Julius-Maximilians-UNIVERSITÄT WÜRZBURG

Lehrstuhl für INFORMATIK I Algorithmen & Komplexität

Institut für Informatik

Exakte Algorithmen

Lecture 9.2 Reductions and the $W[t]$ Hierarchy

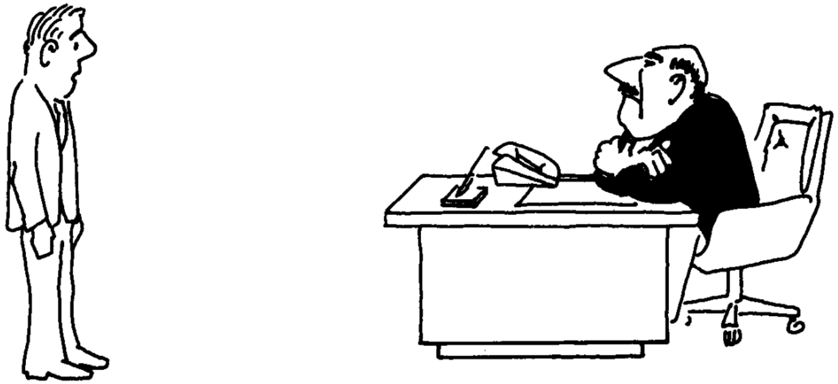
Based on: [Parameterized Algorithms: §13]

(slides by Thomas Bläsius)

Alexander Wolff

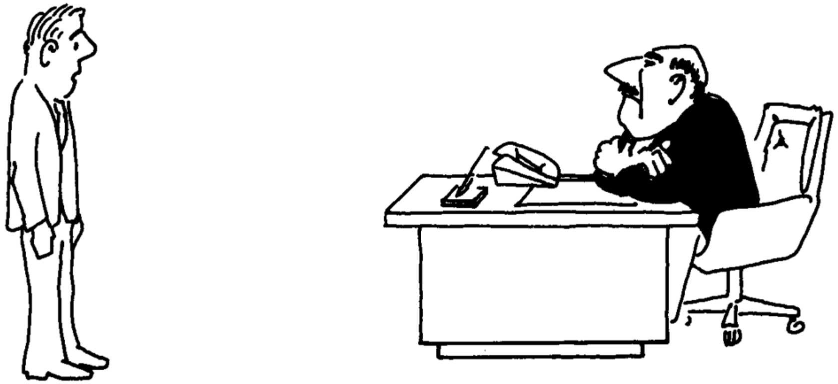
Lehrstuhl für Informatik I

How to obtain lower bounds?

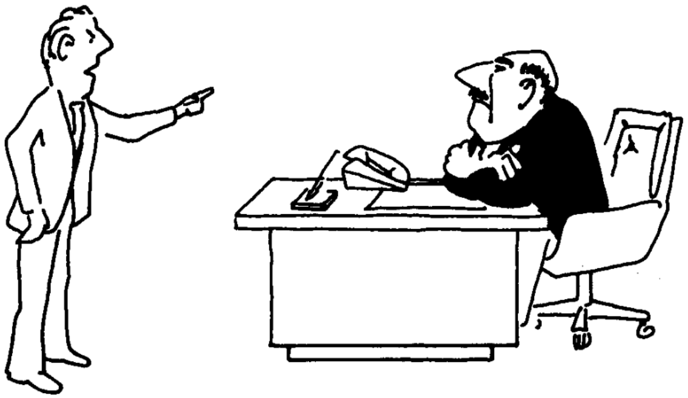


“I can’t find an efficient algorithm, I guess I’m just too dumb.”

How to obtain lower bounds?

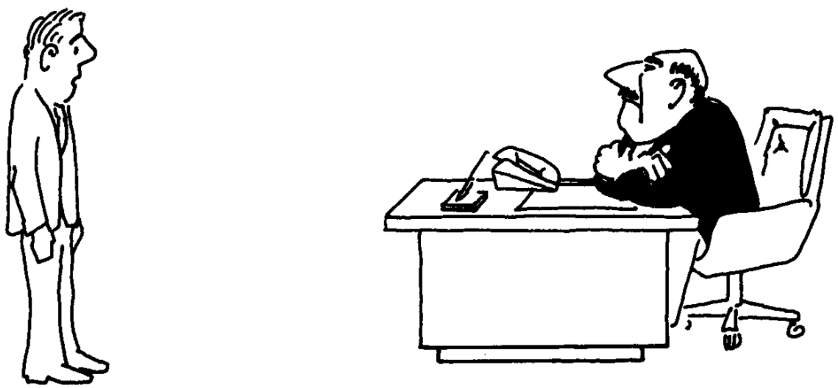


"I can't find an efficient algorithm, I guess I'm just too dumb."

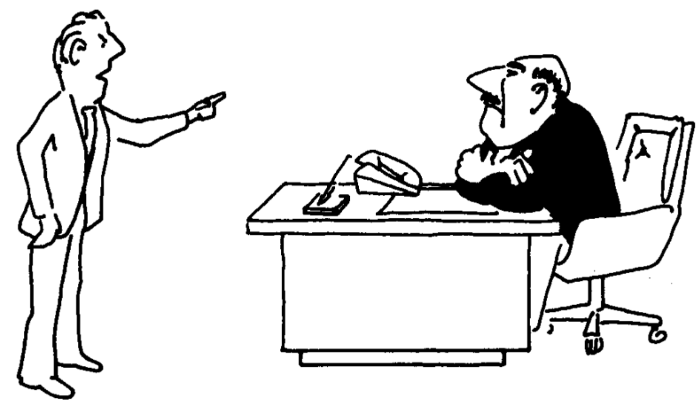


"I can't find an efficient algorithm, because no such algorithm is possible!"

How to obtain lower bounds?



“I can’t find an efficient algorithm, I guess I’m just too dumb.”

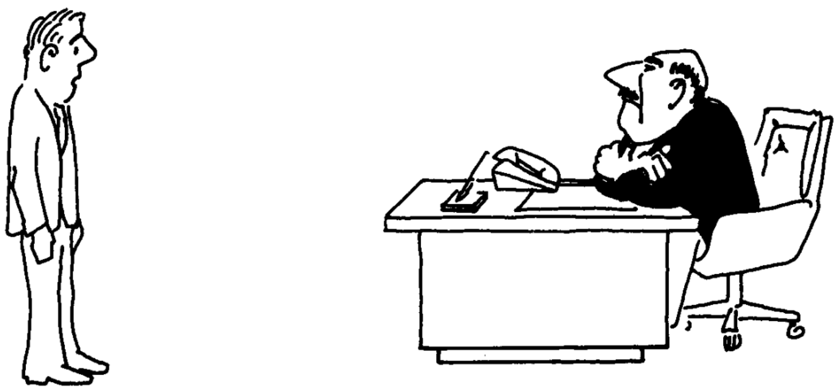


“I can’t find an efficient algorithm, because no such algorithm is possible!”

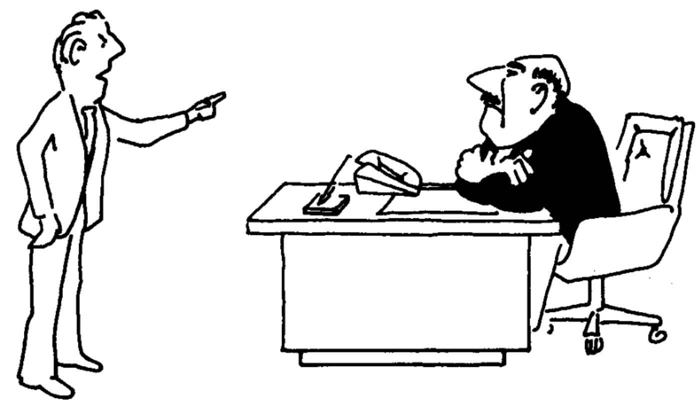
Problem

• difficult to show absence of “nice” algorithms

How to obtain lower bounds?



"I can't find an efficient algorithm, I guess I'm just too dumb."



"I can't find an efficient algorithm, because no such algorithm is possible!"

Problem

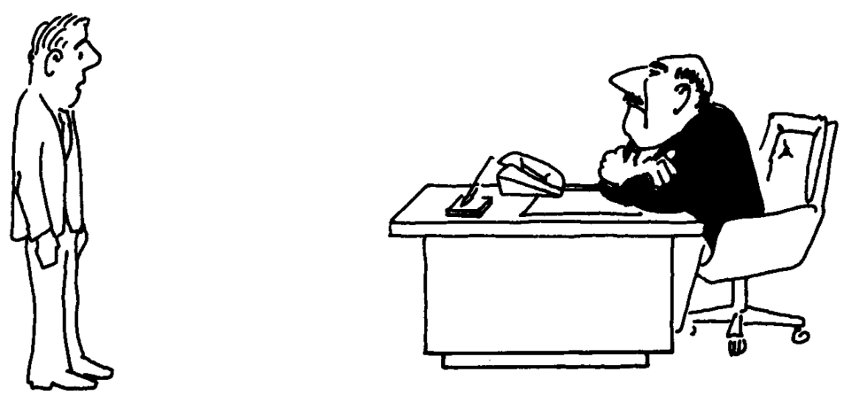
- difficult to show absence of "nice" algorithms

Solution

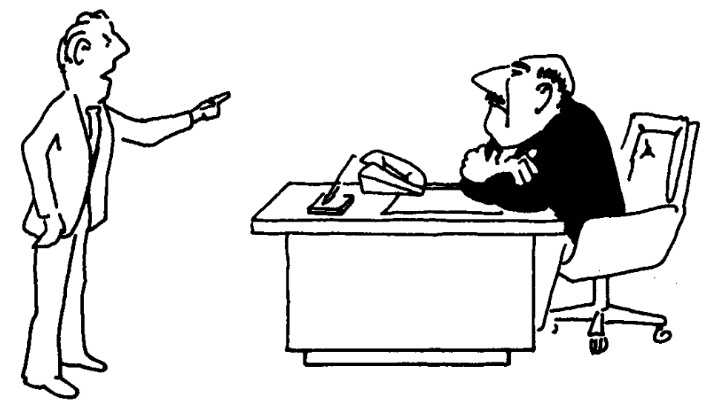
- argue that having an efficient algorithm⇒ some well studied diffcult problem can be solved efficiently

"I can't find an efficient algorithm, but neither can all these famous people."

How to obtain lower bounds?



"I can't find an efficient algorithm, I guess I'm just too dumb."



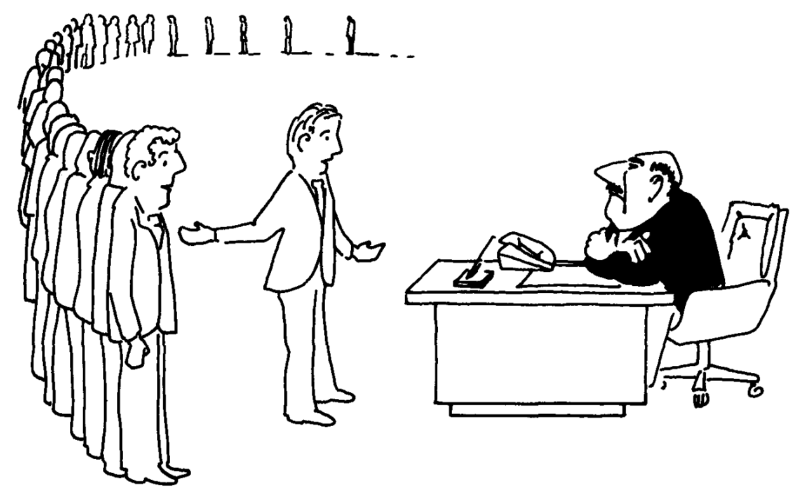
"I can't find an efficient algorithm, because no such algorithm is possible!"

Problem

- difficult to show absence of "nice" algorithms

Solution

- argue that having an efficient algorithm$\Rightarrow$ some well studied diffcult problem can be solved efficiently
- Tool: reduction between problems



"I can't find an efficient algorithm, but neither can all these famous people."

Polynomial Reduction

Reduction from Problem $\mathcal{L}$ to Problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$ so that
- I is a YES-instance $\Leftrightarrow$ I' is a YES-instance

Polynomial Reduction

Reduction from Problem $\mathcal{L}$ to Problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$ so that
- I is a YES-instance $\Leftrightarrow$ I' is a YES-instance
- the mapping can be computed in polynomial time

Polynomial Reduction

Reduction from Problem $\mathcal{L}$ to Problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$ so that
- I is a YES-instance $\Leftrightarrow$ I' is a YES-instance
- the mapping can be computed in polynomial time

Implications

- polynomial time algorithm for $\mathcal{L}'$ $\Rightarrow$ polynomial time algorithm for $\mathcal{L}$

Polynomial Reduction

Reduction from Problem $\mathcal{L}$ to Problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$ so that
- I is a YES-instance $\Leftrightarrow$ I' is a YES-instance
- the mapping can be computed in polynomial time

Implications

- polynomial time algorithm for $\mathcal{L}'$ $\Rightarrow$ polynomial time algorithm for $\mathcal{L}$
- solving my problem $\mathcal{L}'$ efficiently $\Rightarrow$ solving "difficult" problem $\mathcal{L}$ efficiently

(here, we use "efficient" $\sim$ "polynomial")

Polynomial Reduction

Reduction from Problem $\mathcal{L}$ to Problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$ so that
- I is a YES-instance $\Leftrightarrow$ I' is a YES-instance
- the mapping can be computed in polynomial time

Implications

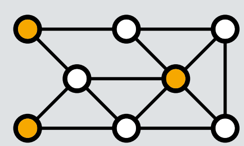
- polynomial time algorithm for $\mathcal{L}'$ $\Rightarrow$ polynomial time algorithm for $\mathcal{L}$
- solving my problem $\mathcal{L}'$ efficiently $\Rightarrow$ solving “difficult” problem $\mathcal{L}$ efficiently

(here, we use “efficient” $\sim$ “polynomial”)

Easy Example

- reduce INDEPENDENT SET to VERTEX COVER

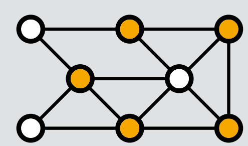
INDEPENDENT SET



$V' \subseteq V$, so that

$\forall e \in E\colon |e \cap V'| \leq 1$

VERTEX COVER



$V' \subseteq V$, so that

$\forall e \in E\colon |e \cap V'| \geq 1$

Polynomial Reduction

Reduction from Problem $\mathcal{L}$ to Problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$ so that
- I is a YES-instance $\Leftrightarrow$ I' is a YES-instance
- the mapping can be computed in polynomial time

Implications

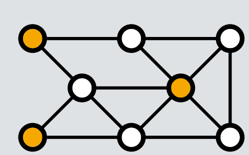
- polynomial time algorithm for $\mathcal{L}'$ $\Rightarrow$ polynomial time algorithm for $\mathcal{L}$
- solving my problem $\mathcal{L}'$ efficiently $\Rightarrow$ solving "difficult" problem $\mathcal{L}$ efficiently

(here, we use "efficient" $\sim$ "polynomial")

Easy Example

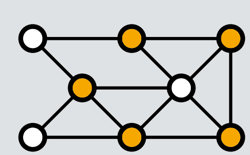
- reduce Independent Set to Vertex Cover
- trivial reduction: G has IS of size k $\Leftrightarrow$ G has VC of size $n - k$

Independent Set



$V' \subseteq V$, so that
$\forall e \in E\colon |e \cap V'| \leq 1$

Vertex Cover



$V' \subseteq V$, so that
$\forall e \in E\colon |e \cap V'| \geq 1$

Polynomial Reduction

Reduction from Problem $\mathcal{L}$ to Problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$ so that
- I is a YES-instance $\Leftrightarrow$ I' is a YES-instance
- the mapping can be computed in polynomial time

Implications

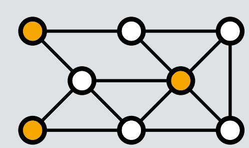
- polynomial time algorithm for $\mathcal{L}'$ $\Rightarrow$ polynomial time algorithm for $\mathcal{L}$
- solving my problem $\mathcal{L}'$ efficiently $\Rightarrow$ solving “difficult” problem $\mathcal{L}$ efficiently

(here, we use “efficient” $\sim$ “polynomial”)

Easy Example

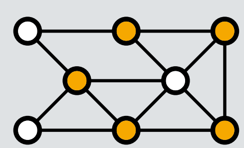
- reduce INDEPENDENT SET to VERTEX COVER
- trivial reduction: G has IS of size k $\Leftrightarrow$ G has VC of size $n-k$
- also: VC $\in$ P $\Rightarrow$ IS $\in$ P

INDEPENDENT SET



$V' \subseteq V$, so that
$\forall e \in E\colon |e \cap V'| \leq 1$

VERTEX COVER



$V' \subseteq V$, so that
$\forall e \in E\colon |e \cap V'| \geq 1$

Polynomial Reduction

Reduction from Problem $\mathcal{L}$ to Problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$ so that
- I is a YES-instance $\Leftrightarrow$ I' is a YES-instance
- the mapping can be computed in polynomial time

Implications

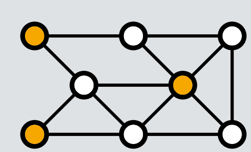
- polynomial time algorithm for $\mathcal{L}'$ $\Rightarrow$ polynomial time algorithm for $\mathcal{L}$
- solving my problem $\mathcal{L}'$ efficiently $\Rightarrow$ solving “difficult” problem $\mathcal{L}$ efficiently

(here, we use “efficient” $\sim$ “polynomial”)

Easy Example

- reduce Independent Set to Vertex Cover
- trivial reduction: G has IS of size k $\Leftrightarrow$ G has VC of size $n - k$
- also: VC $\in$ P $\Rightarrow$ IS $\in$ P
- what about VC $\in$ FPT $\Rightarrow$ IS $\in$ FPT?

