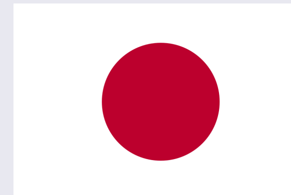
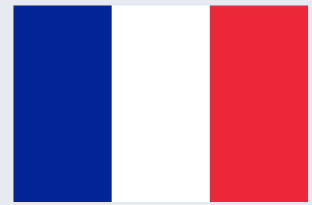
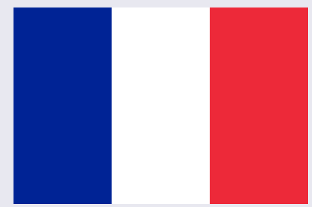
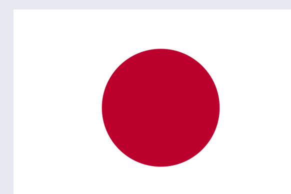
Grundy Distinguishes Treewidth from Pathwidth

Michael Lampis
LAMSADE
Université Paris Dauphine

Sep 7th 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. Work to appear in ESA 2020.

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

What is this talk about?

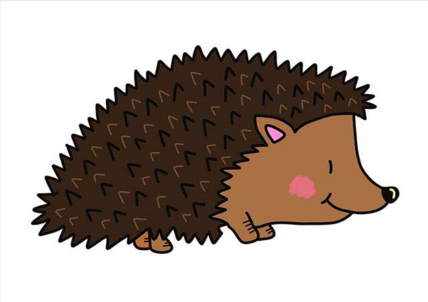
Two ways to look at this work

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?

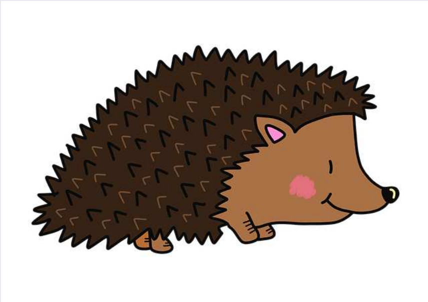
Two ways to look at this work

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

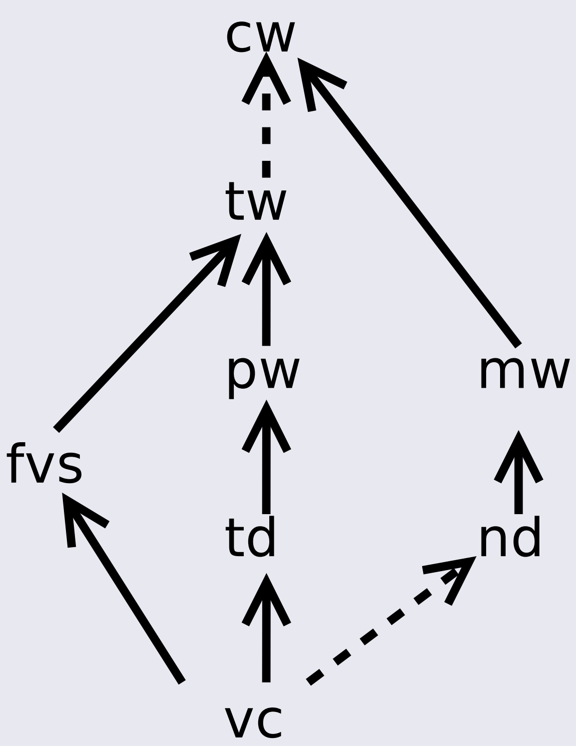
- Well-known optimization problem
- MaxMin variant of Coloring
    - Find a proper coloring that uses the **max** number of colors but the color of no vertex can be decreased.



"The fox knows many things, but the hedgehog knows one big thing", Aesop's fables

What does the fox say?

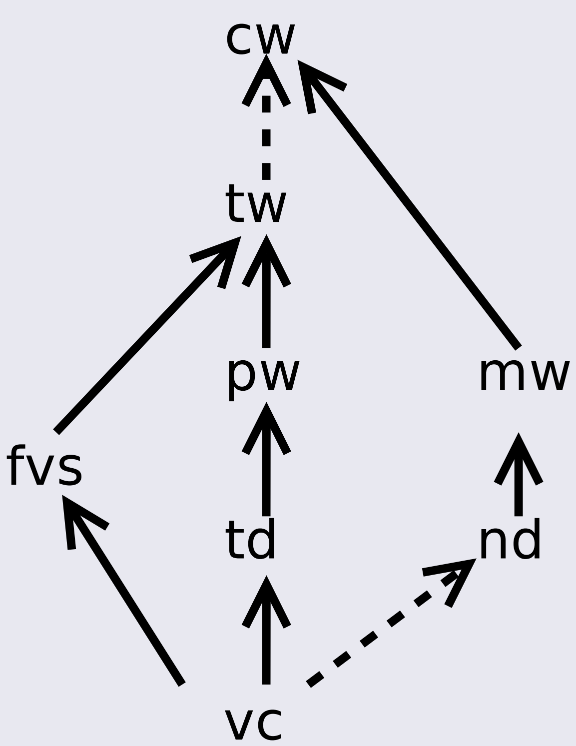
Structural Parameters



- We use a structural parameter w to measure how "easy" a graph is. Examples:
  - Treewidth w
  - Clique-width w
  - Forest+w vertices
  - Independent set+w vertices
- Arrows indicate "inclusion".
  - E.g. graphs of pathwidth k, also have treewidth $\leq k$.

- We want to measure the complexity as function of input structure.
- More general width $\rightarrow$ Larger class of instances for each w $\rightarrow$
  - More generality (good!)
  - Problems become more intractable (bad!)

Price of Generality

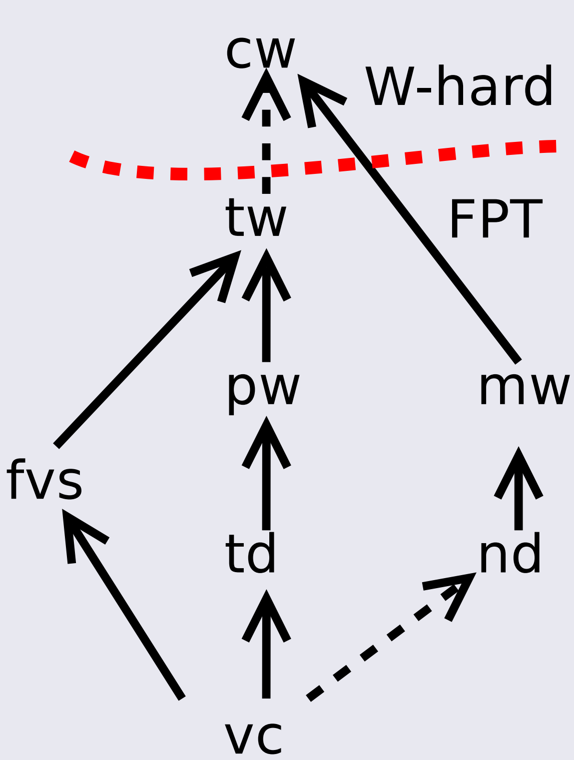


Each problem/parameter pair is typically either:

- 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

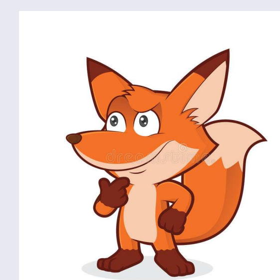


Each problem/parameter pair is typically either:

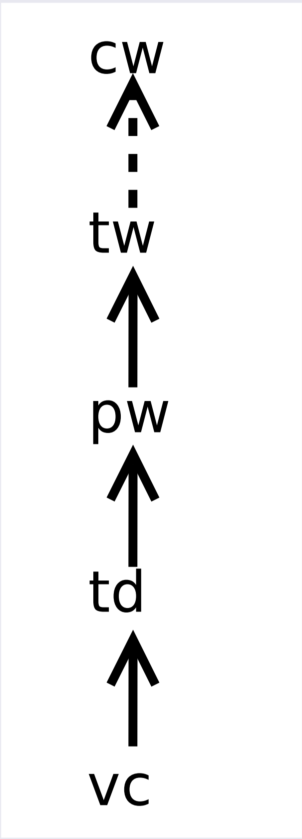
- FPT: solvable in $f(w)n^{O(1)}$
- XP and W-hard: solvable in $n^{g(w)}$, not FPT
- paraNP-hard: NP-hard for $w = O(1)$

- 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.



Price of Generality Continued

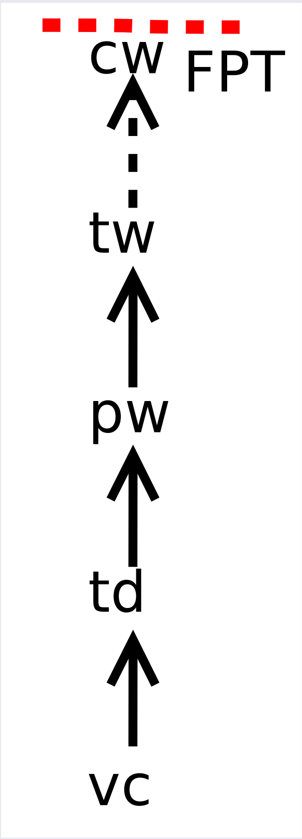


Comments

Price of Generality Examples

Clique-width	
Treewidth	
Pathwidth	
Tree-depth	
Vertex Cover	

Price of Generality Continued

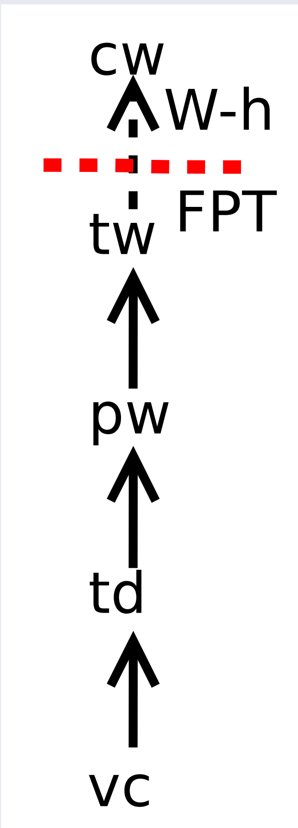


Price of Generality Examples

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

Comments

Price of Generality Continued



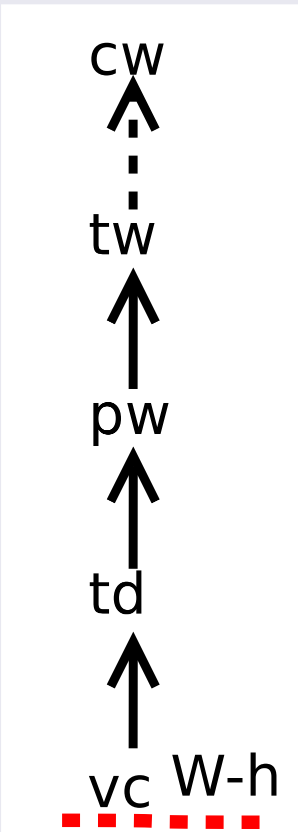
Price of Generality Examples

	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



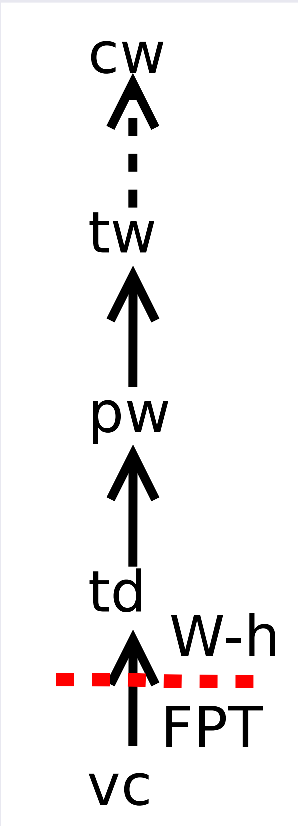
Price of Generality Examples

	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



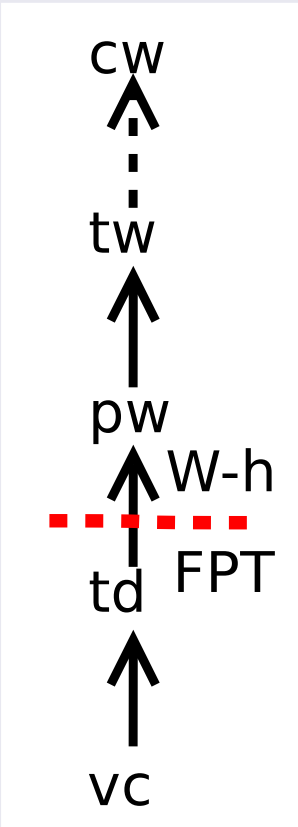
Price of Generality Examples

	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

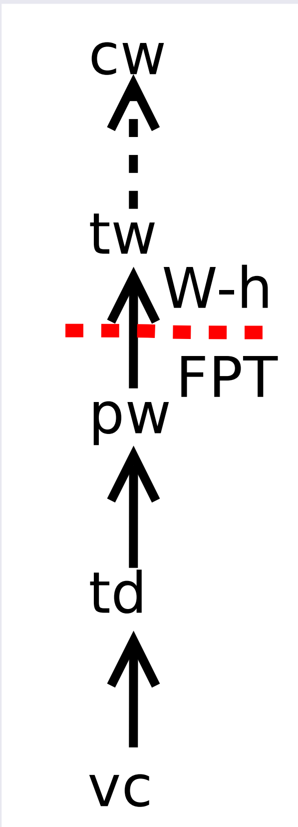


Price of Generality Examples

	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.

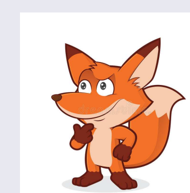


Price of Generality Examples

	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??



Between Treewidth and Pathwidth

We are looking for a **natural** problem which is

- FPT for pathwidth
- W[1]-hard for treewidth

Between Treewidth and Pathwidth

We are looking for a **natural** problem which is

- FPT for pathwidth
- W[1]-hard for treewidth

- “artificial” problem may be easy to construct, not so interesting
- Natural: “has been defined in a previous paper” (per M. Wahlström)

Between Treewidth and Pathwidth

We are looking for a **natural** problem which is

- FPT for pathwidth
- W[1]-hard for treewidth

- "artificial" problem may be easy to construct, not so interesting
- Natural: "has been defined in a previous paper" (per M. Wahlström)

- Is no such problem known?
  - In full paper we survey dozens of problems W-hard by treewidth
  - (Nice compendium for future reference!)
  - Most are W-hard for tree-depth
  - **All** are W-hard for pathwidth!!

Between Treewidth and Pathwidth

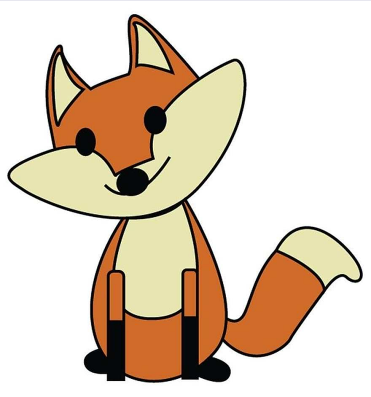
We are looking for a **natural** problem which is

- FPT for pathwidth
- W[1]-hard for treewidth

- “artificial” problem may be easy to construct, not so interesting
- Natural: “has been defined in a previous paper” (per M. Wahlström)

- Is no such problem known?
  - In full paper we survey dozens of problems W-hard by treewidth
  - (Nice compendium for future reference!)
  - Most are W-hard for tree-depth
  - **All** are W-hard for pathwidth!!

  Main result of this talk:
- Grundy Coloring is such a problem!



Are you convinced?

- How do we know that no such other problem is already known?

Are you convinced?

- How do we know that no such other problem is already known?
- We don't but…
- `https://cstheory.stackexchange.com/questions/27590/`

Algorithmic advantages of pathwidth over treewidth

Asked 5 years, 9 months ago Active 2 years ago Viewed 374 times

18

5

Treewidth plays an important role in FPT algorithms, in part because many problems are FPT parameterized by treewidth. A related, more restricted, notion is that of pathwidth. If a graph has pathwidth k, it also has treewidth at most k, while in the converse direction, treewidth k only implies pathwidth at most $k \log n$ and this is tight.

Given the above, one may expect that there may be a significant algorithmic advantage to graphs of bounded pathwidth. However, it seems that most problems which are FPT for one parameter are FPT for the other. I'm curious to know of any counter-examples to this, that is, problems which are "easy" for pathwidth but "hard" for treewidth.

Let me mention that I was motivated to ask this question by running into a recent paper by Igor Razgon ("On OBDDs for CNFs of bounded treewidth", KR'14) which gave an example of a problem with a $2^k n$ solution when k is the pathwidth and a (roughly) n^k lower bound when k is the treewidth. I am wondering if there exist other specimens with this behavior.

Summary: Are there any examples of natural problems which are W-hard parameterized by treewidth but FPT parameterized by pathwidth? More broadly, are there examples of problems whose complexity is known/believed to be much better when parameterized by pathwidth instead of treewidth?

parameterized-complexity treewidth graph-minor

share cite edit close delete flag

edited Dec 2 '14 at 20:10
Hermann Gruber
4,919 ●1 ●26 ●51

asked Nov 26 '14 at 14:34
Michael Lampis
2,919 ●17 ●25

Are you convinced?

