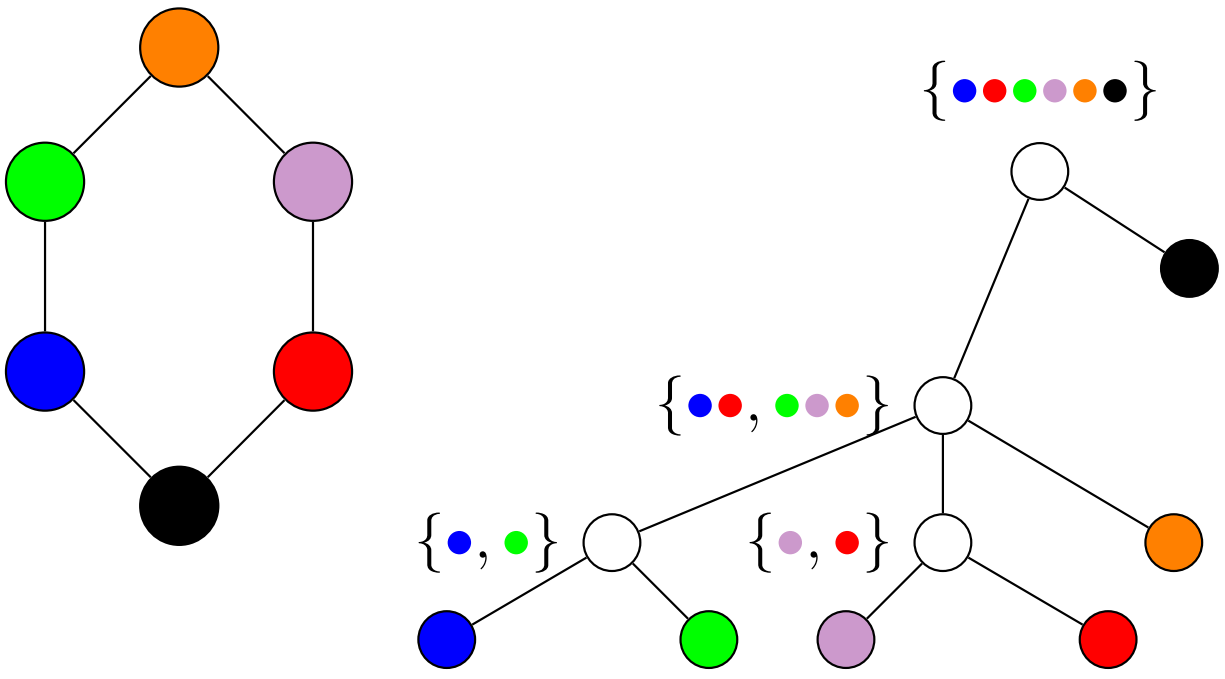
Clique-Width II



Clique-Width II



Clique-Width II



Clique-Width II

Properties of the tree of twin-classes (module decomposition):

- Each node has no or exactly two children.

Clique-Width II

Properties of the tree of twin-classes (module decomposition):

- Each node has no or exactly two children.
- Each twin-class is fully contained in one of the twin-classes of the parent node.

Clique-Width II

Properties of the tree of twin-classes (module decomposition):

- Each node has no or exactly two children.
- Each twin-class is fully contained in one of the twin-classes of the parent node.
- The root node has only a single twin-class containing all vertices (of G).

Clique-Width III

u, v, w: nodes (where u, w are the children of v (if they exist))
i, j, l: twin-classes of v
v_i, v_j, v_l: arbitrary vertices in the twin-classes i, j, l, respectively
M_h^x: set of all twin-classes of x that are subsets of h

Clique-Width III

u, v, w: nodes (where u, w are the children of v (if they exist))
i, j, l: twin-classes of v
v_i, v_j, v_l: arbitrary vertices in the twin-classes i, j, l, respectively
M_h^x: set of all twin-classes of x that are subsets of h

Dynamic program:
$E_{i,j}^v$: all edges between vertices in i and j
$T_{i,j,l}^v$: all triangles containing vertices in i, j and l

Clique-Width III

u, v, w: nodes (where u, w are the children of v (if they exist))
i, j, l: twin-classes of v
v_i, v_j, v_l: arbitrary vertices in the twin-classes i, j, l, respectively
M_h^x: set of all twin-classes of x that are subsets of h

Dynamic program:
$E_{i,j}^v$: all edges between vertices in i and j
$T_{i,j,l}^v$: all triangles containing vertices in i, j and l

A leave node has no edges and no triangles.

Clique-Width III

u, v, w: nodes (where u, w are the children of v (if they exist))
i, j, l: twin-classes of v
v_i, v_j, v_l: arbitrary vertices in the twin-classes i, j, l, respectively
M_h^x: set of all twin-classes of x that are subsets of h

An inner node has

$$E_{i,j}^v = \bigcup_{o \in M_i^u} \bigcup_{p \in M_j^u} E_{o,p}^u \cup \bigcup_{o \in M_i^u} \bigcup_{p \in M_j^w} \{\{x, y\} \mid x \in o, y \in p, \{v_o, v_p\} \in E\}$$

Clique-Width III

u, v, w: nodes (where u, w are the children of v (if they exist))
i, j, l: twin-classes of v
v_i, v_j, v_l: arbitrary vertices in the twin-classes i, j, l, respectively
M_h^x: set of all twin-classes of x that are subsets of h

An inner node has

$$E_{i,j}^v = \bigcup_{o \in M_i^u} \bigcup_{p \in M_j^u} E_{o,p}^u \cup \bigcup_{o \in M_i^u} \bigcup_{p \in M_j^w} \{\{x, y\} \mid x \in o, y \in p, \{v_o, v_p\} \in E\}$$

$$T_{i,j,l}^v = \bigcup_{o \in M_i^u} \bigcup_{p \in M_j^u} \bigcup_{q \in M_l^u} T_{o,p,q}^u \cup$$

$$\bigcup_{o \in M_i^u} \bigcup_{p \in M_j^u} \bigcup_{q \in M_l^w} \{\{x, y, z\} \mid x \in o, y \in p, z \in q, \{v_i, v_l\}, \{v_j, v_l\} \in E\}$$

Clique-Width III

- Each edge and triangle is computed exactly once.
- First step by Bui-Xuan et al. takes $O(n^2)$ time.
- Dynamic program takes $O(m + \#\mathsf{T} + n \cdot k^2)$ time.