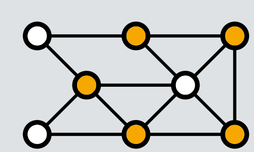
Independent Set



$V' \subseteq V$, so that

$\forall e \in E\colon |e \cap V'| \leq 1$

Vertex Cover



$V' \subseteq V$, so that

$\forall e \in E\colon |e \cap V'| \geq 1$

Polynomial Reduction

Reduction from Problem $\mathcal{L}$ to Problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$ so that
- I is a YES-instance $\Leftrightarrow$ I' is a YES-instance
- the mapping can be computed in polynomial time

Implications

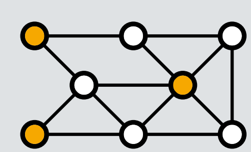
- polynomial time algorithm for $\mathcal{L}'$ $\Rightarrow$ polynomial time algorithm for $\mathcal{L}$
- solving my problem $\mathcal{L}'$ efficiently $\Rightarrow$ solving "difficult" problem $\mathcal{L}$ efficiently

(here, we use "efficient" $\sim$ "polynomial")

Easy Example

- reduce Independent Set to Vertex Cover
- trivial reduction: G has IS of size k $\Leftrightarrow$ G has VC of size $n - k$
- also: VC $\in$ P $\Rightarrow$ IS $\in$ P
- what about VC $\in$ FPT $\Rightarrow$ IS $\in$ FPT?
  $\rightarrow$ NO, the parameter depends on n

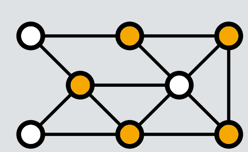
Independent Set



$V' \subseteq V$, so that

$\forall e \in E \colon |e \cap V'| \leq 1$

Vertex Cover



$V' \subseteq V$, so that

$\forall e \in E \colon |e \cap V'| \geq 1$

Polynomial Reduction

Reduction from Problem $\mathcal{L}$ to Problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$ so that
- I is a YES-instance $\Leftrightarrow$ I' is a YES-instance
- the mapping can be computed in polynomial time

Implications

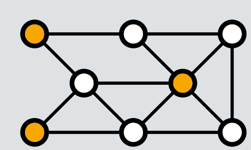
- polynomial time algorithm for $\mathcal{L}'$ $\Rightarrow$ polynomial time algorithm for $\mathcal{L}$
- solving my problem $\mathcal{L}'$ efficiently $\Rightarrow$ solving "difficult" problem $\mathcal{L}$ efficiently

(here, we use "efficient" $\sim$ "polynomial")

Easy Example

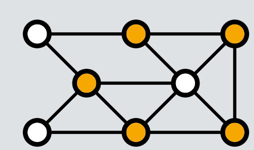
- reduce INDEPENDENT SET to VERTEX COVER
- trivial reduction: G has IS of size k $\Leftrightarrow$ G has VC of size $n-k$
- also: VC $\in$ P $\Rightarrow$ IS $\in$ P
- what about VC $\in$ FPT $\Rightarrow$ IS $\in$ FPT?
  $\rightarrow$ NO, the parameter depends on n
- for "efficient" $\sim$ "FPT" we need a different type of reduction

INDEPENDENT SET



$V' \subseteq V$, so that
$\forall e \in E \colon |e \cap V'| \leq 1$

VERTEX COVER



$V' \subseteq V$, so that
$\forall e \in E \colon |e \cap V'| \geq 1$

Parameterized Reductions

Reduction from Problem $\mathcal{L}$ to Problem $\mathcal{L}'$

- map each instance (I, k) of $\mathcal{L}$ to an instance (I', k') of $\mathcal{L}'$ so that
- (I, k) is a YES-instance $\Leftrightarrow$ (I', k') is a YES-instance where $k' \leq g(k)$
- the map must be computable in FPT-time $(f(k) \cdot |I|^{O(1)})$

(f and g must also be computable functions)

Parameterized Reductions

Reduction from Problem $\mathcal{L}$ to Problem $\mathcal{L}'$

- map each instance (I, k) of $\mathcal{L}$ to an instance (I', k') of $\mathcal{L}'$ so that
- (I, k) is a YES-instance $\Leftrightarrow$ (I', k') is a YES-instance where $k' \leq g(k)$
- the map must be computable in FPT-time $(f(k) \cdot |I|^{O(1)})$

(f and g must also be computable functions)

Implications

- FPT-algorithm for $\mathcal{L}'$ $\Rightarrow$ FPT algorithm for $\mathcal{L}$

Parameterized Reductions

Reduction from Problem $\mathcal{L}$ to Problem $\mathcal{L}'$

- map each instance (I, k) of $\mathcal{L}$ to an instance (I', k') of $\mathcal{L}'$ so that
- (I, k) is a YES-instance $\Leftrightarrow$ (I', k') is a YES-instance where $k' \leq g(k)$
- the map must be computable in FPT-time $(f(k) \cdot |I|^{O(1)})$

(f and g must also be computable functions)

Implications

- FPT-algorithm for $\mathcal{L}'$ $\Rightarrow$ FPT algorithm for $\mathcal{L}$
- solving my problem $\mathcal{L}'$ efficiently $\Rightarrow$ solving "difficult" problem $\mathcal{L}$ efficiently

(now, "efficient" $\sim$ "FPT")

Parameterized Reductions

Reduction from Problem $\mathcal{L}$ to Problem $\mathcal{L}'$

- map each instance (I, k) of $\mathcal{L}$ to an instance (I', k') of $\mathcal{L}'$ so that
- (I, k) is a YES-instance $\Leftrightarrow$ (I', k') is a YES-instance where $k' \leq g(k)$
- the map must be computable in FPT-time $(f(k) \cdot |I|^{O(1)})$

(f and g must also be computable functions)

Implications

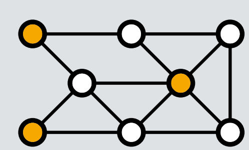
- FPT-algorithm for $\mathcal{L}'$ $\Rightarrow$ FPT algorithm for $\mathcal{L}$
- solving my problem $\mathcal{L}'$ efficiently $\Rightarrow$ solving "difficult" problem $\mathcal{L}$ efficiently

(now, "efficient" $\sim$ "FPT")

Easy Example

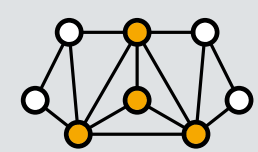
- reduce Independent Set to Clique

Independent Set



$V' \subseteq V$, so that
$\forall e \in E\colon |e \cap V'| \leq 1$

Clique



$V' \subseteq V$, so that
$G[V']$ complete

Parameterized Reductions

Reduction from Problem $\mathcal{L}$ to Problem $\mathcal{L}'$

- map each instance (I, k) of $\mathcal{L}$ to an instance (I', k') of $\mathcal{L}'$ so that
- (I, k) is a YES-instance $\Leftrightarrow$ (I', k') is a YES-instance where $k' \leq g(k)$
- the map must be computable in FPT-time $(f(k) \cdot |I|^{O(1)})$

(f and g must also be computable functions)

Implications

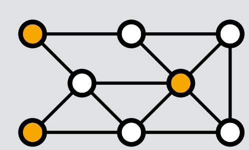
- FPT-algorithm for $\mathcal{L}'$ $\Rightarrow$ FPT algorithm for $\mathcal{L}$
- solving my problem $\mathcal{L}'$ efficiently $\Rightarrow$ solving "difficult" problem $\mathcal{L}$ efficiently

(now, "efficient" $\sim$ "FPT")

Easy Example

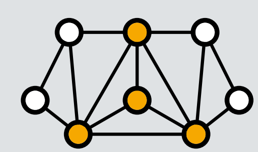
- reduce Independent Set to Clique
- G has a size k IS $\Leftrightarrow$ edge-complement of G has a size k Clique.

Independent Set



$V' \subseteq V$, so that $\forall e \in E\colon |e \cap V'| \leq 1$

Clique



$V' \subseteq V$, so that $G[V']$ complete

Parameterized Reductions

Reduction from Problem $\mathcal{L}$ to Problem $\mathcal{L}'$

- map each instance (I, k) of $\mathcal{L}$ to an instance (I', k') of $\mathcal{L}'$ so that
- (I, k) is a YES-instance $\Leftrightarrow$ (I', k') is a YES-instance where $k' \leq g(k)$
- the map must be computable in FPT-time $(f(k) \cdot |I|^{O(1)})$

(f and g must also be computable functions)

Implications

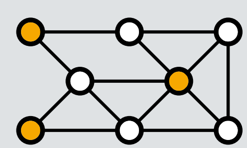
- FPT-algorithm for $\mathcal{L}'$ $\Rightarrow$ FPT algorithm for $\mathcal{L}$
- solving my problem $\mathcal{L}'$ efficiently $\Rightarrow$ solving "difficult" problem $\mathcal{L}$ efficiently

(now, "efficient" $\sim$ "FPT")

Easy Example

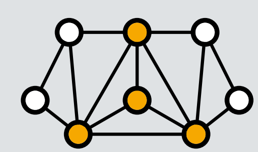
- reduce Independent Set to Clique
- G has a size k IS $\Leftrightarrow$ edge-complement of G has a size k Clique.
- also: Clique $\in$ FPT $\Rightarrow$ IS $\in$ FPT

Independent Set



$V' \subseteq V$, so that
$\forall e \in E \colon |e \cap V'| \leq 1$

Clique



$V' \subseteq V$, so that
$G[V']$ complete

Parameterized Reductions

Reduction from Problem $\mathcal{L}$ to Problem $\mathcal{L}'$

- map each instance (I, k) of $\mathcal{L}$ to an instance (I', k') of $\mathcal{L}'$ so that
- (I, k) is a YES-instance $\Leftrightarrow$ (I', k') is a YES-instance where $k' \leq g(k)$
- the map must be computable in FPT-time $(f(k) \cdot |I|^{O(1)})$

(f and g must also be computable functions)

Implications

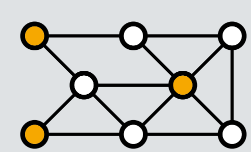
- FPT-algorithm for $\mathcal{L}'$ $\Rightarrow$ FPT algorithm for $\mathcal{L}$
- solving my problem $\mathcal{L}'$ efficiently $\Rightarrow$ solving "difficult" problem $\mathcal{L}$ efficiently

(now, "efficient" $\sim$ "FPT")

Easy Example

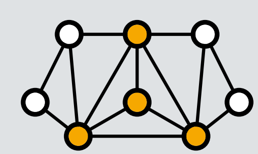
- reduce INDEPENDENT SET to CLIQUE
- G has a size k IS $\Leftrightarrow$ edge-complement of G has a size k Clique.
- also: CLIQUE $\in$ FPT $\Rightarrow$ IS $\in$ FPT
- reverse reduction also applies:
  CLIQUE $\in$ FPT $\Leftrightarrow$ IS $\in$ FPT

INDEPENDENT SET



$V' \subseteq V$, so that
$\forall e \in E\colon |e \cap V'| \leq 1$

CLIQUE



$V' \subseteq V$, so that
$G[V']$ complete

Parameterized Reductions

Reduction from Problem $\mathcal{L}$ to Problem $\mathcal{L}'$

- map each instance (I, k) of $\mathcal{L}$ to an instance (I', k') of $\mathcal{L}'$ so that
- (I, k) is a YES-instance $\Leftrightarrow$ (I', k') is a YES-instance where $k' \leq g(k)$
- the map must be computable in FPT-time $(f(k) \cdot |I|^{O(1)})$

(f and g must also be computable functions)

Implications

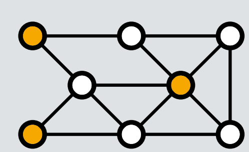
- FPT-algorithm for $\mathcal{L}'$ $\Rightarrow$ FPT algorithm for $\mathcal{L}$
- solving my problem $\mathcal{L}'$ efficiently $\Rightarrow$ solving "difficult" problem $\mathcal{L}$ efficiently

(now, "efficient" $\sim$ "FPT")

Easy Example

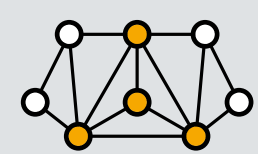
- reduce INDEPENDENT SET to CLIQUE
- G has a size k IS $\Leftrightarrow$ edge-complement of G has a size k Clique.
- also: CLIQUE $\in$ FPT $\Rightarrow$ IS $\in$ FPT
- reverse reduction also applies:
  CLIQUE $\in$ FPT $\Leftrightarrow$ IS $\in$ FPT
- expectation: CLIQUE, IS $\notin$ FPT

INDEPENDENT SET



$V' \subseteq V$, so that
$\forall e \in E \colon |e \cap V'| \leq 1$

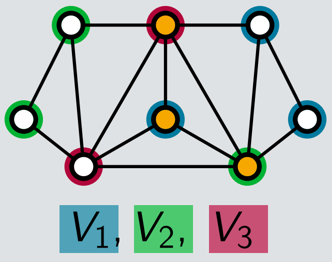
CLIQUE



$V' \subseteq V$, so that
$G[V']$ complete

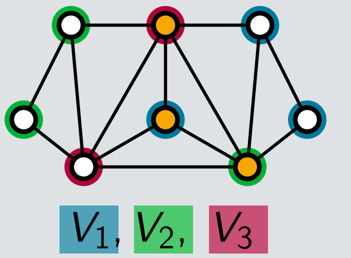
Colored Cliques

Problem: Multicolored Clique
Given: Graph $G = (V, E)$, parameter k,
and partition $(V_1, \ldots, V_k)$ of V.
Find: Clique $V' \subseteq V$ of size k, so that $|V' \cap V_i| = 1$ for each i.

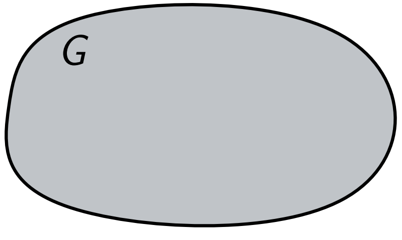


Colored Cliques

Problem: **MULTICOLORED CLIQUE**
Given: Graph $G = (V, E)$, parameter k,
and partition $(V_1, \ldots, V_k)$ of V.
Find: Clique $V' \subseteq V$ of size k, so that $|V' \cap V_i| = 1$ for each i.



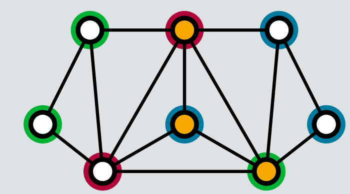
Reduction: from CLIQUE to MULTICOLORED CLIQUE



Instance $(G, 3)$ of CLIQUE

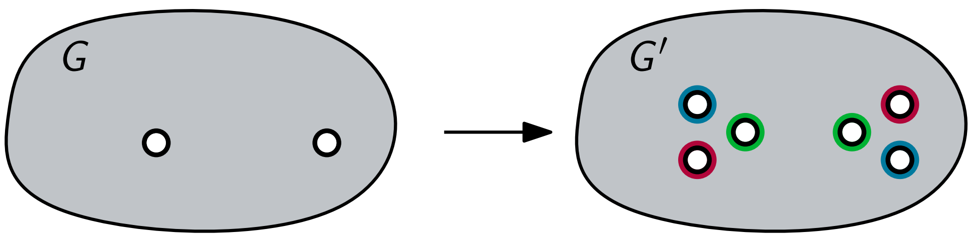
Colored Cliques

Problem: MULTICOLORED CLIQUE
Given: Graph $G = (V, E)$, parameter k,
and partition $(V_1, \ldots, V_k)$ of V.
Find: Clique $V' \subseteq V$ of size k, so that $|V' \cap V_i| = 1$ for each i.



V_1, V_2, V_3

Reduction: from CLIQUE to MULTICOLORED CLIQUE



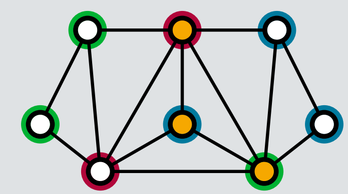
Instance $(G, 3)$ of CLIQUE

Instance of MC CLIQUE: $(G', 3, (V_1, V_2, V_3))$

- copy each $v \in V$ to $v^1, \ldots, v^k$, and set $v^i \in V_i$

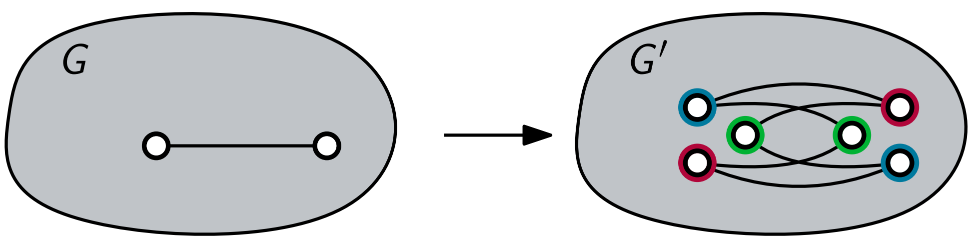
Colored Cliques

Problem: **Multicolored Clique**
Given: Graph $G = (V, E)$, parameter k,
and partition $(V_1, \ldots, V_k)$ of V.
Find: Clique $V' \subseteq V$ of size k, so that $|V' \cap V_i| = 1$ for each i.



V_1, V_2, V_3

Reduction: from Clique to Multicolored Clique



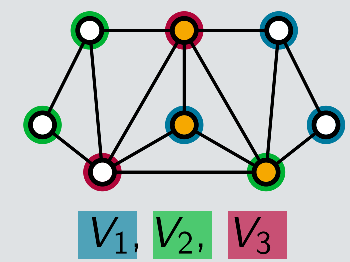
Instance $(G, 3)$ of Clique

Instance of MC Clique: $(G', 3, (V_1, V_2, V_3))$

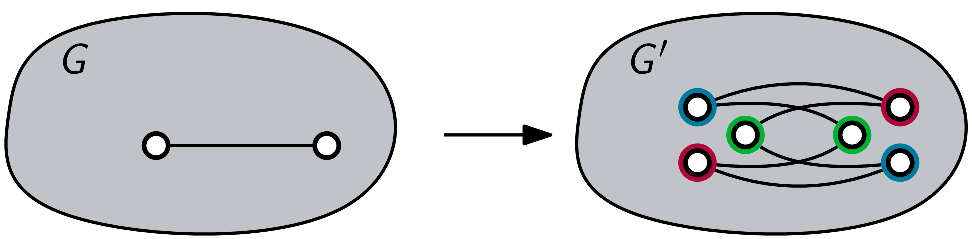
- copy each $v \in V$ to $v^1, \ldots, v^k$, and set $v^i \in V_i$
- for each $uv \in E$ connect u^i with v^j for each $i \neq j$
- $k = k'$

Colored Cliques

Problem: MULTICOLORED CLIQUE
Given: Graph $G = (V, E)$, parameter k,
and partition $(V_1, \ldots, V_k)$ of V.
Find: Clique $V' \subseteq V$ of size k, so that $|V' \cap V_i| = 1$ for each i.



