Grundy Distinguishes Treewidth from Pathwidth

Michael Lampis
LAMSADE
Université Paris Dauphine

ESA 2020

Acknowledgements

This is joint work with:

Rémy Belmonte UEC

Eun Jung Kim LAMSADE

Valia Mitsou IRIF

Yota Otachi Nagoya U

Funded by the bilateral French-Japanese project PARAGA.

Full paper available at: `https://arxiv.org/abs/2008.07425`

What is this talk about?

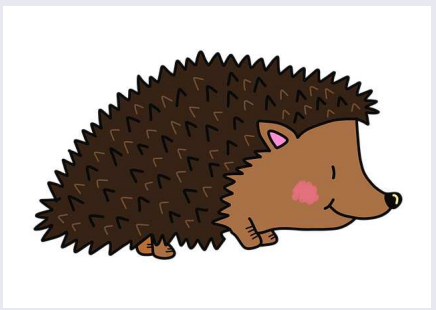
Two ways to look at this work

A talk about structural parameters

- Treewidth
- Pathwidth
- Treedepth, Cliquewidth, ...
- **Price of Generality**
  - Which problems are “easy” for pathwidth but “hard” for treewidth?

A talk about Grundy Coloring

- Well-known optimization problem
- MaxMin variant of Coloring
  - Find a proper coloring that uses the **max** number of colors but the color of no vertex can be decreased.



What is this talk about?

Two ways to look at this work

A talk about structural parameters

- Treewidth
- Pathwidth
- Treedepth, Cliquewidth, ...
- **Price of Generality**
  - Which problems are “easy” for pathwidth but “hard” for treewidth?

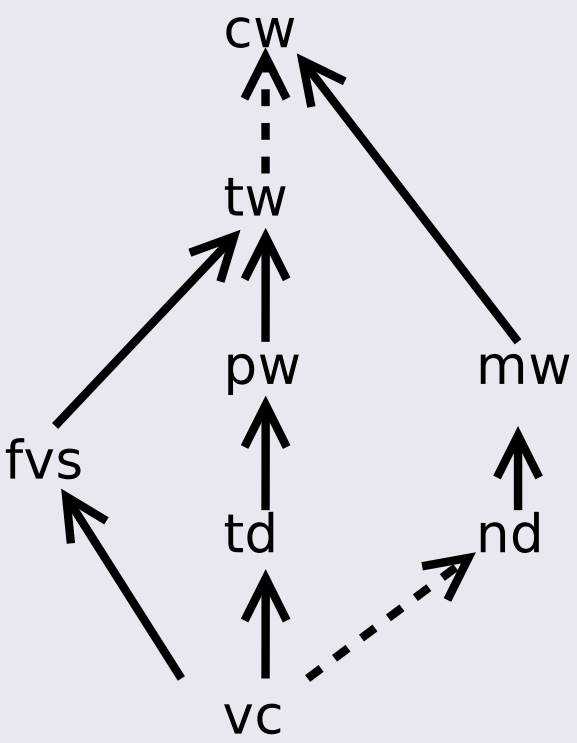
A talk about Grundy Coloring

- Well-known optimization problem
- MaxMin variant of Coloring
  - Find a proper coloring that uses the **max** number of colors but the color of no vertex can be decreased.

“The fox knows many things, but the hedgehog knows one big thing”, Aesop’s fables

What does the fox say?

Price of Generality – Structural Parameters

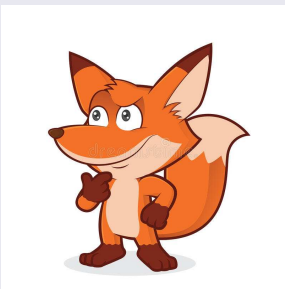


Each problem/parameter pair is typically either:

- FPT: solvable in $f(w)n^{O(1)}$
- XP and W-hard: solvable in $n^{g(w)}$, not FPT
- paraNP-hard: NP-hard for $w = O(1)$

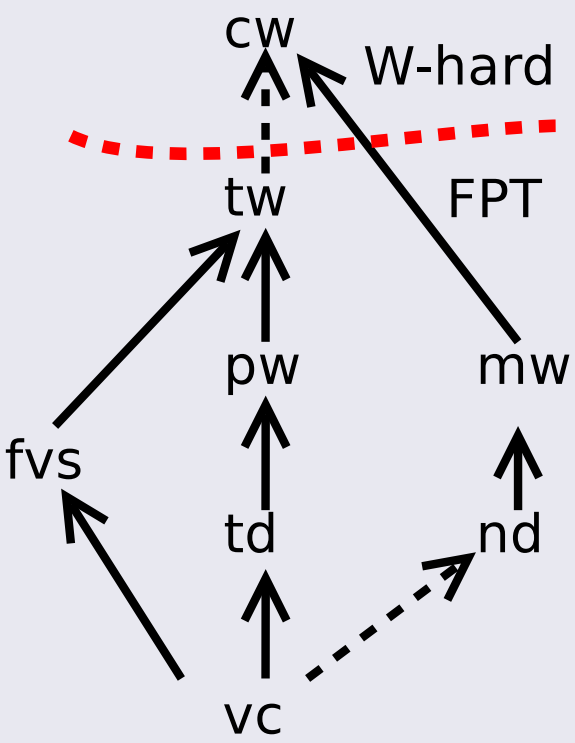
- Tractability propagates "downwards", hardness "upwards"
- Big Picture Question: Which problems do we "lose" when we transition between parameters?

- **Price of Generality**
  - [Fomin, Golovach, Lokshtanov, Saurabh, SODA'09]
  - Showed EDS, MaxCut, Coloring, Hamiltonicity FPT for tw, W-hard for cw.



DAUPHINE UNIVERSITÉ PARIS

Price of Generality – Structural Parameters

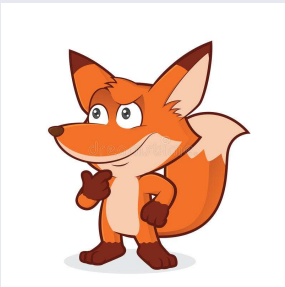


Each problem/parameter pair is typically either:

- FPT: solvable in $f(w)n^{O(1)}$
- XP and W-hard: solvable in $n^{g(w)}$, not FPT
- paraNP-hard: NP-hard for $w = O(1)$

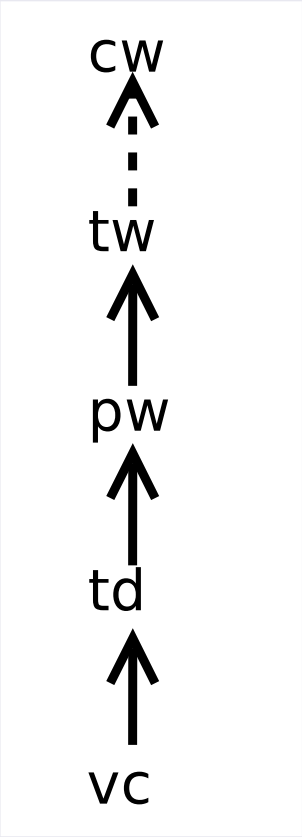
- Tractability propagates "downwards", hardness "upwards"
- Big Picture Question: Which problems do we "lose" when we transition between parameters?

- **Price of Generality**
  - [Fomin, Golovach, Lokshtanov, Saurabh, SODA'09]
  - Showed EDS, MaxCut, Coloring, Hamiltonicity FPT for tw, W-hard for cw.



DAUPHINE UNIVERSITÉ PARIS

Price of Generality Continued

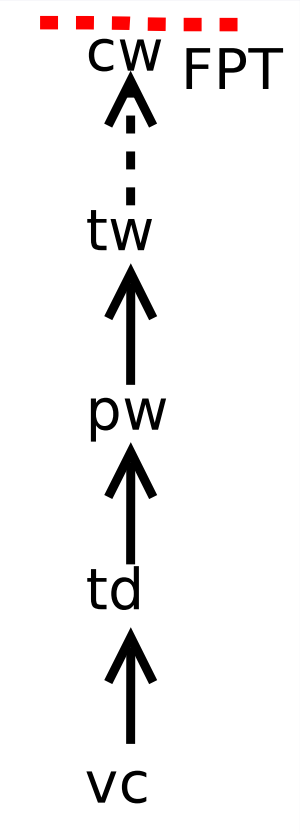


Comments

Price of Generality Examples

Clique-width	
Treewidth	
Pathwidth	
Tree-depth	
Vertex Cover	

Price of Generality Continued

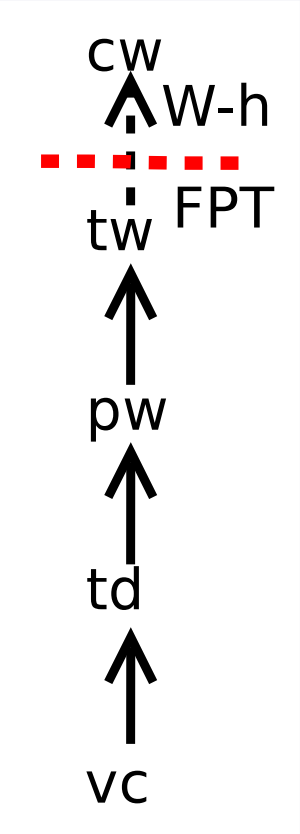


Comments

Price of Generality Examples

	All MSO_1, Dominating Set, Vertex Cover
Clique-width	
Treewidth	
Pathwidth	
Tree-depth	
Vertex Cover	

Price of Generality Continued



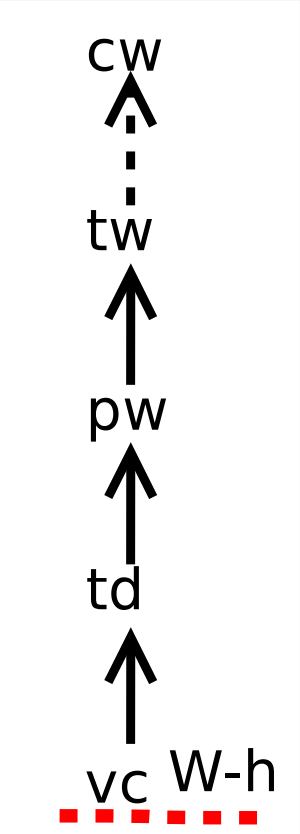
Price of Generality Examples

	All MSO_1, Dominating Set, Vertex Cover
Clique-width	
	Coloring, EDS, SAT, #Matching
Treewidth	
Pathwidth	
Tree-depth	
Vertex Cover	

Comments

- SAT: [Ordyniak, Paulusma, Szeider, TCS '13]
- #Matching: [Curticapean, Marx, SODA '16]

Price of Generality Continued



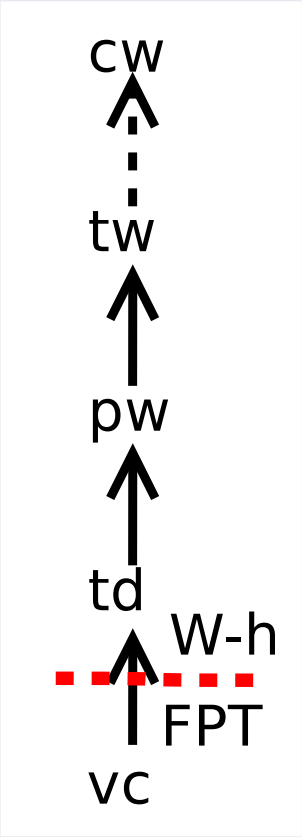
Price of Generality Examples

	All MSO_1, Dominating Set, Vertex Cover
Clique-width	
	Coloring, EDS, SAT, #Matching
Treewidth	
Pathwidth	
Tree-depth	
Vertex Cover	
	List Coloring, r-Dom Set, d-Ind Set

Comments

- List Coloring: [Fellows et al. Inf Comp '11]. First such problem!
- r-DS: [Katsikarelis, L., Paschos, DAM '19]
- Very few problems here!

Price of Generality Continued



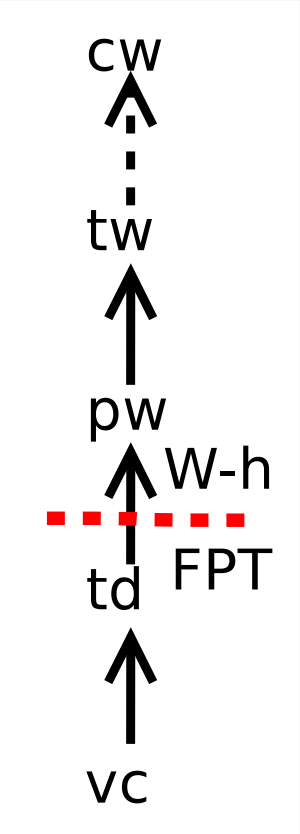
Price of Generality Examples

	All MSO_1, Dominating Set, Vertex Cover
Clique-width	
	Coloring, EDS, SAT, #Matching
Treewidth	
Pathwidth	
Tree-depth	
	Capacitated DS/VC, BDD,...
Vertex Cover	
	List Coloring, r-Dom Set, d-Ind Set

Comments

- Cap VC/DS: [Dom et al. IWPEC 2008]
- **Most problems W[1]-hard for tw are here**!

Price of Generality Continued



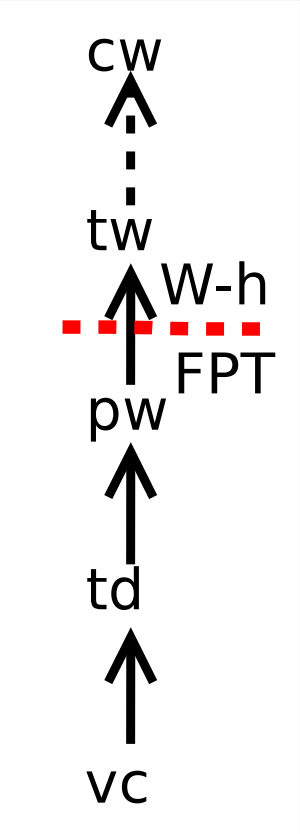
Price of Generality Examples

	All MSO_1, Dominating Set, Vertex Cover
Clique-width	
	Coloring, EDS, SAT, #Matching
Treewidth	
Pathwidth	
	Mixed Chinese Postman, r-DS
Tree-depth	
	Capacitated DS/VC, BDD,...
Vertex Cover	
	List Coloring, r-Dom Set, d-Ind Set

Comments

- MCP: [Gutin, Jones, Wahlström, SIDMA '16]. First of this type!
- Also: Bounded-Length Cut, Geodetic Set, ILP.