Distance to d-Degenerate Graphs and Maximum Degree

Theorem

Triangle Enumeration *parameterized by distance to d-degenerate graphs and maximum degree Δ_D in a set D such that $G - D$ is d-degenerate is solvable in $O(|D| \cdot \Delta_D^2 + n \cdot d^2)$ time provided that the set D is given.*

Distance to d-Degenerate Graphs and Maximum Degree

1. List all triangles which do not contain any vertices in D. To this end, compute the d-degenerate graph $G' = G - D$ and list all triangles contained in G' in $O(n \cdot d^2)$ time using the degeneracy order of G'.

Distance to d-Degenerate Graphs and Maximum Degree

1. List all triangles which do not contain any vertices in D. To this end, compute the d-degenerate graph $G' = G - D$ and list all triangles contained in G' in $O(n \cdot d^2)$ time using the degeneracy order of G'.
2. List all triangles with at least one vertex contained in D. For each $u \in D$, iterate over all of the at most Δ_D^2 possible pairs of neighbors $v, w \in N(u)$. For each pair $v, w \in N(u)$, check in constant time whether $\{u, v, w\}$ is a new triangle.

General Lemma for Distance to Π

Lemma

Triangle Enumeration *parametrized by deletion set* K *to* Π *is solvable in* $O(m \cdot |K| + n + x)$ *time if* Triangle Enumeration *on a graph in* Π *is solvable in* $O(x)$ *time.*

General Lemma for Distance to Π

Let K be a set of vertices such that $G' = G - K$ is a graph in Π. By assumption, all triangles within G' can be listed in $O(x)$ time.

General Lemma for Distance to Π

Let K be a set of vertices such that $G' = G - K$ is a graph in Π.
By assumption, all triangles within G' can be listed in $O(x)$ time.
All triangles with at least one vertex in K can be listed
in $O(m \cdot |K| + n)$ time by the following algorithm.

General Lemma for Distance to Π

Let K be a set of vertices such that $G' = G - K$ is a graph in Π. By assumption, all triangles within G' can be listed in $O(x)$ time. All triangles with at least one vertex in K can be listed in $O(m \cdot |K| + n)$ time by the following algorithm.

- Read the whole input and fix an arbitrary order $\leq_a$ of the vertices in K.

General Lemma for Distance to Π

Let K be a set of vertices such that $G' = G - K$ is a graph in Π. By assumption, all triangles within G' can be listed in $O(x)$ time. All triangles with at least one vertex in K can be listed in $O(m \cdot |K| + n)$ time by the following algorithm.

- Read the whole input and fix an arbitrary order $\leq_a$ of the vertices in K.
- Check for each edge $\{u, w\}$ and each vertex $v \in K$ whether $\{u, v, w\}$ is a triangle and for all $x \in \{u, w\} \cap K$ it holds that $v \leq_a x$. If both conditions hold, then list $\{u, v, w\}$ as a new triangle.

General Lemma for Distance to Π

Let K be a set of vertices such that $G' = G - K$ is a graph in Π. By assumption, all triangles within G' can be listed in $O(x)$ time. All triangles with at least one vertex in K can be listed in $O(m \cdot |K| + n)$ time by the following algorithm.

- Read the whole input and fix an arbitrary order $\leq_a$ of the vertices in K.
- Check for each edge $\{u, w\}$ and each vertex $v \in K$ whether $\{u, v, w\}$ is a triangle and for all $x \in \{u, w\} \cap K$ it holds that $v \leq_a x$. If both conditions hold, then list $\{u, v, w\}$ as a new triangle.

Since $v \in K$ holds, this algorithm does not list any triangles which do not contain vertices in K and it lists all triangles with at least one vertex in K exactly once.

Distance to Cographs

Theorem

Triangle Enumeration *parametrized by deletion set K to cographs is solvable in $O(\#\mathsf{T} + n + m \cdot |K|)$ time.*

Distance to Cographs

Each cograph has a binary cotree representation that can be computed in linear time.

Distance to Cographs

Each cograph has a binary cotree representation that can be computed in linear time.
A cotree is a tree with V as a set of leaves and inner vertices that are either labeled with union or join.
A union of two graphs is disjoint and a join of two graphs is the same vertices and edges plus all edges between the two sets of vertices.

Distance to Cographs

Dynamic program that stores for each node in the cotree all vertices $V(p)$, all edges $E(p)$ and all triangles $T(p)$ in the corresponding subgraph of G.

Distance to Cographs

Dynamic program that stores for each node in the cotree all vertices $V(p)$, all edges $E(p)$ and all triangles $T(p)$ in the corresponding subgraph of G.
Let q_1, q_2 be the children of an inner node p in the cotree.

Distance to Cographs

Dynamic program that stores for each node in the cotree all vertices $V(p)$, all edges $E(p)$ and all triangles $T(p)$ in the corresponding subgraph of G.