Reduction: from CLIQUE to MULTICOLORED CLIQUE



Instance $(G, 3)$ of CLIQUE

Instance of MC CLIQUE: $(G', 3, (V_1, V_2, V_3))$

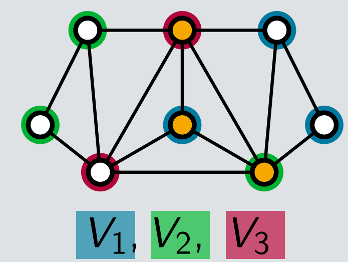
- copy each $v \in V$ to $v^1, \ldots, v^k$, and set $v^i \in V_i$
- for each $uv \in E$ connect u^i with v^j for each $i \neq j$
- $k = k'$

G has a size k clique $\Rightarrow$ G' has a size k colored clique

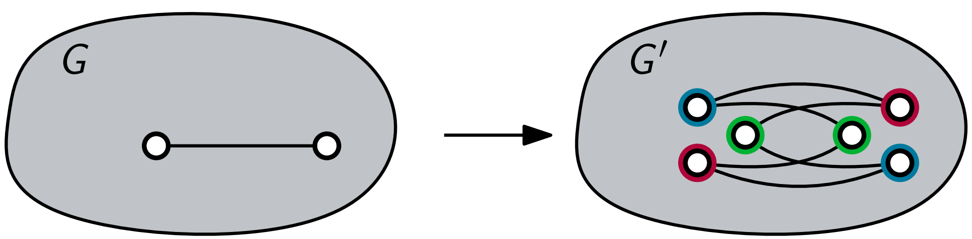
- Let $v_1, \ldots, v_k$ be a clique in G
- then $v_1^1, \ldots, v_k^k$ is a clique in G'

Colored Cliques

Problem: **Multicolored Clique**
Given: Graph $G = (V, E)$, parameter k,
and partition $(V_1, \ldots, V_k)$ of V.
Find: Clique $V' \subseteq V$ of size k, so that $|V' \cap V_i| = 1$ for each i.



Reduction: from Clique to Multicolored Clique



Instance $(G, 3)$ of Clique

Instance of MC Clique: $(G', 3, (V_1, V_2, V_3))$

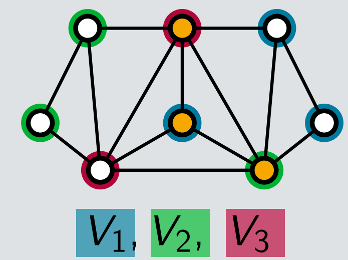
- copy each $v \in V$ to $v^1, \ldots, v^k$, and set $v^i \in V_i$
- for each $uv \in E$ connect u^i with v^j for each $i \neq j$
- $k = k'$

G has a size k clique $\Rightarrow$ G' has a size k colored clique

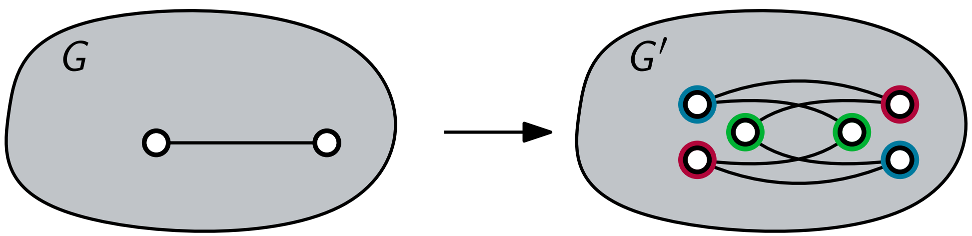
- Let $v_1, \ldots, v_k$ be a clique in G
- then $v_1^1, \ldots, v_k^k$ is a clique in G'
- all these vertices have distinct colors

Colored Cliques

Problem: Multicolored Clique
Given: Graph $G = (V, E)$, parameter k,
and partition $(V_1, \dots, V_k)$ of V.
Find: Clique $V' \subseteq V$ of size k, so that $|V' \cap V_i| = 1$ for each i.



Reduction: from Clique to Multicolored Clique



Instance $(G, 3)$ of Clique

Instance of MC Clique: $(G', 3, (V_1, V_2, V_3))$

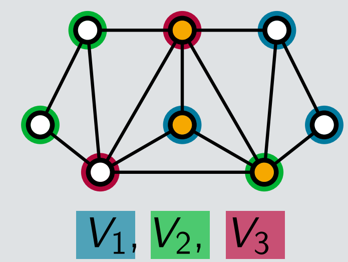
- copy each $v \in V$ to $v^1, \dots, v^k$, and set $v^i \in V_i$
- for each $uv \in E$ connect u^i with v^j for each $i \neq j$
- $k = k'$

G' has a size k colored clique $\Rightarrow$ G has a size k clique

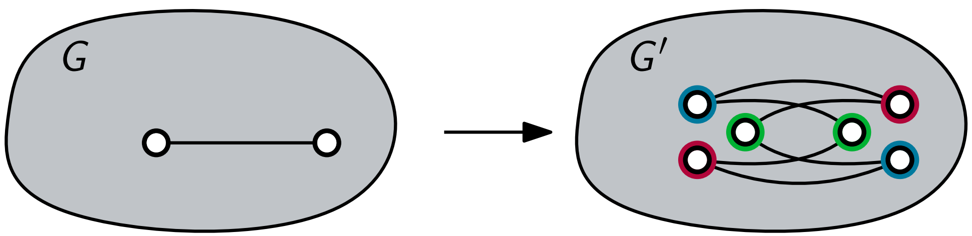
- Let $v^1_{\pi(1)}, \dots, v^k_{\pi(k)}$ be a colored clique in G with $\pi\colon \{1, \dots, k\} \to \{1, \dots n\}$

Colored Cliques

Problem: **Multicolored Clique**
Given: Graph $G = (V, E)$, parameter k,
and partition $(V_1, \ldots, V_k)$ of V.
Find: Clique $V' \subseteq V$ of size k, so that $|V' \cap V_i| = 1$ for each i.



Reduction: from Clique to Multicolored Clique



Instance $(G, 3)$ of Clique

Instance of MC Clique: $(G', 3, (V_1, V_2, V_3))$

- copy each $v \in V$ to $v^1, \ldots, v^k$, and set $v^i \in V_i$
- for each $uv \in E$ connect u^i with v^j for each $i \neq j$
- $k = k'$

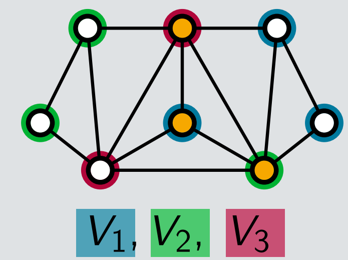
G' has a size k colored clique $\Rightarrow$ G has a size k clique

- Let $v^1_{\pi(1)}, \ldots, v^k_{\pi(k)}$ be a colored clique in G with $\pi\colon \{1, \ldots, k\} \to \{1, \ldots n\}$
- π is injective

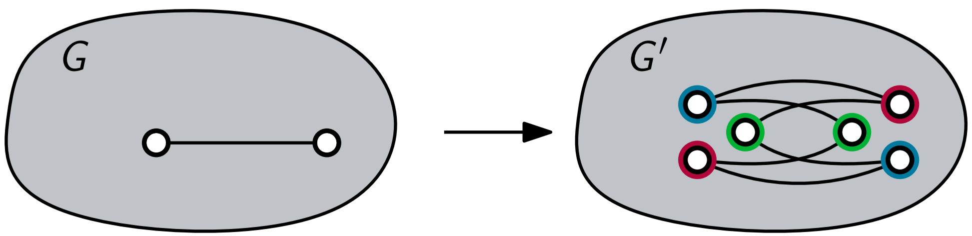
(the colored clique does not contain two copies of the same vertex)

Colored Cliques

Problem: Multicolored Clique
Given: Graph $G = (V, E)$, parameter k,
and partition $(V_1, \ldots, V_k)$ of V.
Find: Clique $V' \subseteq V$ of size k, so that $|V' \cap V_i| = 1$ for each i.



Reduction: from Clique to Multicolored Clique



Instance $(G, 3)$ of Clique

Instance of MC Clique: $(G', 3, (V_1, V_2, V_3))$

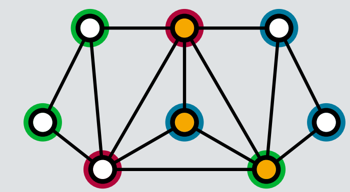
- copy each $v \in V$ to $v^1, \ldots, v^k$, and set $v^i \in V_i$
- for each $uv \in E$ connect u^i with v^j for each $i \neq j$
- $k = k'$

G' has a size k colored clique $\Rightarrow$ G has a size k clique

- Let $v^1_{\pi(1)}, \ldots, v^k_{\pi(k)}$ be a colored clique in G with $\pi\colon \{1, \ldots, k\} \to \{1, \ldots n\}$
- π is injective (the colored clique does not contain two copies of the same vertex)
- thus, $v_{\pi(1)}, \ldots, v_{\pi(k)}$ are k distinct nodes that form a clique in G

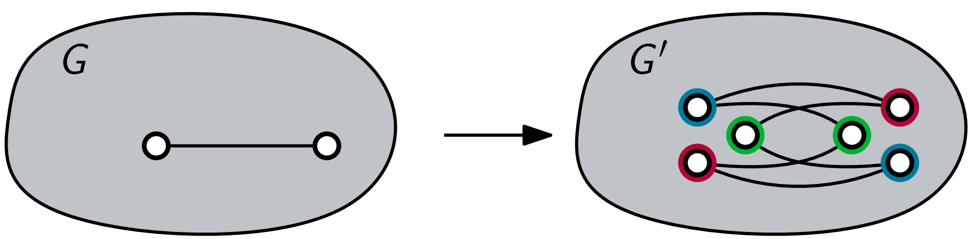
Colored Cliques

Problem: MULTICOLORED CLIQUE
Given: Graph $G = (V, E)$, parameter k,
and partition $(V_1, \ldots, V_k)$ of V.
Find: Clique $V' \subseteq V$ of size k, so that $|V' \cap V_i| = 1$ for each i.



V_1, V_2, V_3

Reduction: from CLIQUE to MULTICOLORED CLIQUE



Instance $(G, 3)$ of CLIQUE

Instance of MC CLIQUE: $(G', 3, (V_1, V_2, V_3))$

- copy each $v \in V$ to $v^1, \ldots, v^k$, and set $v^i \in V_i$
- for each $uv \in E$ connect u^i with v^j for each $i \neq j$
- $k = k'$

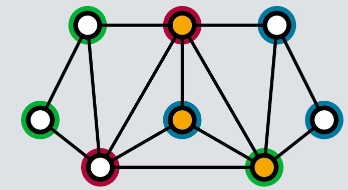
***G'* has a size *k* colored clique $\Rightarrow$ *G* has a size *k* clique**

- Let $v^1_{\pi(1)}, \ldots, v^k_{\pi(k)}$ be a colored clique in G with $\pi\colon \{1, \ldots, k\} \to \{1, \ldots n\}$
- π is injective
  (the colored clique does not contain two copies of the same vertex)
- thus, $v_{\pi(1)}, \ldots, v_{\pi(k)}$ are k distinct nodes that form a clique in G

$\square$

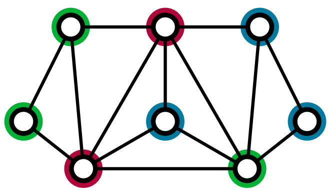
Colored Cliques

Problem: Multicolored Clique
Given: Graph $G = (V, E)$, parameter k,
and partition $(V_1, \ldots, V_k)$ of V.
Find: Clique $V' \subseteq V$ of size k, so that $|V' \cap V_i| = 1$ for each i.



V_1, V_2, V_3

Reduction: from Multicolored Clique to Clique

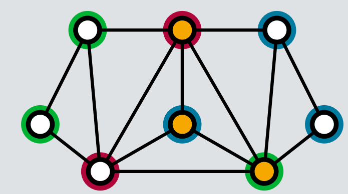


$\longrightarrow$

Instance of MC Clique:
$(G, 3, (V_1, V_2, V_3))$

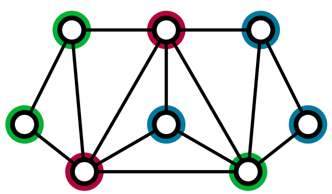
Colored Cliques

Problem: **MULTICOLORED CLIQUE**
Given: Graph $G = (V, E)$, parameter k,
and partition $(V_1, \ldots, V_k)$ of V.
Find: Clique $V' \subseteq V$ of size k, so that $|V' \cap V_i| = 1$ for each i.

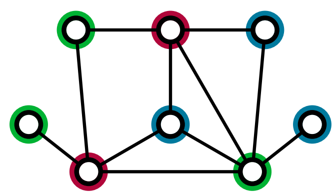


V_1, V_2, V_3

Reduction: from MULTICOLORED CLIQUE to CLIQUE



Instance of MC CLIQUE:
$(G, 3, (V_1, V_2, V_3))$

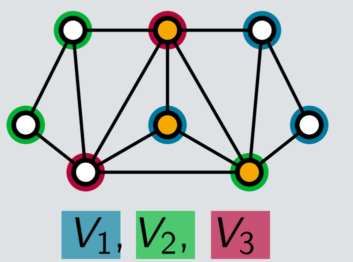


Instance $(G', 3)$ of CLIQUE

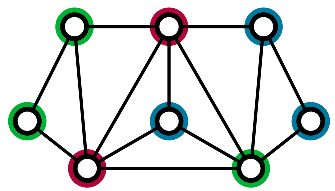
- delete edges within each color class
- $k' = k$

Colored Cliques

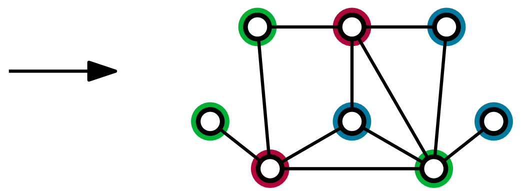
Problem: MULTICOLORED CLIQUE
Given: Graph $G = (V, E)$, parameter k,
and partition $(V_1, \dots, V_k)$ of V.
Find: Clique $V' \subseteq V$ of size k, so that $|V' \cap V_i| = 1$ for each i.



Reduction: from MULTICOLORED CLIQUE to CLIQUE



Instance of MC CLIQUE: $(G, 3, (V_1, V_2, V_3))$



Instance $(G', 3)$ of CLIQUE

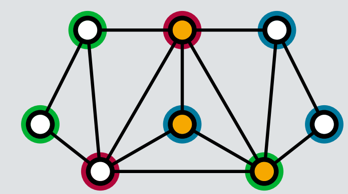
- delete edges within each color class
- $k' = k$

G has a colored size k clique $\Rightarrow$ G' has a size k clique

- the colored clique does not use any edges inside a color class
- thus, G' contains a size k clique

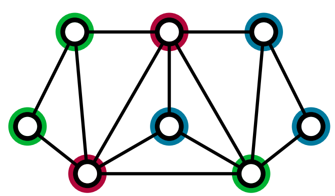
Colored Cliques

Problem: MULTICOLORED CLIQUE
Given: Graph $G = (V, E)$, parameter k,
and partition $(V_1, \ldots, V_k)$ of V.
Find: Clique $V' \subseteq V$ of size k, so that $|V' \cap V_i| = 1$ for each i.

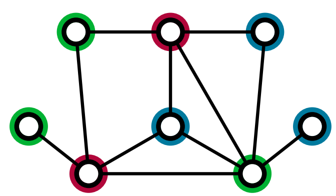


V_1, V_2, V_3

Reduction: from MULTICOLORED CLIQUE to CLIQUE



Instance of MC CLIQUE: $(G, 3, (V_1, V_2, V_3))$



Instance $(G', 3)$ of CLIQUE

- delete edges within each color class
- $k' = k$

G has a colored size k clique $\Rightarrow$ G' has a size k clique

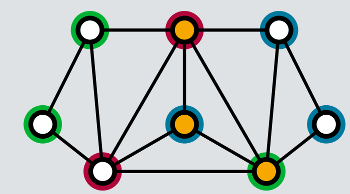
- the colored clique does not use any edges inside a color class
- thus, G' contains a size k clique

G' has a size k clique $\Rightarrow$ G has a colored size k clique

- in G', no monochromatic vertices are adjacent
- thus, each clique must be a colored clique

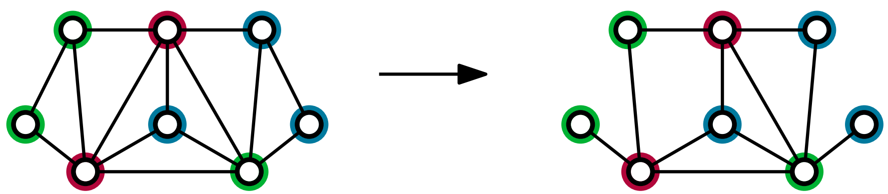
Colored Cliques

Problem: **MULTICOLORED CLIQUE**
Given: Graph $G = (V, E)$, parameter k,
and partition $(V_1, \ldots, V_k)$ of V.
Find: Clique $V' \subseteq V$ of size k, so that $|V' \cap V_i| = 1$ for each i.