Price of Generality Continued

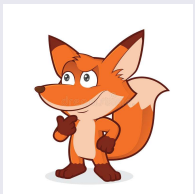


Price of Generality Examples

	All MSO_1, Dominating Set, Vertex Cover
Clique-width	
	Coloring, EDS, SAT, #Matching
Treewidth	
	???
Pathwidth	
	Mixed Chinese Postman, r-DS
Tree-depth	
	Capacitated DS/VC, BDD,...
Vertex Cover	
	List Coloring, r-Dom Set, d-Ind Set

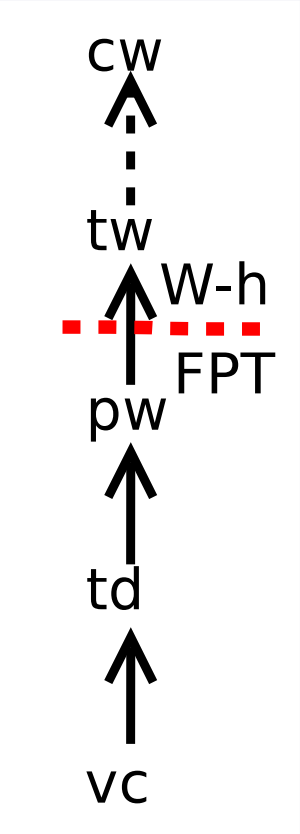
Comments

No **natural** problem known??



DAUPHINE UNIVERSITÉ PARIS

Price of Generality Continued



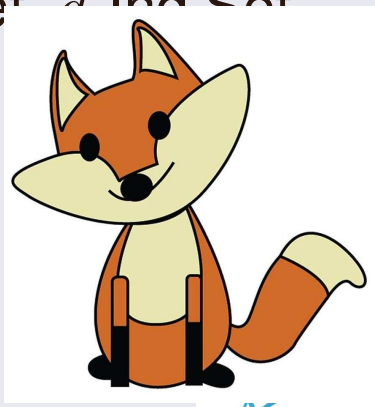
Price of Generality Examples

	All MSO_1, Dominating Set, Vertex Cover
Clique-width	
	Coloring, EDS, SAT, #Matching
Treewidth	
	Grundy Coloring!
Pathwidth	
	Mixed Chinese Postman, r-DS
Tree-depth	
	Capacitated DS/VC, BDD,...
Vertex Cover	
	List Coloring, r-Dom Set, d-Ind Set

Comments

Main result of this talk:

Grundy Coloring is such a problem!



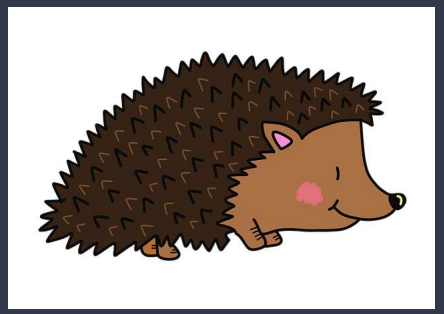
DAUPHINE UNIVERSITÉ PARIS

A Lesson from the fox

Price of Generality and Combinatorics

- Sometimes, the reason a problem becomes FPT for a more restricted parameter is more combinatorial than algorithmic.
- Example:
  - Coloring is FPT for tw, W-hard for cw.
  - But algorithm runs in k^{tw}. Is this FPT?
  - Yes! Because in all graphs $\chi(G) \leq tw(G)$.
  - This bound makes all the difference: Coloring is FPT by $cw + k$.
- Example:
  - r-Dom Set is FPT for td, W-hard for pw.
  - Why W-hard for pw? DP runs in $r^{O(pw)}$. But r could be large!
  - Why FPT for td? Graphs of tree-depth t have no simple path of length $> 2^t$, so $r \leq 2^{td}$.
  - Again saved by combinatorial bound on optimal!

Let’s nail this problem!

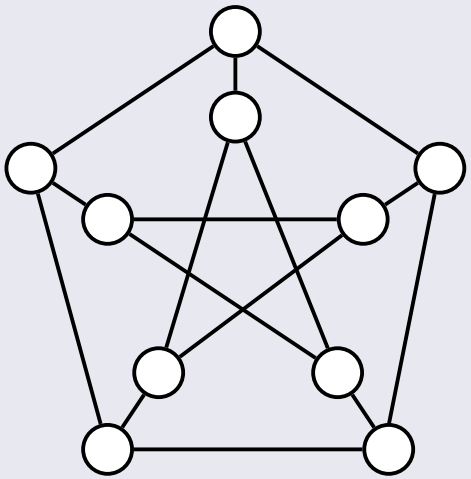


Grundy Coloring

- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.

Grundy Coloring

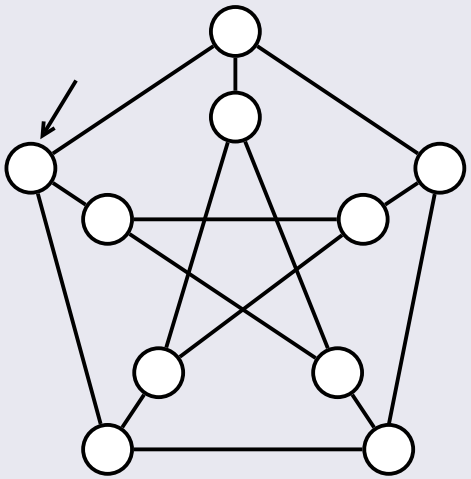
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

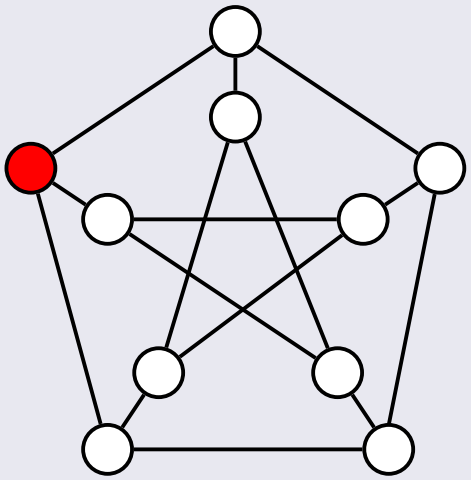
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

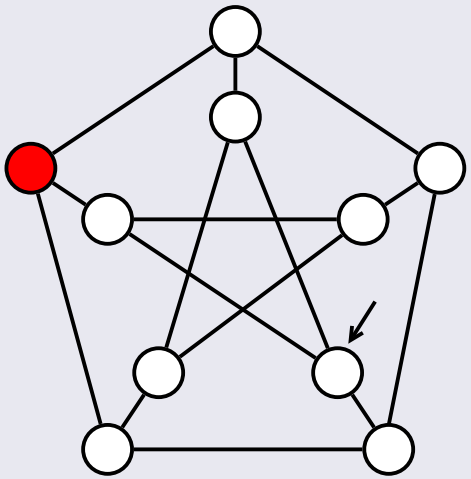
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

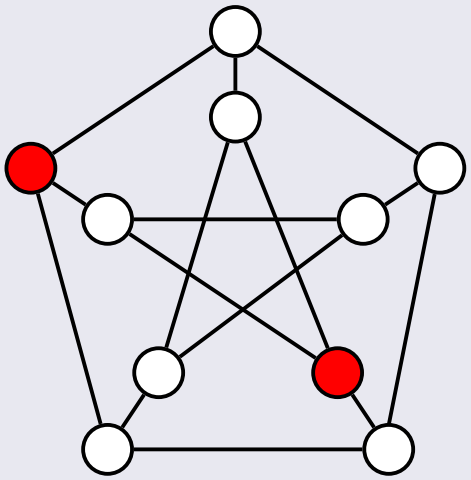
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

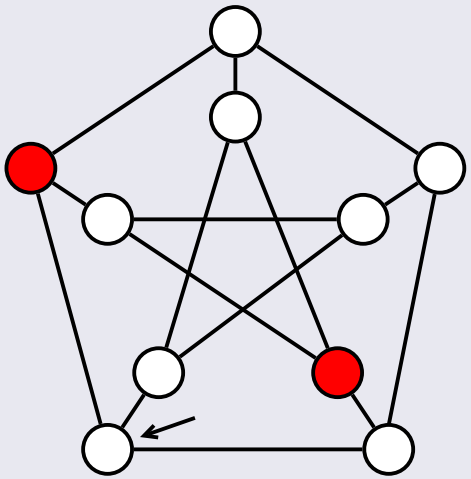
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

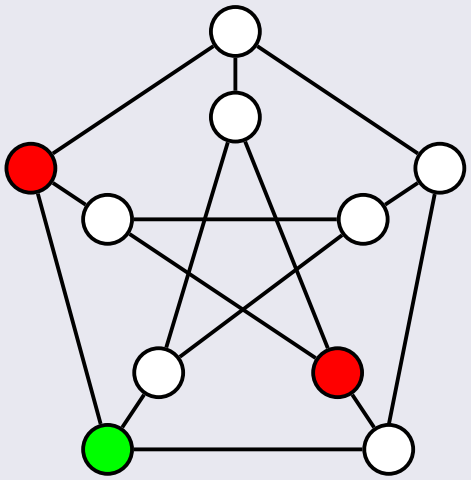
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

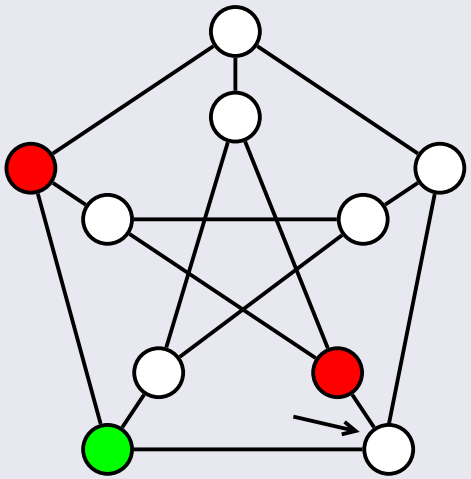
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

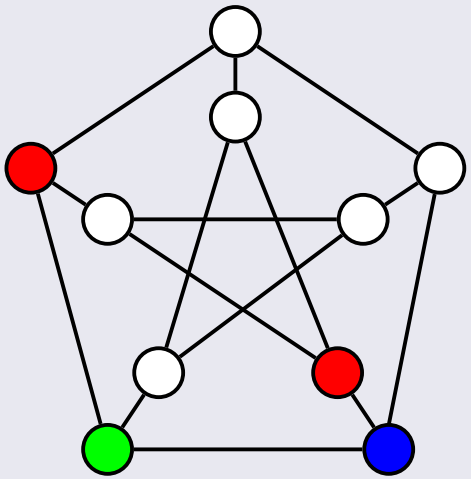
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

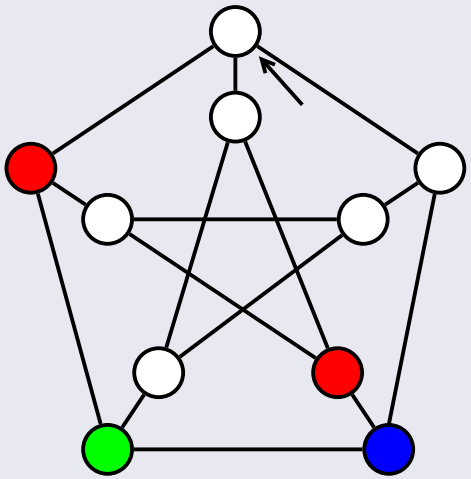
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

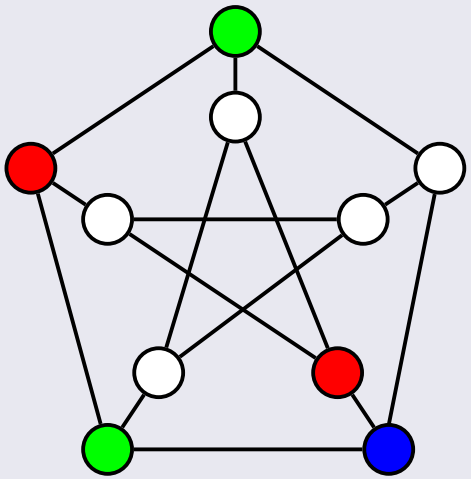
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

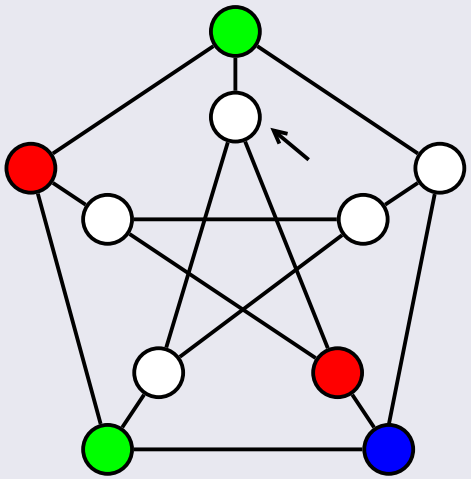
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

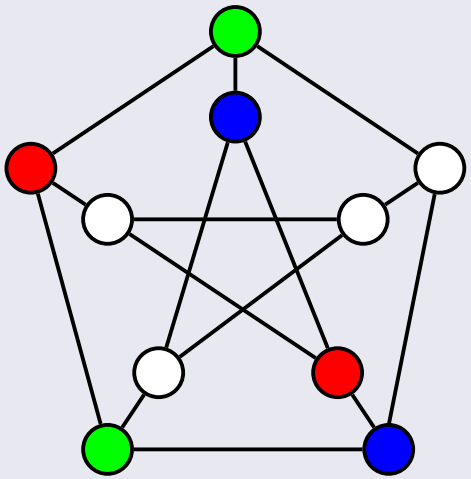
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