Let q_1, q_2 be the children of an inner node p in the cotree.

- A single leaf node has one vertex and no edges or triangles.
- A union node has vertices $V(q_1) \cup V(q_2)$, edges $E(q_1) \cup E(q_2)$ and triangles $T(q_1) \cup T(q_2)$.

Distance to Cographs

Dynamic program that stores for each node in the cotree all vertices $V(p)$, all edges $E(p)$ and all triangles $T(p)$ in the corresponding subgraph of G.

Let q_1, q_2 be the children of an inner node p in the cotree.

- A single leaf node has one vertex and no edges or triangles.
- A union node has vertices $V(q_1) \cup V(q_2)$, edges $E(q_1) \cup E(q_2)$ and triangles $T(q_1) \cup T(q_2)$.
- A join node has

$$V(p) = V(q_1) \cup V(q_2),$$
$$E(p) = E(q_1) \cup E(q_2) \cup \{\{x, y\} \mid x \in V(q_1) \wedge y \in V(q_2)\} \text{ and}$$
$$T(p) = T(q_1) \cup T(q_2) \cup \{\{x, y, z\} \mid x \in V(q_1) \wedge \{y, z\} \in E(q_2)\} \cup$$
$$\{\{x, y, z\} \mid x \in V(q_2) \wedge \{y, z\} \in E(q_1)\}.$$

Distance to Cographs

- There are n leaf-nodes in the cotree each of which require a constant amount of time to compute.
- There are at most $n - 1$ union nodes each of which only require a constant amount of time as they only need to point on their children's values.

Distance to Cographs

- There are n leaf-nodes in the cotree each of which require a constant amount of time to compute.
- There are at most $n - 1$ union nodes each of which only require a constant amount of time as they only need to point on their children's values.
- There are at most $n - 1$ join nodes. Each edge and triangle is only added once and all other values do not need to be recomputed.

Distance to Cographs

- There are n leaf-nodes in the cotree each of which require a constant amount of time to compute.
- There are at most $n-1$ union nodes each of which only require a constant amount of time as they only need to point on their children's values.
- There are at most $n-1$ join nodes. Each edge and triangle is only added once and all other values do not need to be recomputed.

The global running time of this algorithm is in $O(\#\mathsf{T}+n+m)$.

Enum-Advice Kernelization $\Longrightarrow$ Solving Algorithm

Lemma

Let $\mathcal{R}$ be an enum-advice kernelization of a parameterized enumeration problem P such that for every instance (x, k) of P:

- *$\mathcal{R}$ runs in $O((|x| + k)^c)$ time for some constant c;*
- *the unparameterized version of P can be solved in $g(|x|)$ time;*
- *the kernelization computes the pair (I, A) where $|I| \leq h(k)$, and $\mathcal{T}_f$ takes $O(|I|^d)$ time between generating two solutions for some constant d;*
- *$\#s$ denotes the number of solutions in I and $\#S$ denotes the number of solutions in x.*

Then, P can be solved in $O((|x| + k)^c + g(h(k)) + (\#s + \#S) \cdot h(k)^d)$ time.

Enum-Advice Kernelization $\Longrightarrow$ Solving Algorithm

1. Compute the kernel (I, A) in $O((|x| + k)^c)$ time.
2. Find all solutions in I in $g(|I|) \in O(f(h(k)))$ time.

Enum-Advice Kernelization $\Longrightarrow$ Solving Algorithm

1. Compute the kernel (I, A) in $O((|x| + k)^c)$ time.
2. Find all solutions in I in $g(|I|) \in O(f(h(k)))$ time.
3. Compute all solutions of I in $O(g(|I|) \subseteq O(g(h(k)))$ time.
4. Apply $\mathcal{T}_f$ on all solutions of I.
   This takes $O((\#\mathsf{s} + \#\mathsf{S}) \cdot |I|^d) \subseteq O((\#\mathsf{s} + \#\mathsf{S}) \cdot h(k)^d)$ time.

Enum-Advice Kernelization $\Longrightarrow$ Solving Algorithm

1. Compute the kernel (I, A) in $O((|x| + k)^c)$ time.
2. Find all solutions in I in $g(|I|) \in O(f(h(k)))$ time.
3. Compute all solutions of I in $O(g(|I|) \subseteq O(g(h(k)))$ time.
4. Apply $\mathcal{T}_f$ on all solutions of I.
   This takes $O((\#\mathsf{s} + \#\mathsf{S}) \cdot |I|^d) \subseteq O((\#\mathsf{s} + \#\mathsf{S}) \cdot h(k)^d)$ time.

This algorithm takes $O((|x| + k)^c + g(h(k)) + (\#\mathsf{s} + \#\mathsf{S}) \cdot h(k)^d)$ time and lists all triangles in x.

Feedback Edge Number I

Lemma

Let $G = (V, E)$ be an undirected graph and let F be a feedback edge set in G. All triangles $\{u, v, w\}$ where at least one of the edges between the three vertices is not in F can be enumerated in $O(n + m)$ time. There are at most $2|F|$ such triangles.

Feedback Edge Number I

- Every triangle $\{u, v, w\}$ in G where at least one of the edges between the three vertices is not in F is of the form $\{u, v, p(v)\}$ for some vertices u, v.