- How do we know that no such other problem is already known?
- We don't but. . .
- `https://cstheory.stackexchange.com/questions/27590/`

Algorithmic advantages of pathwidth over treewidth

Asked 5 years, 9 months ago Active 2 years ago Viewed 374 times

18

Treewidth plays an important role in FPT algorithms, in part because many problems are FPT parameterized by treewidth. A related, more restricted, notion is that of pathwidth. If a graph has pathwidth k, it also has treewidth at most k, while in the converse direction, treewidth k only implies pathwidth at most $k \log n$ and this is tight.

Given the above, one may expect that there may be a significant algorithmic advantage to graphs of bounded pathwidth. However, it seems that most problems which are FPT for one parameter are FPT for the other. I'm curious to know of any counter-examples to this, that is, problems which are "easy" for pathwidth but "hard" for treewidth.

Let me mention that I was motivated to ask this question by running into a recent paper by Igor Razgon ("On OBDDs for CNFs of bounded treewidth", KR'14) which gave an example of a problem with a $2^k n$ solution when k is the pathwidth and a (roughly) n^k lower bound when k is the treewidth. I am wondering if there exist other specimens with this behavior.

Summary: Are there any examples of natural problems which are W-hard parameterized by treewidth but FPT parameterized by pathwidth? More broadly, are there examples of problems whose complexity is known/believed to be much better when parameterized by pathwidth instead of treewidth?

parameterized-complexity treewidth graph-minor

share cite edit close delete flag

edited Dec 2 '14 at 20:10
Hermann Gruber
4,919 • 1 • 26 • 51

asked Nov 26 '14 at 14:34
Michael Lampis
2,919 • 17 • 25

Algorithmic advantages of pathwidth over treewidth

Asked 5 years, 9 months ago Active 2 years ago Viewed 374 times

DAUPHINE UNIVERSITÉ PARIS

Are you convinced?

- How do we know that no such other problem is already known?
- We don't but...
- `https://cstheory.stackexchange.com/questions/27590/`

Algorithmic advantages of pathwidth over treewidth

Asked 5 years, 9 months ago Active 2 years ago Viewed 374 times

18

5

Treewidth plays an important role in FPT algorithms, in part because many problems are FPT parameterized by treewidth. A related, more restricted, notion is that of pathwidth. If a graph has pathwidth k, it also has treewidth at most k, while in the converse direction, treewidth k only implies pathwidth at most $k \log n$ and this is tight.

Given the above, one may expect that there may be a significant algorithmic advantage to graphs of bounded pathwidth. However, it seems that most problems which are FPT for one parameter are FPT for the other. I'm curious to know of any counter-examples to this, that is, problems which are "easy" for pathwidth but "hard" for treewidth.

Let me mention that I was motivated to ask this question by running into a recent paper by Igor Razgon ("On OBDDs for CNFs of bounded treewidth", KR'14) which gave an example of a problem with a $2^k n$ solution when k is the pathwidth and a (roughly) n^k lower bound when k is the treewidth. I am wondering if there exist other specimens with this behavior.

Summary: Are there any examples of natural problems which are W-hard parameterized by treewidth but FPT parameterized by pathwidth? More broadly, are there examples of problems whose complexity is known/believed to be much better when parameterized by pathwidth instead of treewidth?

parameterized-complexity treewidth graph-minor

share cite edit close delete flag

edited Dec 2 '14 at 20:10
Hermann Gruber
4,919 • 1 • 26 • 51

asked Nov 26 '14 at 14:34
Michael Lampis
2,919 • 17 • 25

Algorithmic advantages of pathwidth over treewidth

Asked 5 years, 9 months ago Active 2 years ago Viewed 374 times

Summary: Are there any examples of natural problems which are W-hard parameterized by treewidth but FPT parameterized by pathwidth? More broadly, are there examples of problems whose complexity is known/believed to be much better when parameterized by pathwidth instead of treewidth?

Are you convinced?

- How do we know that no such other problem is already known?
- We don't but…
- `https://cstheory.stackexchange.com/questions/27590/`

Algorithmic advantages of pathwidth over treewidth

Asked 5 years, 9 months ago Active 2 years ago Viewed 374 times

18

5

Treewidth plays an important role in FPT algorithms, in part because many problems are FPT parameterized by treewidth. A related, more restricted, notion is that of pathwidth. If a graph has pathwidth k, it also has treewidth at most k, while in the converse direction, treewidth k only implies pathwidth at most $k \log n$ and this is tight.

Given the above, one may expect that there may be a significant algorithmic advantage to graphs of bounded pathwidth. However, it seems that most problems which are FPT for one parameter are FPT for the other. I'm curious to know of any counter-examples to this, that is, problems which are "easy" for pathwidth but "hard" for treewidth.

Let me mention that I was motivated to ask this question by running into a recent paper by Igor Razgon ("On OBDDs for CNFs of bounded treewidth", KR'14) which gave an example of a problem with a $2^k n$ solution when k is the pathwidth and a (roughly) n^k lower bound when k is the treewidth. I am wondering if there exist other specimens with this behavior.

Summary: Are there any examples of natural problems which are W-hard parameterized by treewidth but FPT parameterized by pathwidth? More broadly, are there examples of problems whose complexity is known/believed to be much better when parameterized by pathwidth instead of treewidth?

parameterized-complexity treewidth graph-minor

share cite edit close delete flag

edited Dec 2 '14 at 20:10 Hermann Gruber 4,919 1 26 51

asked Nov 26 '14 at 14:34 Michael Lampis 2,919 17 25

Algorithmic advantages of pathwidth over treewidth

Asked 5 years, 9 months ago Active 2 years ago Viewed 374 times

Summary: Are there any examples of natural problems which are W-hard parameterized by treewidth but FPT parameterized by pathwidth? More broadly, are there examples of problems whose complexity is known/believed to be much better when parameterized by pathwidth instead of treewidth?

asked Nov 26 '14 at 14:34 Michael Lampis 2,919 17 25

Are you convinced?

- How do we know that no such other problem is already known?
- We don't but…
- `https://cstheory.stackexchange.com/questions/27590/`

Algorithmic advantages of pathwidth over treewidth

Asked 5 years, 9 months ago Active 2 years ago Viewed 374 times

18

5

Treewidth plays an important role in FPT algorithms, in part because many problems are FPT parameterized by treewidth. A related, more restricted, notion is that of pathwidth. If a graph has pathwidth k, it also has treewidth at most k, while in the converse direction, treewidth k only implies pathwidth at most $k \log n$ and this is tight.

Given the above, one may expect that there may be a significant algorithmic advantage to graphs of bounded pathwidth. However, it seems that most problems which are FPT for one parameter are FPT for the other. I'm curious to know of any counter-examples to this, that is, problems which are "easy" for pathwidth but "hard" for treewidth.

Let me mention that I was motivated to ask this question by running into a recent paper by Igor Razgon ("On OBDDs for CNFs of bounded treewidth", KR'14) which gave an example of a problem with a $2^k n$ solution when k is the pathwidth and a (roughly) n^k lower bound when k is the treewidth. I am wondering if there exist other specimens with this behavior.

Summary: Are there any examples of natural problems which are W-hard parameterized by treewidth but FPT parameterized by pathwidth? More broadly, are there examples of problems whose complexity is known/believed to be much better when parameterized by pathwidth instead of treewidth?

parameterized-complexity treewidth graph-minor

share cite edit close delete flag

edited Dec 2 '14 at 20:10
Hermann Gruber
4,919 • 1 • 26 • 51

asked Nov 26 '14 at 14:34
Michael Lampis
2,919 • 17 • 25

Algorithmic advantages of pathwidth over treewidth

Asked 5 years, 9 months ago Active 2 years ago Viewed 374 times

Summary: Are there any examples of natural problems which are W-hard parameterized by treewidth but FPT parameterized by pathwidth? More broadly, are there examples of problems whose complexity is known/believed to be much better when parameterized by pathwidth instead of treewidth?

asked Nov 26 '14 at 14:34
Michael Lampis
2,919 • 17 • 25

3 The best algorithm for Hamiltonian Cycle parameterized by pathwidth has runtime $(2+\sqrt{2})^{pw}$ (arxiv.org/abs/1211.1506) while the best treewidth one is 4^{tw} (arxiv.org/abs/1103.0534) This is probably just a gap waiting to be closed, though. – daniello Nov 27 '14 at 20:55

Are you convinced?

- How do we know that no such other problem is already known?
- We don't but…
- `https://cstheory.stackexchange.com/questions/27590/`

Algorithmic advantages of pathwidth over treewidth

Asked 5 years, 9 months ago Active 2 years ago Viewed 374 times

18

Treewidth plays an important role in FPT algorithms, in part because many problems are FPT parameterized by treewidth. A related, more restricted, notion is that of pathwidth. If a graph has pathwidth k, it also has treewidth at most k, while in the converse direction, treewidth k only implies pathwidth at most $k \log n$ and this is tight.

Given the above, one may expect that there may be a significant algorithmic advantage to graphs of bounded pathwidth. However, it seems that most problems which are FPT for one parameter are FPT for the other. I'm curious to know of any counter-examples to this, that is, problems which are "easy" for pathwidth but "hard" for treewidth.

Let me mention that I was motivated to ask this question by running into a recent paper by Igor Razgon ("On OBDDs for CNFs of bounded treewidth", KR'14) which gave an example of a problem with a $2^k n$ solution when k is the pathwidth and a (roughly) n^k lower bound when k is the treewidth. I am wondering if there exist other specimens with this behavior.

Summary: Are there any examples of natural problems which are W-hard parameterized by treewidth but FPT parameterized by pathwidth? More broadly, are there examples of problems whose complexity is known/believed to be much better when parameterized by pathwidth instead of treewidth?

parameterized-complexity treewidth graph-minor

share cite edit close delete flag

edited Dec 2 '14 at 20:10
Hermann Gruber
4,919 • 1 • 26 • 51

asked Nov 26 '14 at 14:34
Michael Lampis
2,919 • 17 • 25

Algorithmic advantages of pathwidth over treewidth

Asked 5 years, 9 months ago Active 2 years ago Viewed 374 times

Summary: Are there any examples of natural problems which are W-hard parameterized by treewidth but FPT parameterized by pathwidth? More broadly, are there examples of problems whose complexity is known/believed to be much better when parameterized by pathwidth instead of treewidth?

asked Nov 26 '14 at 14:34
Michael Lampis
2,919 • 17 • 25

3 The best algorithm for Hamiltonian Cycle parameterized by pathwidth has runtime $(2+\sqrt{2})^{pw}$ (arxiv.org/abs/1211.1506) while the best treewidth one is 4^{tw} (arxiv.org/abs/1103.0534) This is probably just a gap waiting to be closed, though. – daniello Nov 27 '14 at 20:55

- Grundy Coloring seems to be the first problem of this type!
- Why don't we know any others??

DAUPHINE UNIVERSITÉ PARIS

Let’s recall some basics

Treewidth – Pathwidth

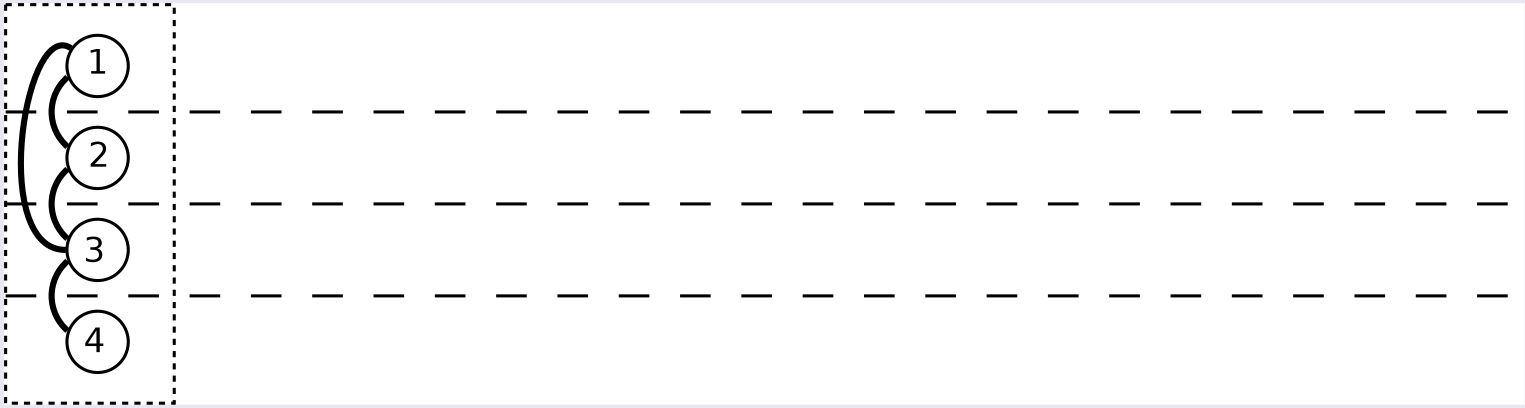
Gentle definition of pathwidth k:

- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.

Treewidth – Pathwidth

Gentle definition of pathwidth k:

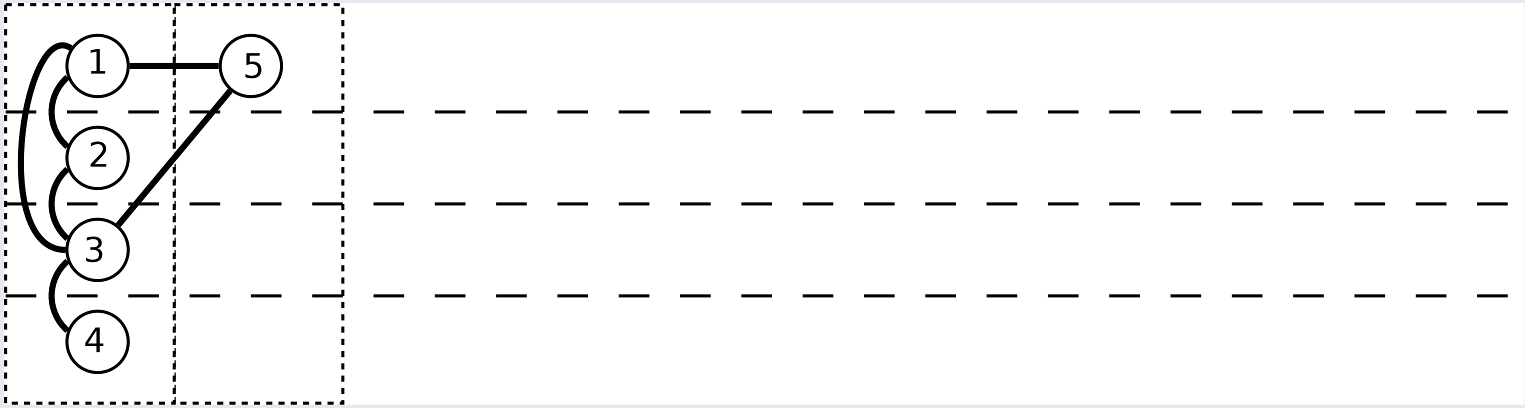
- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



Treewidth – Pathwidth

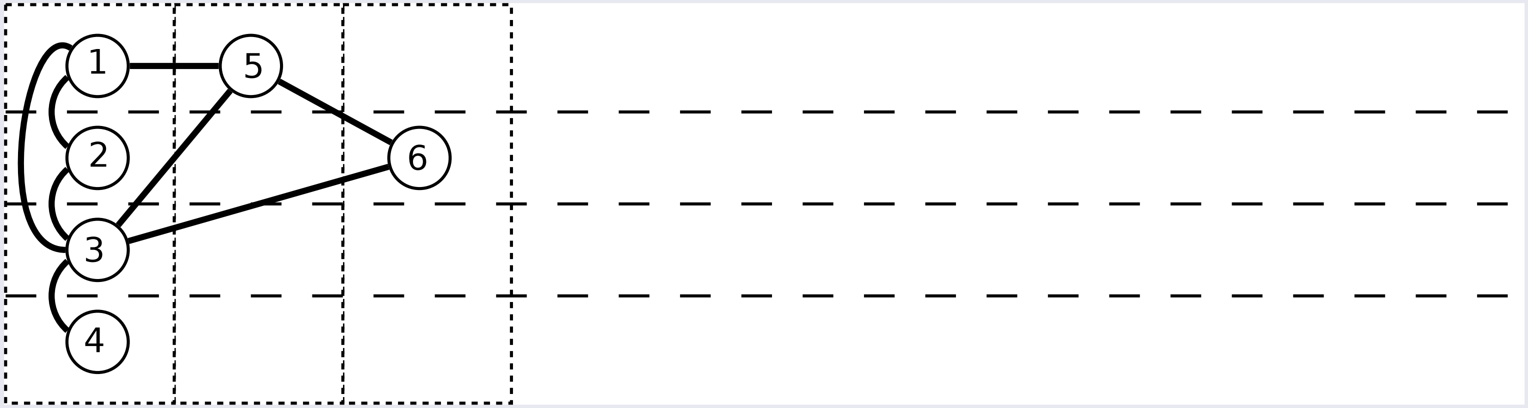
Gentle definition of pathwidth k:

- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



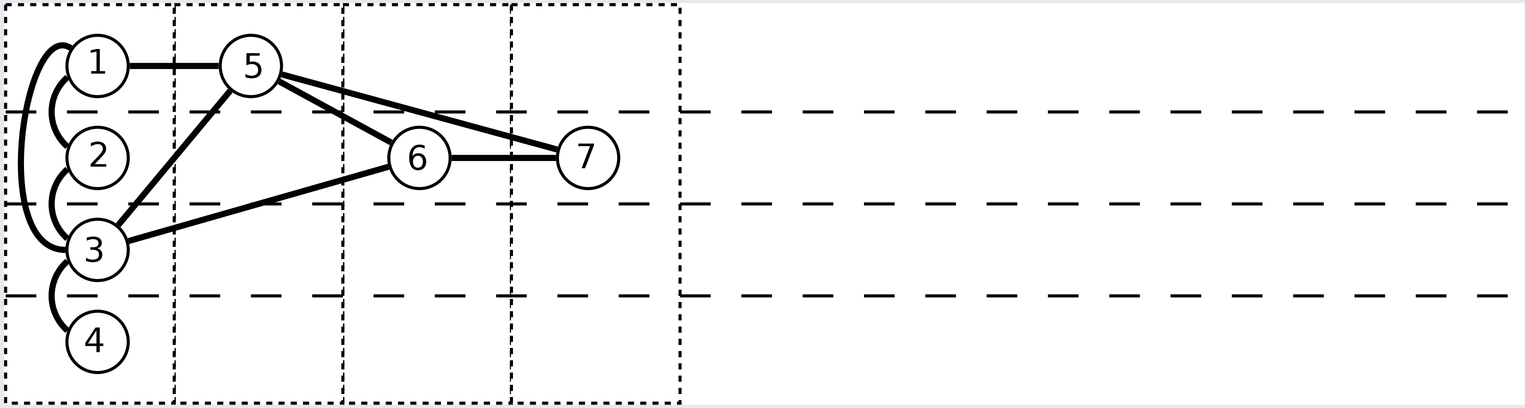
Gentle definition of pathwidth k:

- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



Gentle definition of pathwidth k:

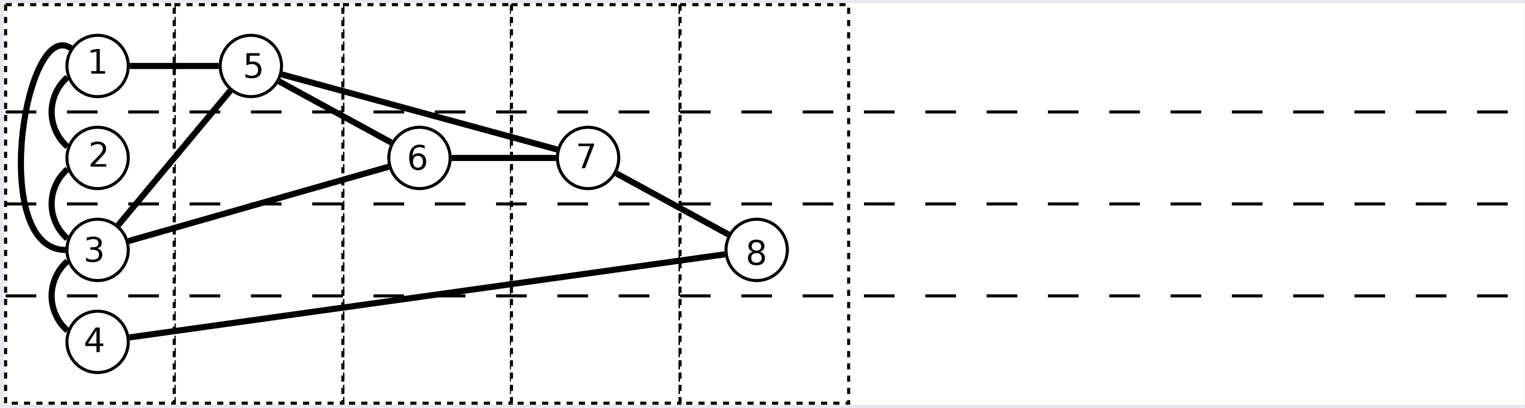
- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



Treewidth – Pathwidth

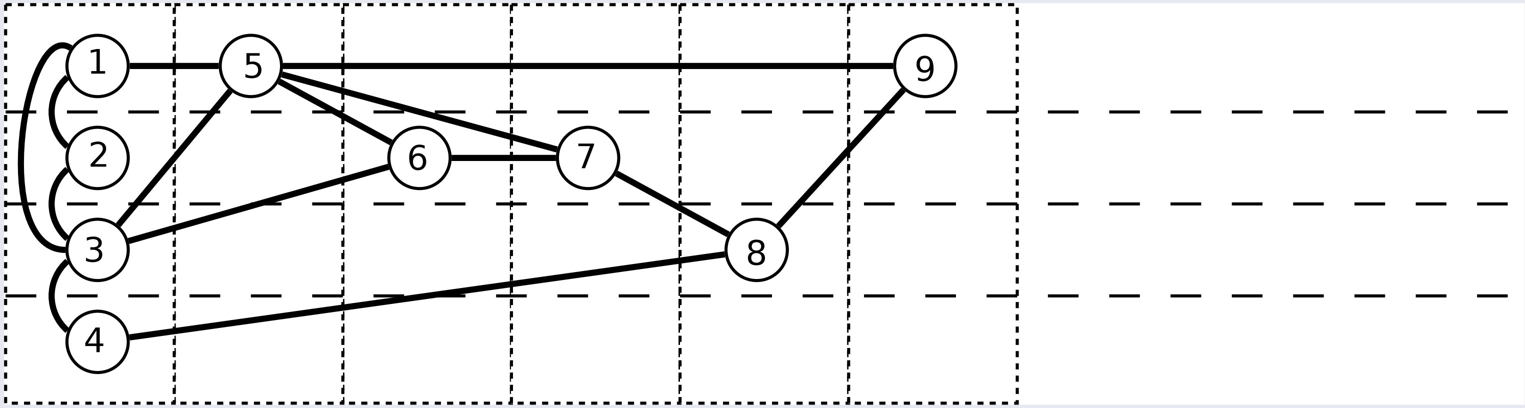
Gentle definition of pathwidth k:

- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



Gentle definition of pathwidth k:

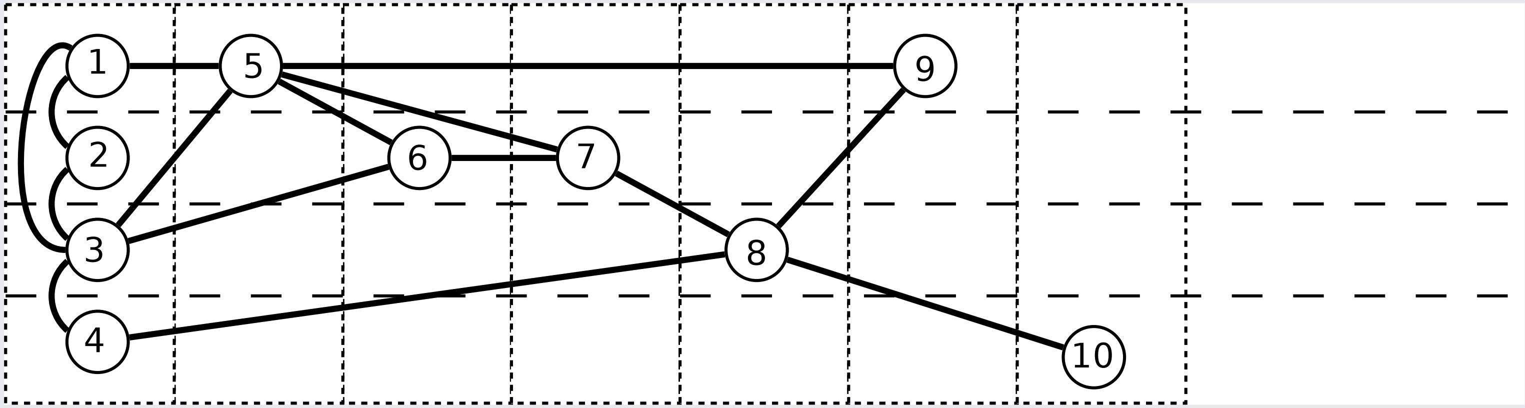
- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



Treewidth – Pathwidth

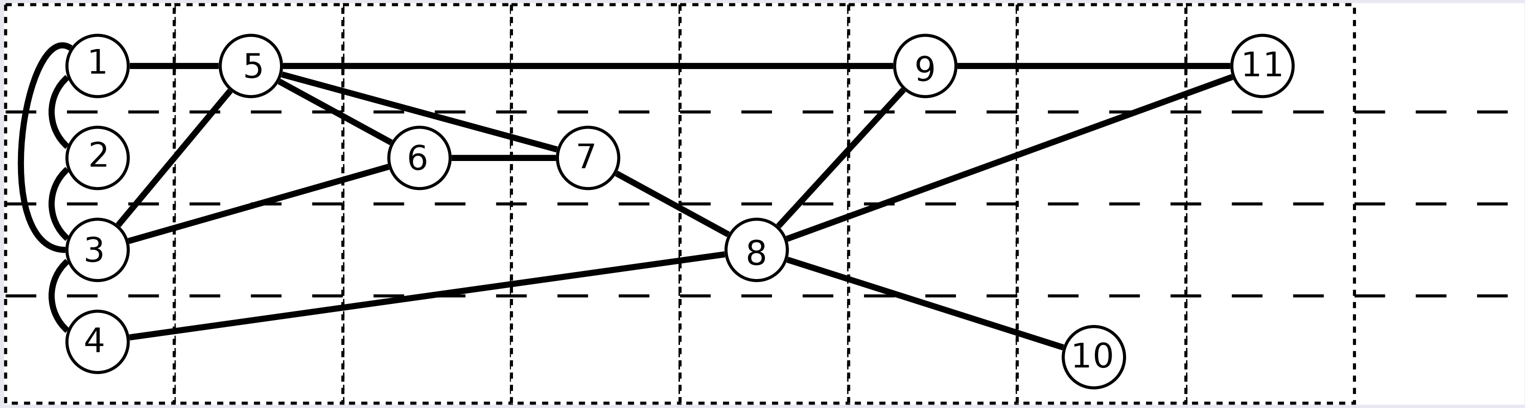
Gentle definition of pathwidth k:

- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



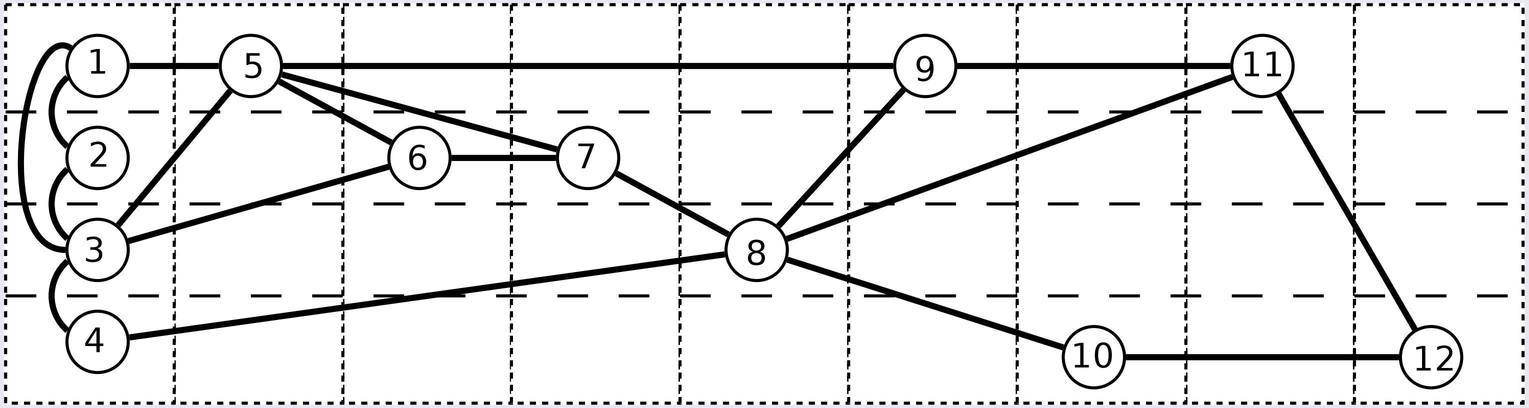
Gentle definition of pathwidth k:

- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



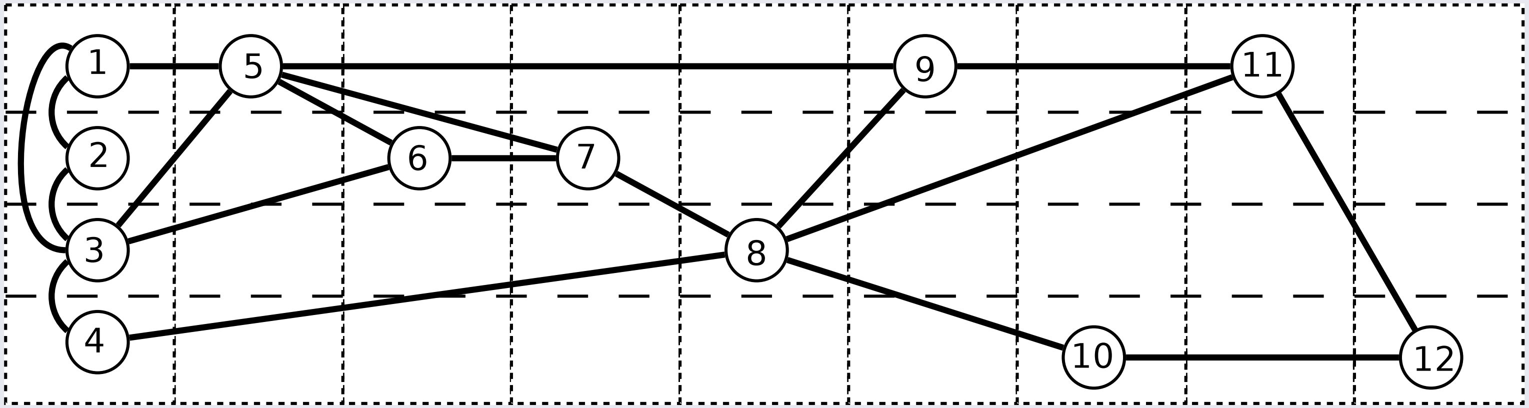
Gentle definition of pathwidth k:

- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



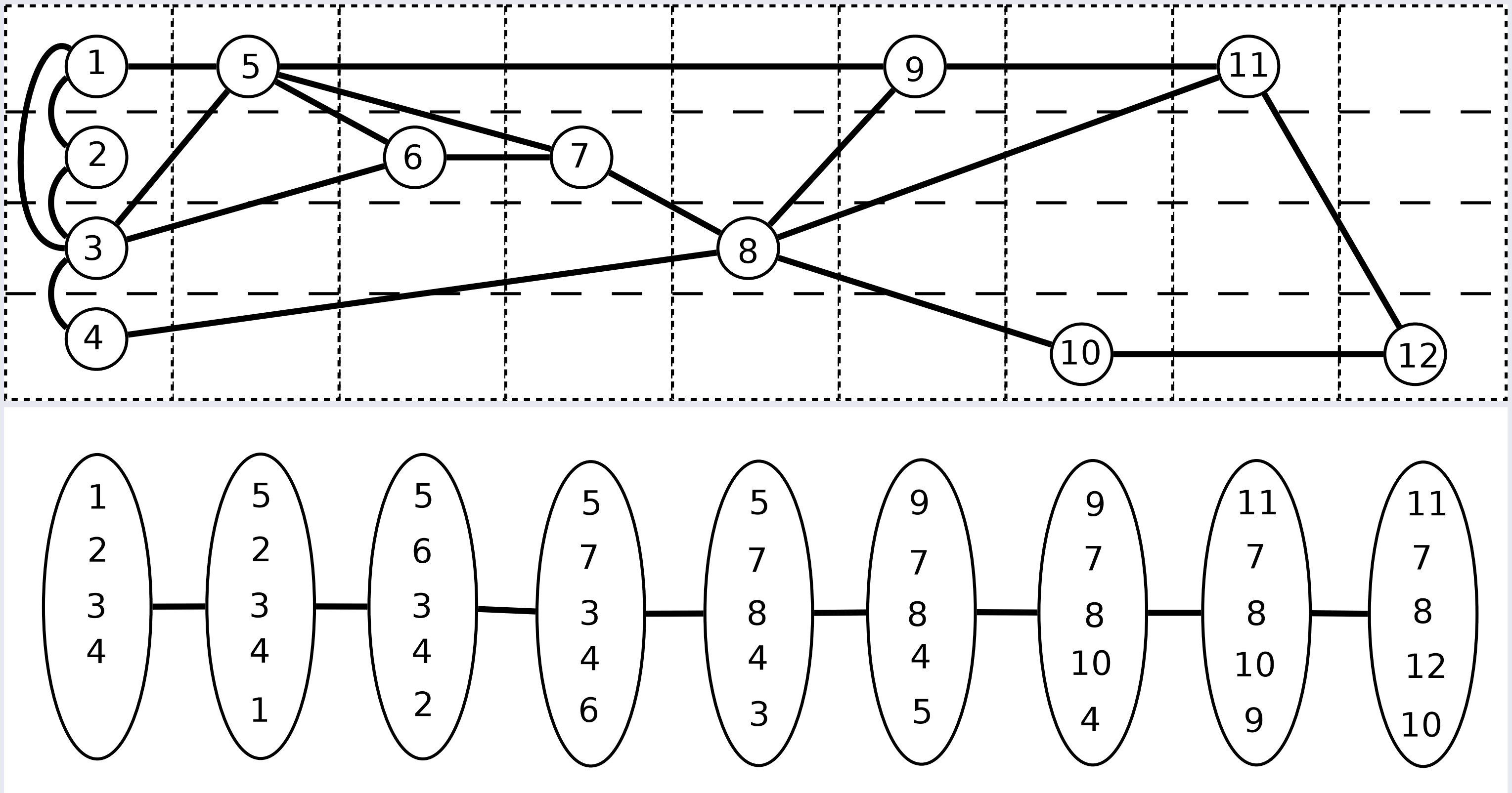
Treewidth – Pathwidth

Note that this is equivalent to the standard definition of path decompositions.