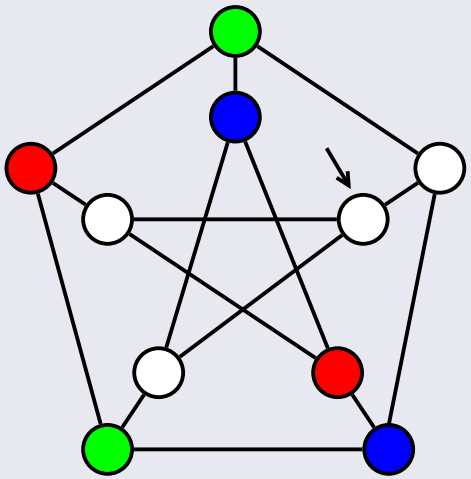
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

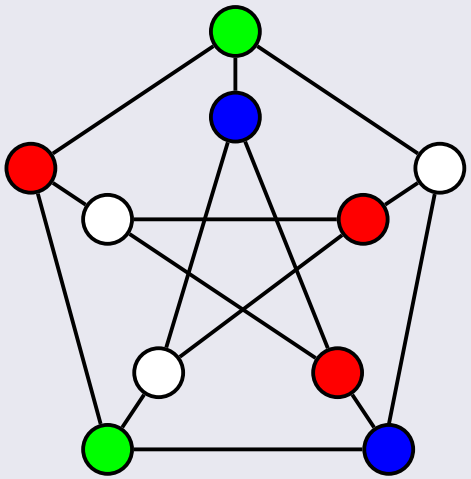
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

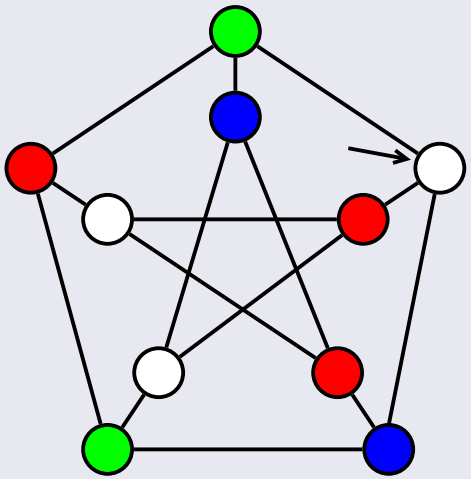
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

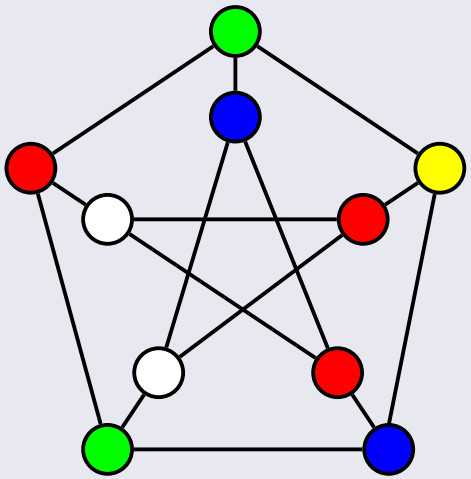
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

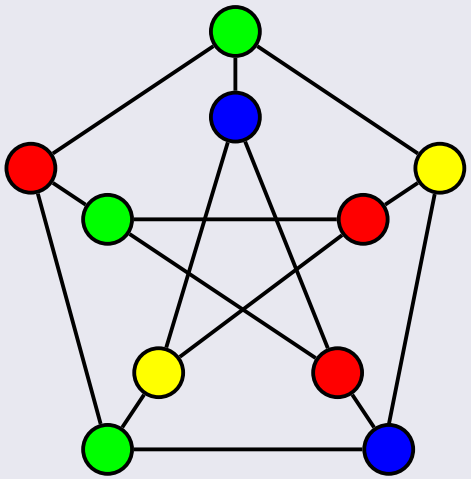
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



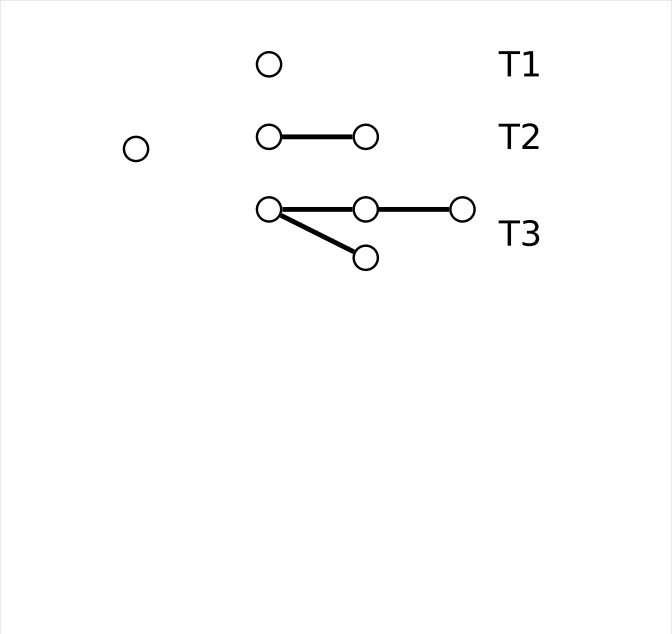
Red	1
Green	2
Blue	3
Yellow	4

Binomial Trees

- The Binomial Tree T_k has a Grundy Coloring which assigns color k to the root

Binomial Trees

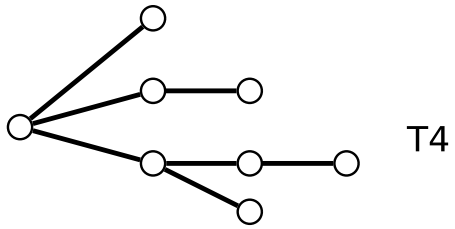
- The Binomial Tree T_k has a Grundy Coloring which assigns color k to the root



- Two recursive constructions
- T_1 is a vertex.
- T_k is a new root connected to $T_{k-1}, T_{k-2}, \ldots, T_1$.

Binomial Trees

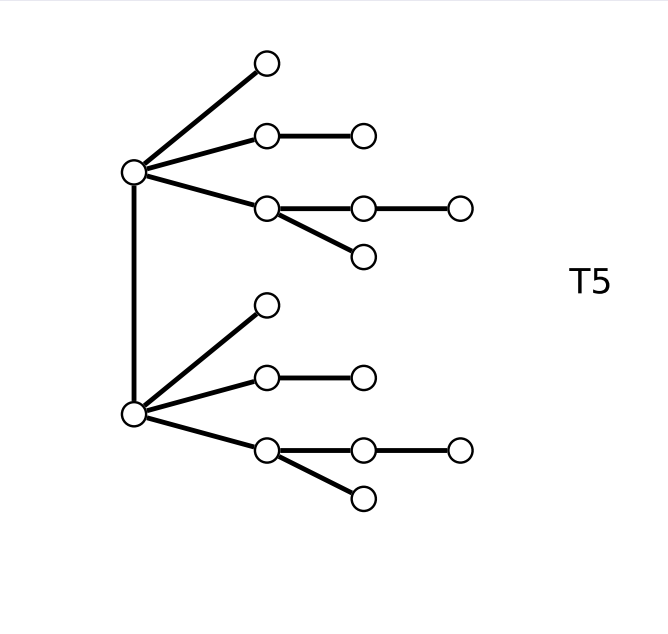
- The Binomial Tree T_k has a Grundy Coloring which assigns color k to the root



- Two recursive constructions
- T_1 is a vertex.
- T_k is a new root connected to $T_{k-1}, T_{k-2}, \ldots, T_1$.

Binomial Trees

- The Binomial Tree T_k has a Grundy Coloring which assigns color k to the root



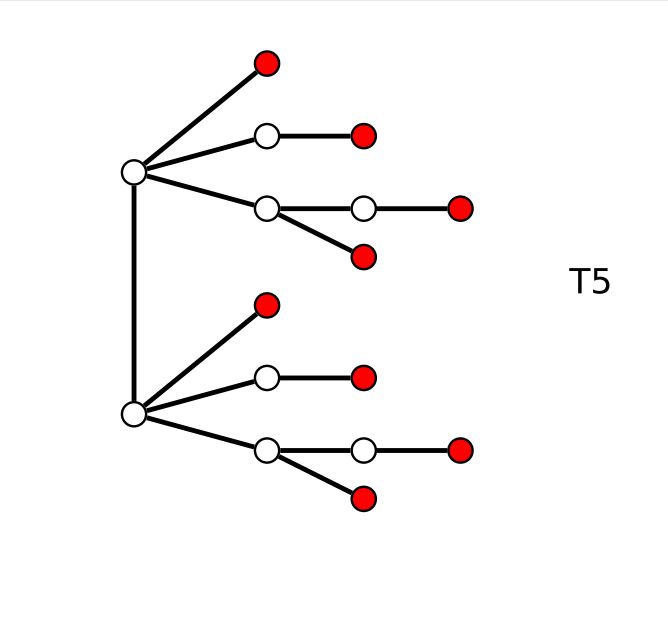
- Two recursive constructions
- T_1 is a vertex.
- T_k is a new root connected to $T_{k-1}, T_{k-2}, \ldots, T_1$.

  Or

- T_k is formed by connecting two copies of T_{k-1}

Binomial Trees

- The Binomial Tree T_k has a Grundy Coloring which assigns color k to the root



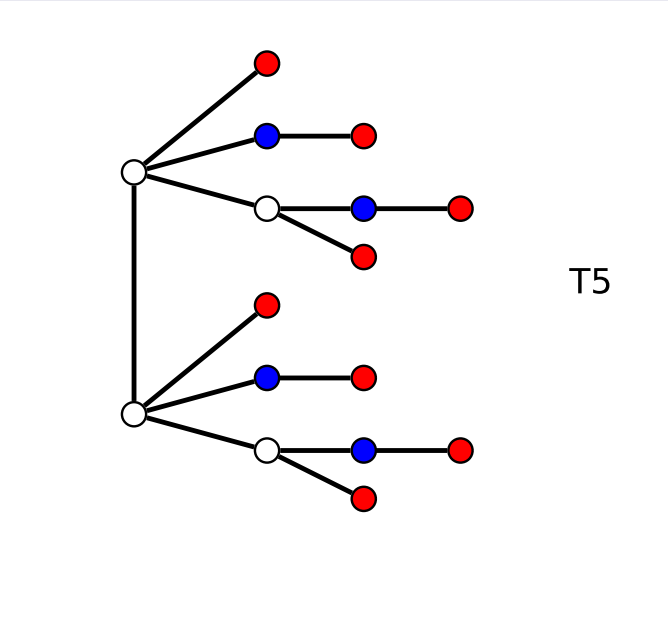
- Two recursive constructions
- T_1 is a vertex.
- T_k is a new root connected to $T_{k-1}, T_{k-2}, \ldots, T_1$.

  Or

- T_k is formed by connecting two copies of T_{k-1}

Binomial Trees

- The Binomial Tree T_k has a Grundy Coloring which assigns color k to the root



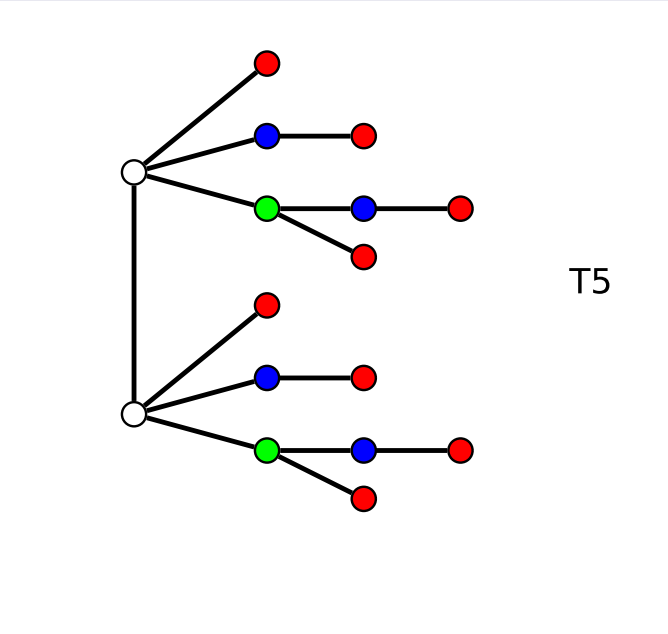
- Two recursive constructions
- T_1 is a vertex.
- T_k is a new root connected to $T_{k-1}, T_{k-2}, \ldots, T_1$.

  Or

- T_k is formed by connecting two copies of T_{k-1}

Binomial Trees

- The Binomial Tree T_k has a Grundy Coloring which assigns color k to the root



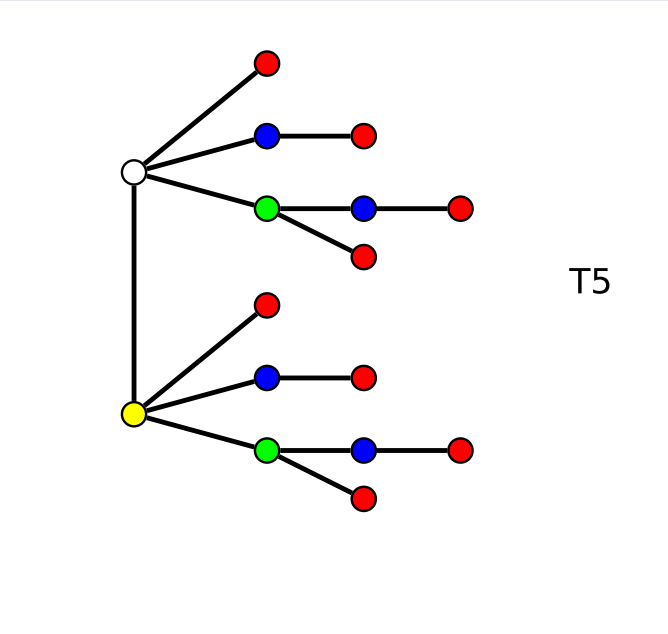
- Two recursive constructions
- T_1 is a vertex.
- T_k is a new root connected to $T_{k-1}, T_{k-2}, \ldots, T_1$.