Feedback Edge Number I

- Every triangle $\{u, v, w\}$ in G where at least one of the edges between the three vertices is not in F is of the form $\{u, v, p(v)\}$ for some vertices u, v.
- Check for each edge $\{u, v\} \in F$ whether $p(u) = p(v)$, $\{p(u), v\} \in E$ or $\{u, p(v)\} \in E$:

Feedback Edge Number I

- Every triangle $\{u, v, w\}$ in G where at least one of the edges between the three vertices is not in F is of the form $\{u, v, p(v)\}$ for some vertices u, v.
- Check for each edge $\{u, v\} \in F$ whether $p(u) = p(v)$, $\{p(u), v\} \in E$ or $\{u, p(v)\} \in E$:
  - In the first case we list $\{u, v, p(u)\}$ as a triangle and do not consider the other cases.

Feedback Edge Number I

- Every triangle $\{u, v, w\}$ in G where at least one of the edges between the three vertices is not in F is of the form $\{u, v, p(v)\}$ for some vertices u, v.
- Check for each edge $\{u, v\} \in F$ whether $p(u) = p(v)$, $\{p(u), v\} \in E$ or $\{u, p(v)\} \in E$:
  - In the first case we list $\{u, v, p(u)\}$ as a triangle and do not consider the other cases.
  - If $p(u) \neq p(v)$ and $\{p(u), v\} \in E$, then we list $\{u, v, p(u)\}$ as a triangle.
  - If $p(u) \neq p(v)$ and $\{u, p(v)\} \in E$, then we list $\{u, v, p(v)\}$ as a triangle.

Feedback Edge Number I

- Every triangle $\{u, v, w\}$ in G where at least one of the edges between the three vertices is not in F is of the form $\{u, v, p(v)\}$ for some vertices u, v.
- Check for each edge $\{u, v\} \in F$ whether $p(u) = p(v)$, $\{p(u), v\} \in E$ or $\{u, p(v)\} \in E$:
  - In the first case we list $\{u, v, p(u)\}$ as a triangle and do not consider the other cases.
  - If $p(u) \neq p(v)$ and $\{p(u), v\} \in E$, then we list $\{u, v, p(u)\}$ as a triangle.
  - If $p(u) \neq p(v)$ and $\{u, p(v)\} \in E$, then we list $\{u, v, p(v)\}$ as a triangle.

Note that for every edge in F we list at most two triangles.

Feedback Edge Number II

Theorem

Triangle Enumeration *parameterized by feedback edge number k admits a constant-delay enum-advice kernel with at most $2k + 3$ vertices and $k + 3$ edges which can be computed in $O(n + m)$ time.*

Feedback Edge Number II

For every edge $e \in F$ put e and both of its endpoints into the kernel graph. Compute the feedback edge number $k' \leq k$.

Feedback Edge Number II

For every edge $e \in F$ put e and both of its endpoints into the kernel graph. Compute the feedback edge number $k' \leq k$. Compute all triangles in G with at least one edge in $E \setminus F$ and set A to be the set of all triangles found.

Feedback Edge Number II

For every edge $e \in F$ put e and both of its endpoints into the kernel graph. Compute the feedback edge number $k' \leq k$.
Compute all triangles in G with at least one edge in $E \setminus F$ and set A to be the set of all triangles found.
If $A \neq \emptyset$, then add one extra triangle $\{x, y, z\}$ where $x, y, z \notin V$.

Feedback Edge Number II

For every edge $e \in F$ put e and both of its endpoints into the kernel graph. Compute the feedback edge number $k' \leq k$.
Compute all triangles in G with at least one edge in $E \setminus F$ and set A to be the set of all triangles found.
If $A \neq \emptyset$, then add one extra triangle $\{x, y, z\}$ where $x, y, z \notin V$.

$$f(w, A) = \begin{cases} A & \text{if } w = \{x, y, z\}, \\ \{w\} & \text{else.} \end{cases}$$

Feedback Edge Number II

- Each step can be done in $O(n + m)$ time.

Feedback Edge Number II

- Each step can be done in $O(n + m)$ time.
- It holds that $|G_I| \leq 3 \cdot k + 3$.

Feedback Edge Number II

- Each step can be done in $O(n+m)$ time.
- It holds that $|G_I| \leq 3 \cdot k + 3$.
- Each triangle T either contains at least one edge in $E \setminus F$ or only edges in F. In the first case G_I contains the triangle $\{x, y, z\}$ and in the second case G_I contains T.

Feedback Edge Number II

- Each step can be done in $O(n+m)$ time.
- It holds that $|G_I| \leq 3 \cdot k + 3$.
- Each triangle T either contains at least one edge in $E \setminus F$ or only edges in F. In the first case G_I contains the triangle $\{x, y, z\}$ and in the second case G_I contains T.
- For two solutions p, q in I that are not $\{x, y, z\}$, if $p \neq q$, then $f(p, A) \cap f(q, A) = \{p\} \cap \{q\} = \emptyset$.

Feedback Edge Number II

- Each step can be done in $O(n+m)$ time.
- It holds that $|G_I| \leq 3 \cdot k + 3$.
- Each triangle T either contains at least one edge in $E \setminus F$ or only edges in F. In the first case G_I contains the triangle $\{x, y, z\}$ and in the second case G_I contains T.
- For two solutions p, q in I that are not $\{x, y, z\}$, if $p \neq q$, then $f(p, A) \cap f(q, A) = \{p\} \cap \{q\} = \emptyset$.
- By construction, $f(\{x, y, z\}, A)$ contains all triangles in G where at least one of the edges is not in F. Since all edges in F are included in G_I, all other triangles are contained in G_I.

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,

The kernel contains the triangle $\{a, b, c\}$ if and only if the original graph contains a triangle with at most one vertex in D.

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,

The kernel contains the triangle $\{a, b, c\}$ if and only if the original graph contains a triangle with at most one vertex in D.