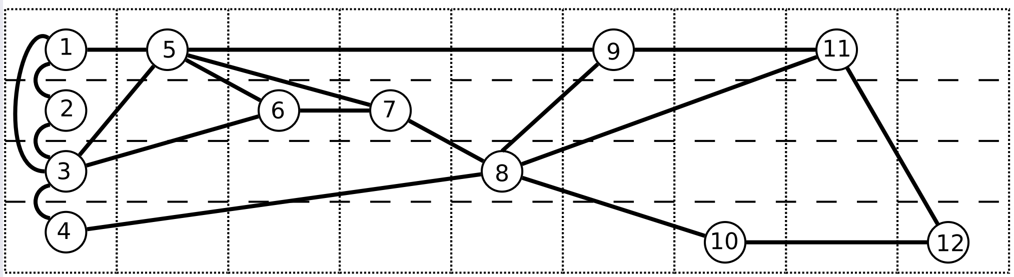
Treewidth – Pathwidth

Note that this is equivalent to the standard definition of path decompositions.



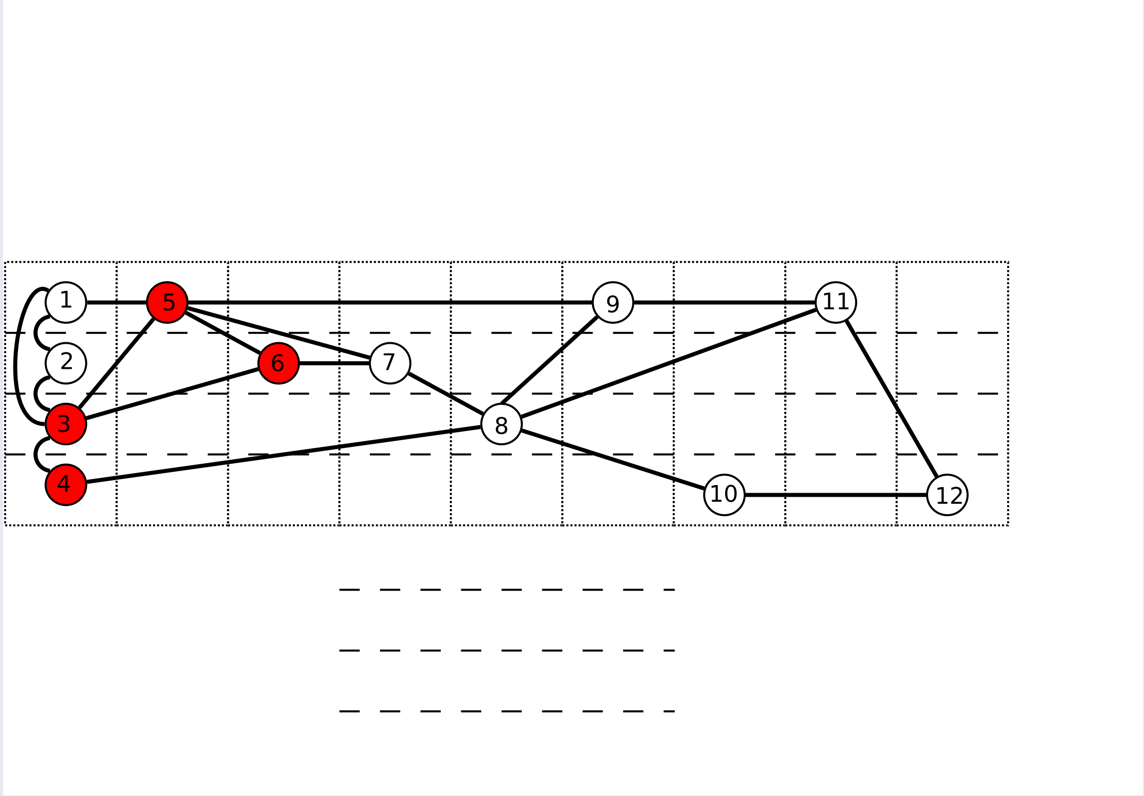
Treewidth

In treewidth we are allowed to branch out, starting from a set of vertices which are simultaneously on the top of their respective stacks.

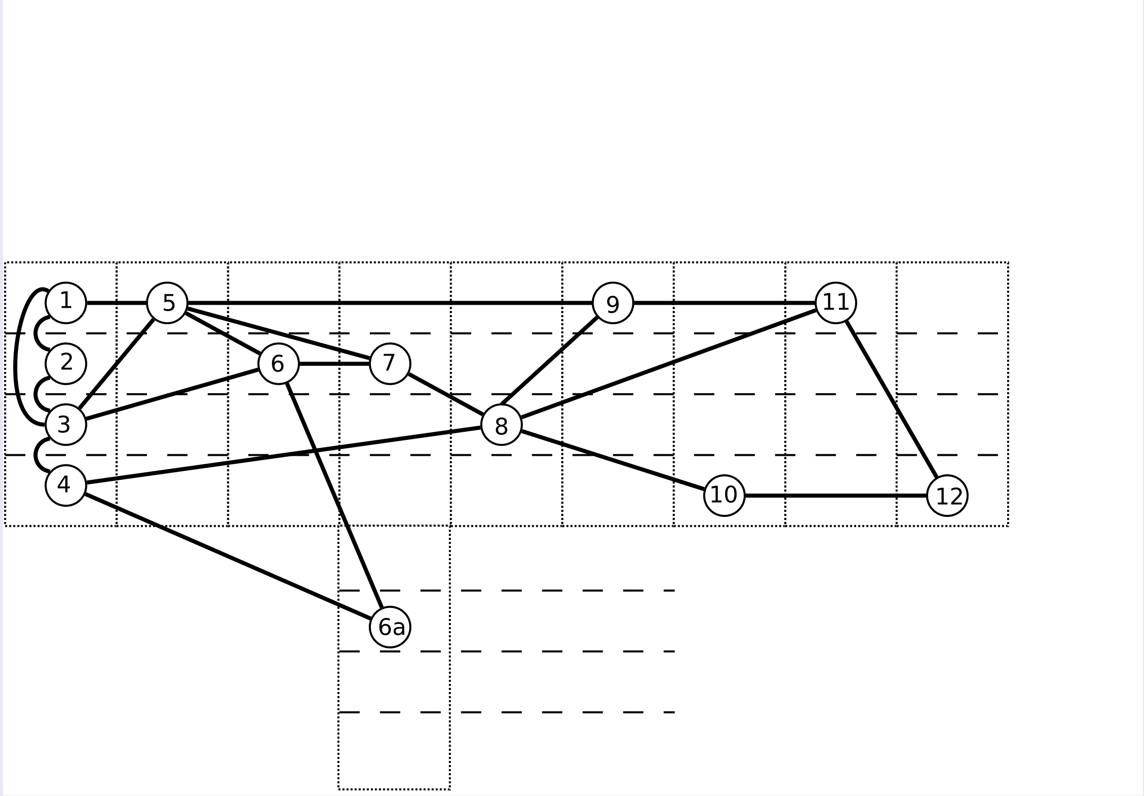


Treewidth

In treewidth we are allowed to branch out, starting from a set of vertices which are simultaneously on the top of their respective stacks.

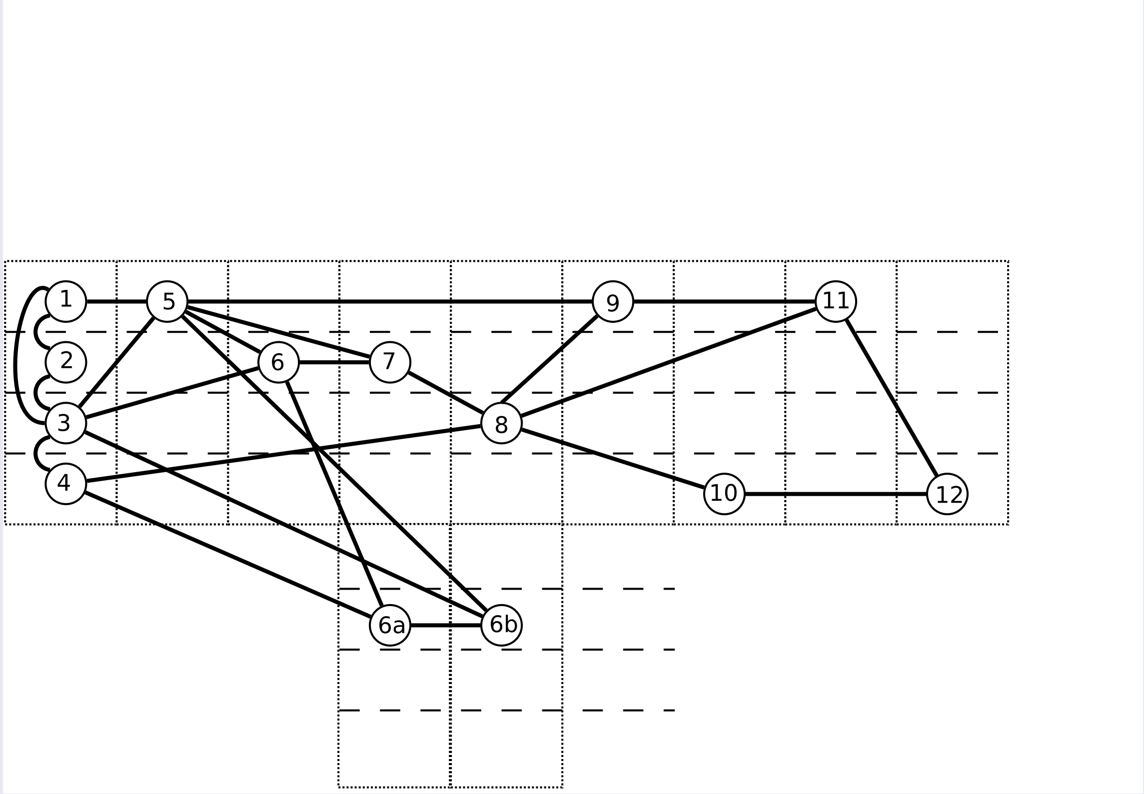


In treewidth we are allowed to branch out, starting from a set of vertices which are simultaneously on the top of their respective stacks.



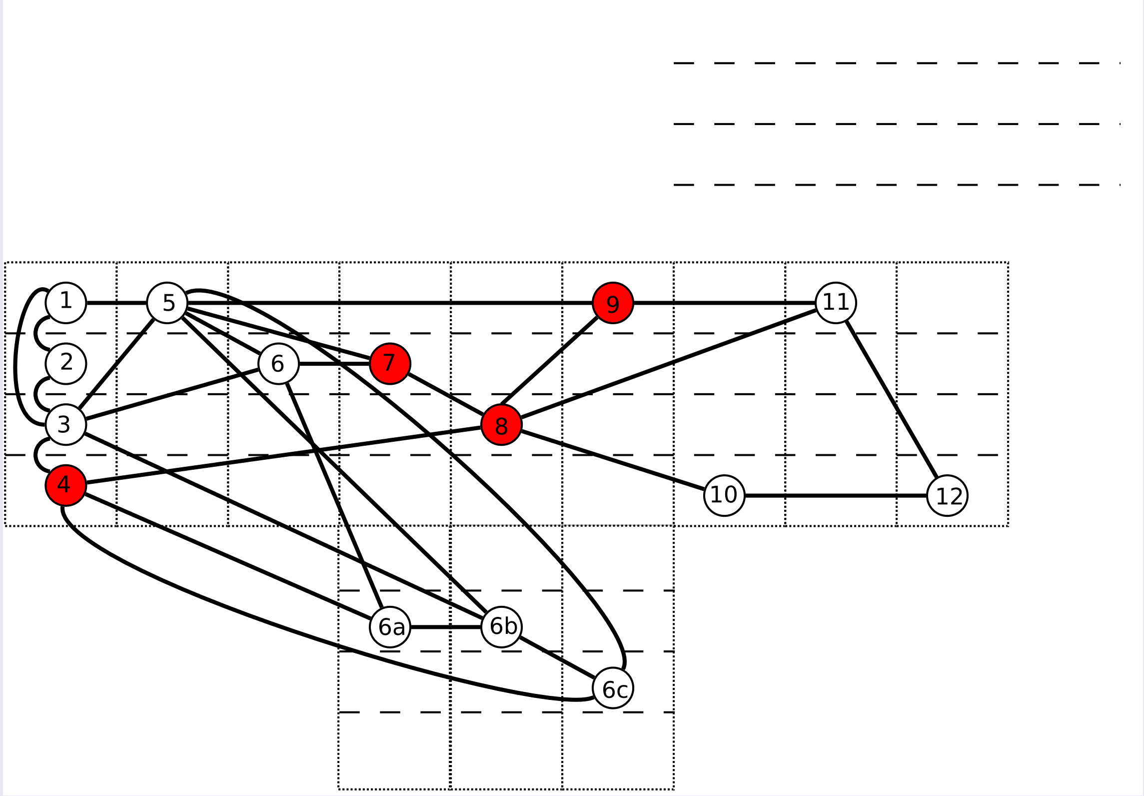
Treewidth

In treewidth we are allowed to branch out, starting from a set of vertices which are simultaneously on the top of their respective stacks.

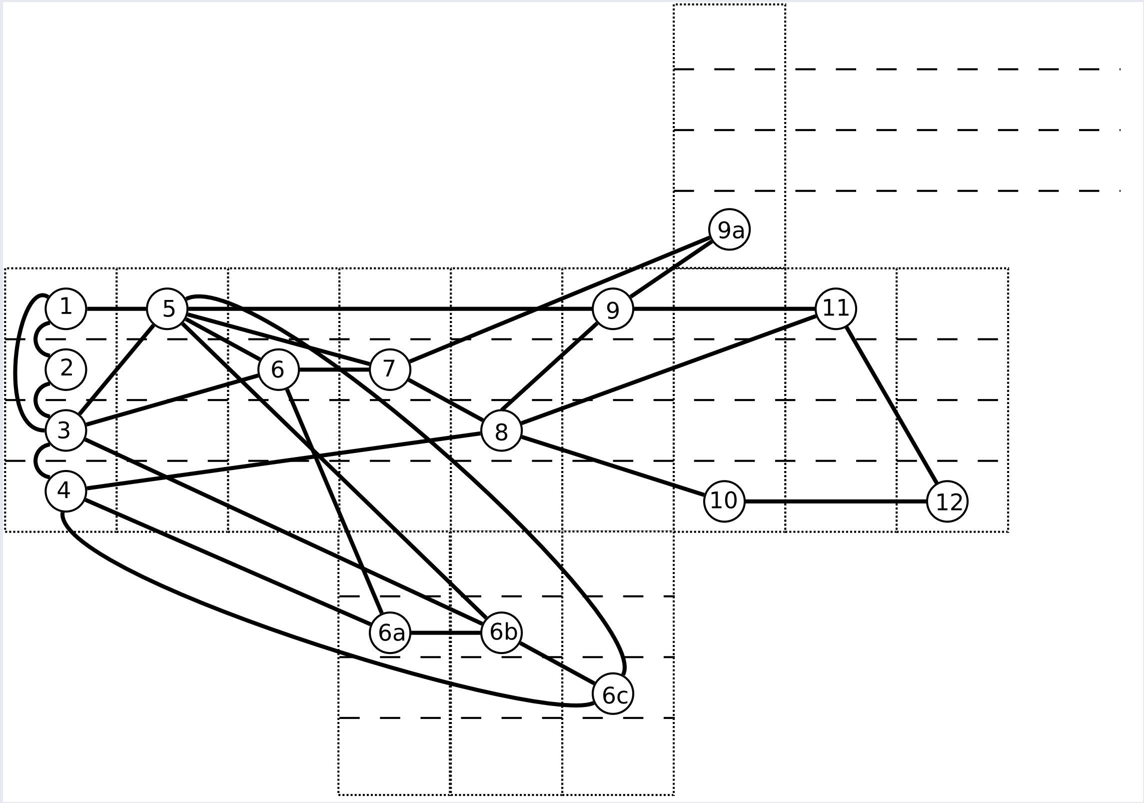


Treewidth

In treewidth we are allowed to branch out, starting from a set of vertices which are simultaneously on the top of their respective stacks.