  Or

- T_k is formed by connecting two copies of T_{k-1}

Binomial Trees

- The Binomial Tree T_k has a Grundy Coloring which assigns color k to the root



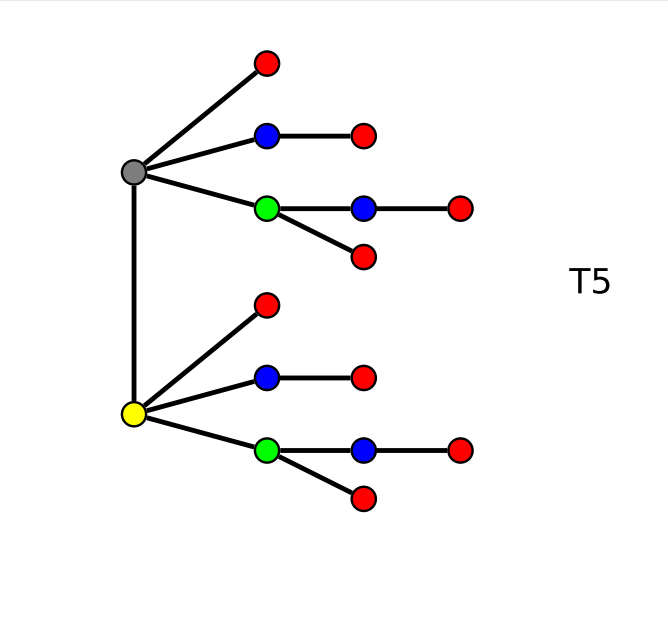
- Two recursive constructions
- T_1 is a vertex.
- T_k is a new root connected to $T_{k-1}, T_{k-2}, \ldots, T_1$.

  Or

- T_k is formed by connecting two copies of T_{k-1}

Binomial Trees

- The Binomial Tree T_k has a Grundy Coloring which assigns color k to the root



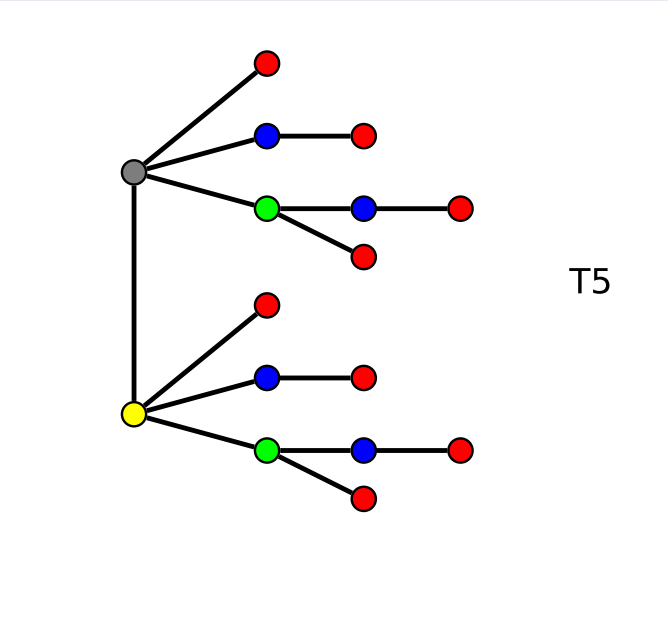
- Two recursive constructions
- T_1 is a vertex.
- T_k is a new root connected to $T_{k-1}, T_{k-2}, \ldots, T_1$.

  Or

- T_k is formed by connecting two copies of T_{k-1}

Binomial Trees

- The Binomial Tree T_k has a Grundy Coloring which assigns color k to the root



- Two recursive constructions
- T_1 is a vertex.
- T_k is a new root connected to $T_{k-1}, T_{k-2}, \ldots, T_1$.

  Or

- T_k is formed by connecting two copies of T_{k-1}

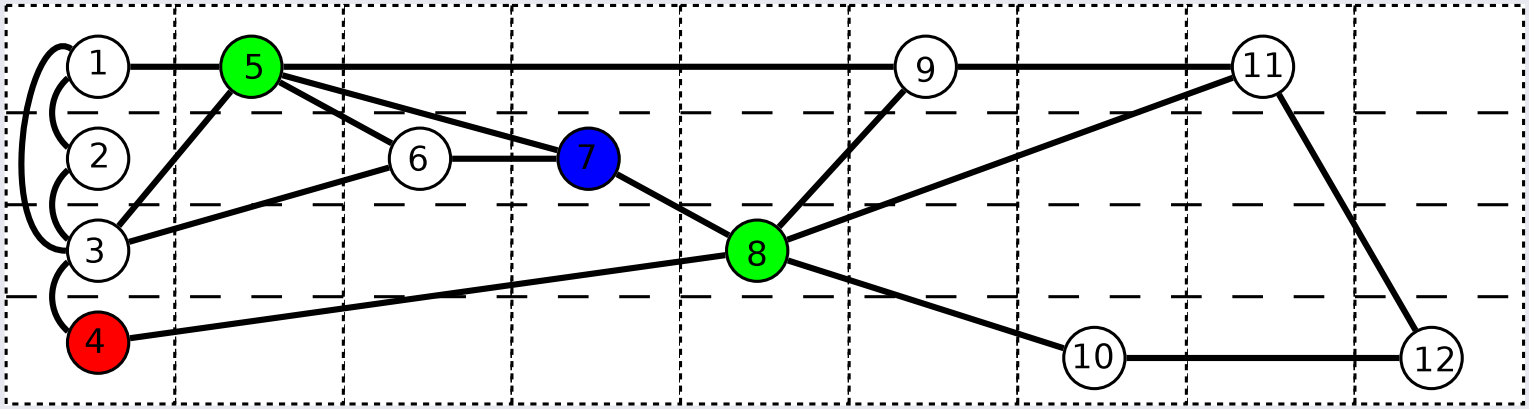
- We have $\Gamma(T_k) = k$ but $\chi(T_k) = 2$.
- $|T_k| = 2^{k-1}$.
- This is tight: for all trees $\Gamma(T) \leq \log n$.

- **More generally:** for all graphs $\Gamma(G) \leq tw(G) \log n$.

DAUPHINE UNIVERSITÉ PARIS

Algorithm for Grundy and Treewidth

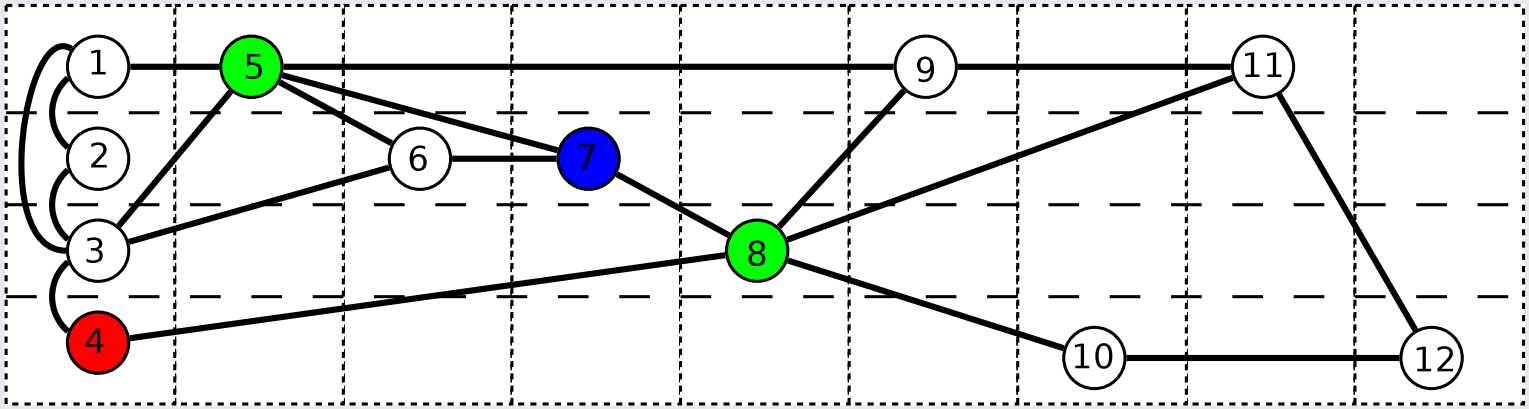
- XP algorithm due to [Telle&Proskurowski SIDMA'97]



- Standard Coloring DP: recall color of each vertex in bag
- Reminder: Bags are separators
- Only need to remember which colorings of the bag can be extended to the left.
  - Complexity: $\rightarrow k^{tw}$

Algorithm for Grundy and Treewidth

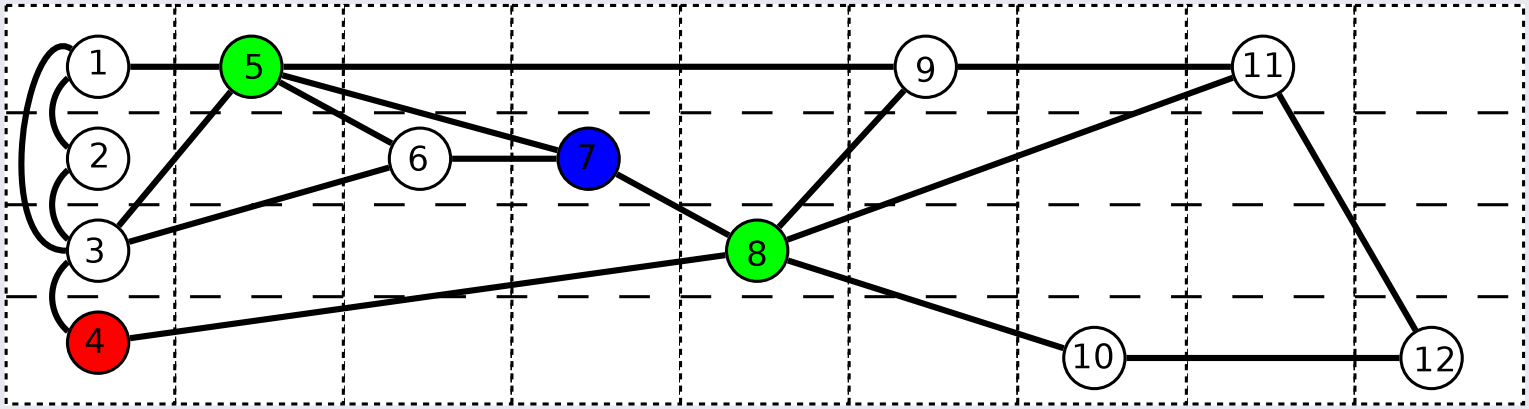
- XP algorithm due to [Telle&Proskurowski SIDMA'97]



- Grundy: for each vertex we also need to make sure that it is dominated by **all** lower colors
  - In this example, this coloring is only valid if 6 takes color Red
- Need to remember for each vertex **the subset** of colors it has seen in its neighborhood
  - $\rightarrow (2^k)^{tw}$

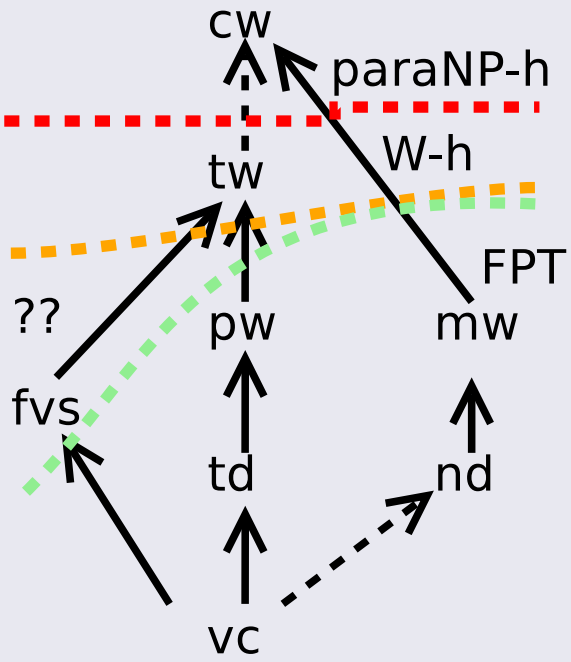
Algorithm for Grundy and Treewidth

- XP algorithm due to [Telle&Proskurowski SIDMA'97]



- Overall running time $O^*((k2^k)^{tw})$.
- Is this XP?
- Yes, if we use that $k \leq tw \log n$
- Running time: $n^{O(tw^2)}$

Our results



Main results:

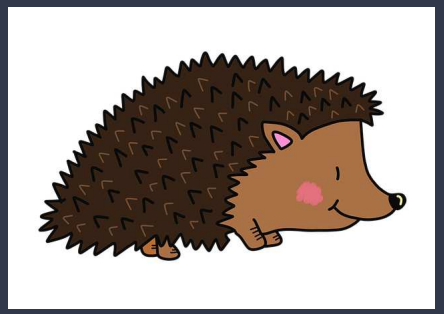
- Grundy Coloring is W[1]-hard by treewidth
- Grundy Coloring is FPT by pathwidth

Also:

- Grundy Coloring is NP-h for clique-width$= 6$
- Grundy Coloring is FPT for modular width

- Key insight: ability to bound $\Gamma(G)$ is crucial
  - For bounded pw we have bounded Γ
  - For bounded tw we have $\Gamma \leq tw \log n$
  - No upper bound on Γ for bounded cw

W-hardness for treewidth



Proof Outline

- Desired result: Grundy Coloring is W[1]-hard by treewidth
- Proof: Reduction from k-MCC
  - k-MCC: given properly k-colored graph, decide if exists k-Clique.

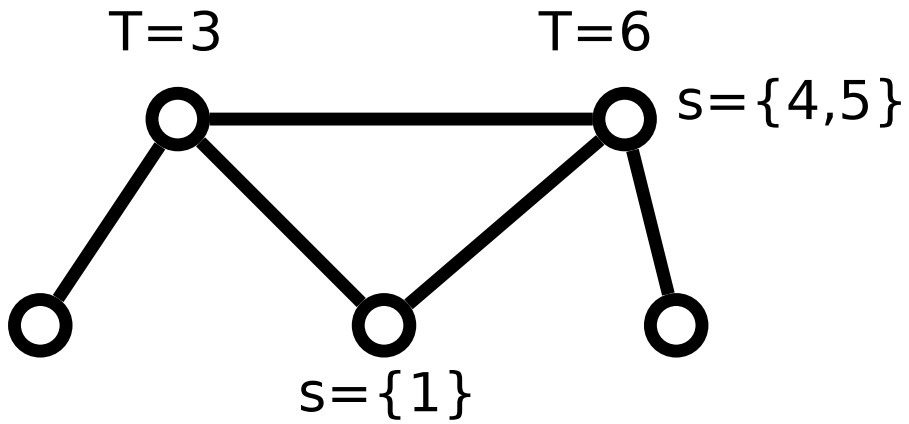
Proof Outline

- Desired result: Grundy Coloring is W[1]-hard by treewidth
- Proof: Reduction from k-MCC
  - k-MCC: given properly k-colored graph, decide if exists k-Clique.