Each vertex or one of its twins is contained in the kernel instance.

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,
- $|I'|, k \leq g(k)$,

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,
- $|I'|, k \leq g(k)$,

- At most $2^{|D|}$ vertices that are not in $D \cup \{a, b, c\}$
  $\rightsquigarrow |I'| \leq 2^{|D|} + |D| + 3$

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,
- $|I'|, k \leq g(k)$,

- At most $2^{|D|}$ vertices that are not in $D \cup \{a, b, c\}$
  $\rightsquigarrow |I'| \leq 2^{|D|} + |D| + 3$
- $k' = k$

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,
- $|I'|, k \leq g(k)$,
- $\bigcup_{w:\mathtt{Sol}(I',k')} f(w, A) = \mathtt{Sol}(I, k)$,

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,
- $|I'|, k \leq g(k)$,
- $\bigcup_{w:\mathtt{Sol}(I',k')} f(w, A) = \mathtt{Sol}(I, k)$,

- $f(\{a, b, c\}, A)$: all triangles with at most one vertex in D

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,
- $|I'|, k \leq g(k)$,
- $\bigcup_{w:\mathtt{Sol}(I',k')} f(w, A) = \mathtt{Sol}(I, k)$,

- $f(\{a, b, c\}, A)$: all triangles with at most one vertex in D
- $\bigcup_{w:\mathtt{Sol}(I',k')\setminus\{\{a,b,c\}\}} f(w, A)$: all triangles with at least two vertices in D.

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,
- $|I'|, k \leq g(k)$,
- $\bigcup_{w:\mathtt{Sol}(I',k')} f(w, A) = \mathtt{Sol}(I, k)$,
- $f(p, A) \cap f(q, A) = \emptyset$, and

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,
- $|I'|, k \leq g(k)$,
- $\bigcup_{w:\mathtt{Sol}(I',k')} f(w, A) = \mathtt{Sol}(I, k)$,
- $f(p, A) \cap f(q, A) = \emptyset$, and

$$p = \{a, b, c\} \Rightarrow |f(p, A) \cap D| > |f(q, A) \cap D|$$

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,
- $|I'|, k \leq g(k)$,
- $\bigcup_{w:\mathtt{Sol}(I',k')} f(w, A) = \mathtt{Sol}(I, k)$,
- $f(p, A) \cap f(q, A) = \emptyset$, and

$$p = \{a, b, c\} \Rightarrow |f(p, A) \cap D| > |f(q, A) \cap D|$$

Twins of twins are twins.

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,
- $|I'|, k \leq g(k)$,
- $\bigcup_{w:\mathtt{Sol}(I',k')} f(w, A) = \mathtt{Sol}(I, k)$,
- $f(p, A) \cap f(q, A) = \emptyset$, and
- $\mathcal{R}$ in FPT and f in FPT-delay time.

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,
- $|I'|, k \leq g(k)$,
- $\bigcup_{w:\mathtt{Sol}(I',k')} f(w, A) = \mathtt{Sol}(I, k)$,
- $f(p, A) \cap f(q, A) = \emptyset$, and
- $\mathcal{R}$ in FPT and f in FPT-delay time.

- $\mathcal{R}$:

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,
- $|I'|, k \leq g(k)$,
- $\bigcup_{w:\mathtt{Sol}(I',k')} f(w, A) = \mathtt{Sol}(I, k)$,
- $f(p, A) \cap f(q, A) = \emptyset$, and
- $\mathcal{R}$ in FPT and f in FPT-delay time.

- $\mathcal{R}$:
  - T_0 in $O(m \cdot d)$ time (Chiba and Nishizeki)

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,
- $|I'|, k \leq g(k)$,
- $\bigcup_{w:\mathtt{Sol}(I',k')} f(w, A) = \mathtt{Sol}(I, k)$,
- $f(p, A) \cap f(q, A) = \emptyset$, and
- $\mathcal{R}$ in FPT and f in FPT-delay time.

- $\mathcal{R}$:
  - T_0 in $O(m \cdot d)$ time (Chiba and Nishizeki)
  - T_1 in $O(n \cdot d \cdot |D|)$ (almost brute force)

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,
- $|I'|, k \leq g(k)$,
- $\bigcup_{w:\mathtt{Sol}(I',k')} f(w, A) = \mathtt{Sol}(I, k)$,
- $f(p, A) \cap f(q, A) = \emptyset$, and
- $\mathcal{R}$ in FPT and f in FPT-delay time.

- $\mathcal{R}$:
  - T_0 in $O(m \cdot d)$ time (Chiba and Nishizeki)
  - T_1 in $O(n \cdot d \cdot |D|)$ (almost brute force)
  - edge deletion in linear time

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,
- $|I'|, k \leq g(k)$,
- $\bigcup_{w:\mathtt{Sol}(I',k')} f(w, A) = \mathtt{Sol}(I, k)$,
- $f(p, A) \cap f(q, A) = \emptyset$, and
- $\mathcal{R}$ in FPT and f in FPT-delay time.

- $\mathcal{R}$:
  - T_0 in $O(m \cdot d)$ time (Chiba and Nishizeki)
  - T_1 in $O(n \cdot d \cdot |D|)$ (almost brute force)
  - edge deletion in linear time
  - all twins in linear time (partition refinement)
  - deleting all but one twin in linear time

Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,
- $|I'|, k \leq g(k)$,
- $\bigcup_{w:\mathtt{Sol}(I',k')} f(w, A) = \mathtt{Sol}(I, k)$,
- $f(p, A) \cap f(q, A) = \emptyset$, and
- $\mathcal{R}$ in FPT and f in FPT-delay time.