V_1, V_2, V_3

Reduction: from MULTICOLORED CLIQUE to CLIQUE



Instance of MC CLIQUE: $(G, 3, (V_1, V_2, V_3))$

Instance $(G', 3)$ of CLIQUE

- delete edges within each color class
- $k' = k$

G has a colored size k clique $\Rightarrow$ G' has a size k clique

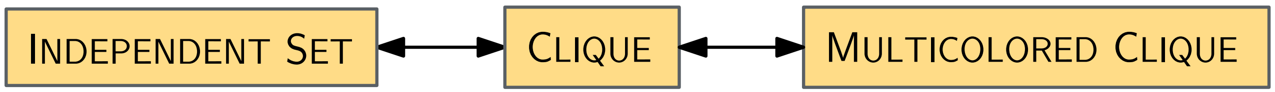
- the colored clique does not use any edges inside a color class
- thus, G' contains a size k clique

G' has a size k clique $\Rightarrow$ G has a colored size k clique

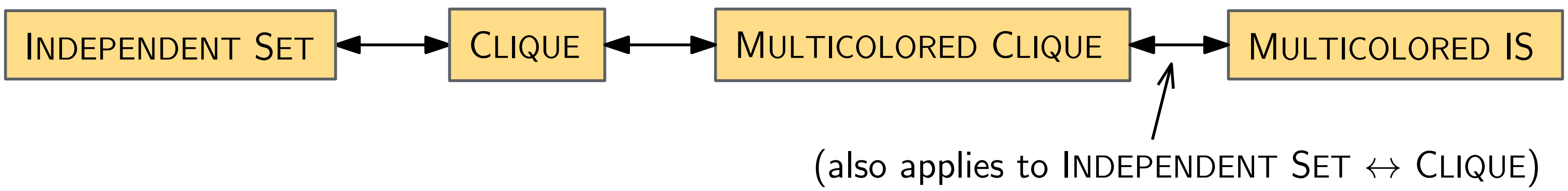
- in G', no monochromatic vertices are adjacent
- thus, each clique must be a colored clique

$\square$

FPT-Reductions so far



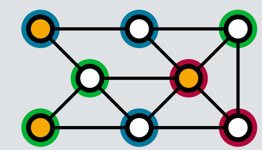
FPT-Reductions so far



Dominating Set

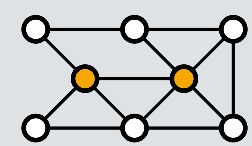
Reduce Multicolored Independent Set to Dominating Set

MC Independent Set



$V' \subseteq V$ colored, so that
$\forall e \in E \colon |e \cap V'| \leq 1$

Dominating Set



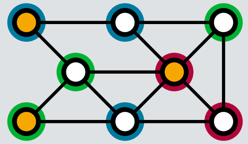
$V' \subseteq V$, so that
$\forall v \in V \colon |N[v] \cap V'| \geq 1$

DOMINATING SET

Reduce MULTICOLORED INDEPENDENT SET to DOMINATING SET

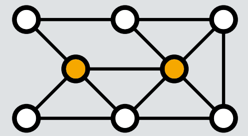
- enforce the following properties on the DOMINATING SET instance
  - select exactly one element from each color class
  - avoid simultaneous selection of designated node pairs

MC INDEPENDENT SET



$V' \subseteq V$ colored, so that
$\forall e \in E\colon |e \cap V'| \leq 1$

DOMINATING SET



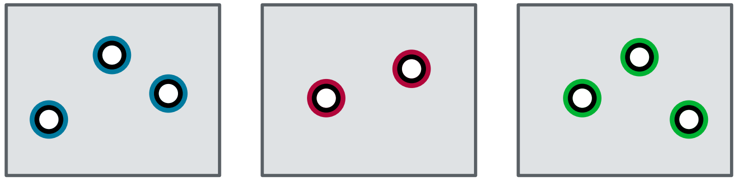
$V' \subseteq V$, so that
$\forall v \in V\colon |N[v] \cap V'| \geq 1$

Dominating Set

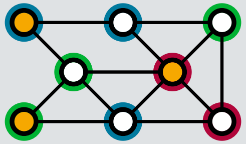
Reduce Multicolored Independent Set to Dominating Set

- enforce the following properties on the Dominating Set instance
  - select exactly one element from each color class
  - avoid simultaneous selection of designated node pairs

One element in each color class

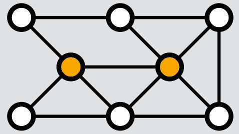


MC Independent Set



$V' \subseteq V$ colored, so that
$\forall e \in E\colon |e \cap V'| \leq 1$

Dominating Set



$V' \subseteq V$, so that
$\forall v \in V\colon |N[v] \cap V'| \geq 1$

Dominating Set

Reduce Multicolored Independent Set to Dominating Set

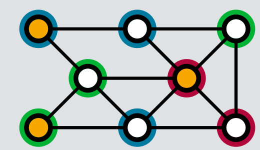
- enforce the following properties on the Dominating Set instance
  - select exactly one element from each color class
  - avoid simultaneous selection of designated node pairs

One element in each color class

- vertex x_i adjacent to all vertices in V_i and no others

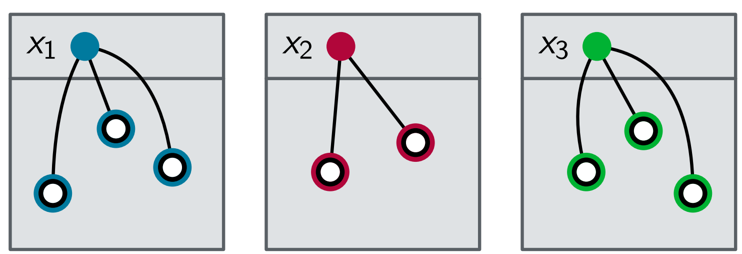
(forces the selection of at least one of $V_i \cup \{x_i\}$)

MC Independent Set

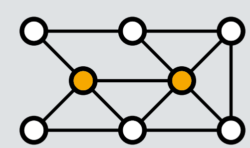


$V' \subseteq V$ colored, so that

$\forall e \in E\colon |e \cap V'| \leq 1$



Dominating Set



$V' \subseteq V$, so that

$\forall v \in V\colon |N[v] \cap V'| \geq 1$

Dominating Set

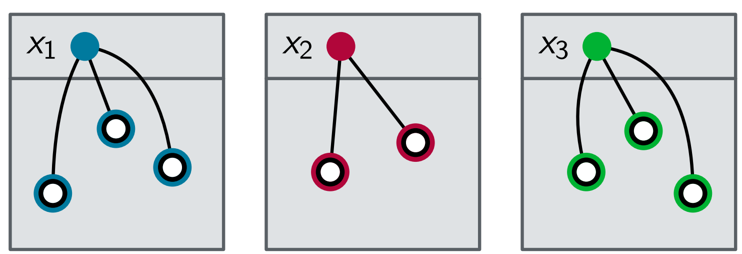
Reduce Multicolored Independent Set to Dominating Set

- enforce the following properties on the Dominating Set instance
  - select exactly one element from each color class
  - avoid simultaneous selection of designated node pairs

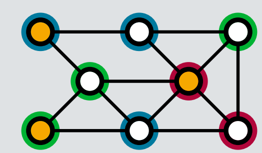
One element in each color class

- vertex x_i adjacent to all vertices in V_i and no others

  (forces the selection of at least one of $V_i \cup \{x_i\}$)
- Problem: can still choose x_i

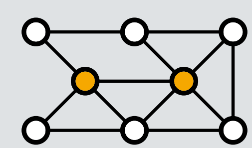


MC Independent Set



$V' \subseteq V$ colored, so that
$\forall e \in E\colon |e \cap V'| \leq 1$

Dominating Set



$V' \subseteq V$, so that
$\forall v \in V\colon |N[v] \cap V'| \geq 1$

Dominating Set

Reduce Multicolored Independent Set to Dominating Set

- enforce the following properties on the Dominating Set instance
  - select exactly one element from each color class
  - avoid simultaneous selection of designated node pairs

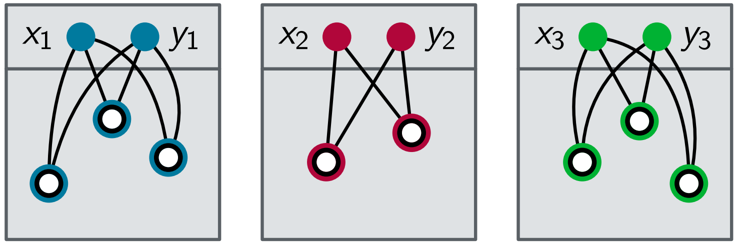
One element in each color class

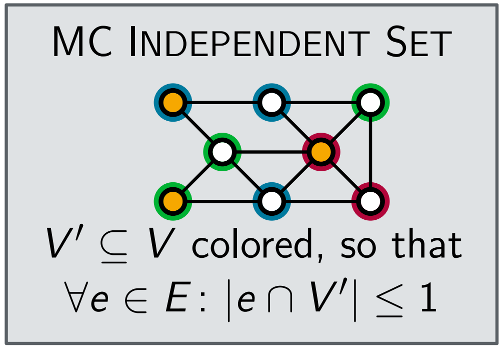
- vertex x_i adjacent to all vertices in V_i and no others

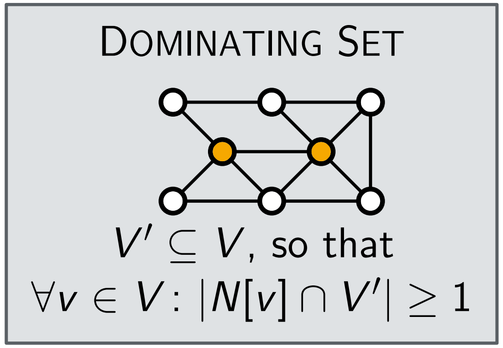
(forces the selection of at least one of $V_i \cup \{x_i\}$)

- Problem: can still choose x_i
- Solution: create a copy y_i of x_i

(picking both x_i and y_i is too costly to form a size k DS)







Dominating Set

Reduce Multicolored Independent Set to Dominating Set

- enforce the following properties on the Dominating Set instance
  - select exactly one element from each color class
  - avoid simultaneous selection of designated node pairs

One element in each color class

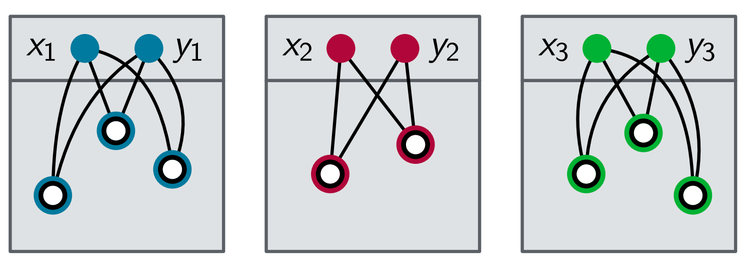
- vertex x_i adjacent to all vertices in V_i and no others

(forces the selection of at least one of $V_i \cup \{x_i\}$)

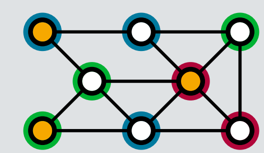
- Problem: can still choose x_i
- Solution: create a copy y_i of x_i

(picking both x_i and y_i is too costly to form a size k DS)

- to allow any vertex from V_i to dominate it and both x_i and y_i.

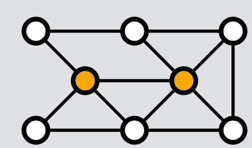


MC Independent Set



$V' \subseteq V$ colored, so that
$\forall e \in E : |e \cap V'| \leq 1$

Dominating Set



$V' \subseteq V$, so that
$\forall v \in V : |N[v] \cap V'| \geq 1$

Dominating Set

Reduce Multicolored Independent Set to Dominating Set

- enforce the following properties on the Dominating Set instance
  - select exactly one element from each color class
  - avoid simultaneous selection of designated node pairs

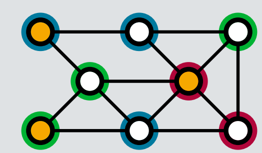
One element in each color class

- vertex x_i adjacent to all vertices in V_i and no others

(forces the selection of at least one of $V_i \cup \{x_i\}$)

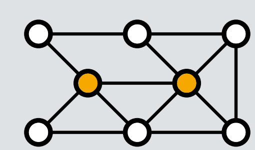
- Problem: can still choose x_i
- Solution: create a copy y_i of x_i
  (picking both x_i and y_i is too costly to form a size k DS)
- make V_i into a clique as to allow any vertex from V_i to dominate it and both x_i and y_i.

MC Independent Set

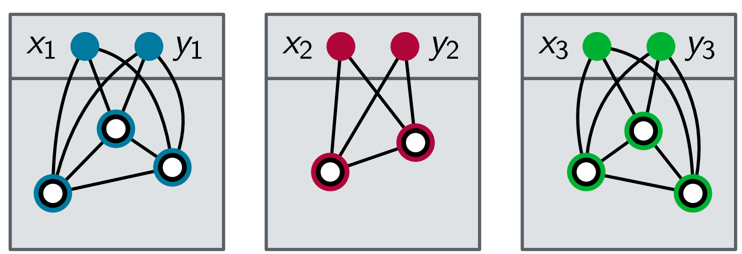


$V' \subseteq V$ colored, so that
$\forall e \in E: |e \cap V'| \leq 1$

Dominating Set



$V' \subseteq V$, so that
$\forall v \in V: |N[v] \cap V'| \geq 1$



Dominating Set

Reduce Multicolored Independent Set to Dominating Set

- enforce the following properties on the Dominating Set instance
  - select exactly one element from each color class
  - avoid simultaneous selection of designated node pairs

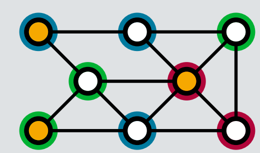
One element in each color class

- vertex x_i adjacent to all vertices in V_i and no others

(forces the selection of at least one of $V_i \cup \{x_i\}$)

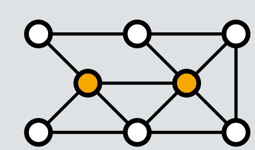
- Problem: can still choose x_i
- Solution: create a copy y_i of x_i
  (picking both x_i and y_i is too costly to form a size k DS)
- make V_i into a clique as to allow any vertex from V_i to dominate it and both x_i and y_i.

MC Independent Set

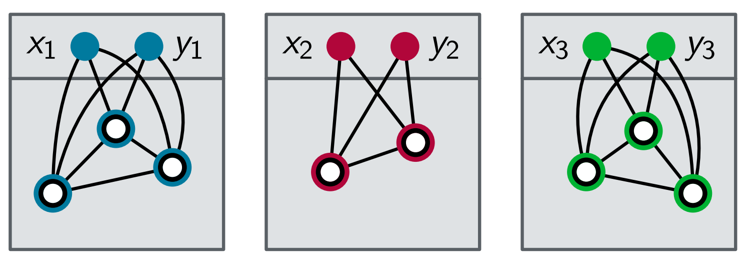


$V' \subseteq V$ colored, so that
$\forall e \in E : |e \cap V'| \leq 1$

Dominating Set



$V' \subseteq V$, so that
$\forall v \in V : |N[v] \cap V'| \geq 1$

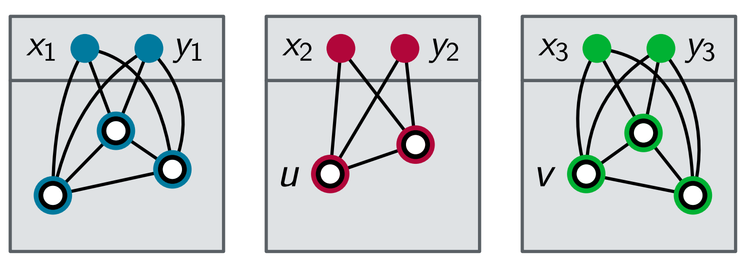


Dominating Set

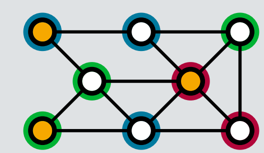
Reduce Multicolored Independent Set to Dominating Set

- enforce the following properties on the Dominating Set instance
  - select exactly one element from each color class
  - avoid simultaneous selection of designated node pairs

Avoid selecting both *u* and *v*

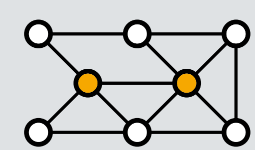


MC Independent Set



$V' \subseteq V$ colored, so that
$\forall e \in E\colon |e \cap V'| \leq 1$

Dominating Set



$V' \subseteq V$, so that
$\forall v \in V\colon |N[v] \cap V'| \geq 1$

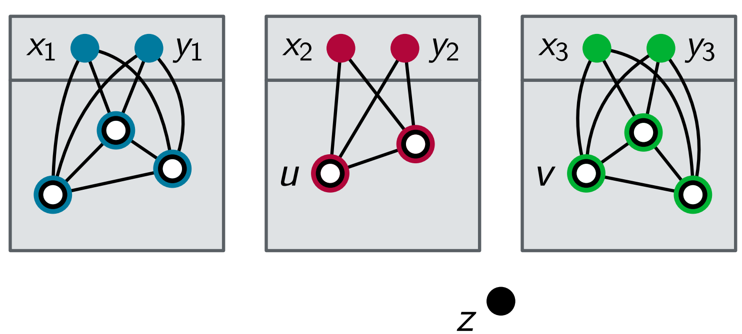
Dominating Set

Reduce Multicolored Independent Set to Dominating Set

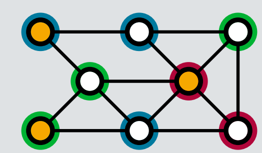
- enforce the following properties on the Dominating Set instance
  - select exactly one element from each color class
  - avoid simultaneous selection of designated node pairs

Avoid selecting both u and v

- Idea: insert a vertex z that is dominated unless both u and v are picked

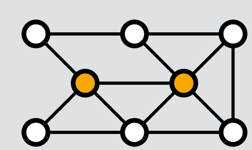


MC Independent Set



$V' \subseteq V$ colored, so that
$\forall e \in E : |e \cap V'| \leq 1$

Dominating Set



$V' \subseteq V$, so that
$\forall v \in V : |N[v] \cap V'| \geq 1$

Dominating Set

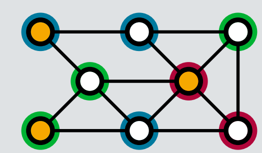
Reduce Multicolored Independent Set to Dominating Set

- enforce the following properties on the Dominating Set instance
  - select exactly one element from each color class
  - avoid simultaneous selection of designated node pairs

Avoid selecting both *u* and *v*

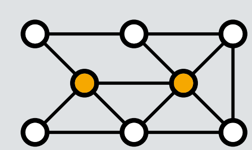
- Idea: insert a vertex z that is dominated unless both u and v are picked
  - z is not adjacent to u and not adjacent to v but is adjacent to all other vertices in the color classes of u and v