Steps:

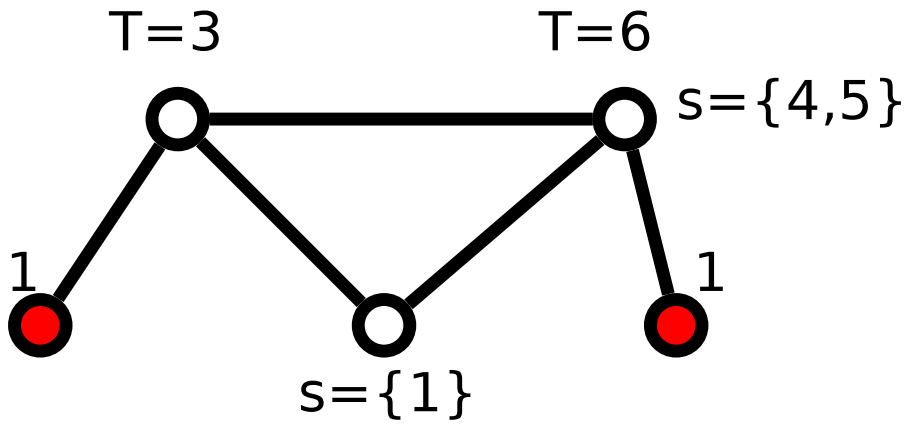
- Define more general “Grundy with Targets and Supports”
- Show that GwTS is W[1]-hard parameterized by **pathwidth**
  - Not a typo! More info later...
- Use binomial trees to reduce GwTS/pw to Grundy/tw

Grundy with Supports and Targets – Example



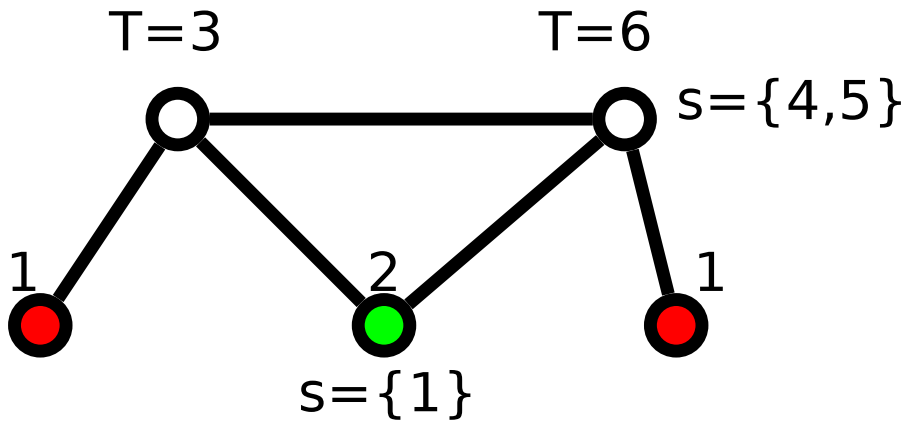
- Example of generalized problem instance.
- Two vertices have a target we want to achieve.
- Some vertices have a support set: we don’t need to assign them neighbors of these colors to obtain a higher color.

Grundy with Supports and Targets – Example



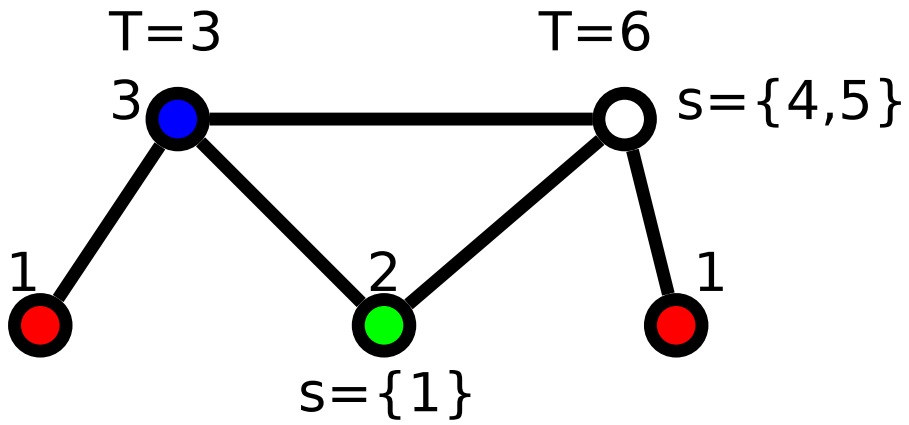
- Example of generalized problem instance.
- Two vertices have a target we want to achieve.
- Some vertices have a support set: we don’t need to assign them neighbors of these colors to obtain a higher color.

Grundy with Supports and Targets – Example



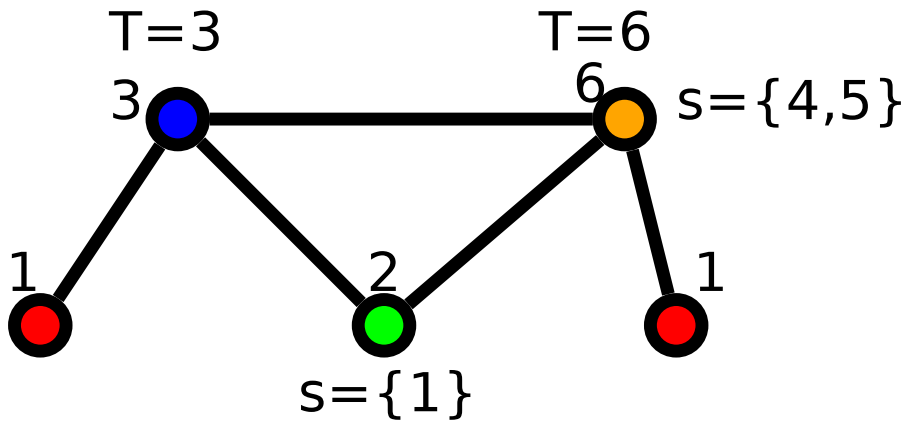
- Example of generalized problem instance.
- Two vertices have a target we want to achieve.
- Some vertices have a support set: we don’t need to assign them neighbors of these colors to obtain a higher color.

Grundy with Supports and Targets – Example



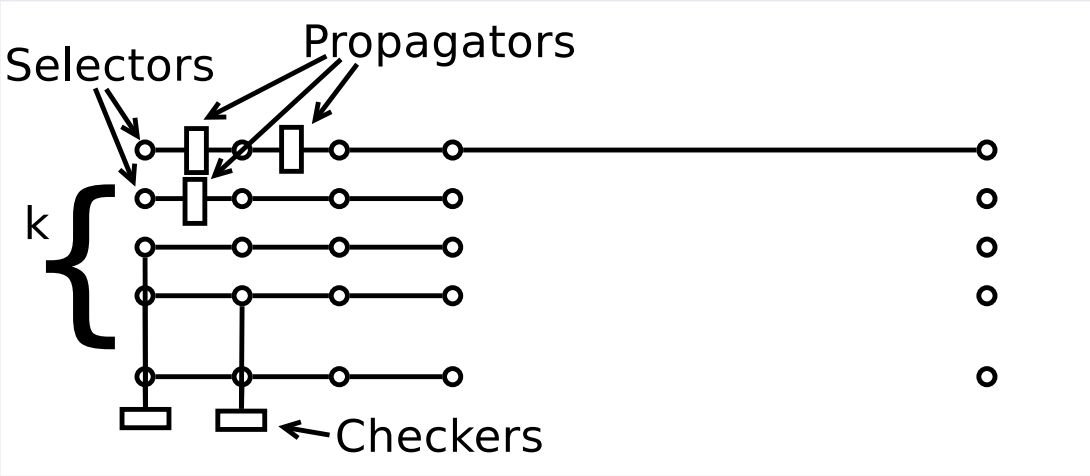
- Example of generalized problem instance.
- Two vertices have a target we want to achieve.
- Some vertices have a support set: we don’t need to assign them neighbors of these colors to obtain a higher color.

Grundy with Supports and Targets – Example



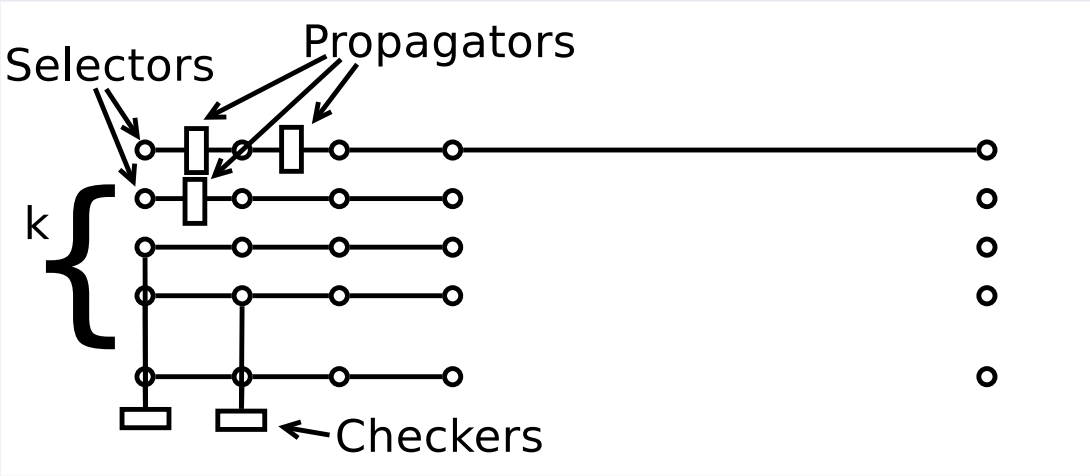
- Example of generalized problem instance.
- Two vertices have a target we want to achieve.
- Some vertices have a support set: we don’t need to assign them neighbors of these colors to obtain a higher color.

Outline of hardness for GwTS



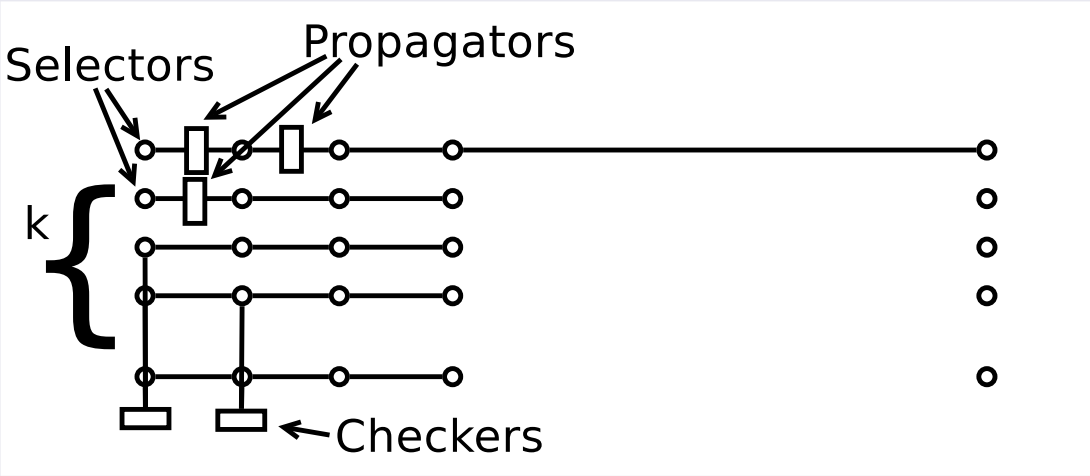
- $k \times m$ “grid” where each row represents a color class

Outline of hardness for GwTS



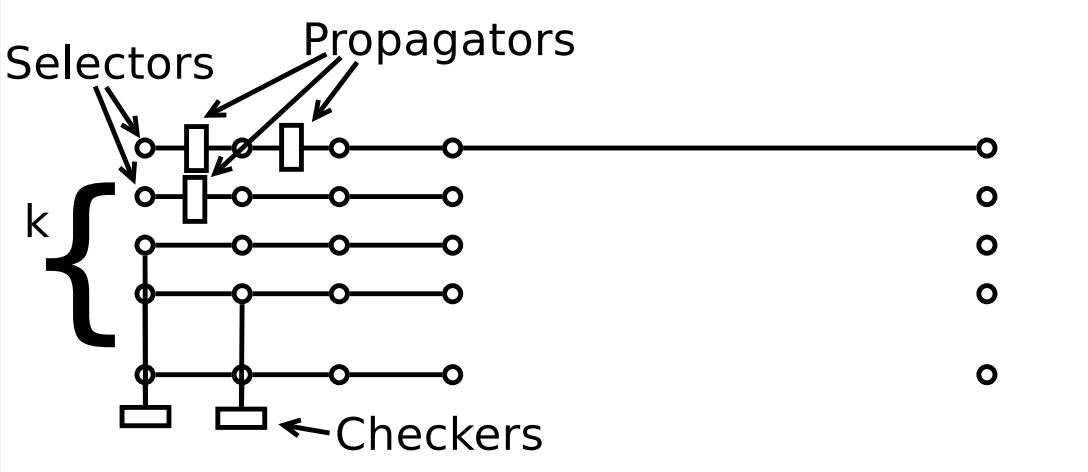
- $k \times m$ "grid" where each row represents a color class
- Selector gadget: has n "reasonable" Grundy colorings. Each encodes a selection of a vertex in original k-MCC instance.