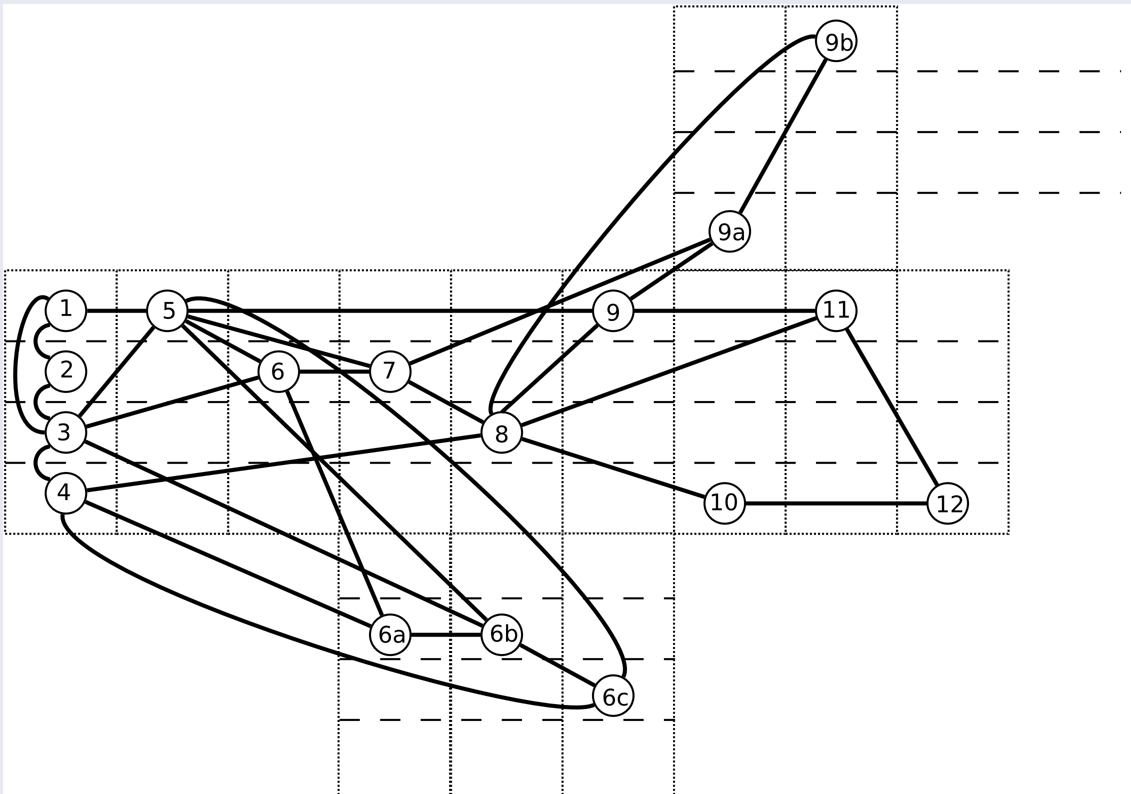


In treewidth we are allowed to branch out, starting from a set of vertices which are simultaneously on the top of their respective stacks.



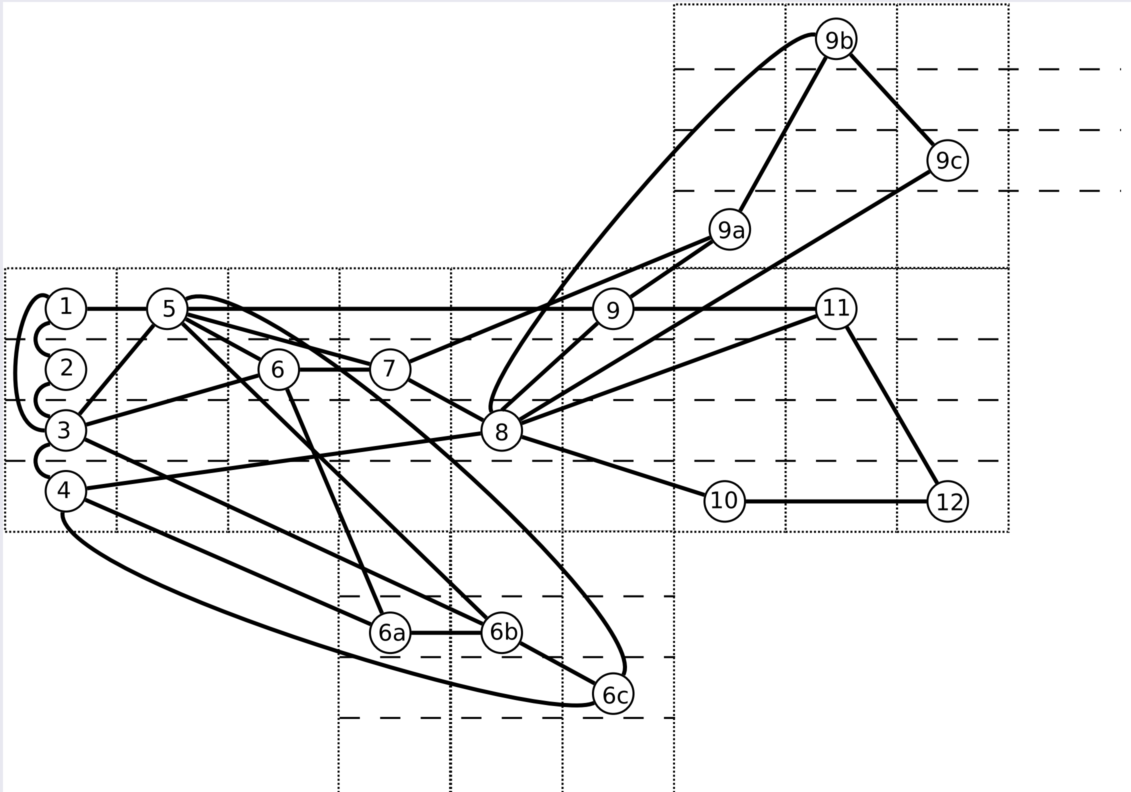
Treewidth

In treewidth we are allowed to branch out, starting from a set of vertices which are simultaneously on the top of their respective stacks.



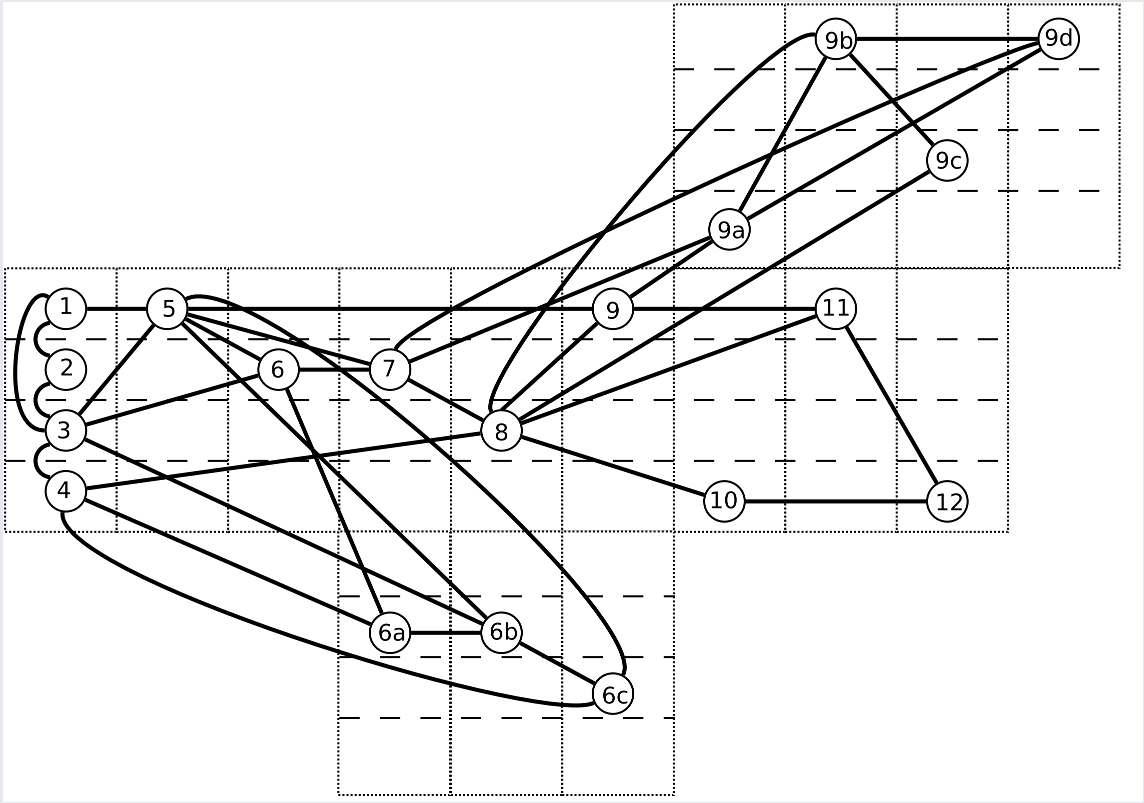
Treewidth

In treewidth we are allowed to branch out, starting from a set of vertices which are simultaneously on the top of their respective stacks.



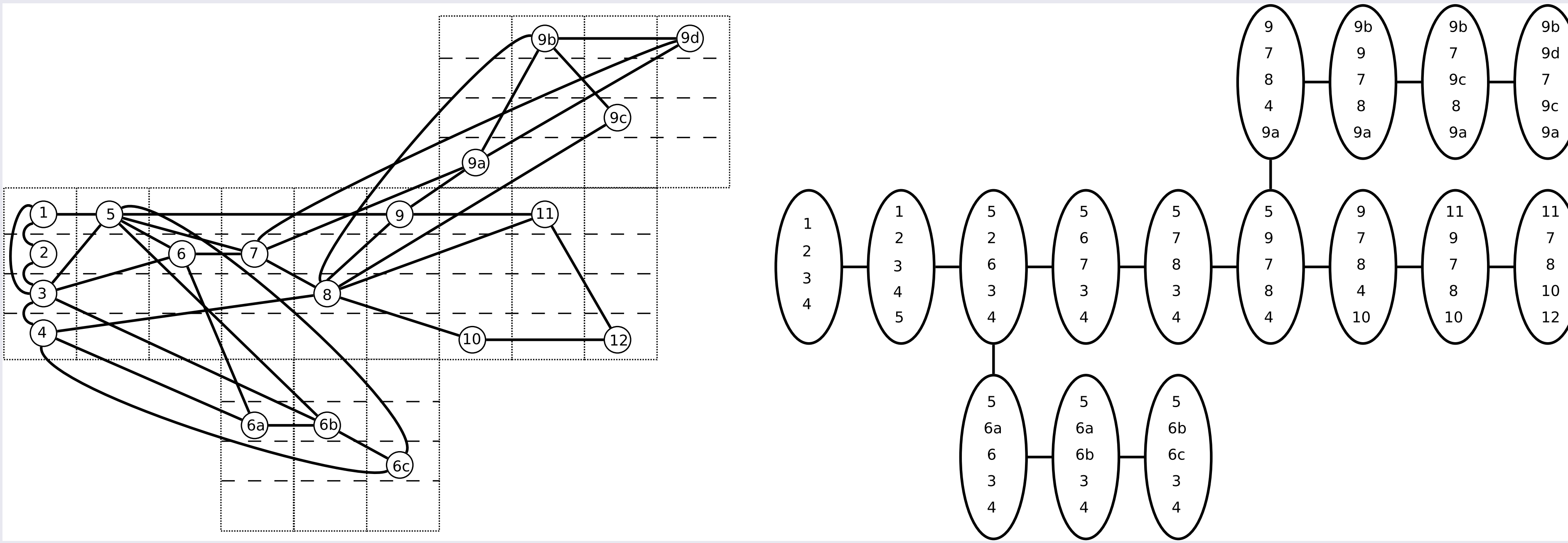
Treewidth

In treewidth we are allowed to branch out, starting from a set of vertices which are simultaneously on the top of their respective stacks.



Treewidth

In treewidth we are allowed to branch out, starting from a set of vertices which are simultaneously on the top of their respective stacks.



Treewidth – Pathwidth – Tree-depth

- Suppose at each step we add all allowed edges:
  - Pathwidth $\rightarrow$ interval graph with $\omega(G) = k + 1$
  - Treewidth $\rightarrow$ chordal graph with $\omega(G) = k + 1$
- We get the following equivalent definitions:

Treewidth(G)	$\min \omega(G')$	where G' is chordal supergraph of G
Pathwidth(G)	$\min \omega(G')$	where G' is interval supergraph of G
Treedepth(G)	$\min \omega(G')$	where G' is trivially perfect supergraph of G

Treewidth – Pathwidth – Tree-depth

- Suppose at each step we add all allowed edges:
  - Pathwidth $\rightarrow$ interval graph with $\omega(G) = k + 1$
  - Treewidth $\rightarrow$ chordal graph with $\omega(G) = k + 1$
- We get the following equivalent definitions:

Treewidth(G)	$\min \omega(G')$	where G' is chordal supergraph of G
Pathwidth(G)	$\min \omega(G')$	where G' is interval supergraph of G
Treedepth(G)	$\min \omega(G')$	where G' is trivially perfect supergraph of G

- Connection to interval graphs will be useful later.

Treewidth – Pathwidth – Tree-depth

- Suppose at each step we add all allowed edges:
  - Pathwidth $\rightarrow$ interval graph with $\omega(G) = k + 1$
  - Treewidth $\rightarrow$ chordal graph with $\omega(G) = k + 1$
- We get the following equivalent definitions:

Treewidth(G)	$\min \omega(G')$	where G' is chordal supergraph of G
Pathwidth(G)	$\min \omega(G')$	where G' is interval supergraph of G
Treedepth(G)	$\min \omega(G')$	where G' is trivially perfect supergraph of G

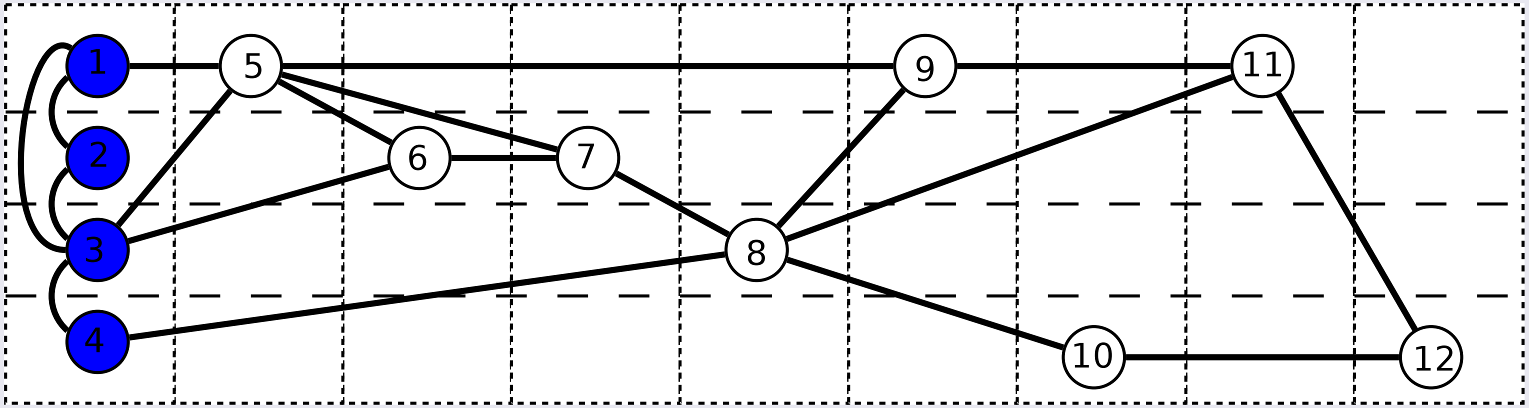
- Connection to interval graphs will be useful later.
- What about clique-width?
- Clique-width == treewidth + large bicliques
  - If G has treewidth t and no $K_{c,c}$ subgraph, then G has clique-width $O(ct)$. [Gurski&Wanke]

Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that “sweeps” the graph.