- $\mathcal{R}$:
  - T_0 in $O(m \cdot d)$ time (Chiba and Nishizeki)
  - T_1 in $O(n \cdot d \cdot |D|)$ (almost brute force)
  - edge deletion in linear time
  - all twins in linear time (partition refinement)
  - deleting all but one twin in linear time

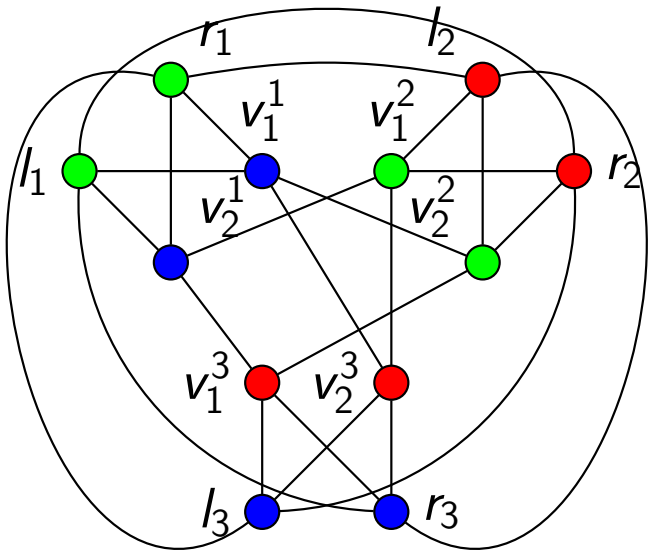
Distance to d-Degenerate Graphs (Proof)

- $(I, k) \in L \Leftrightarrow (I', k') \in L$,
- $|I'|, k \leq g(k)$,
- $\bigcup_{w:\mathtt{Sol}(I',k')} f(w, A) = \mathtt{Sol}(I, k)$,
- $f(p, A) \cap f(q, A) = \emptyset$, and
- $\mathcal{R}$ in FPT and f in FPT-delay time.

- $\mathcal{R}$:
  - T_0 in $O(m \cdot d)$ time (Chiba and Nishizeki)
  - T_1 in $O(n \cdot d \cdot |D|)$ (almost brute force)
  - edge deletion in linear time
  - all twins in linear time (partition refinement)
  - deleting all but one twin in linear time
- f: constant delay time (trivial lookup)

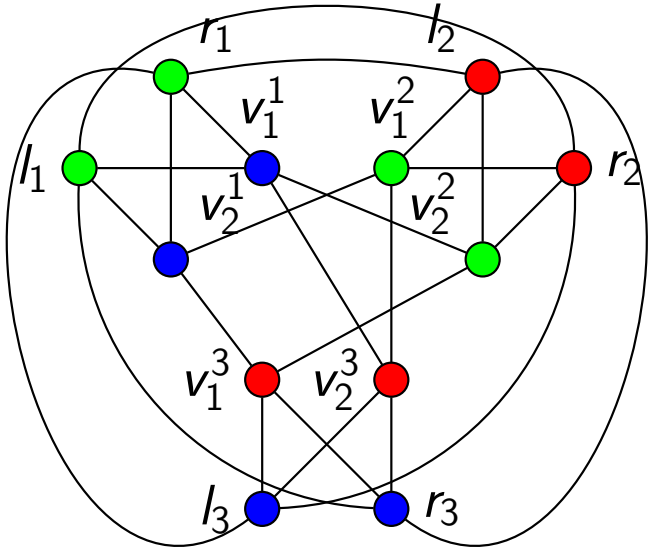
One Reduction to Rule Them All (Proof)

1. $k' \leq 9$



One Reduction to Rule Them All (Proof)

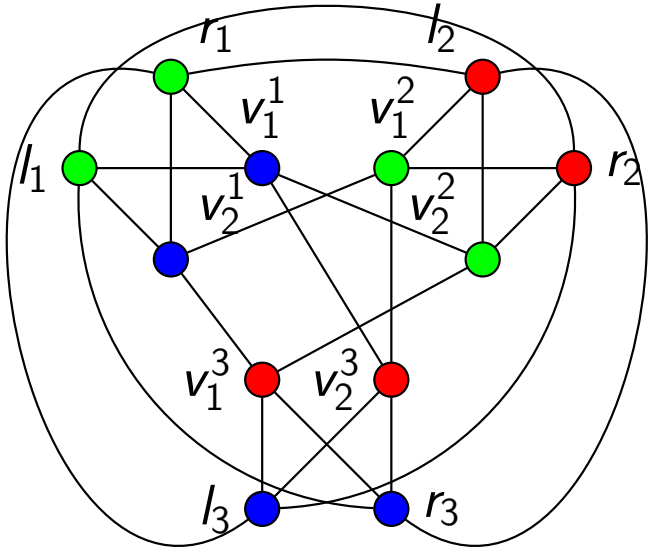
1. $k' \leq 9$



- chromatic number

One Reduction to Rule Them All (Proof)