Outline of hardness for GwTS



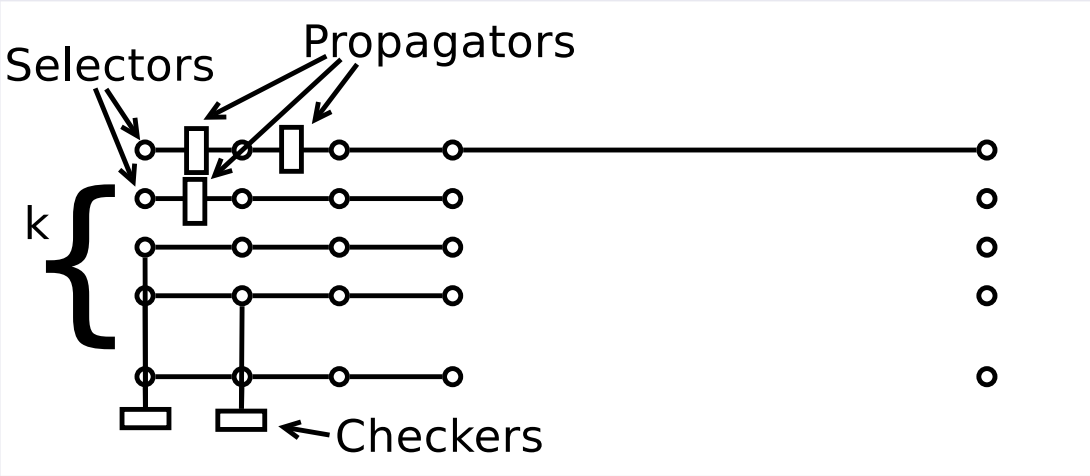
- $k \times m$ "grid" where each row represents a color class
- Selector gadget: has n "reasonable" Grundy colorings. Each encodes a selection of a vertex in original k-MCC instance.
- Propagator gadget: makes sure consecutive selectors encode same vertex.

Outline of hardness for GwTS



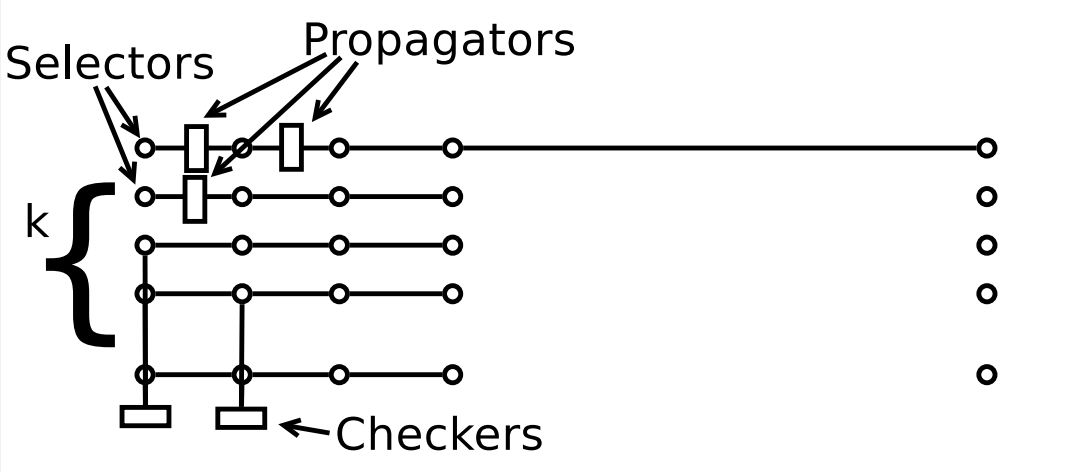
- $k \times m$ "grid" where each row represents a color class
- Selector gadget: has n "reasonable" Grundy colorings. Each encodes a selection of a vertex in original k-MCC instance.
- Propagator gadget: makes sure consecutive selectors encode same vertex.
- Checker gadget: one for each edge of G. Connected to two selectors, is activated if we encode the endpoints of this edge.

Outline of hardness for GwTS



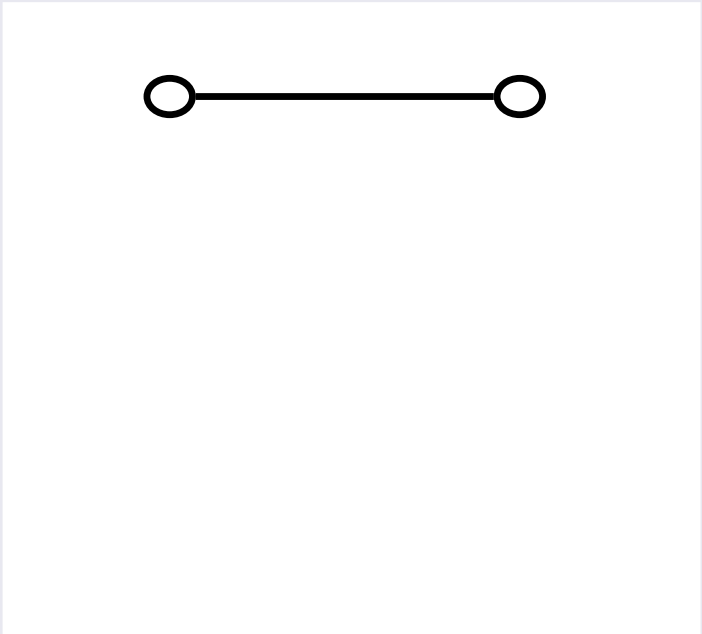
- $k \times m$ "grid" where each row represents a color class
- Selector gadget: has n "reasonable" Grundy colorings. Each encodes a selection of a vertex in original k-MCC instance.
- Propagator gadget: makes sure consecutive selectors encode same vertex.
- Checker gadget: one for each edge of G. Connected to two selectors, is activated if we encode the endpoints of this edge.
- Goal: activate $\binom{k}{2}$ checkers.

Outline of hardness for GwTS



- $k \times m$ "grid" where each row represents a color class
- Selector gadget: has n "reasonable" Grundy colorings. Each encodes a selection of a vertex in original k-MCC instance.
- Propagator gadget: makes sure consecutive selectors encode same vertex.
- Checker gadget: one for each edge of G. Connected to two selectors, is activated if we encode the endpoints of this edge.
- Goal: activate $\binom{k}{2}$ checkers.
- Main difficulty: selectors and propagators

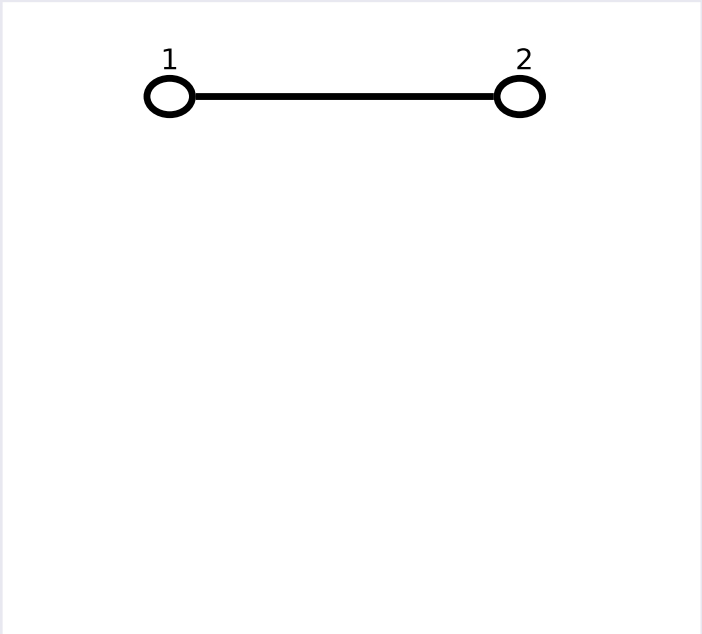
Selector Gadget



Intuition:

- We construct $\log n$ independent edges, numbered $1 \ldots \log n$.
- Endpoints of edge i get support $[1 \ldots 2i-2]$.
- $\rightarrow$ they can be colored with $2i-1, 2i$.
- For each edge we have a choice to put the larger color left or right.
- $2^{\log n} = n$ choices can be encoded.

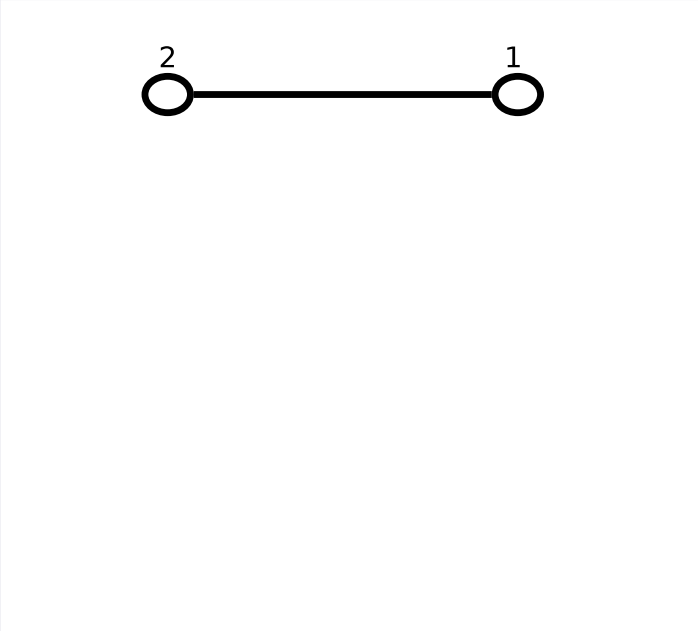
Selector Gadget



Intuition:

- We construct $\log n$ independent edges, numbered $1 \ldots \log n$.
- Endpoints of edge i get support $[1 \ldots 2i-2]$.
- $\rightarrow$ they can be colored with $2i-1, 2i$.
- For each edge we have a choice to put the larger color left or right.
- $2^{\log n} = n$ choices can be encoded.

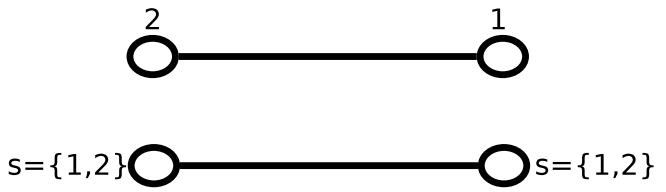
Selector Gadget



Intuition:

- We construct $\log n$ independent edges, numbered $1 \ldots \log n$.
- Endpoints of edge i get support $[1 \ldots 2i-2]$.
- $\rightarrow$ they can be colored with $2i-1, 2i$.
- For each edge we have a choice to put the larger color left or right.
- $2^{\log n} = n$ choices can be encoded.

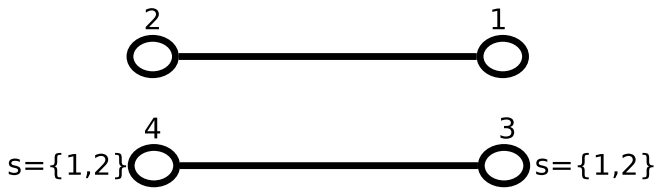
Selector Gadget



Intuition:

- We construct $\log n$ independent edges, numbered $1 \dots \log n$.
- Endpoints of edge i get support $[1 \dots 2i-2]$.
- $\rightarrow$ they can be colored with $2i-1, 2i$.
- For each edge we have a choice to put the larger color left or right.
- $2^{\log n} = n$ choices can be encoded.

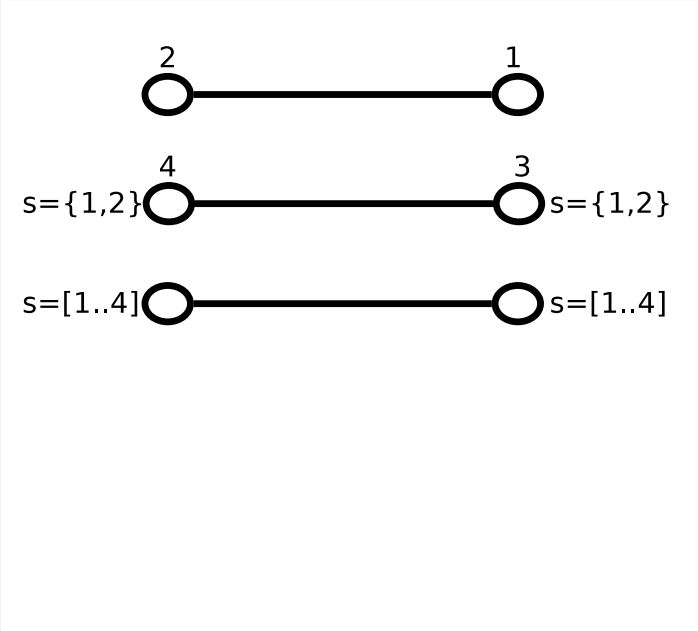
Selector Gadget



Intuition:

- We construct $\log n$ independent edges, numbered $1 \ldots \log n$.
- Endpoints of edge i get support $[1 \ldots 2i-2]$.
- $\rightarrow$ they can be colored with $2i-1, 2i$.
- For each edge we have a choice to put the larger color left or right.
- $2^{\log n} = n$ choices can be encoded.

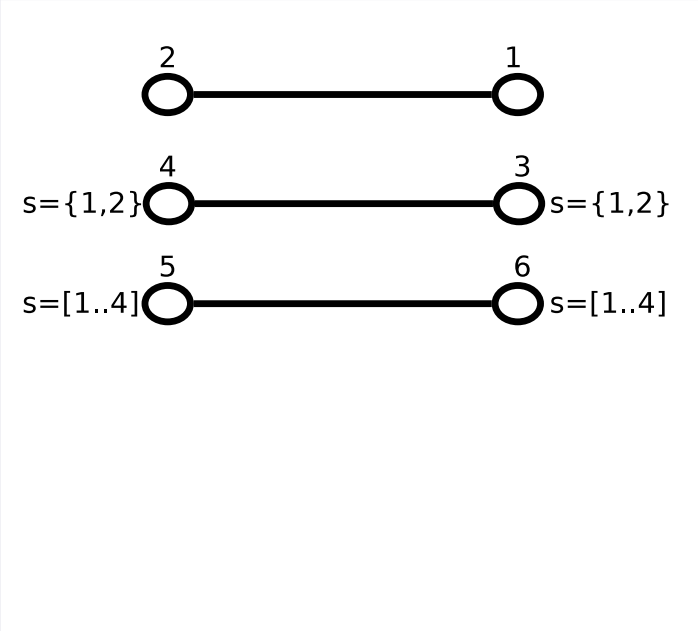
Selector Gadget



Intuition:

- We construct $\log n$ independent edges, numbered $1 \ldots \log n$.
- Endpoints of edge i get support $[1 \ldots 2i-2]$.
- $\rightarrow$ they can be colored with $2i-1, 2i$.
- For each edge we have a choice to put the larger color left or right.
- $2^{\log n} = n$ choices can be encoded.