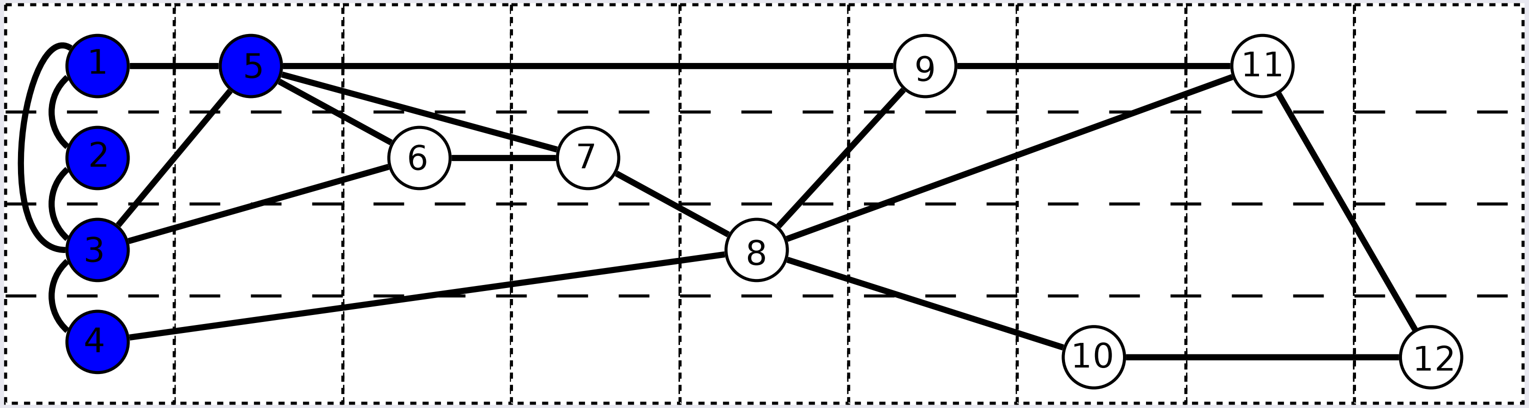
Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.



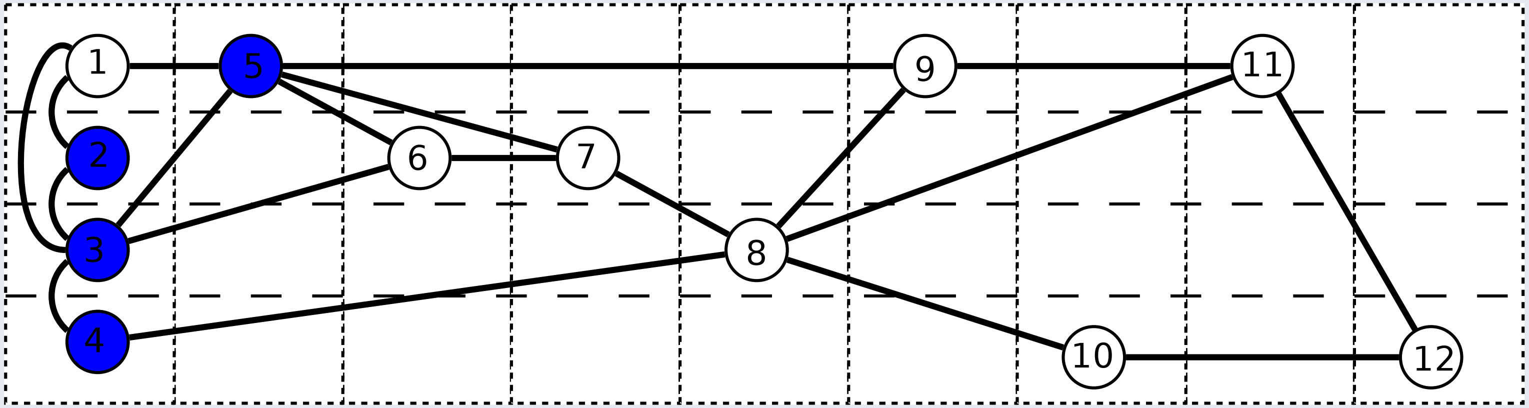
Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that “sweeps” the graph.



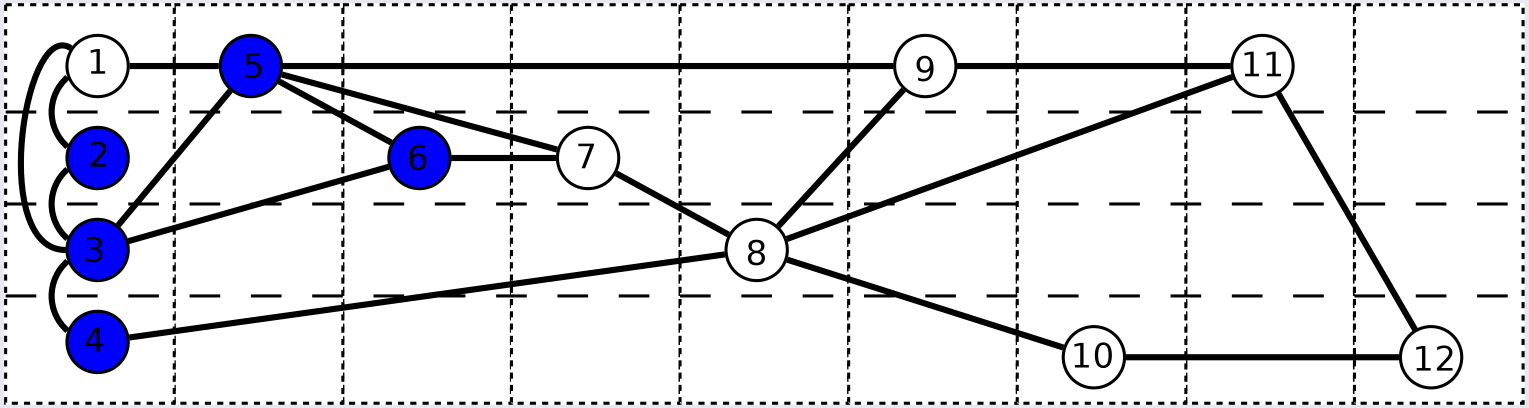
Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that “sweeps” the graph.



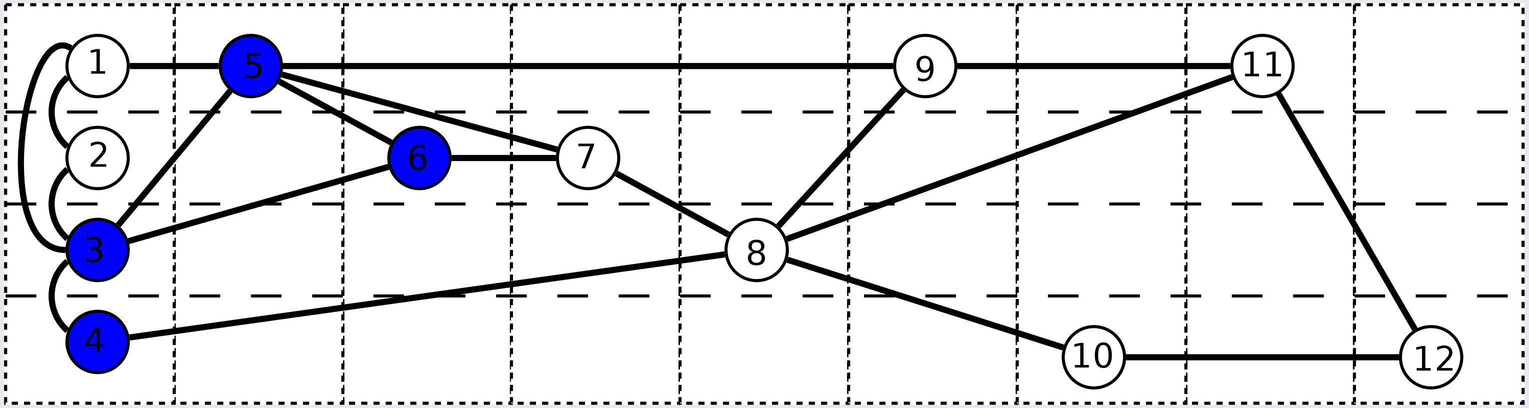
Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that “sweeps” the graph.



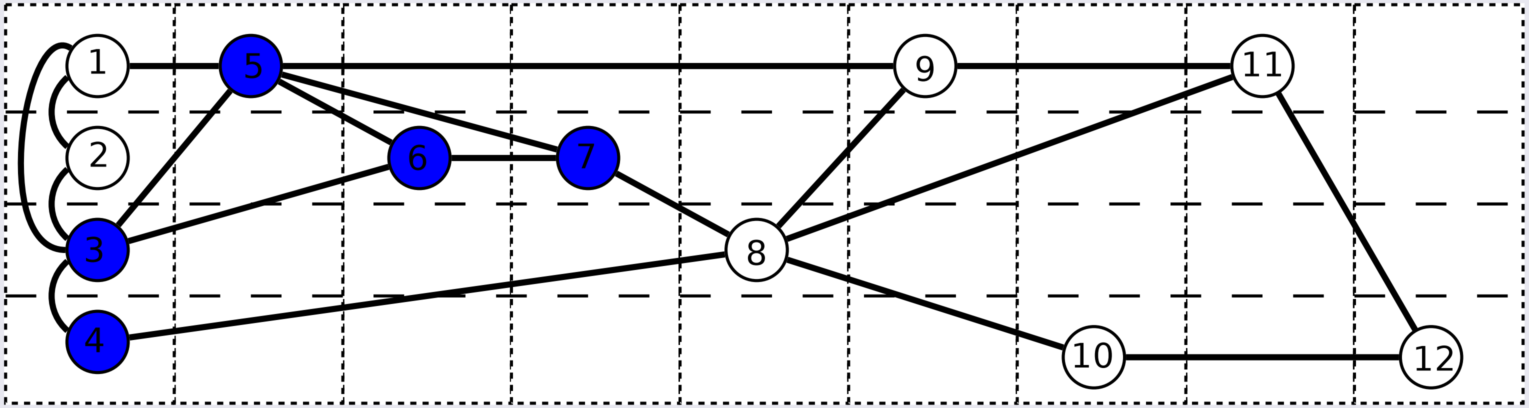
Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that “sweeps” the graph.



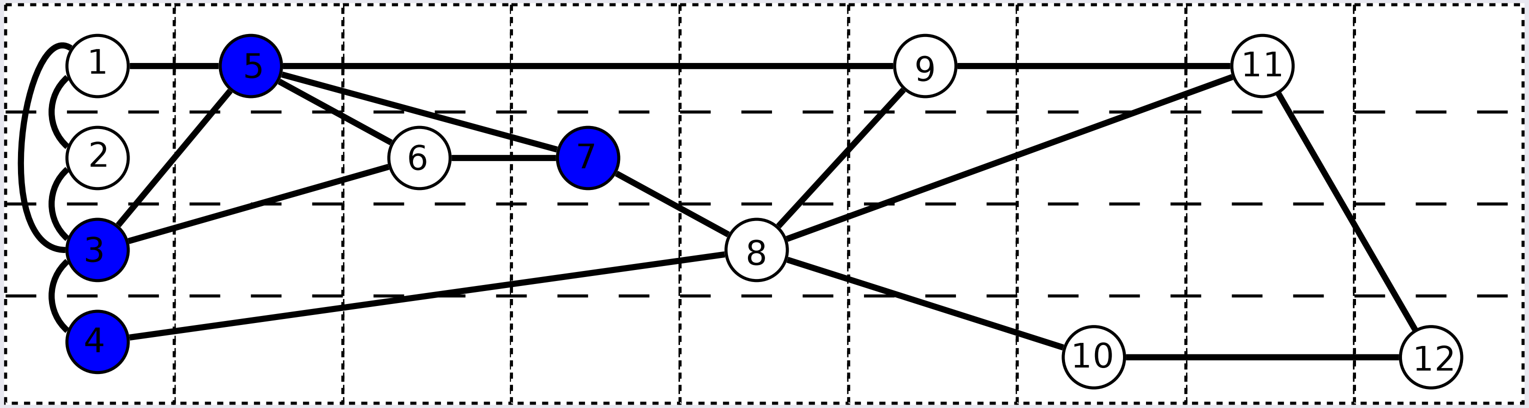
Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.



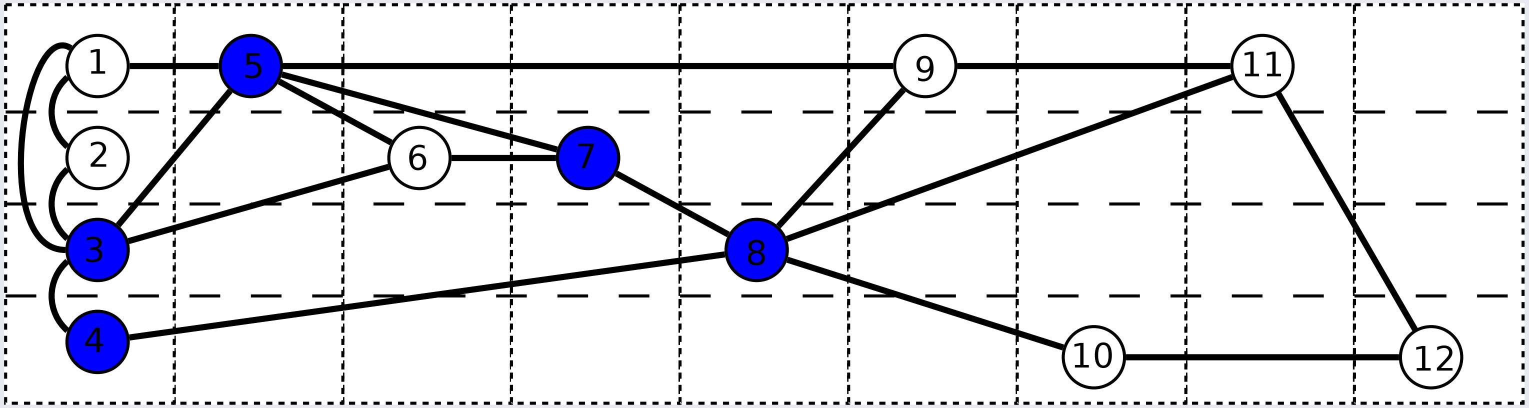
Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.



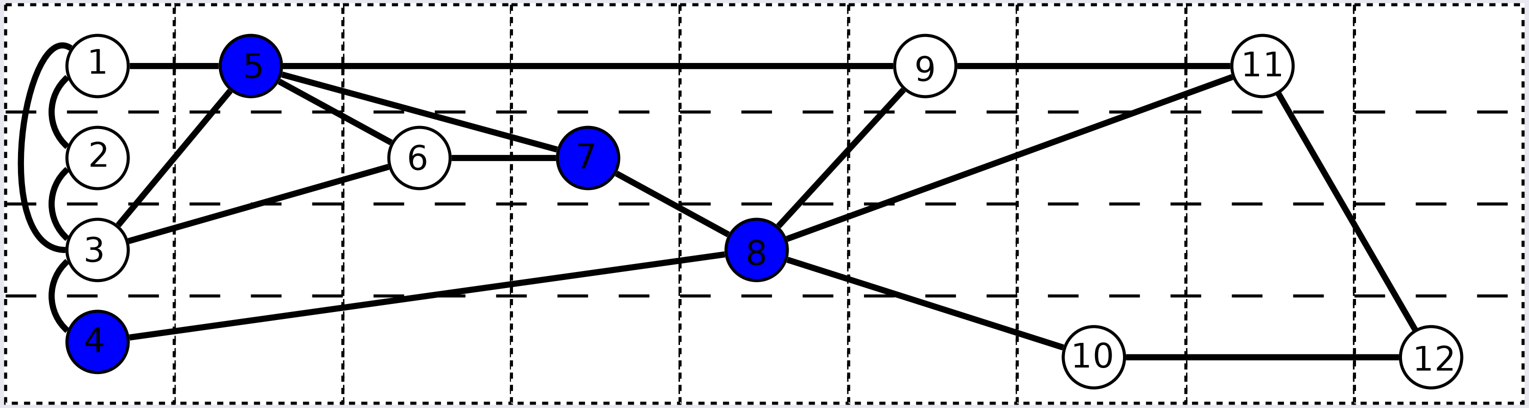
Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.



Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that “sweeps” the graph.



Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that “sweeps” the graph.

For 3-COLORING only need to remember information about boundary

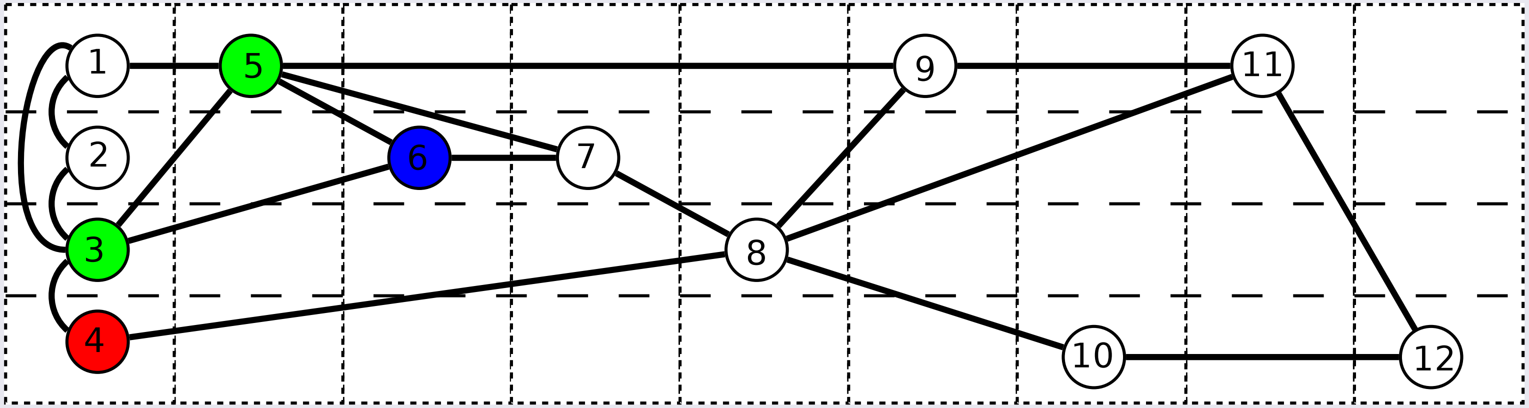
Which colorings of the boundary are properly extendible to the left?

Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.

For 3-COLORING only need to remember information about boundary

Which colorings of the boundary are properly extendible to the left?



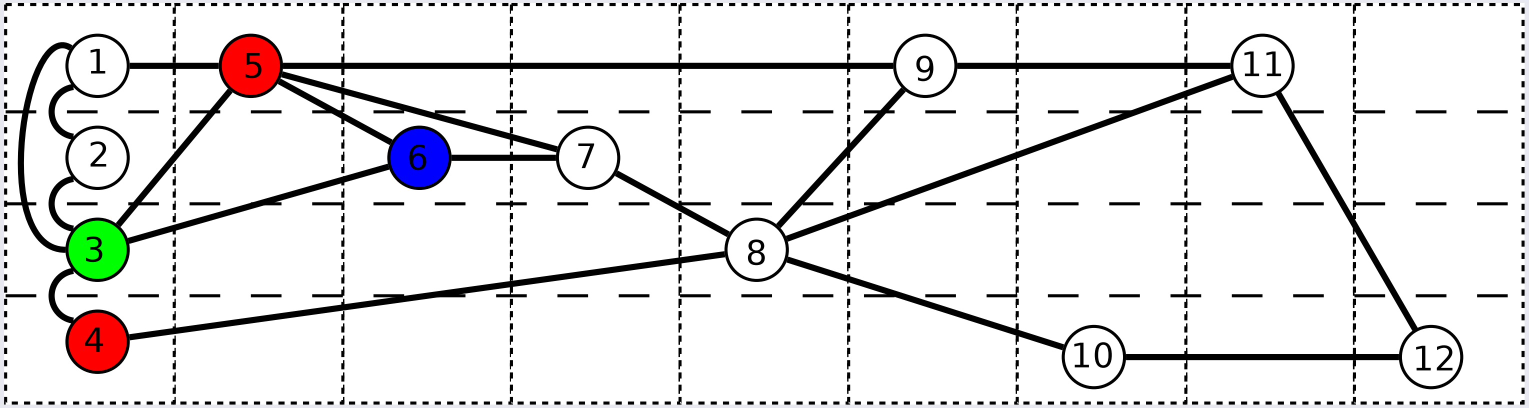
Separator: $\{3, 4, 5, 6\}$ includes tuple (3,4,5,6;No) because this coloring does not work

Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that “sweeps” the graph.

For 3-COLORING only need to remember information about boundary

Which colorings of the boundary are properly extendible to the left?



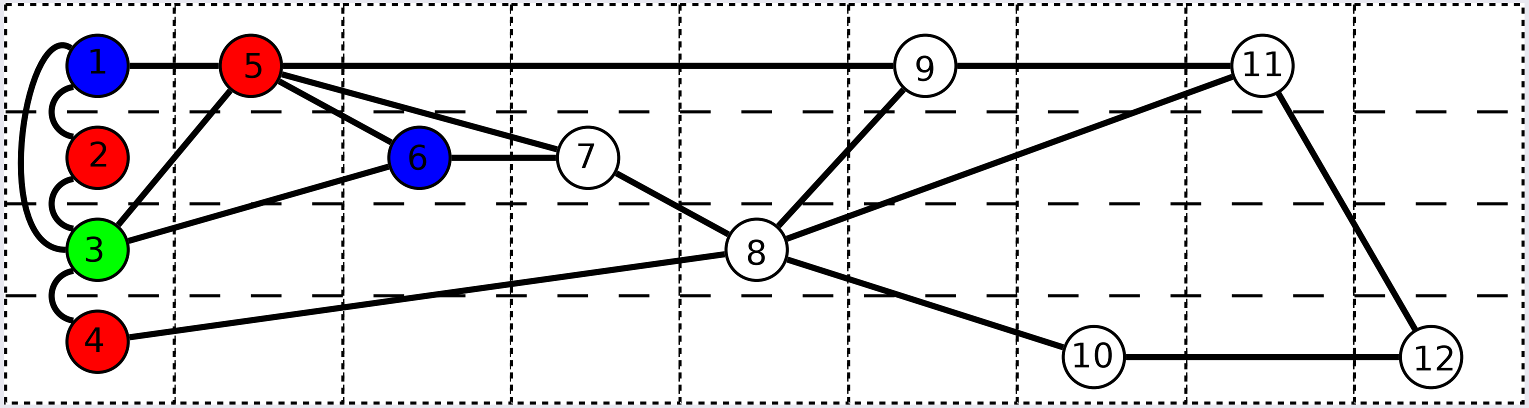
Separator: $\{3, 4, 5, 6\}$ includes tuple (3,4,5,6;Yes) because this coloring can be extended to the left

Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.

For 3-COLORING only need to remember information about boundary

Which colorings of the boundary are properly extendible to the left?



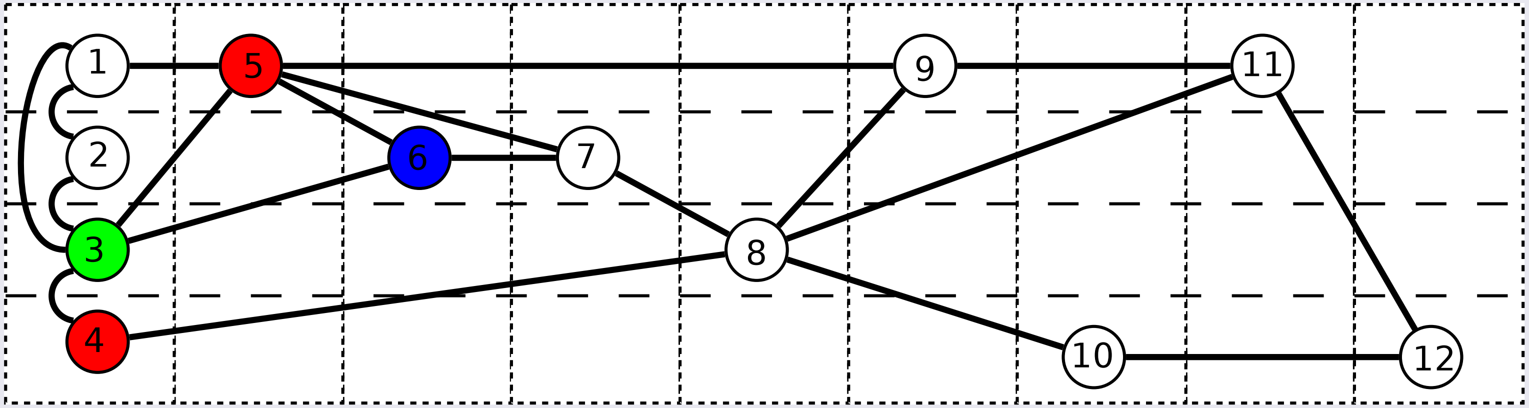
Separator: $\{3, 4, 5, 6\}$ includes tuple (3,4,5,6;Yes) because this coloring can be extended to the left

Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.

For 3-COLORING only need to remember information about boundary

Which colorings of the boundary are properly extendible to the left?



We now need to decide which are the good colorings for the separator $(3, 4, 5, 7)$.

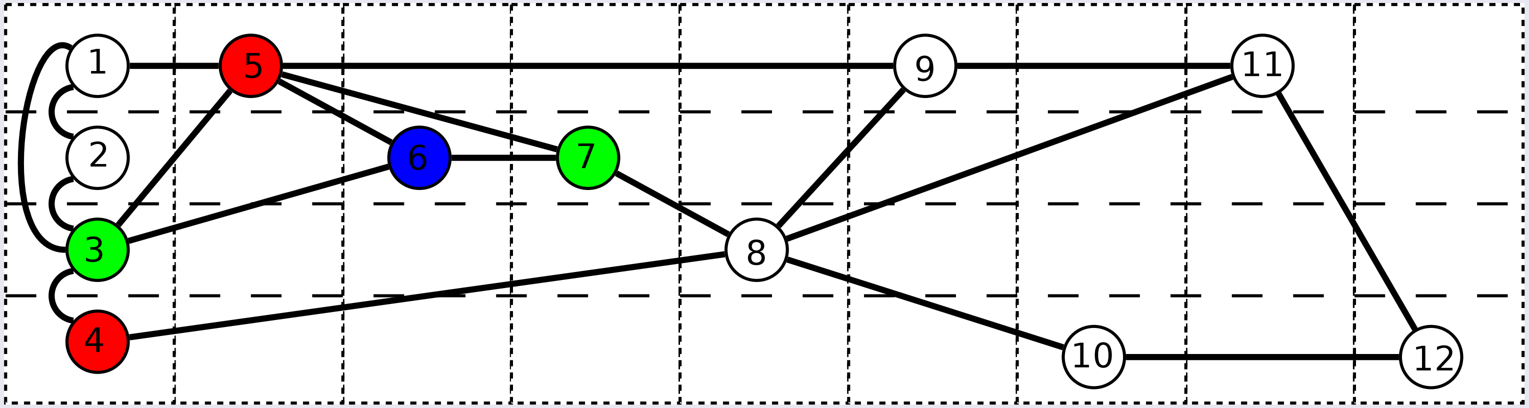
We consider each good coloring of $(3, 4, 5, 6)$.

Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.

For 3-COLORING only need to remember information about boundary

Which colorings of the boundary are properly extendible to the left?



We now need to decide which are the good colorings for the separator $(3, 4, 5, 7)$.
We consider each good coloring of $(3, 4, 5, 6)$.
We see that $({\color{green}3}, {\color{red}4}, {\color{red}5}, {\color{green}7})$ is a good coloring.
Important: we know the colors of all neighbors of 7.

Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.

For 3-COLORING only need to remember information about boundary

Which colorings of the boundary are properly extendible to the left?

- DP tables have size 3^w.
- Things work in similar way for treewidth.
- Perhaps not surprising that complexity is the same for most problems??
    - Big back story we skip: Fast Subset Convolution

Lessons from the fox

Price of Generality and Combinatorics

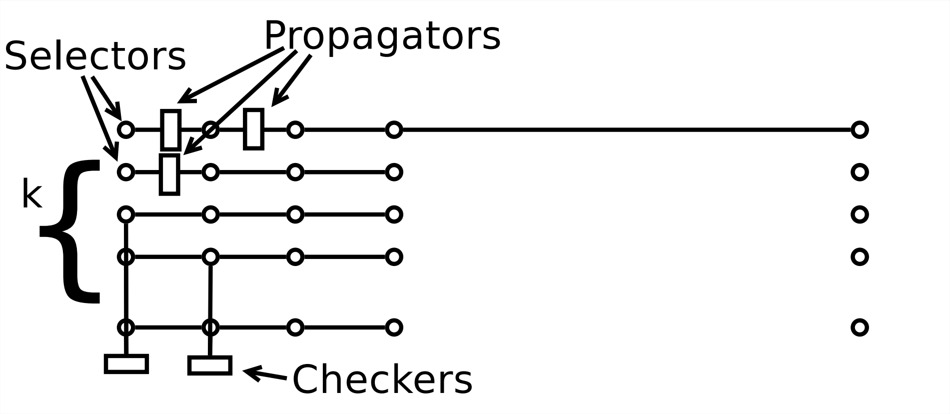
- Sometimes, the reason a problem becomes FPT for a more restricted parameter is more combinatorial than algorithmic.
- Example:
  - Coloring is FPT for tw, W-hard for cw.
  - But algorithm runs in k^{tw}. Is this FPT?
  - Yes! Because in all graphs $\chi(G) \leq tw(G)$.
  - This bound makes all the difference: Coloring is FPT by $cw + k$.

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!

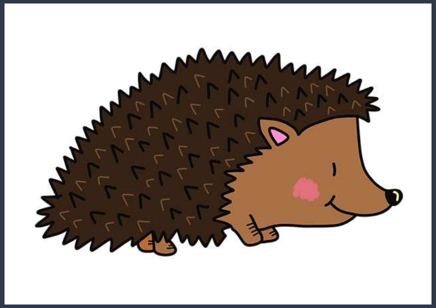
Hardness for pathwidth and treewidth

- Typical W-hard problem for tw/pw:
  - Basic DP must decide a value in $1 \ldots n$ for each vertex in bag.
  - Given n^{tw} algorithm.
- How to prove this is optimal?
  - Reduce from k-MC-Clique
  - Choice for each vertex in bag $\Leftrightarrow$ choice for each color class
- Typical Structure:



- Key fact: $k \times n$ grid has both pathwidth and treewidth k.

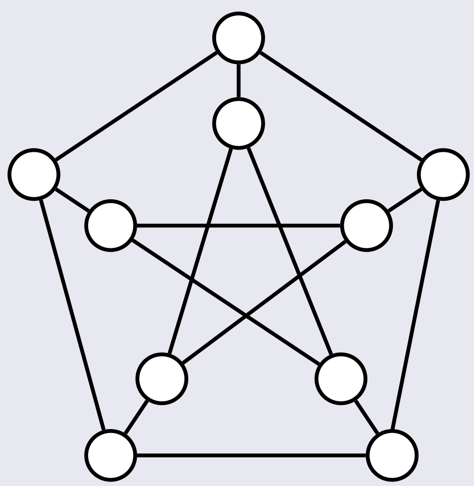
Let’s nail this problem!



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**.

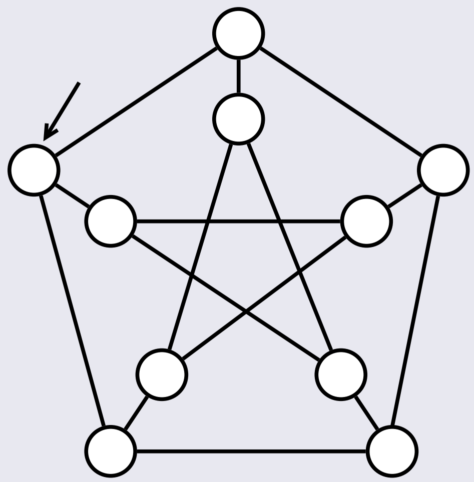
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

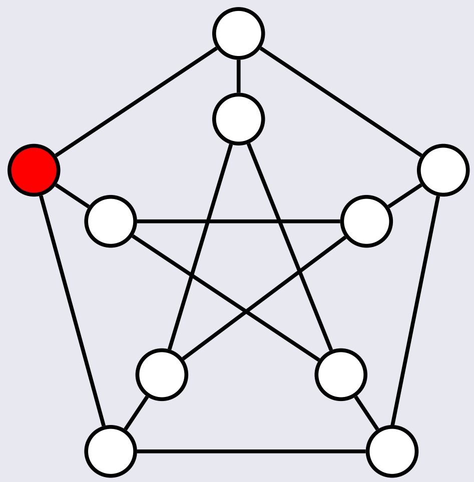
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

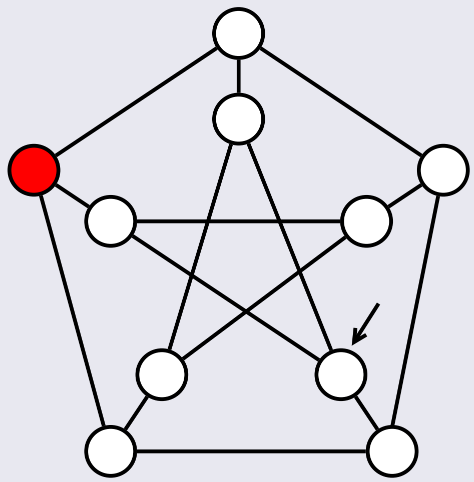
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

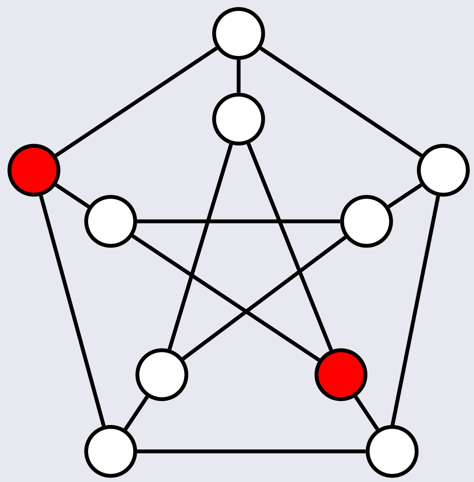
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