MC Independent Set

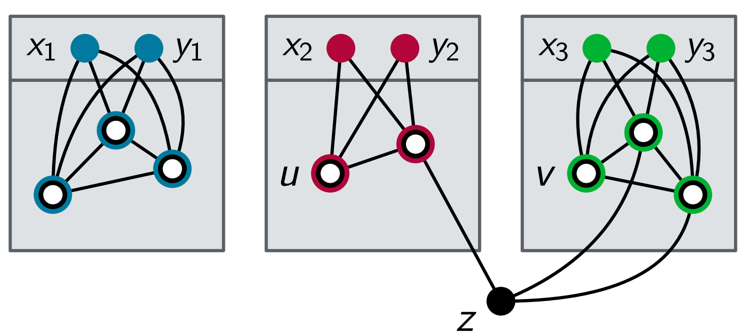


$V' \subseteq V$ colored, so that
$\forall e \in E\colon |e \cap V'| \leq 1$

Dominating Set



$V' \subseteq V$, so that
$\forall v \in V\colon |N[v] \cap V'| \geq 1$



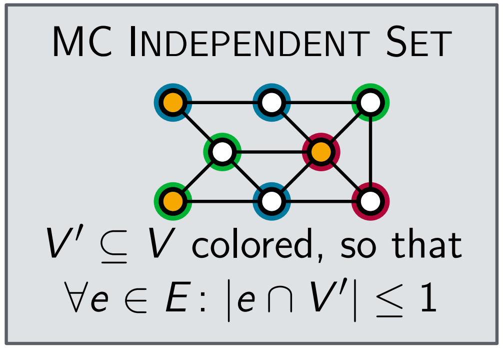
Dominating Set

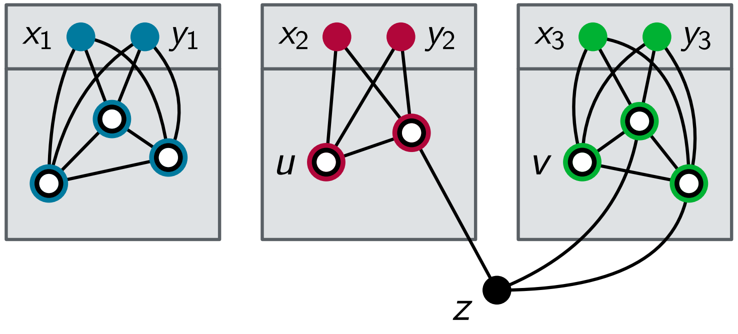
Reduce Multicolored Independent Set to Dominating Set

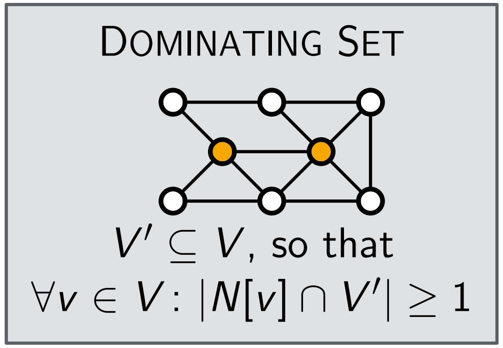
- enforce the following properties on the Dominating Set instance
  - select exactly one element from each color class
  - avoid simultaneous selection of designated node pairs

Avoid selecting both *u* and *v*

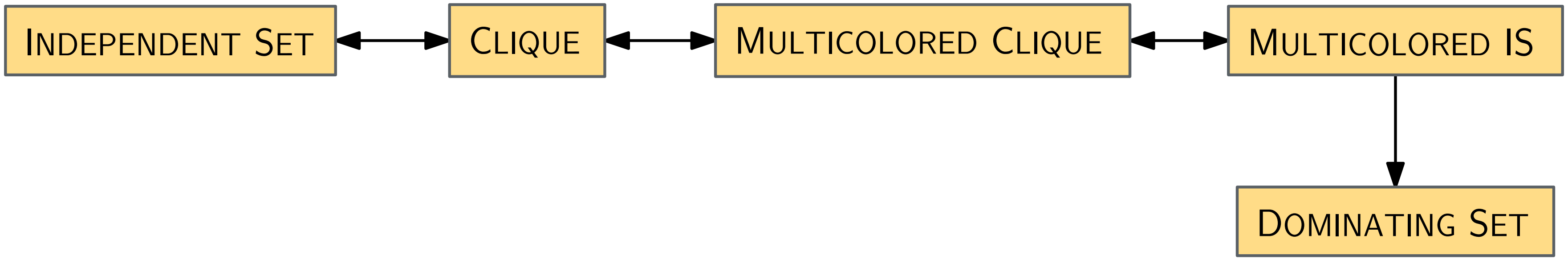
- Idea: insert a vertex z that is dominated unless both u and v are picked
  - z is not adjacent to u and not adjacent to v but is adjacent to all other vertices in the color classes of u and v



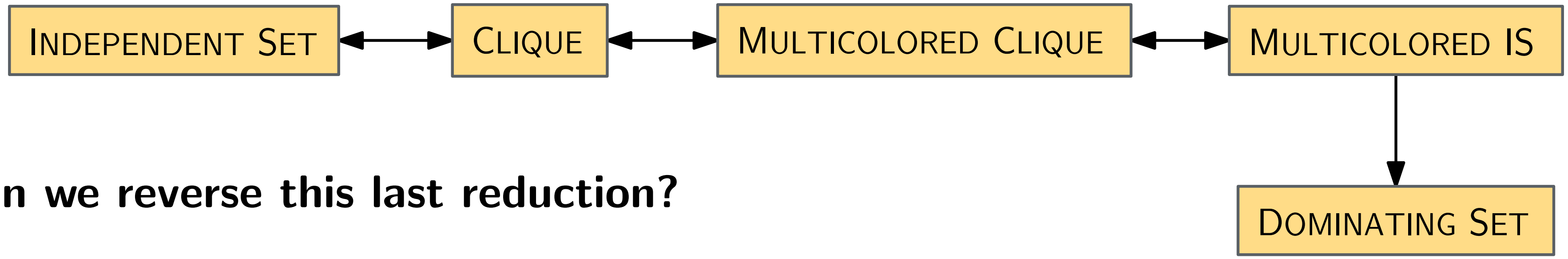




FPT-Reductions so far



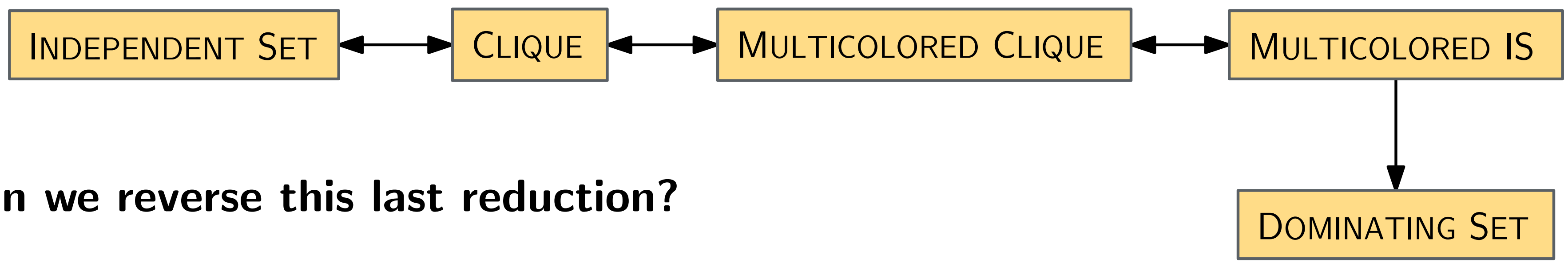
FPT-Reductions so far



Can we reverse this last reduction?

- it is not known ... but seems very unlikely

FPT-Reductions so far

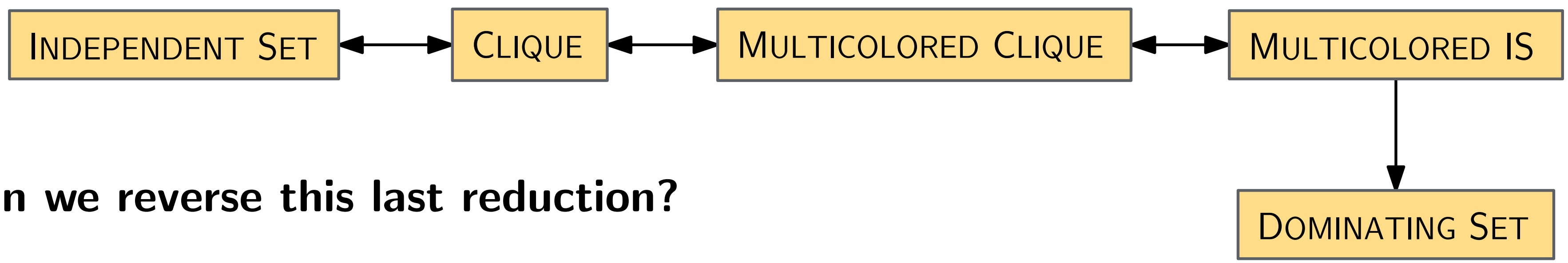


Can we reverse this last reduction?

- it is not known ... but seems very unlikely
- DOMINATING SET is (probably) more difficult than CLIQUE oder INDEPENDENT SET

(DS $\in$ FPT $\Rightarrow$ IS $\in$ FPT, but not the other way around)

FPT-Reductions so far



Can we reverse this last reduction?

- it is not known ... but seems very unlikely
- DOMINATING SET is (probably) more difficult than CLIQUE oder INDEPENDENT SET

  (DS $\in$ FPT $\Rightarrow$ IS $\in$ FPT, but not the other way around)
- we need a refined notion of "hardness"

FPT-Reductions so far

$$\begin{tikzcd} \textsc{Independent Set} \arrow[r, leftrightarrow] & \textsc{Clique} \arrow[r, leftrightarrow] & \textsc{Multicolored Clique} \arrow[r, leftrightarrow] & \textsc{Multicolored IS} \arrow[d] \\ & & & \textsc{Dominating Set} \end{tikzcd}$$

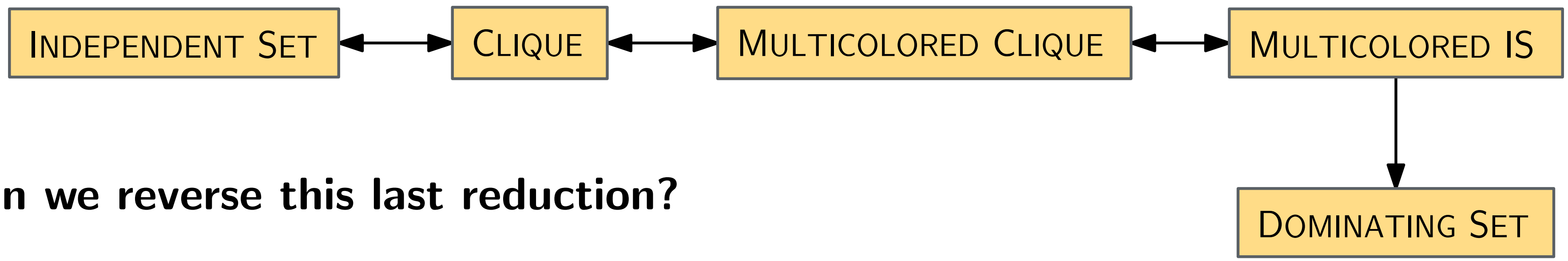
Can we reverse this last reduction?

- it is not known ... but seems very unlikely
- DOMINATING SET is (probably) more difficult than CLIQUE oder INDEPENDENT SET

  (DS $\in$ FPT $\Rightarrow$ IS $\in$ FPT, but not the other way around)
- we need a refined notion of "hardness"

Similar to the complexity class P

FPT-Reductions so far



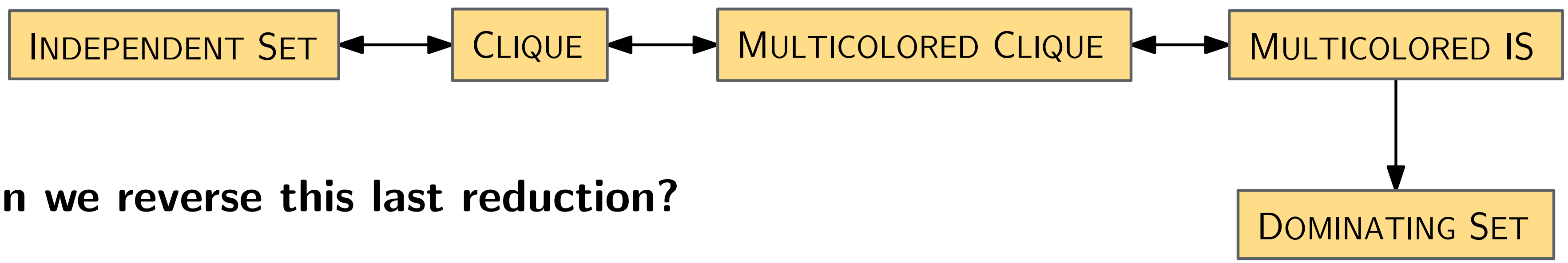
Can we reverse this last reduction?

- it is not known ... but seems very unlikely
- DOMINATING SET is (probably) more difficult than CLIQUE oder INDEPENDENT SET
  (DS $\in$ FPT $\Rightarrow$ IS $\in$ FPT, but not the other way around)
- we need a refined notion of "hardness"

Similar to the complexity class P

- usually, one focuses on: NP-hardness
- but there are also **intermediate** problems (assuming P $\neq$ NP)

FPT-Reductions so far



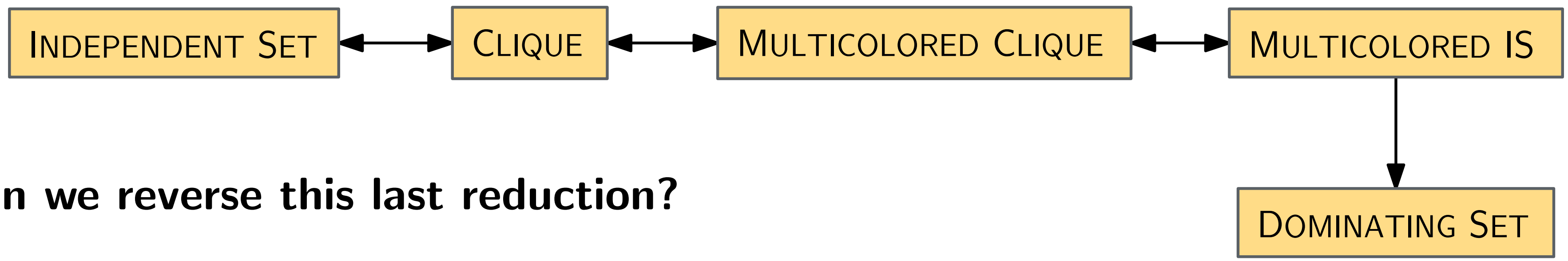
Can we reverse this last reduction?

- it is not known ... but seems very unlikely
- Dominating Set is (probably) more difficult than Clique oder Independent Set
  (DS $\in$ FPT $\Rightarrow$ IS $\in$ FPT, but not the other way around)
- we need a refined notion of "hardness"

Similar to the complexity class P

- usually, one focuses on: NP-hardness
- but there are also **intermediate** problems (assuming P $\neq$ NP)
- however, no natural hierarchy of NP-intermediate problems
  (candidates: prime factorization, graph isomorphism)

FPT-Reductions so far



Can we reverse this last reduction?

- it is not known ... but seems very unlikely
- DOMINATING SET is (probably) more difficult than CLIQUE oder INDEPENDENT SET
  (DS $\in$ FPT $\Rightarrow$ IS $\in$ FPT, but not the other way around)
- we need a refined notion of "hardness"

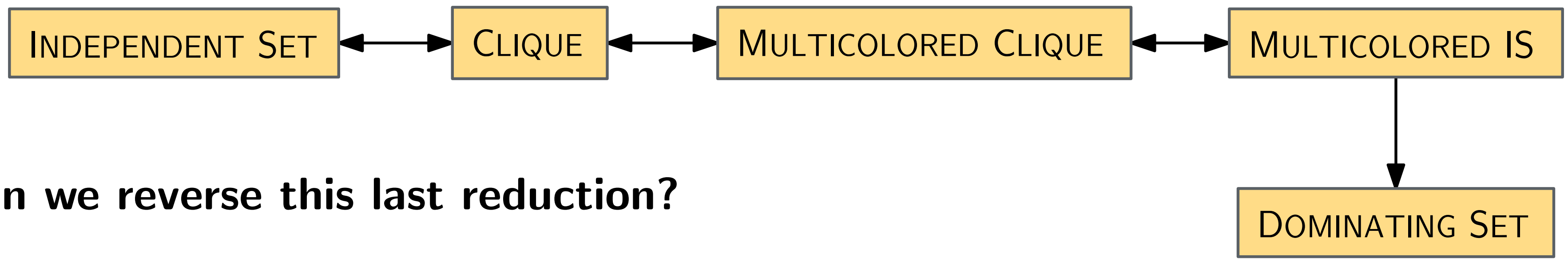
Similar to the complexity class P

- usually, one focuses on: NP-hardness
- but there are also **intermediate** problems (assuming P $\neq$ NP)
- however, no natural hierarchy of NP-intermediate problems
  (candidates: prime factorization, graph isomorphism)

And now? What about FPT?

- define natural hierarchy of complexity classes

FPT-Reductions so far



Can we reverse this last reduction?

- it is not known ... but seems very unlikely
- Dominating Set is (probably) more difficult than Clique oder Independent Set
  (DS $\in$ FPT $\Rightarrow$ IS $\in$ FPT, but not the other way around)
- we need a refined notion of "hardness"

Similar to the complexity class P

- usually, one focuses on: NP-hardness
- but there are also **intermediate** problems (assuming P $\neq$ NP)
- however, no natural hierarchy of NP-intermediate problems
  (candidates: prime factorization, graph isomorphism)

And now? What about FPT?