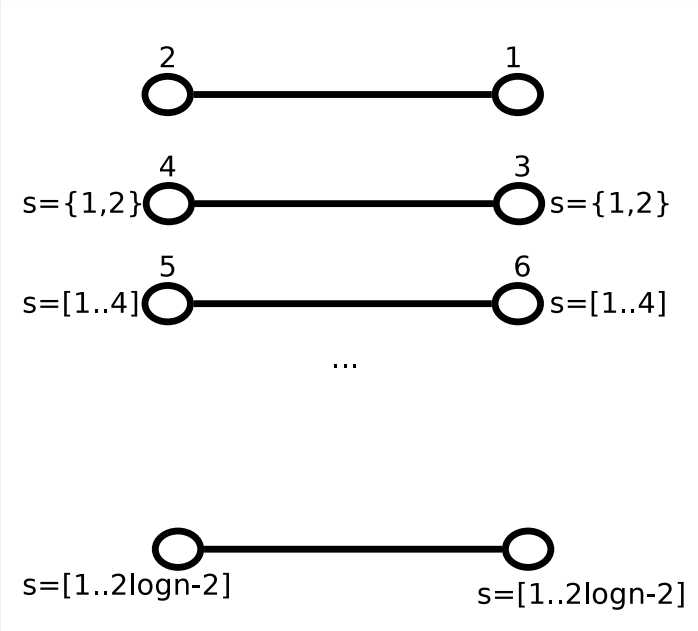
Selector Gadget



Intuition:

- We construct $\log n$ independent edges, numbered $1 \dots \log n$.
- Endpoints of edge i get support $[1 \dots 2i-2]$.
- $\rightarrow$ they can be colored with $2i-1, 2i$.
- For each edge we have a choice to put the larger color left or right.
- $2^{\log n} = n$ choices can be encoded.

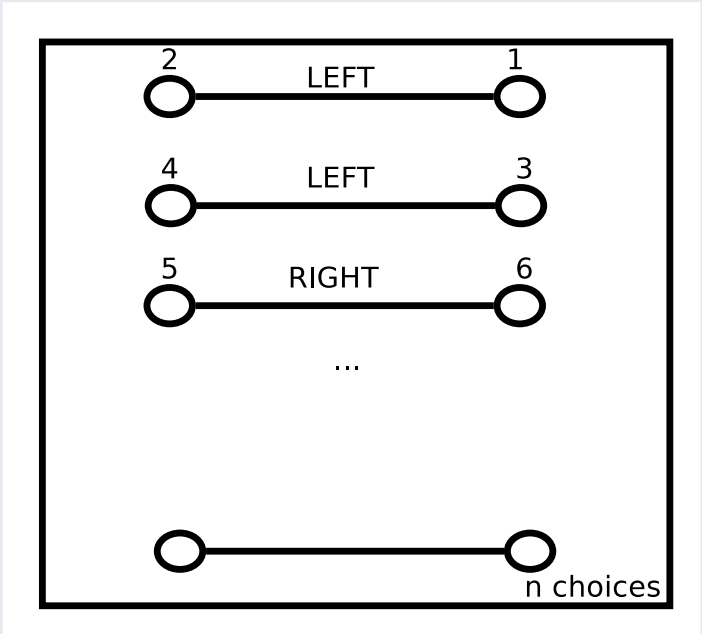
Selector Gadget



Intuition:

- We construct $\log n$ independent edges, numbered $1 \ldots \log n$.
- Endpoints of edge i get support $[1 \ldots 2i-2]$.
- $\rightarrow$ they can be colored with $2i-1, 2i$.
- For each edge we have a choice to put the larger color left or right.
- $2^{\log n} = n$ choices can be encoded.

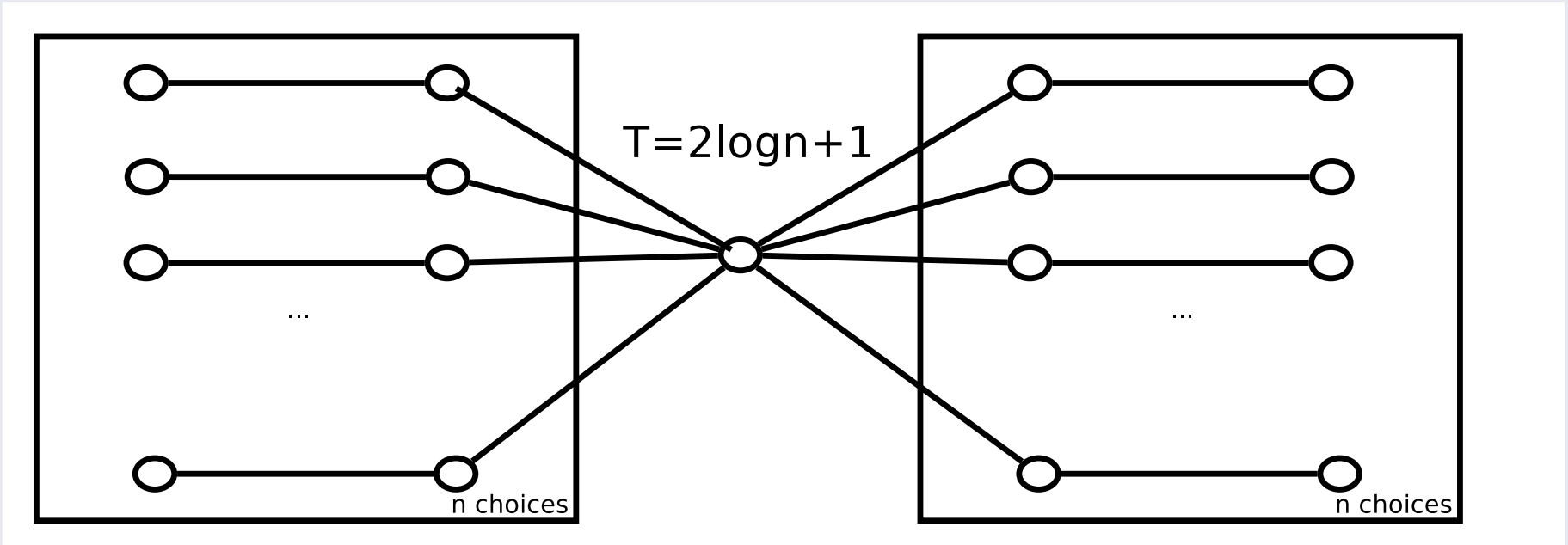
Selector Gadget



Intuition:

- We construct $\log n$ independent edges, numbered $1 \ldots \log n$.
- Endpoints of edge i get support $[1 \ldots 2i-2]$.
- $\rightarrow$ they can be colored with $2i-1, 2i$.
- For each edge we have a choice to put the larger color left or right.
- $2^{\log n} = n$ choices can be encoded.

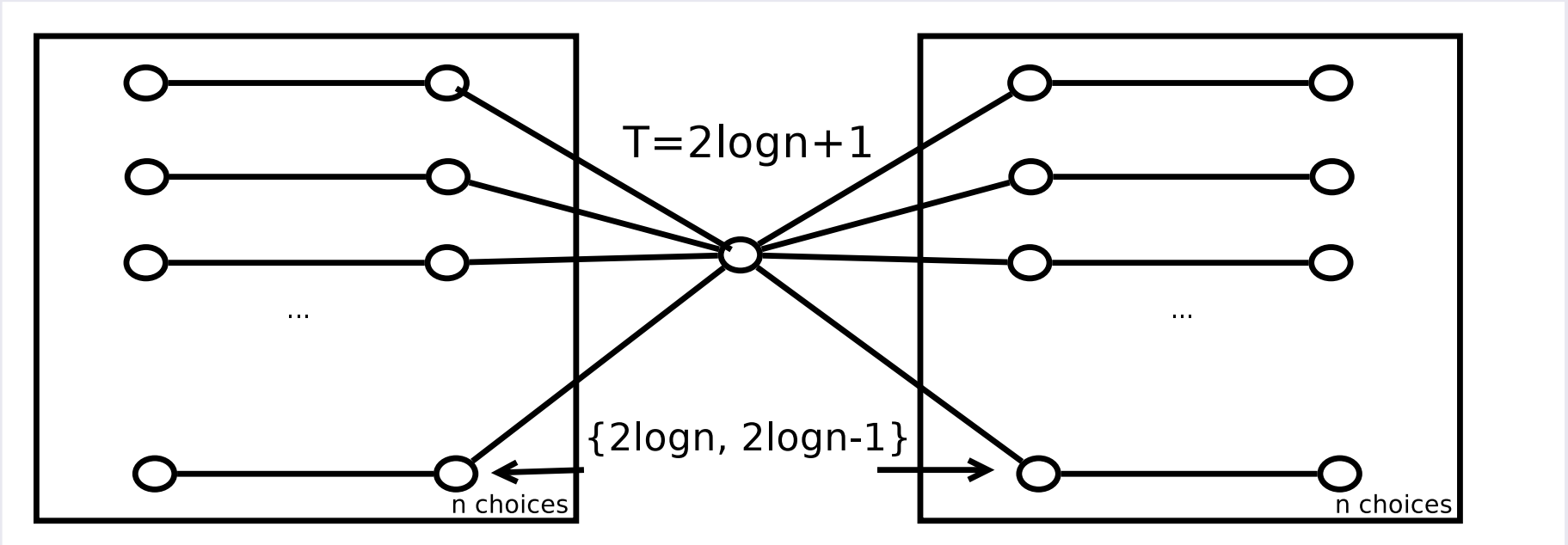
Propagator Gadget



Intuition:

- A propagator is a vertex with target $2 \log n + 1$ connected to different sides of consecutive selectors.
- Its neighborhood must cover all colors in $\{1, \ldots, 2 \log n\}$.
- For each (starting from largest) colors $2i - 1, 2i$ can only be found on i-th edge.
- Therefore, assignment must remain consistent.

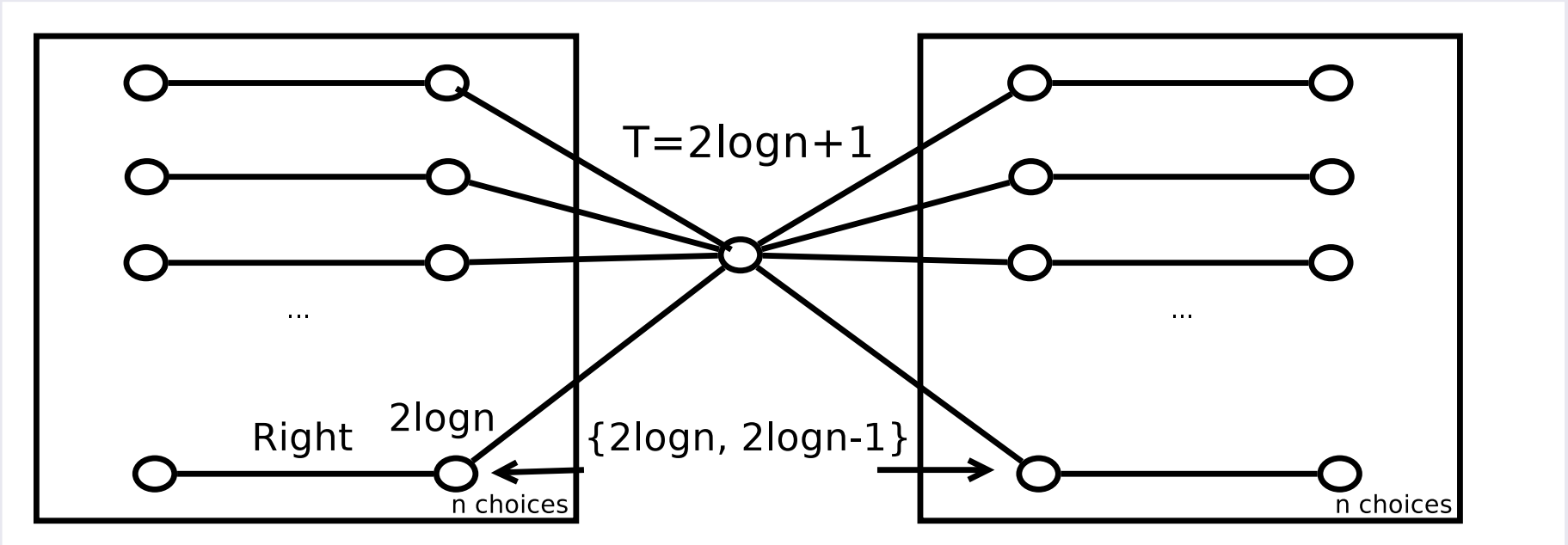
Propagator Gadget



Intuition:

- A propagator is a vertex with target $2 \log n + 1$ connected to different sides of consecutive selectors.
- Its neighborhood must cover all colors in $\{1, \ldots, 2 \log n\}$.
- For each (starting from largest) colors $2i - 1, 2i$ can only be found on i-th edge.
- Therefore, assignment must remain consistent.