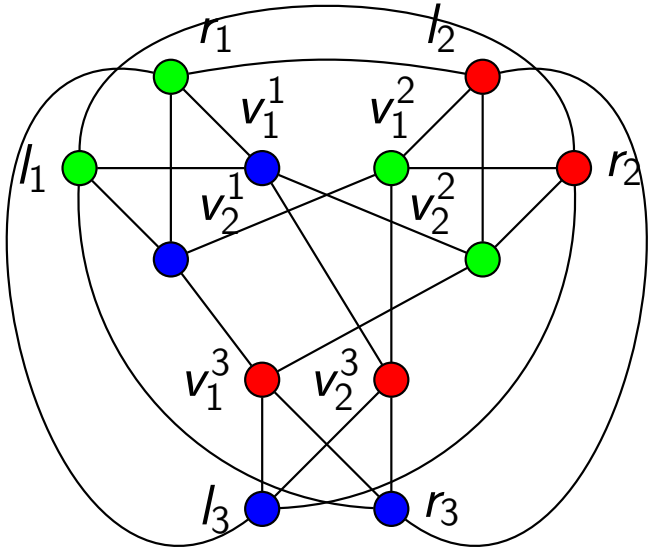
1. $k' \leq 9$



- chromatic number
- domination number (l_1, l_2, l_3)

One Reduction to Rule Them All (Proof)

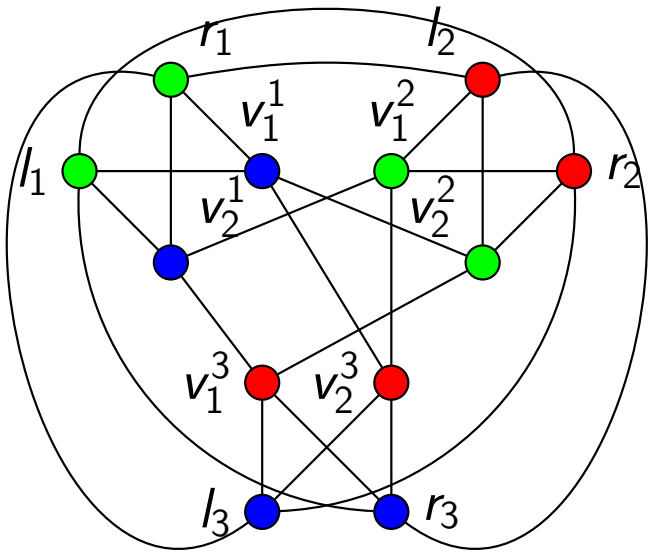
1. $k' \leq 9$



- chromatic number
- domination number (l_1, l_2, l_3)
- diameter

One Reduction to Rule Them All (Proof)

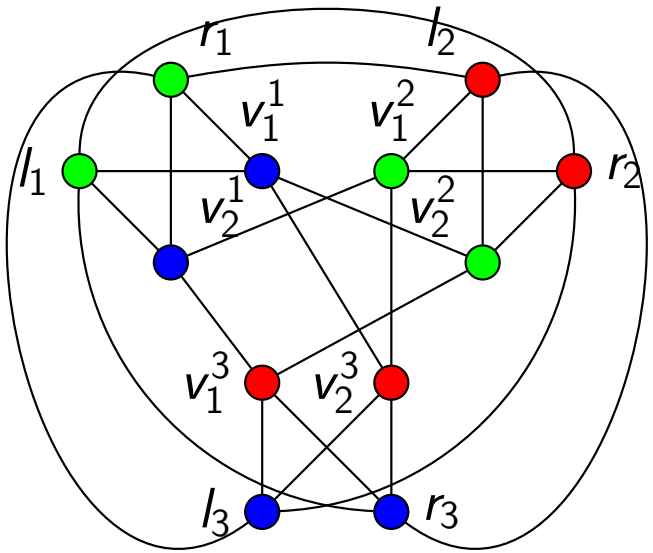
1. $k' \leq 9$
2. $I \in L \Leftrightarrow I' \in L$



One Reduction to Rule Them All (Proof)

1. $k' \leq 9$
2. $I \in L \Leftrightarrow I' \in L$

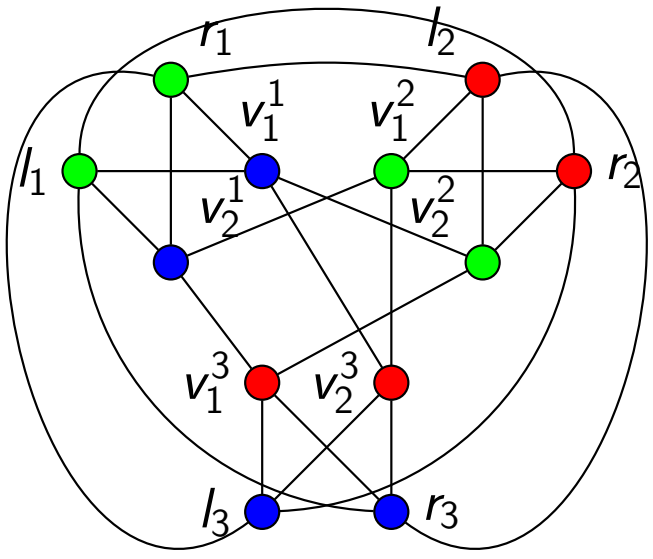
- l_i, r_i are not part of any triangle.



One Reduction to Rule Them All (Proof)

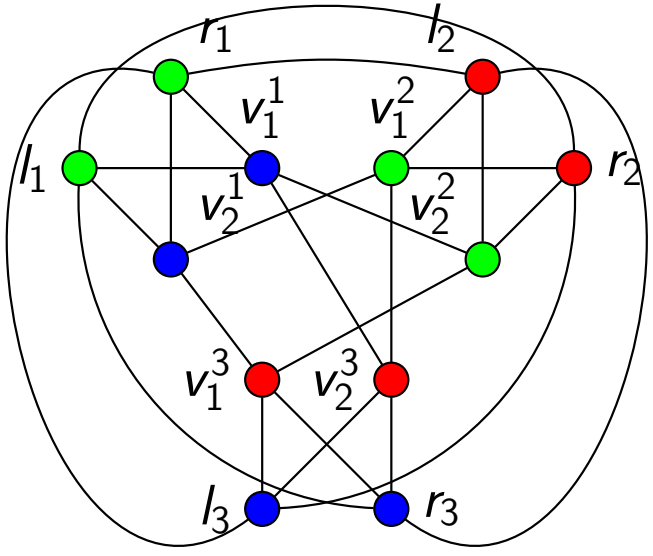
1. $k' \leq 9$
2. $I \in L \Leftrightarrow I' \in L$

- l_i, r_i are not part of any triangle.
- Each triangle has to contain one red, one blue and one green vertex.