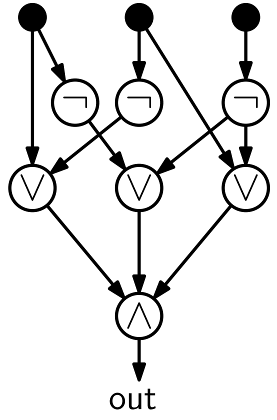
- define natural hierarchy of complexity classes
- establish prototypical problem for each level

Weighted Circuit Satisfiability

Boolean Circuits

- directed acyclic graph (DAG) with the following node types:

example

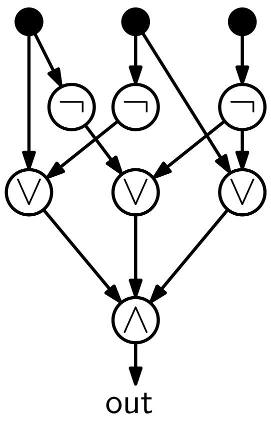


Weighted Circuit Satisfiability

Boolean Circuits

- directed acyclic graph (DAG) with the following node types:
  - sources (in-degree 0) are **input nodes**

example

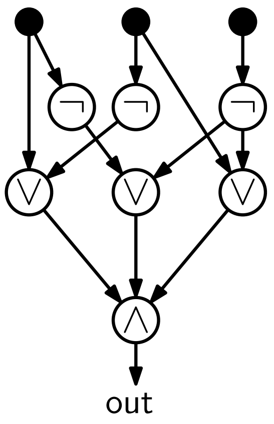


Weighted Circuit Satisfiability

Boolean Circuits

- directed acyclic graph (DAG) with the following node types:
  - sources (in-degree 0) are **input nodes** ●
  - **NEGATION** nodes have in-degree 1 $\neg$

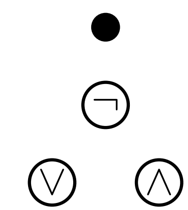
example



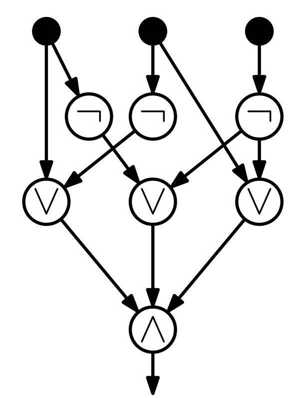
Weighted Circuit Satisfiability

Boolean Circuits

- directed acyclic graph (DAG) with the following node types:
  - sources (in-degree 0) are **input nodes**
  - **NEGATION** nodes have in-degree 1
  - **AND**- resp. **OR**-nodes have in-degree ≥ 2



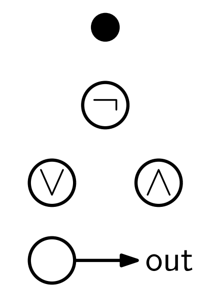
example



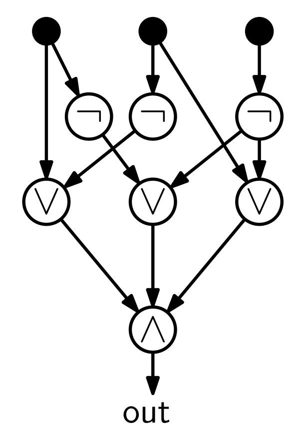
Weighted Circuit Satisfiability

Boolean Circuits

- directed acyclic graph (DAG) with the following node types:
  - sources (in-degree 0) are **input nodes**
  - **NEGATION** nodes have in-degree 1
  - **AND**- resp. **OR**-nodes have in-degree ≥ 2
  - one sink (out-degree 0) is the **output node**



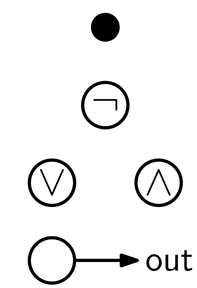
example



Weighted Circuit Satisfiability

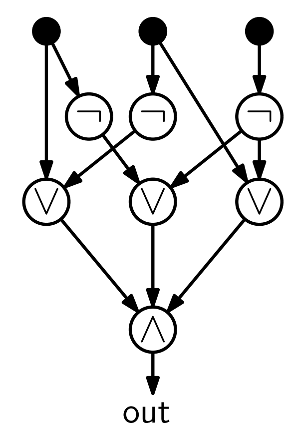
Boolean Circuits

- directed acyclic graph (DAG) with the following node types:
  - sources (in-degree 0) are **input nodes**
  - **NEGATION** nodes have in-degree 1
  - **AND**- resp. **OR**-nodes have in-degree ≥ 2
  - one sink (out-degree 0) is the **output node**



- an assignment of 0 or 1 to each input node propogates through the other nodes, providing an output value. (in the natural way)
- an assignment **statisfying** when the output value is 1

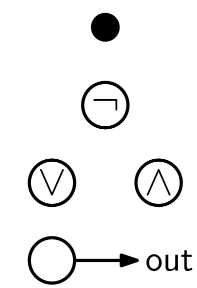
example



Weighted Circuit Satisfiability

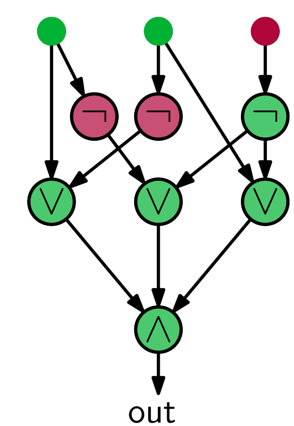
Boolean Circuits

- directed acyclic graph (DAG) with the following node types:
  - sources (in-degree 0) are **input nodes**
  - **NEGATION** nodes have in-degree 1
  - **AND**- resp. **OR**-nodes have in-degree ≥ 2
  - one sink (out-degree 0) is the **output node**



- an assignment of 0 or 1 to each input node propogates through the other nodes, providing an output value. (in the natural way)
- an assignment **statisfying** when the output value is 1

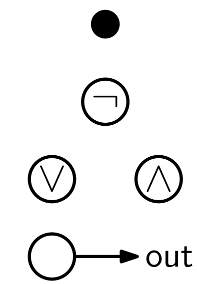
example



Weighted Circuit Satisfiability

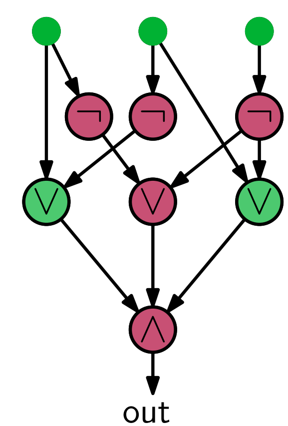
Boolean Circuits

- directed acyclic graph (DAG) with the following node types:
  - sources (in-degree 0) are **input nodes**
  - **NEGATION** nodes have in-degree 1
  - **AND**- resp. **OR**-nodes have in-degree ≥ 2
  - one sink (out-degree 0) is the **output node**



- an assignment of 0 or 1 to each input node propogates through the other nodes, providing an output value. (in the natural way)
- an assignment **statisfying** when the output value is 1

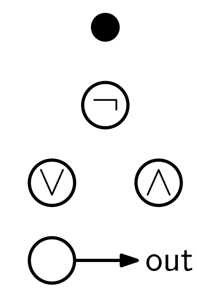
example



Weighted Circuit Satisfiability

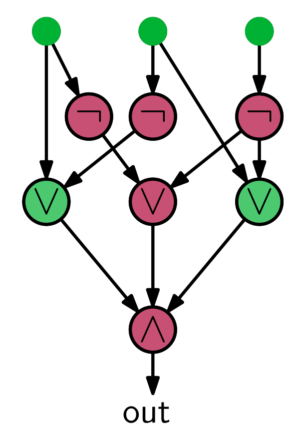
Boolean Circuits

- directed acyclic graph (DAG) with the following node types:
  - sources (in-degree 0) are **input nodes**
  - **NEGATION** nodes have in-degree 1
  - **AND**- resp. **OR**-nodes have in-degree ≥ 2
  - one sink (out-degree 0) is the **output node**



- an assignment of 0 or 1 to each input node propogates through the other nodes, providing an output value. (in the natural way)
- an assignment **statisfying** when the output value is 1
- the **weight** of an assignment is the number of 1s used

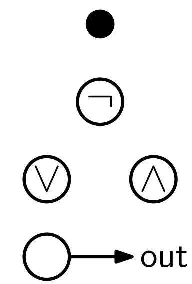
example



Weighted Circuit Satisfiability

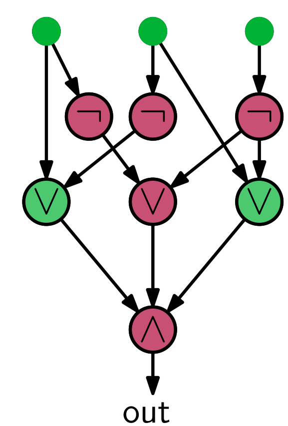
Boolean Circuits

- directed acyclic graph (DAG) with the following node types:
  - sources (in-degree 0) are **input nodes**
  - **NEGATION** nodes have in-degree 1
  - **AND**- resp. **OR**-nodes have in-degree ≥ 2
  - one sink (out-degree 0) is the **output node**



- an assignment of 0 or 1 to each input node propogates through the other nodes, providing an output value. (in the natural way)
- an assignment **statisfying** when the output value is 1
- the **weight** of an assignment is the number of 1s used

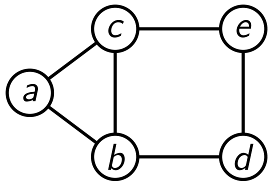
example



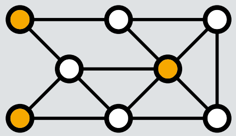
Problem: Weighted Circuit Satisfiability (WCS)
Given a boolean circuit and a parameter k
Find: A weight k satisfying assignment

Reductions visualized

Independent Set → WCS



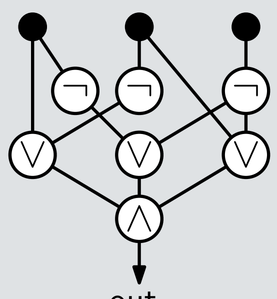
Independent Set



$V' \subseteq V$, so that

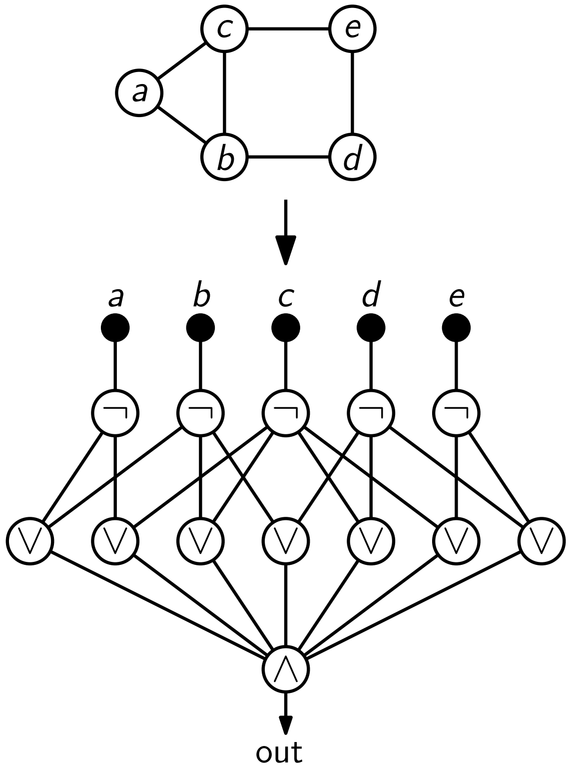
$\forall e \in E\colon |e \cap V'| \leq 1$

WCS

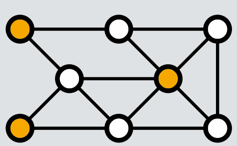


Reductions visualized

Independent Set → WCS



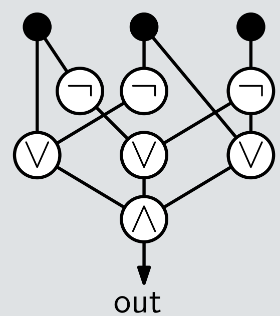
Independent Set



$V' \subseteq V$, so that

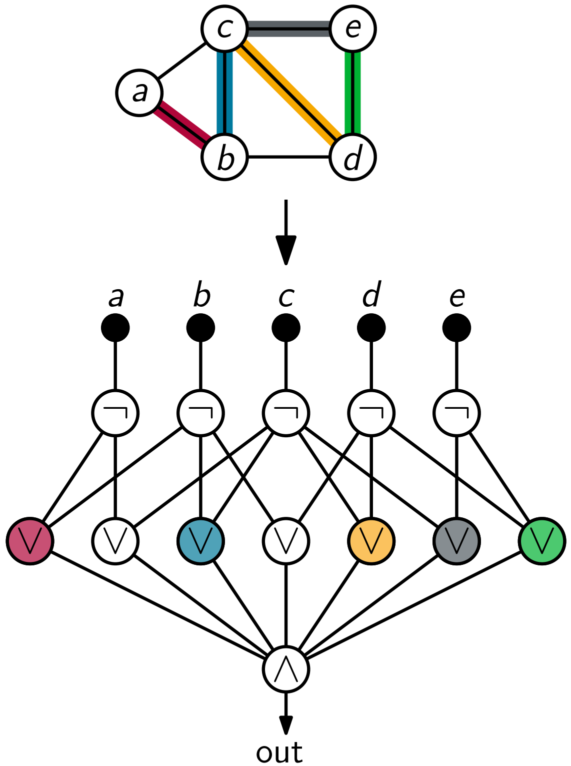
$\forall e \in E \colon |e \cap V'| \leq 1$

WCS

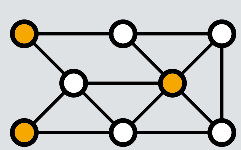


Reductions visualized

INDEPENDENT SET → WCS



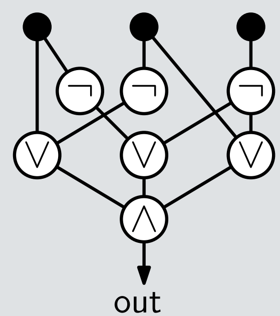
INDEPENDENT SET



$V' \subseteq V$, so that

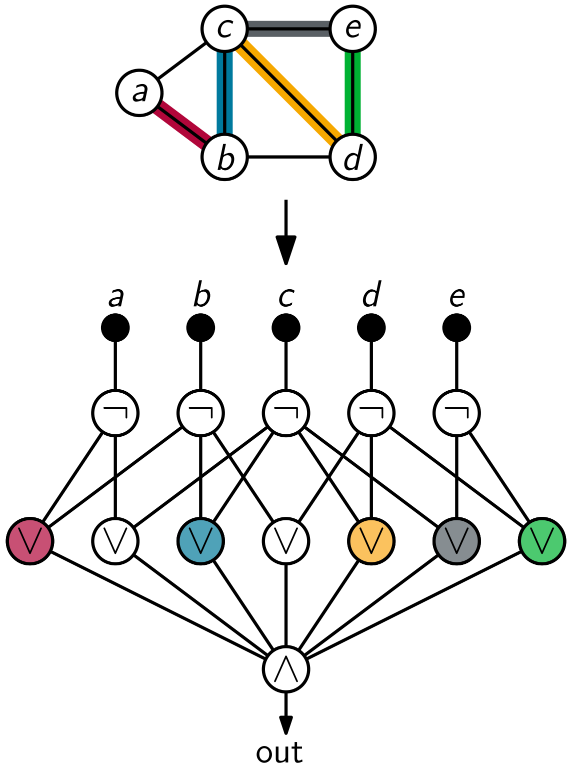
$\forall e \in E\colon |e \cap V'| \leq 1$

WCS

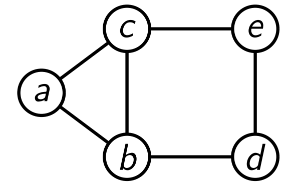


Reductions visualized

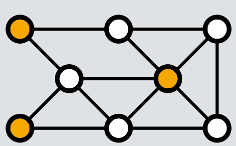
Independent Set → WCS



Dominating Set → WCS

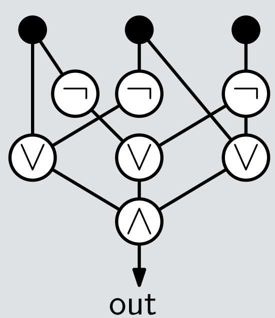


Independent Set

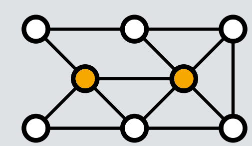


$V' \subseteq V$, so that
$\forall e \in E\colon |e \cap V'| \leq 1$

WCS



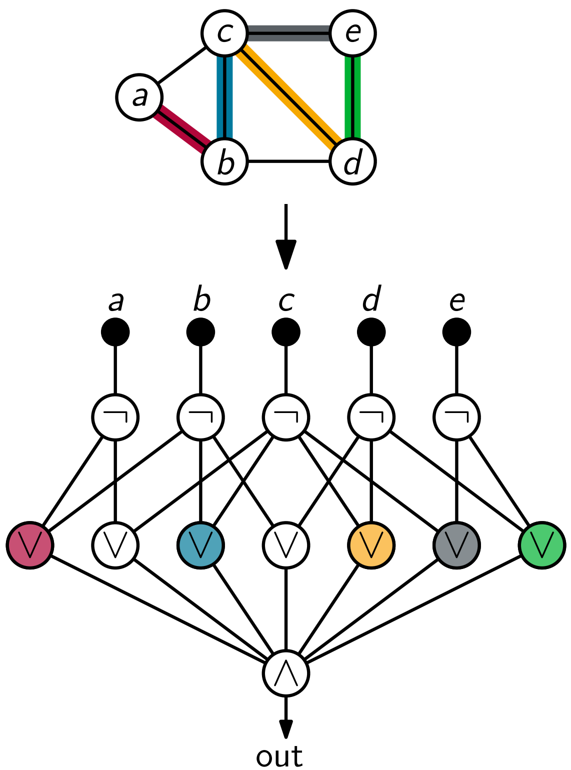
Dominating Set



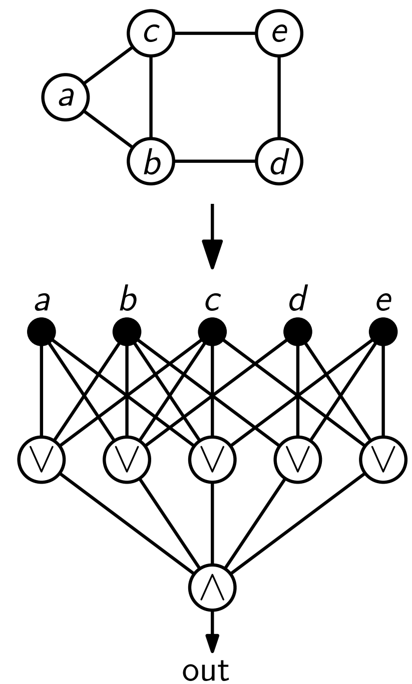
$V' \subseteq V$, so that
$\forall v \in V\colon |N[v] \cap V'| \geq 1$