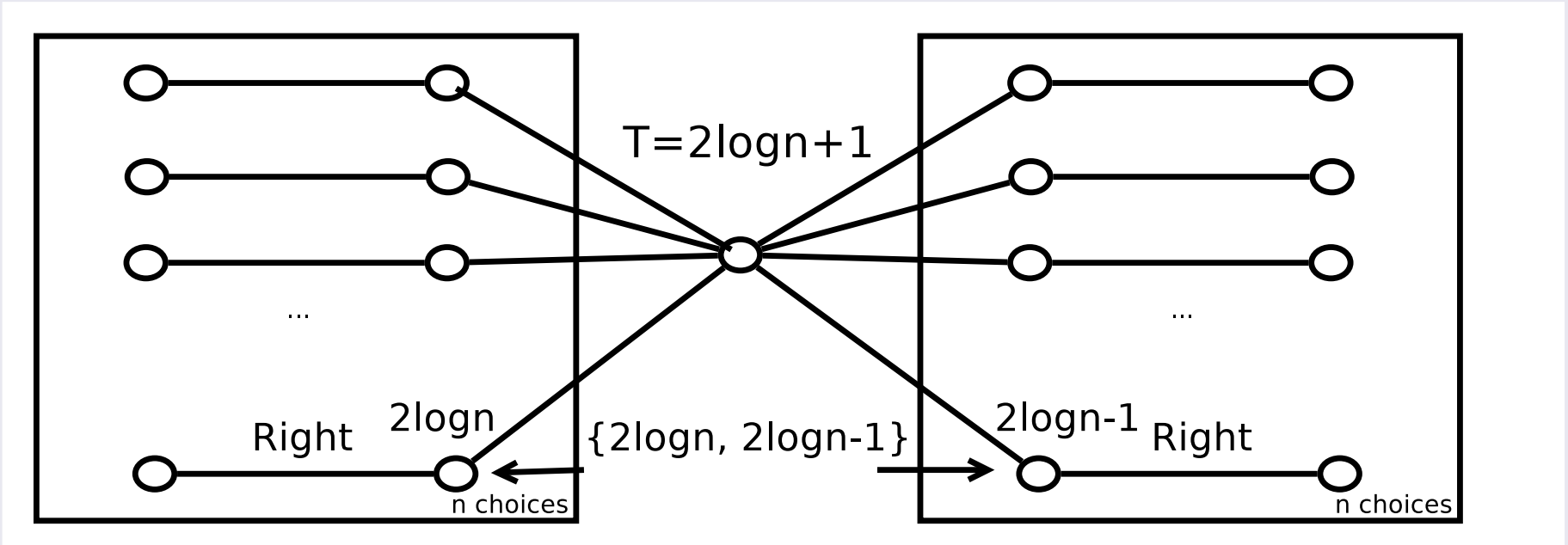
Propagator Gadget



Intuition:

- A propagator is a vertex with target $2 \log n + 1$ connected to different sides of consecutive selectors.
- Its neighborhood must cover all colors in $\{1, \ldots, 2 \log n\}$.
- For each (starting from largest) colors $2i - 1, 2i$ can only be found on i-th edge.
- Therefore, assignment must remain consistent.

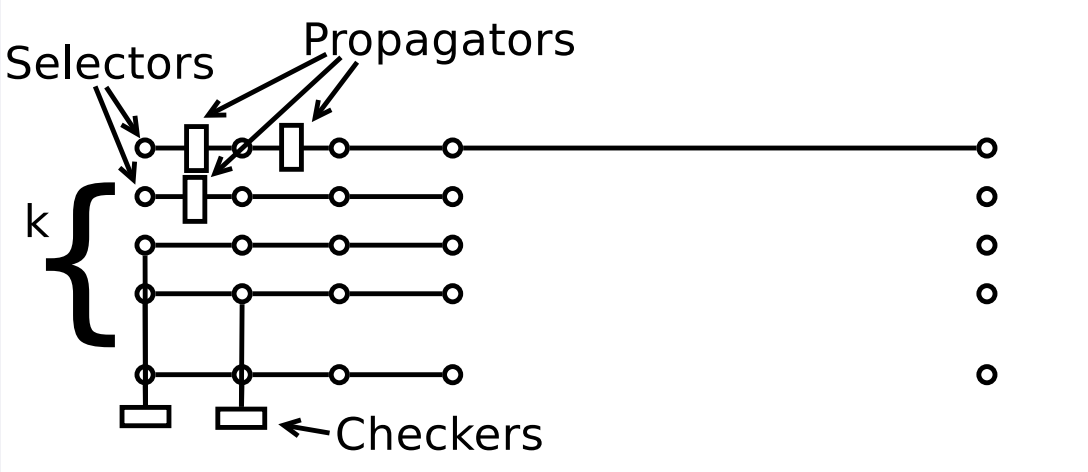
Propagator Gadget



Intuition:

- A propagator is a vertex with target $2 \log n + 1$ connected to different sides of consecutive selectors.
- Its neighborhood must cover all colors in $\{1, \ldots, 2 \log n\}$.
- For each (starting from largest) colors $2i - 1, 2i$ can only be found on i-th edge.
- Therefore, assignment must remain consistent.

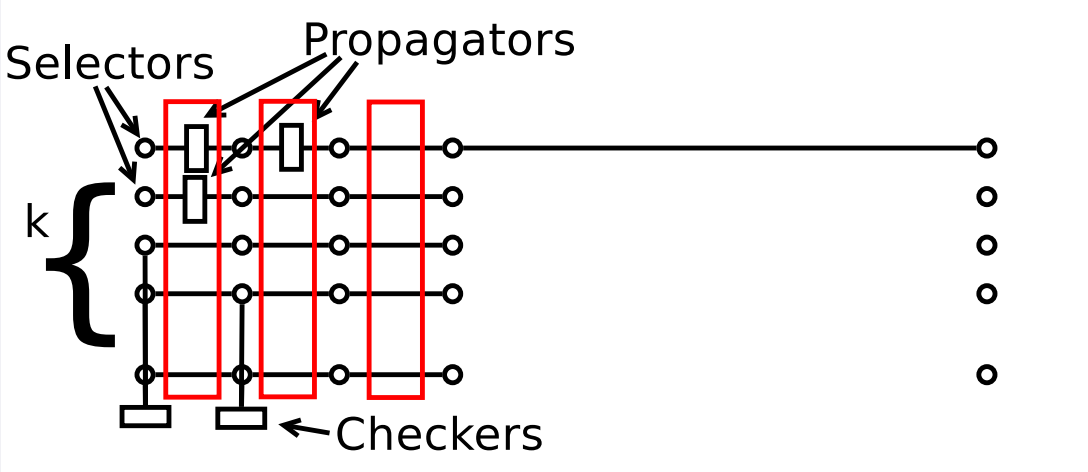
We're on the right track!



How is this reduction going?

- Graph will have pathwidth $\approx k$
  - Propagators are vertices, form separators, bags of decomposition
- Information encoded?
  - Bottleneck of DP: must remember set of colors seen
  - Encoding of selection: set of colors seen by propagator to its left
  - Makes sense!

We're on the right track!



How is this reduction going?

- Graph will have pathwidth $\approx k$
  - Propagators are vertices, form separators, bags of decomposition
- Information encoded?
  - Bottleneck of DP: must remember set of colors seen
  - Encoding of selection: set of colors seen by propagator to its left
  - Makes sense!

Regular Grundy

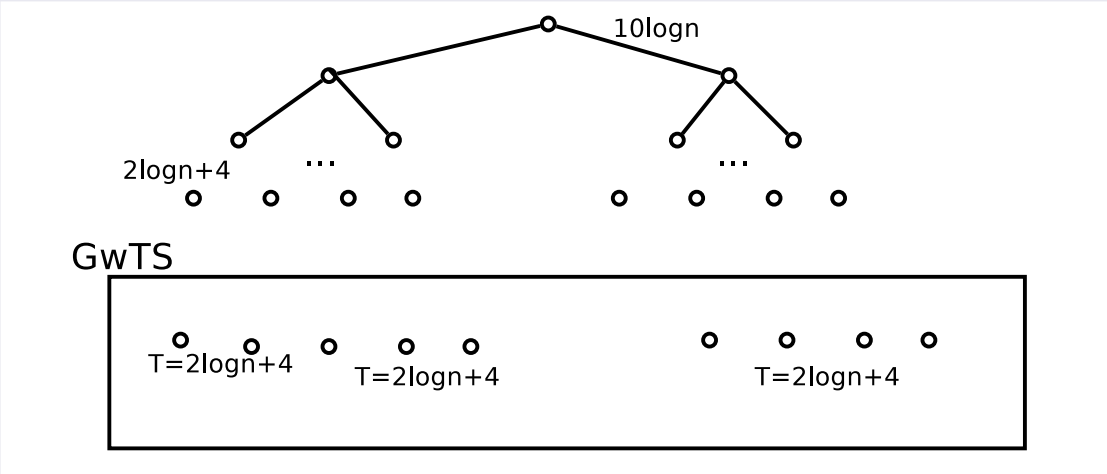
- To implement supports we attach binomial trees to supported vertices.
  - Does not increase treewidth.
  - Crucial: all supports are $O(\log n)$, so binomial trees have polynomial size.
- To implement targets we add a huge binomial tree $T_{10 \log n}$.
- For each vertex with target $\leq 2 \log n + 4$ we find an internal vertex of the tree that is supposed to take the same color and merge them.
- Must be done carefully to keep treewidth low!

Regular Grundy

- To implement supports we attach binomial trees to supported vertices.
  - Does not increase treewidth.
  - Crucial: all supports are $O(\log n)$, so binomial trees have polynomial size.
- To implement targets we add a huge binomial tree $T_{10 \log n}$.
- For each vertex with target $\leq 2 \log n + 4$ we find an internal vertex of the tree that is supposed to take the same color and merge them.
- Must be done carefully to keep treewidth low!

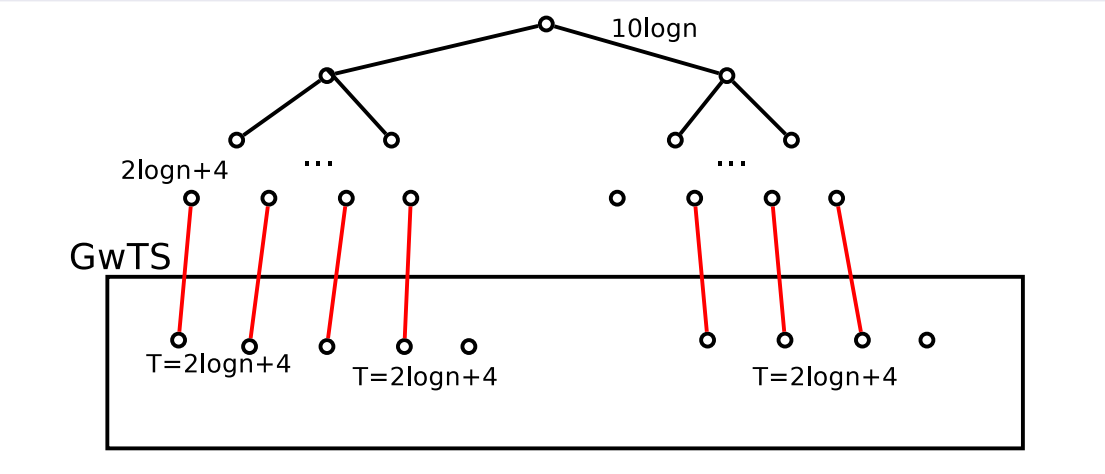
Regular Grundy

- To implement supports we attach binomial trees to supported vertices.
  - Does not increase treewidth.
  - Crucial: all supports are $O(\log n)$, so binomial trees have polynomial size.
- To implement targets we add a huge binomial tree $T_{10 \log n}$.
- For each vertex with target $\leq 2 \log n + 4$ we find an internal vertex of the tree that is supposed to take the same color and merge them.
- Must be done carefully to keep treewidth low!



Regular Grundy

- To implement supports we attach binomial trees to supported vertices.
  - Does not increase treewidth.
  - Crucial: all supports are $O(\log n)$, so binomial trees have polynomial size.
- To implement targets we add a huge binomial tree $T_{10 \log n}$.
- For each vertex with target $\leq 2 \log n + 4$ we find an internal vertex of the tree that is supposed to take the same color and merge them.
- Must be done carefully to keep treewidth low!

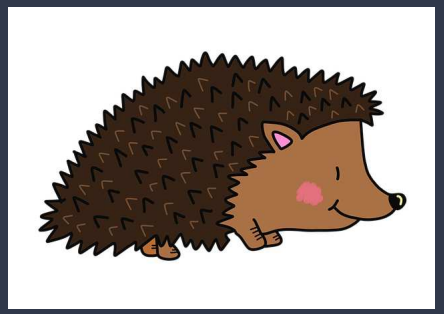


Interesting Trick:
Graph of pathwidth k + Tree
$\Rightarrow$
Graph of treewidth k

Summary

- Grundy is W[1]-hard by treewidth
- Reduction shows Grundy with Targets and Supports is W[1]-hard by pathwidth!
- Key reason why this doesn't work for regular Grundy: we need binomial trees
- Binomial trees have large pathwidth ($\Theta(\log n)$)

FPT for pathwidth



Cmbinatorics to Algorithms

Two ingredients for FPT algorithm by pathwidth:

- DP algorithm running in $2^{k \cdot tw}$ we saw
- A combinatorial bound: for all G, $\Gamma(G) \leq 8pw(G)$
  - Shown in [Dujmovic, Joret, Wood SIDMA'12]
  - Uses connection pathwidth$\leftrightarrow$interval graphs

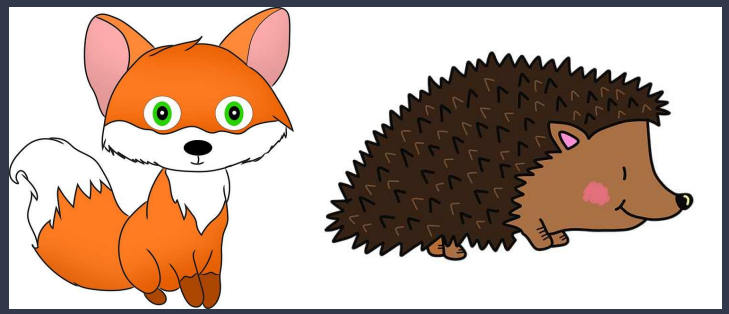
Cmbinatorics to Algorithms

Two ingredients for FPT algorithm by pathwidth:

- DP algorithm running in $2^{k \cdot tw}$ we saw
- A combinatorial bound: for all G, $\Gamma(G) \leq 8pw(G)$
  - Shown in [Dujmovic, Joret, Wood SIDMA'12]
  - Uses connection pathwidth$\leftrightarrow$interval graphs
- Plugging in the bound and using $tw \leq pw$ we get

Thm: Grundy Coloring can be solved in $O^*(2^{O(pw^2)})$

Conclusions



Conclusions – Open Questions

- Grundy Coloring is first (?) natural problem to be FPT for pathwidth, W-hard for treewidth

Open questions:

- Other such problems separating tw/pw?
- Problems separatings them for other reasons?
- FPT by fvs?
- Gap between $n^{o(\sqrt{tw})}$ LB and n^{tw^2} algorithm?

Thank you!

Thank you!
Questions?