One Reduction to Rule Them All (Proof)

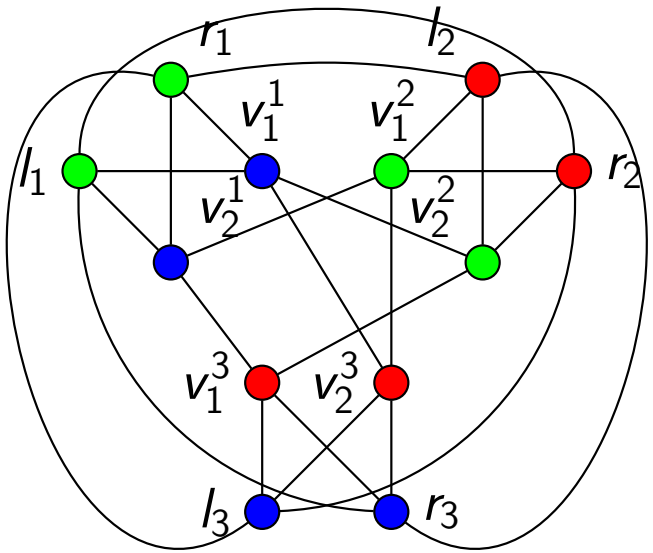
1. $k' \leq 9$
2. $I \in L \Leftrightarrow I' \in L$



- l_i, r_i are not part of any triangle.
- Each triangle has to contain one red, one blue and one green vertex.
- $(v_i^1, v_j^2, v_l^3) \Leftrightarrow (v_i, v_j, v_l)$

One Reduction to Rule Them All (Proof)

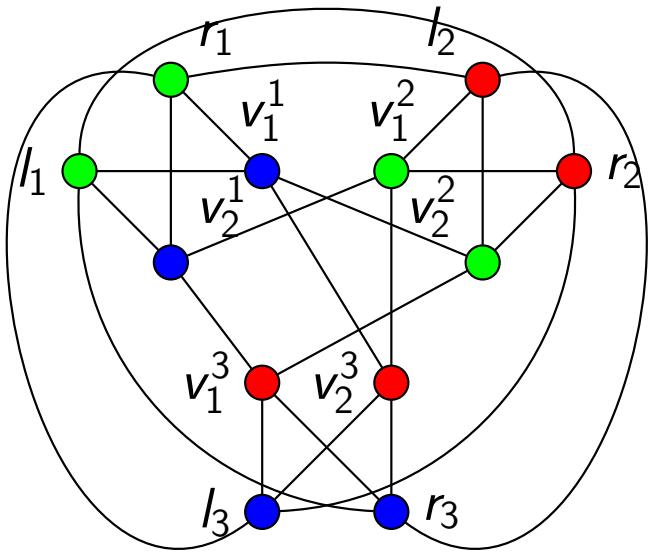
1. $k' \leq 9$
2. $I \in L \Leftrightarrow I' \in L$
3. $|I'| \in O(|I|)$



One Reduction to Rule Them All (Proof)

1. $k' \leq 9$
2. $I \in L \Leftrightarrow I' \in L$
3. $|I'| \in O(|I|)$

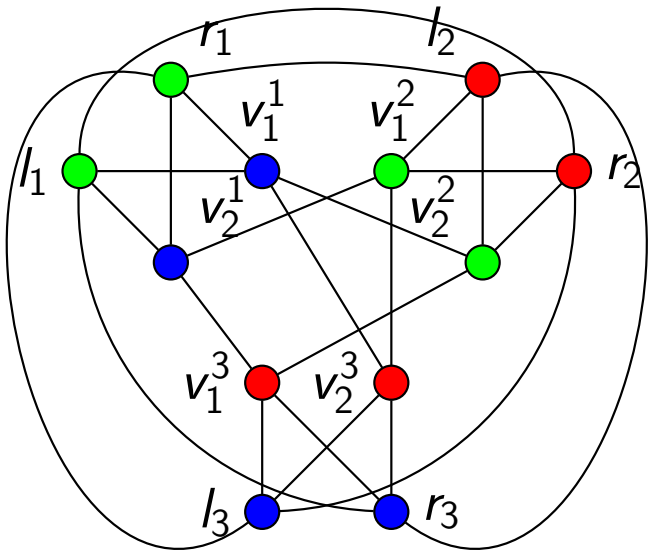
$n' = 3n + 6$
$m' = 6m + 6n + 6$



One Reduction to Rule Them All (Proof)

1. $k' \leq 9$
2. $I \in L \Leftrightarrow I' \in L$
3. $|I'| \in O(|I|)$
4. Takes $O(|I|)$ time to compute.

$n' = 3n + 6$
$m' = 6m + 6n + 6$



One Reduction to Rule Them All (Proof)

1. $k' \leq 9$
2. $I \in L \Leftrightarrow I' \in L$
3. $|I'| \in O(|I|)$
4. Takes $O(|I|)$ time to compute.

$n' = 3n + 6$
$m' = 6m + 6n + 6$

Each operation takes constant time.