Reductions visualized

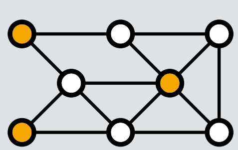
Independent Set → WCS



Dominating Set → WCS



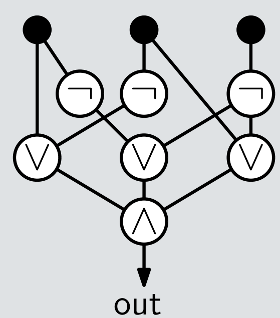
Independent Set



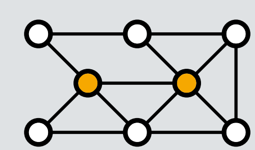
$V' \subseteq V$, so that

$\forall e \in E\colon |e \cap V'| \leq 1$

WCS



Dominating Set

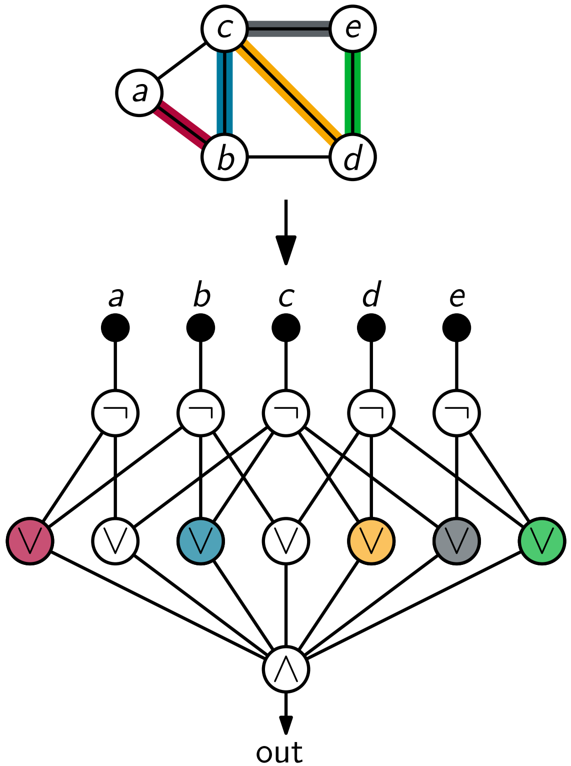


$V' \subseteq V$, so that

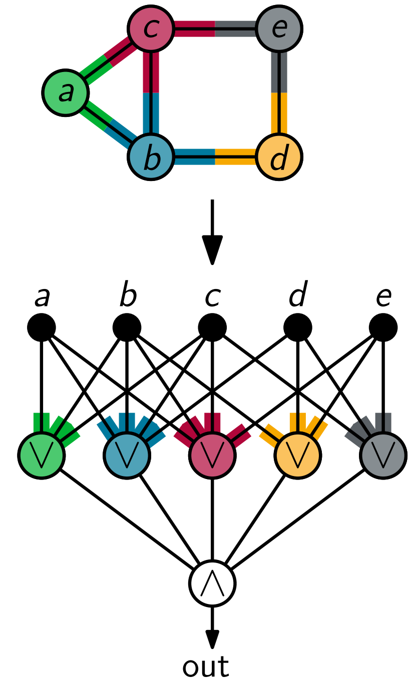
$\forall v \in V\colon |N[v] \cap V'| \geq 1$

Reductions visualized

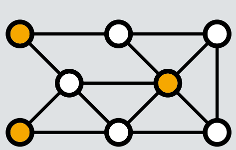
INDEPENDENT SET → WCS



DOMINATING SET → WCS



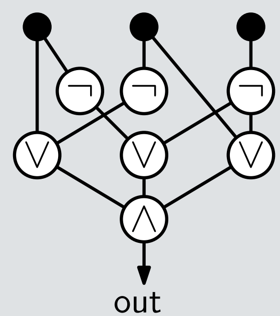
INDEPENDENT SET



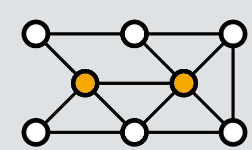
$V' \subseteq V$, so that

$\forall e \in E \colon |e \cap V'| \leq 1$

WCS



DOMINATING SET

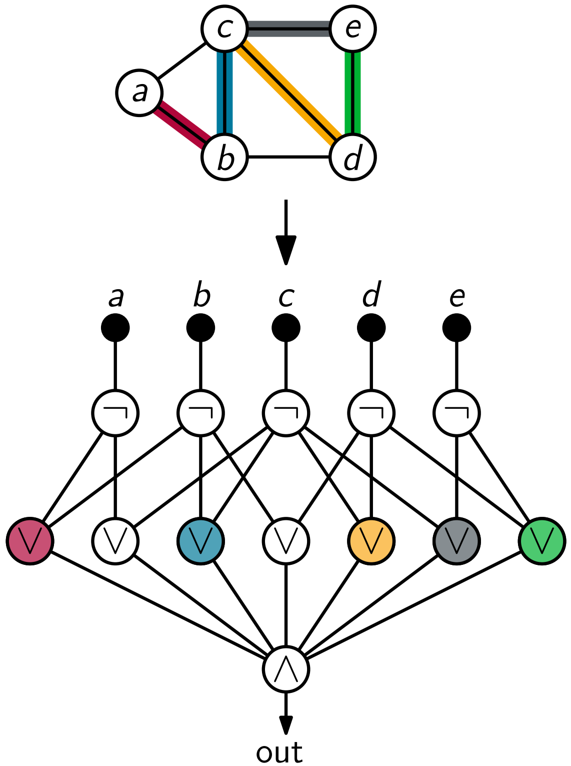


$V' \subseteq V$, so that

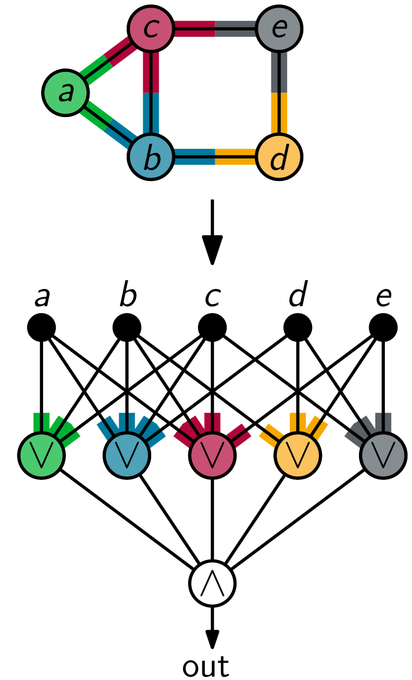
$\forall v \in V \colon |N[v] \cap V'| \geq 1$

Reductions visualized

Independent Set → WCS



Dominating Set → WCS

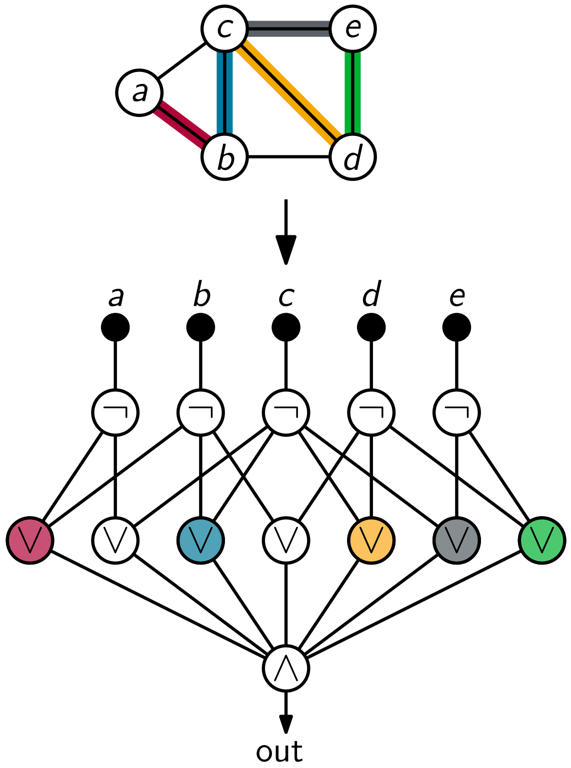


Observations

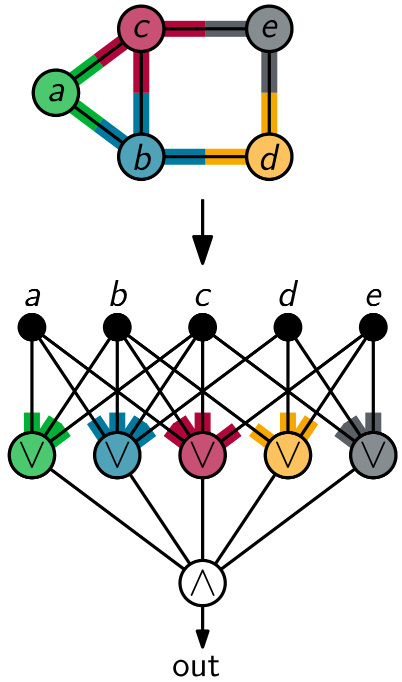
- the circuits have constant depth

Reductions visualized

Independent Set → WCS



Dominating Set → WCS

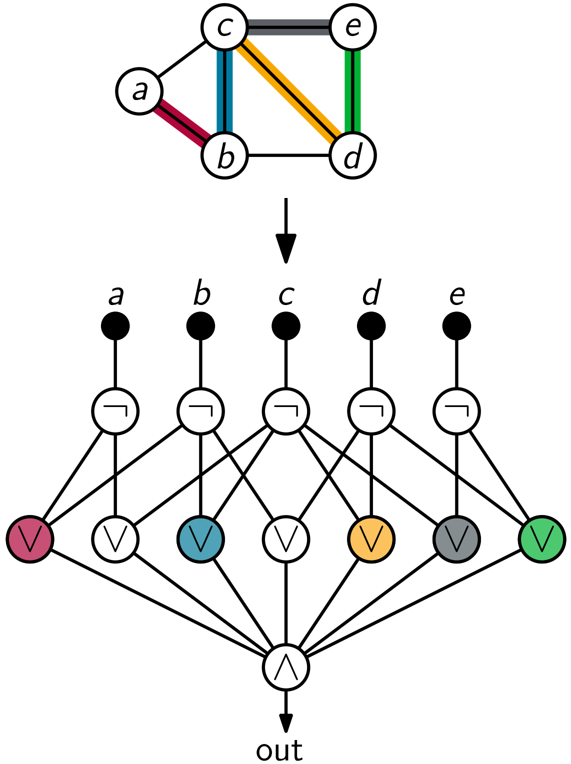


Observations

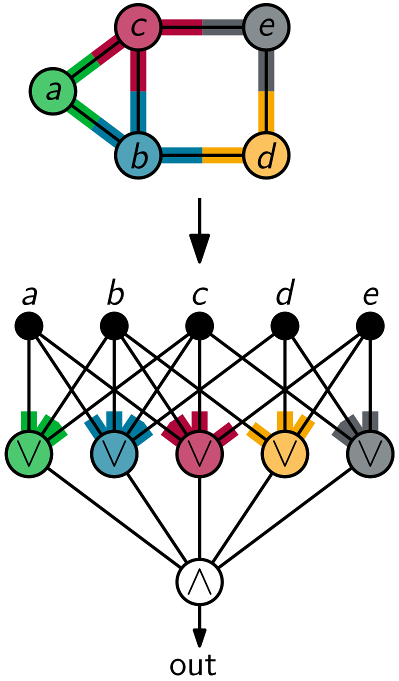
- the circuits have constant depth
- the circuit for DS contains more nodes with in-degree > 2

Reductions visualized

Independent Set → WCS



Dominating Set → WCS

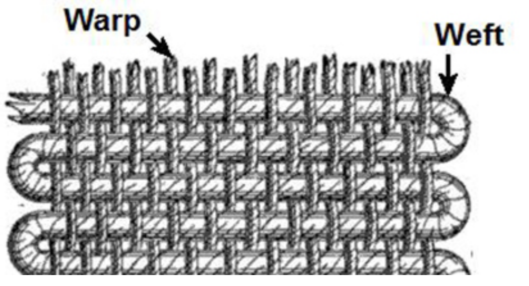


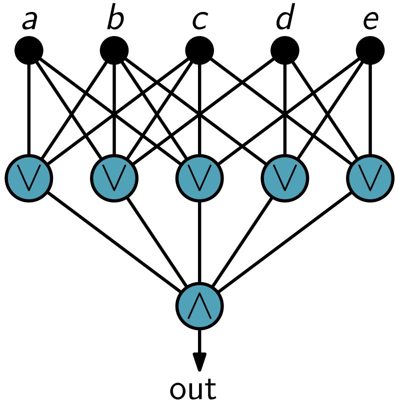
Observations

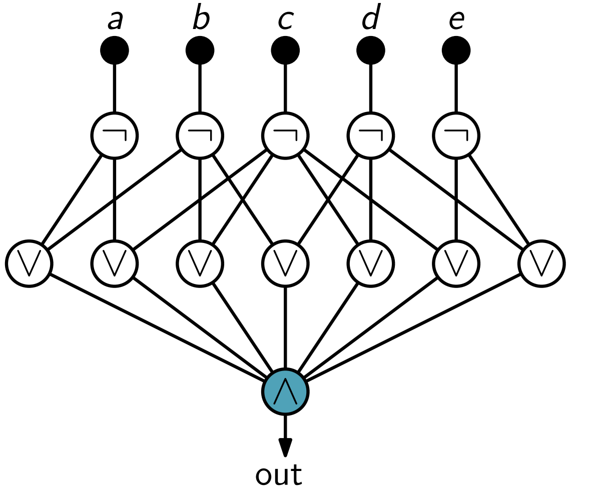
- the circuits have constant depth
- the circuit for DS contains more nodes with in-degree > 2
- is DS harder (in terms of FPT) than IS?

Weft

Definition
The **Weft** of a boolean circuit is the maximum number of nodes with in-degree > 2 on a directed path.

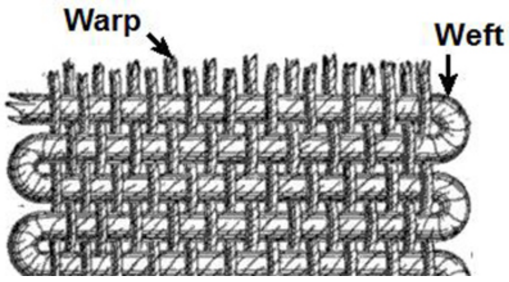


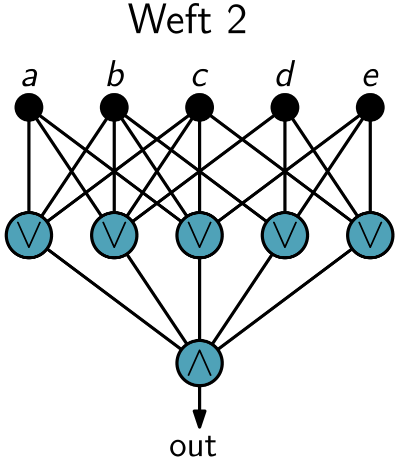


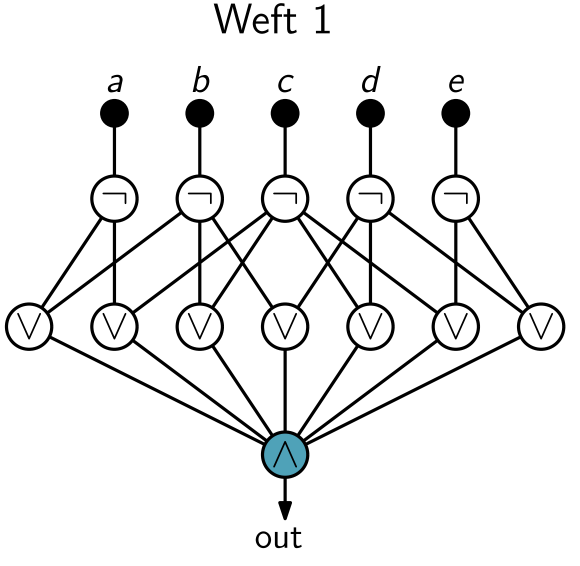


Weft

Definition
The **Weft** of a boolean circuit is the maximum number of nodes with in-degree > 2 on a directed path.



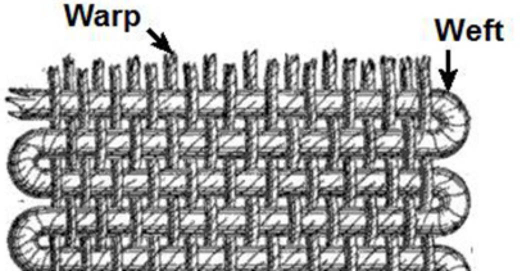




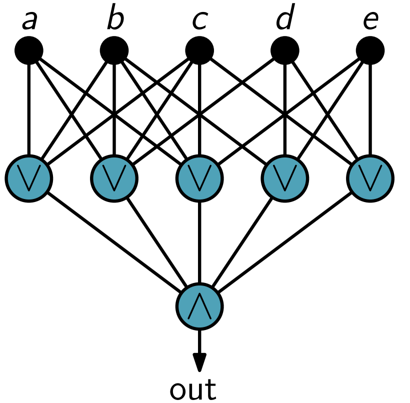
Weft

Definition
The **Weft** of a boolean circuit is the maximum number of nodes with in-degree > 2 on a directed path.

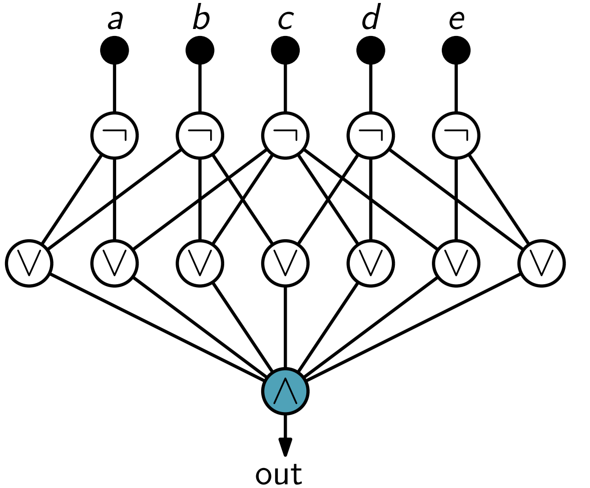
Problem
WCS[*t*] is WCS limited to circuits with constant depth and weft at most t.



Weft 2



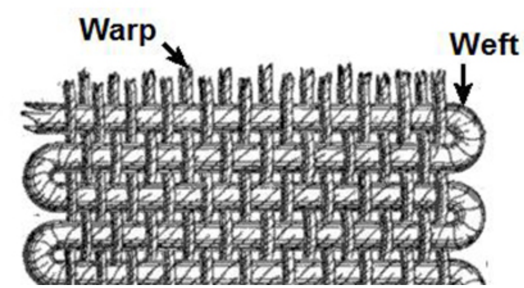
Weft 1



Weft

Definition
The **Weft** of a boolean circuit is the maximum number of nodes with in-degree > 2 on a directed path.

Problem
WCS[*t*] is WCS limited to circuits with constant depth and weft at most t.

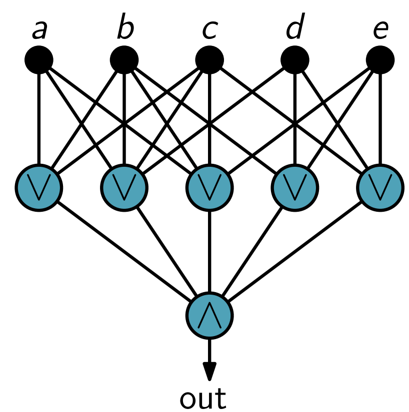


Definition
The class ***W*[*t*]** contains all problems with a paramterized reduction to WCS[t].

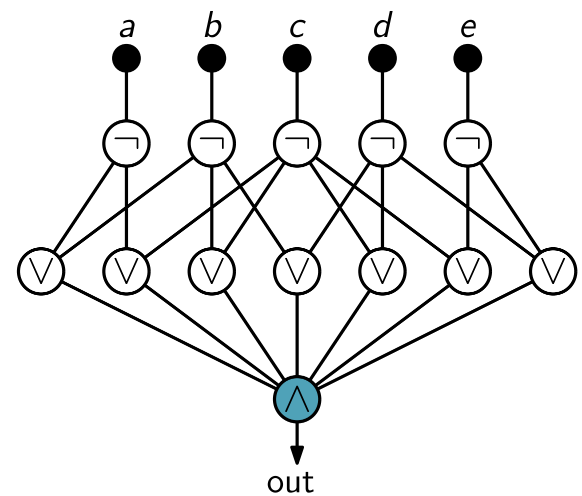
We have seen

- Independent Set $\in W[1] \subseteq W[2]$
- Dominating Set $\in W[2]$

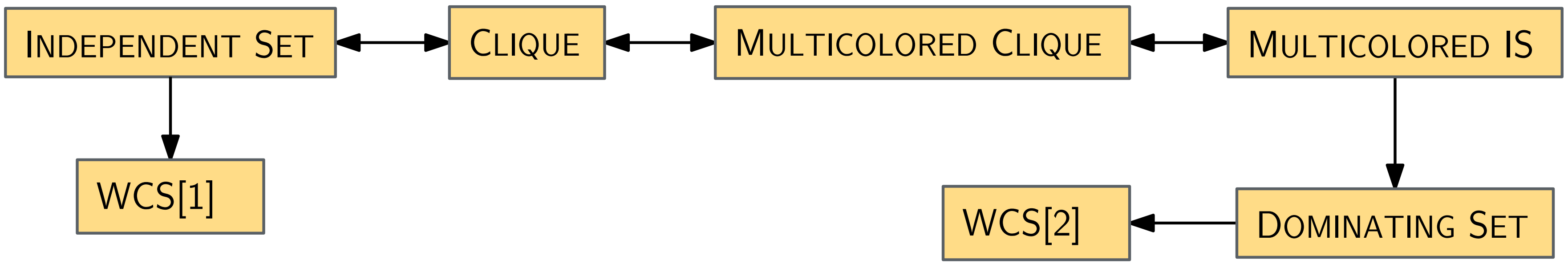
Weft 2



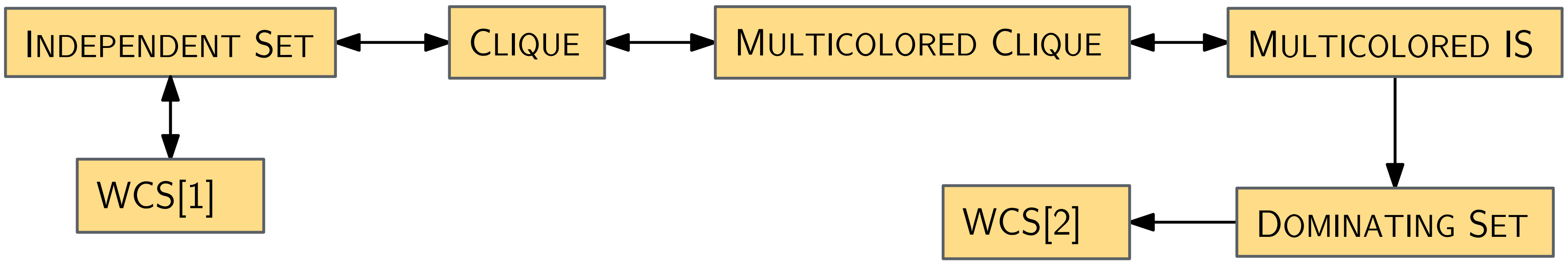
Weft 1



FPT-Reductions so far



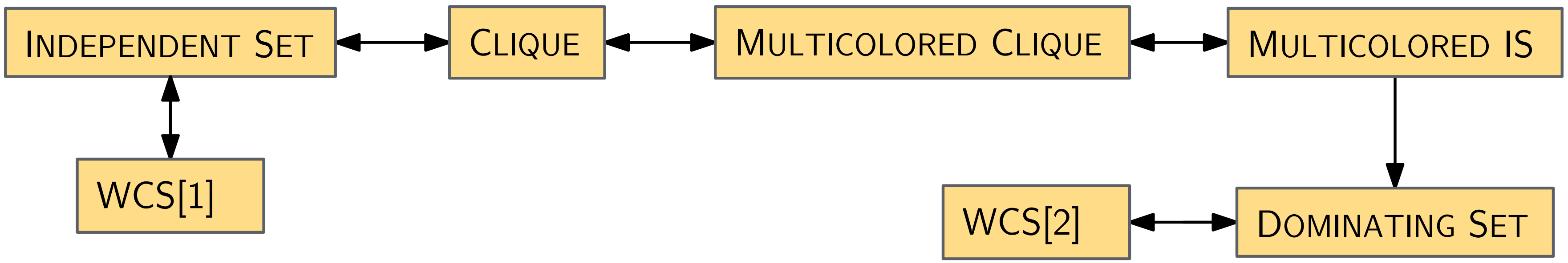
FPT-Reductions so far



Further reductions

- one can also reduce WCS[1] to INDEPENDENT SET

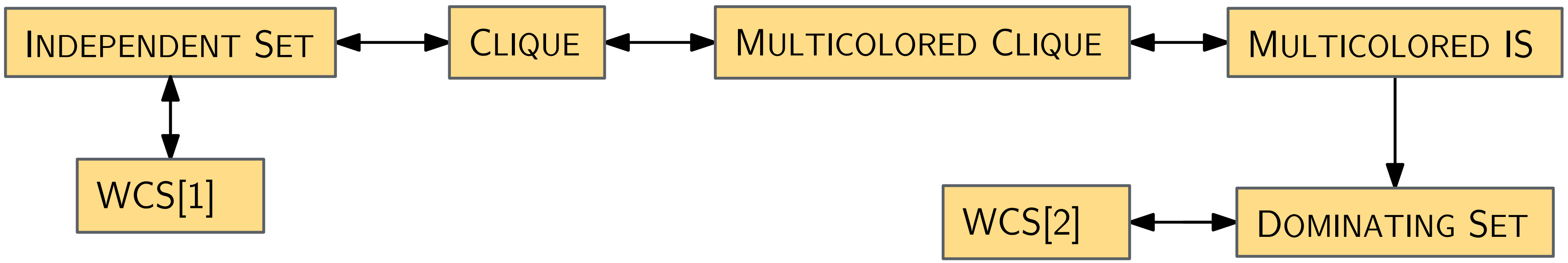
FPT-Reductions so far



Further reductions

- one can also reduce WCS[1] to Independent Set
- and WCS[2] to Dominating Set

FPT-Reductions so far

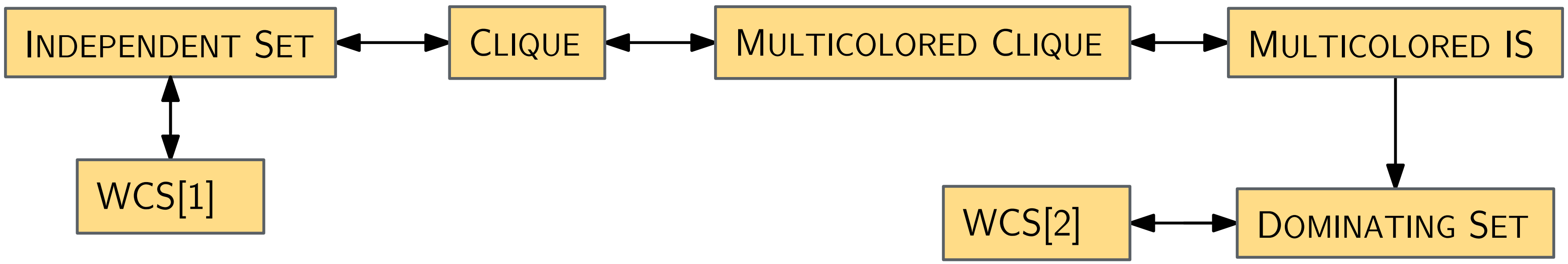


Further reductions

- one can also reduce WCS[1] to INDEPENDENT SET
- and WCS[2] to DOMINATING SET
- so, all problems in $W[1]$ reduce to IS

$\rightsquigarrow$ IS is $W[1]$-complete

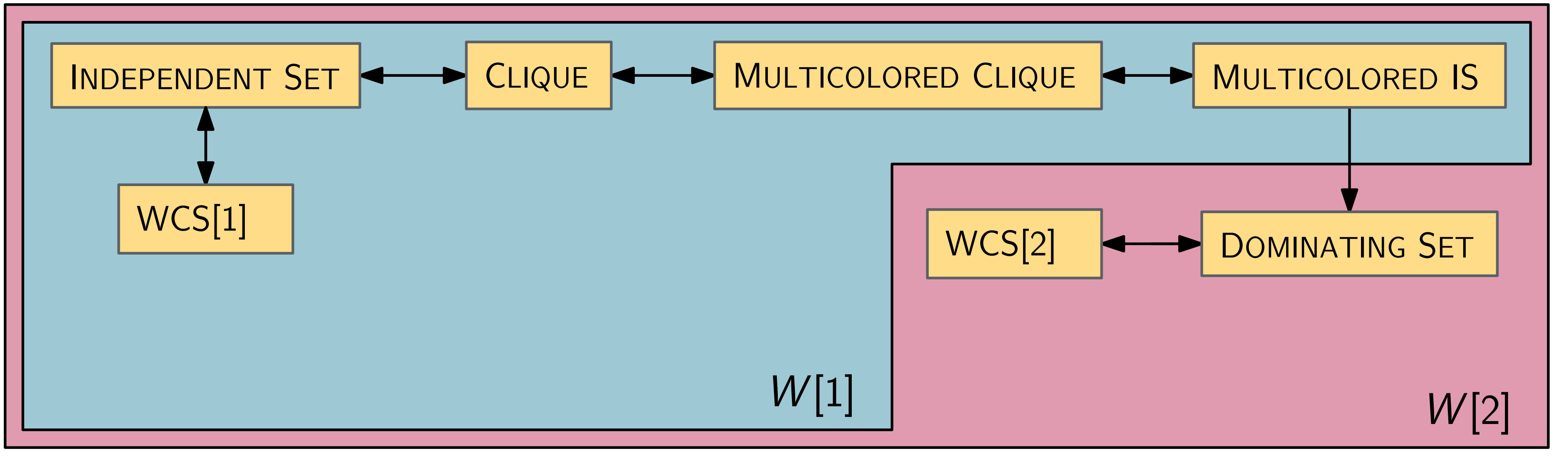
FPT-Reductions so far



Further reductions

- one can also reduce WCS[1] to Independent Set
- and WCS[2] to Dominating Set
- so, all problems in $W[1]$ reduce to IS
  $\rightsquigarrow$ IS is $W[1]$-complete
- similarly DS is $W[2]$-complete

FPT-Reductions so far



Further reductions

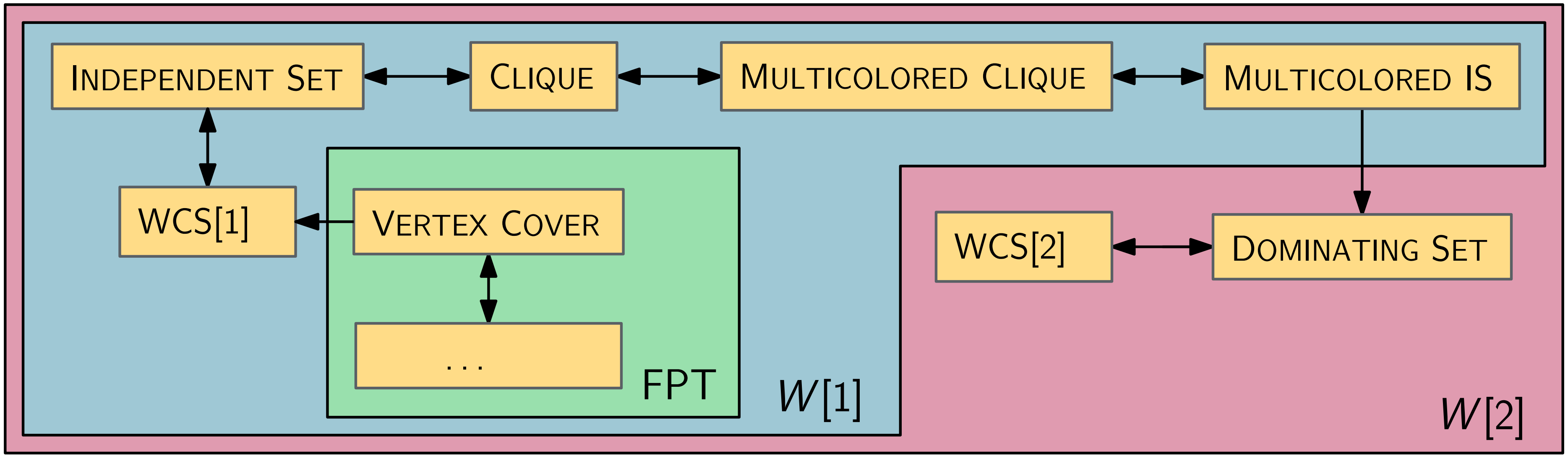
- one can also reduce WCS[1] to INDEPENDENT SET
- and WCS[2] to DOMINATING SET
- so, all problems in $W[1]$ reduce to IS

  $\rightsquigarrow$ IS is $W[1]$-complete

- similarly DS is $W[2]$-complete
- note: $W[1] \subseteq W[2]$

why?

FPT-Reductions so far



Further reductions

- one can also reduce WCS[1] to Independent Set
- and WCS[2] to Dominating Set
- so, all problems in $W[1]$ reduce to IS

  $\rightsquigarrow$ IS is $W[1]$-complete

- similarly DS is $W[2]$-complete
- note: $W[1] \subseteq W[2]$ why?
- also: FPT $\subseteq W[1] \subseteq W[2]$ why?

Summary

The W-Hierarchy

- Complexity Classes FPT $\subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \cdots$
- $W[t]$ defined via a prototypical complete problem WCS[t]:
  $\mathcal{L} \in W[t] \Leftrightarrow \mathcal{L}$ can be reduced to WCS[t] (by FPT-reduction)

Summary

The W-Hierarchy

- Complexity Classes FPT $\subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \cdots$
- $W[t]$ defined via a prototypical complete problem WCS$[t]$:
  $\mathcal{L} \in W[t] \Leftrightarrow \mathcal{L}$ can be reduced to WCS$[t]$ (by FPT-reduction)
- Inclusions **expected** to be strict

Summary

The W-Hierarchy

- Complexity Classes $\mathrm{FPT} \subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \cdots$
- $W[t]$ defined via a prototypical complete problem $\mathrm{WCS}[t]$:
  $\mathcal{L} \in W[t] \Leftrightarrow \mathcal{L}$ can be reduced to $\mathrm{WCS}[t]$ (by FPT-reduction)
- Inclusions **expected** to be strict

Is my $W[t]$-completeness proof useless if $W[t] = \mathrm{FPT}$?

Summary

The W-Hierarchy

- Complexity Classes FPT $\subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \cdots$
- $W[t]$ defined via a prototypical complete problem WCS$[t]$:
  $\mathcal{L} \in W[t] \Leftrightarrow \mathcal{L}$ can be reduced to WCS$[t]$ (by FPT-reduction)
- Inclusions **expected** to be strict

Is my $W[t]$-completeness proof useless if $W[t] =$ FPT?

- finding an FPT-algorithm for a complete problem would provide an FPT-algorithm for all problems in the class.

Summary

The W-Hierarchy

- Complexity Classes FPT $\subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \cdots$
- $W[t]$ defined via a prototypical complete problem WCS[t]:
  $\mathcal{L} \in W[t] \Leftrightarrow \mathcal{L}$ can be reduced to WCS[t] (by FPT-reduction)
- Inclusions **expected** to be strict

Is my $W[t]$-completeness proof useless if $W[t] =$ FPT?

- finding an FPT-algorithm for a complete problem would provide an FPT-algorithm for all problems in the class.

How do I show that my problem is hard?

- reduce a known hard problem to your problem

Summary

The W-Hierarchy

- Complexity Classes FPT $\subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \cdots$
- $W[t]$ defined via a prototypical complete problem WCS$[t]$:
  $\mathcal{L} \in W[t] \Leftrightarrow \mathcal{L}$ can be reduced to WCS$[t]$ (by FPT-reduction)
- Inclusions **expected** to be strict

Is my $W[t]$-completeness proof useless if $W[t] =$ FPT?

- finding an FPT-algorithm for a complete problem would provide an FPT-algorithm for all problems in the class.

How do I show that my problem is hard?

- reduce a known hard problem to your problem
- reducing from MC Independent Set or MC Clique provides $W[1]$-hardness

Summary

The W-Hierarchy

- Complexity Classes FPT $\subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \cdots$
- $W[t]$ defined via a prototypical complete problem WCS$[t]$:
  $\mathcal{L} \in W[t] \Leftrightarrow \mathcal{L}$ can be reduced to WCS$[t]$ (by FPT-reduction)
- Inclusions **expected** to be strict

Is my $W[t]$-completeness proof useless if $W[t] = $ FPT?

- finding an FPT-algorithm for a complete problem would provide an FPT-algorithm for all problems in the class.

How do I show that my problem is hard?

- reduce a known hard problem to your problem
- reducing from MC Independent Set or MC Clique provides $W[1]$-hardness
- reducing from Dominating Set or Set Cover provides $W[2]$-hardness