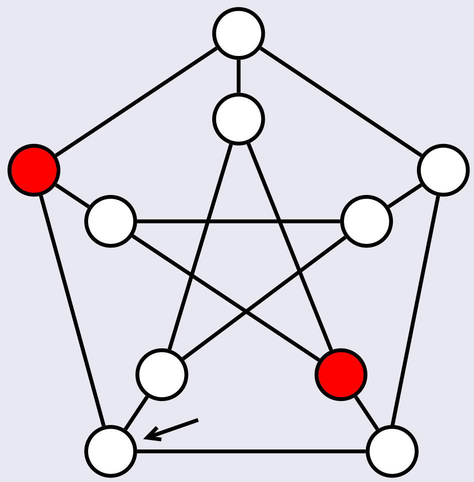
Grundy Coloring

- 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

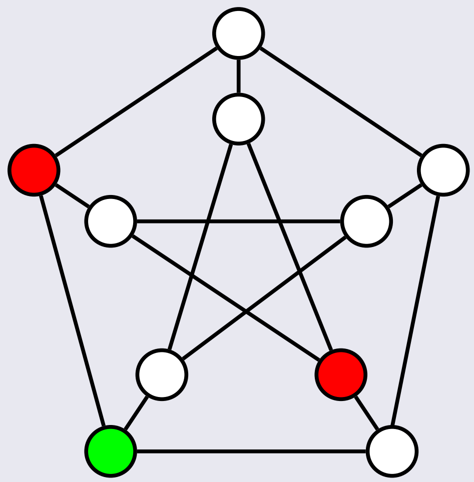
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
    - Select an uncolored vertex u of G
    - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

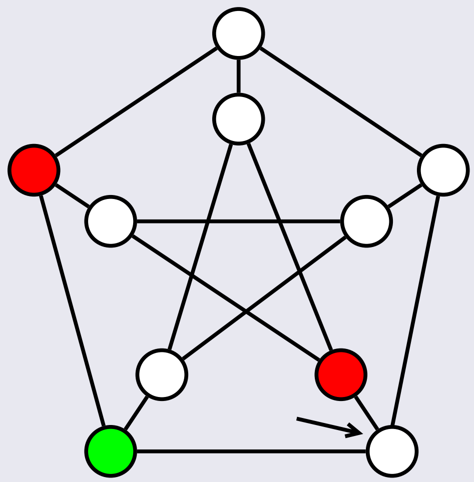
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

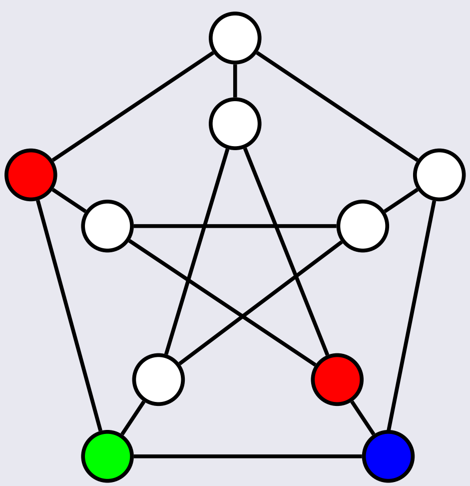
Grundy Coloring

- 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**.



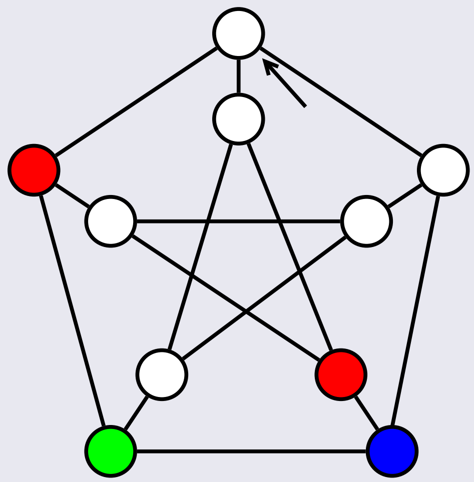
Red	1
Green	2
Blue	3
Yellow	4

- 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

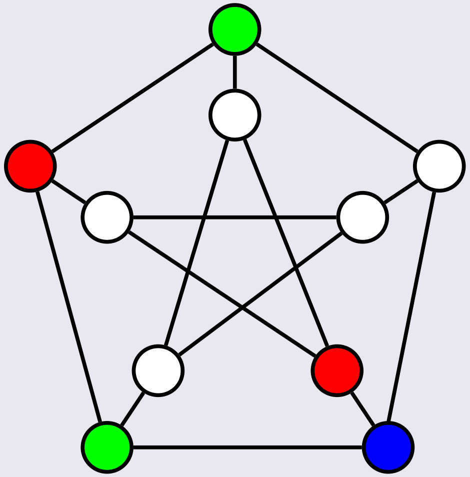
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

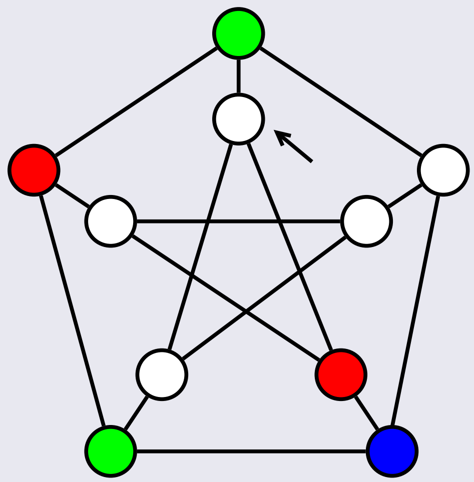
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

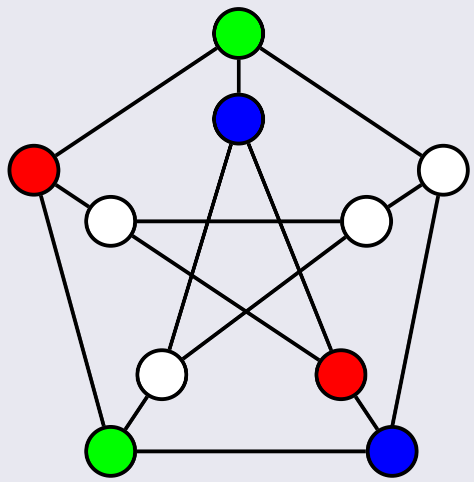
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

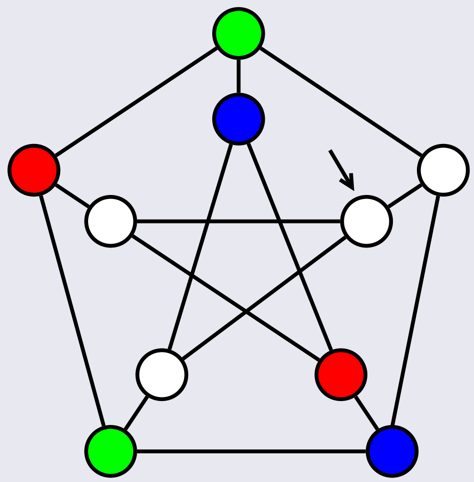
Grundy Coloring

- 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**.



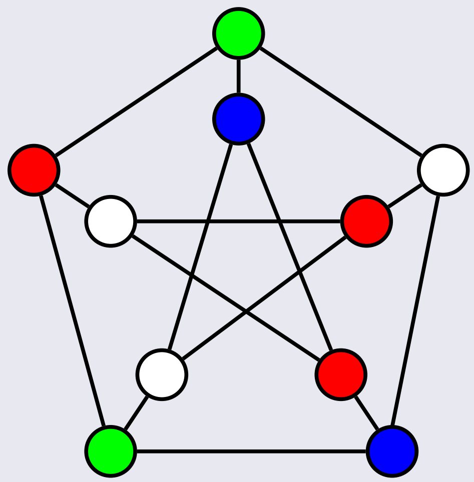
Red	1
Green	2
Blue	3
Yellow	4

- 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

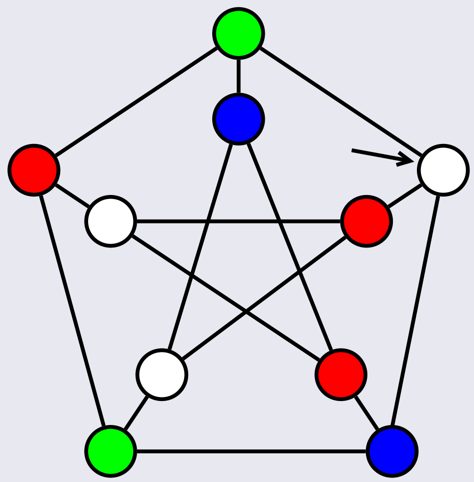
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



Red	1
Green	2
Blue	3
Yellow	4

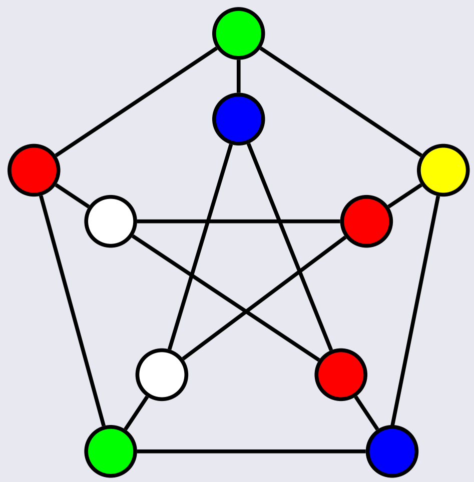
Grundy Coloring

- 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

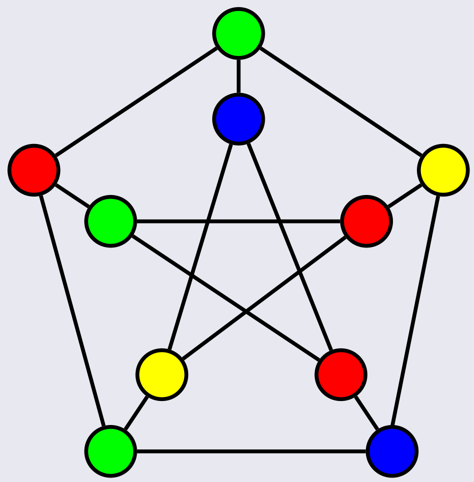
- Input: Graph $G = (V, E)$ on n vertices
- Repeat n times
  - Select an uncolored vertex u of G
  - Assign u the smallest color that is not currently used in any of its neighbors (**First-Fit**)
- Goal: Order the vertices in such a way that number of colors used is **maximized**.



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**.

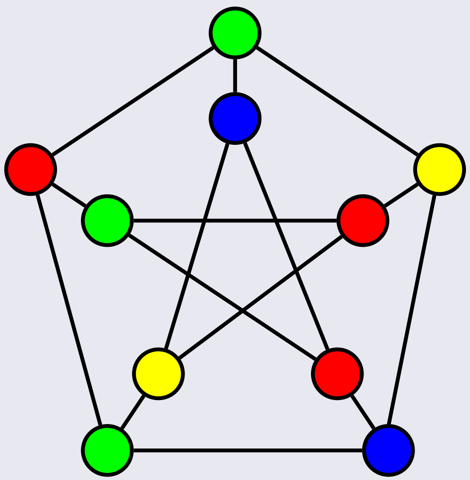


Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

- $\Gamma(G)$: max Grundy Coloring
- $\chi(G)$: chromatic number
- Def1: max # colors used by First-Fit
- Def2: max # colors in proper coloring where $\forall i < j$, color class i dominates color class j

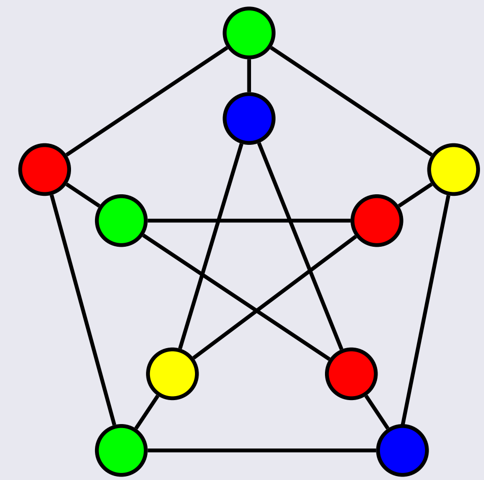
- $\Gamma(G) \geq \chi(G)$ for all graphs.
- $\Gamma(G)$ can be arbitrarily larger than $\chi(G)$.
- For Petersen graph $\chi(G) = 3$ and this coloring shows that $\Gamma(G) \geq 4$
- Is $\Gamma(G) = 4$?



Red	1
Green	2
Blue	3
Yellow	4

Grundy Coloring

- $\Gamma(G)$: max Grundy Coloring
- $\chi(G)$: chromatic number
- Def1: max # colors used by First-Fit
- Def2: max # colors in proper coloring where $\forall i < j$, color class i dominates color class j



- $\Gamma(G) \geq \chi(G)$ for all graphs.
- $\Gamma(G)$ can be arbitrarily larger than $\chi(G)$.
- For Petersen graph $\chi(G) = 3$ and this coloring shows that $\Gamma(G) \geq 4$
- Is $\Gamma(G) = 4$?

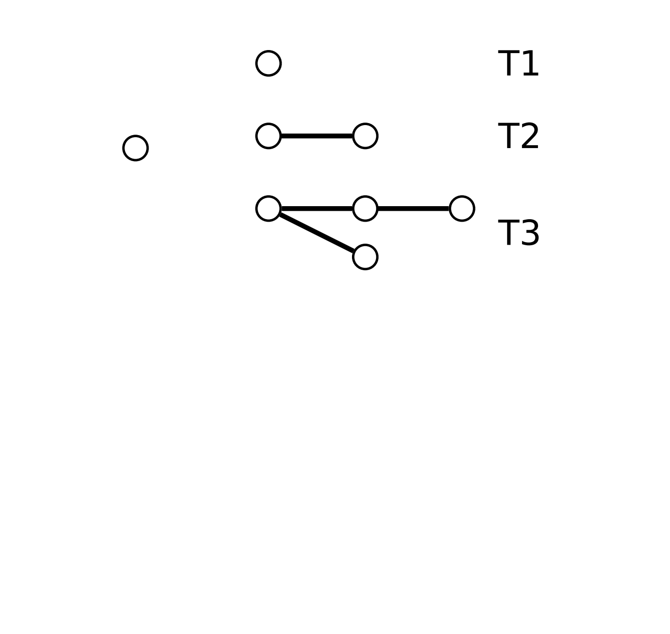
- In all graphs $\Gamma(G) \leq \Delta + 1$, so $\Gamma(G) = 4$ for Petersen.

Binomial Trees

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

Binomial Trees

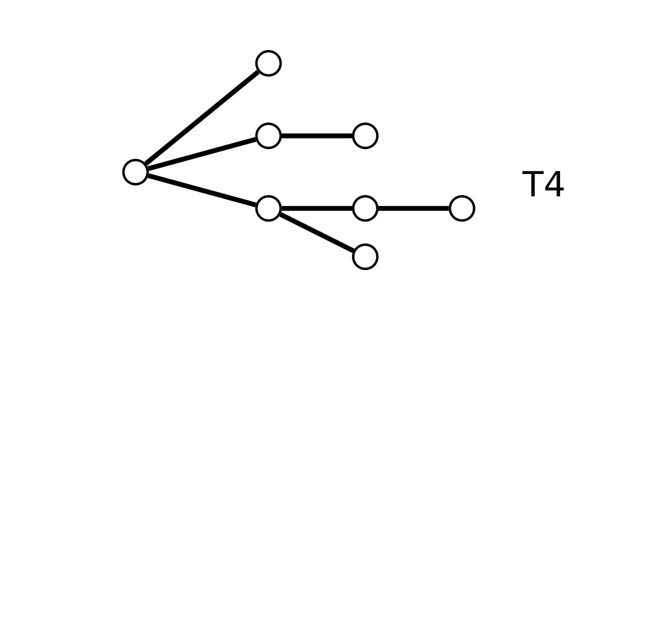
- The Binomial Tree T_k has a Grundy Coloring which assigns color k to the root



- 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

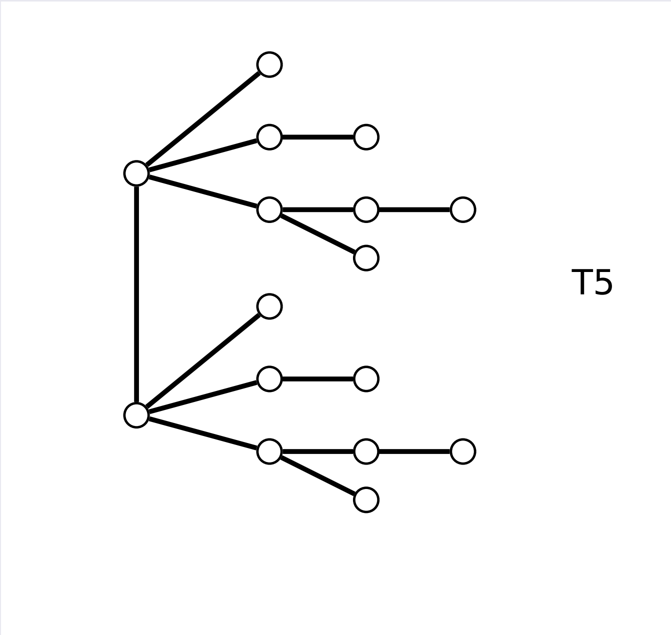
- The Binomial Tree T_k has a Grundy Coloring which assigns color k to the root



- 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



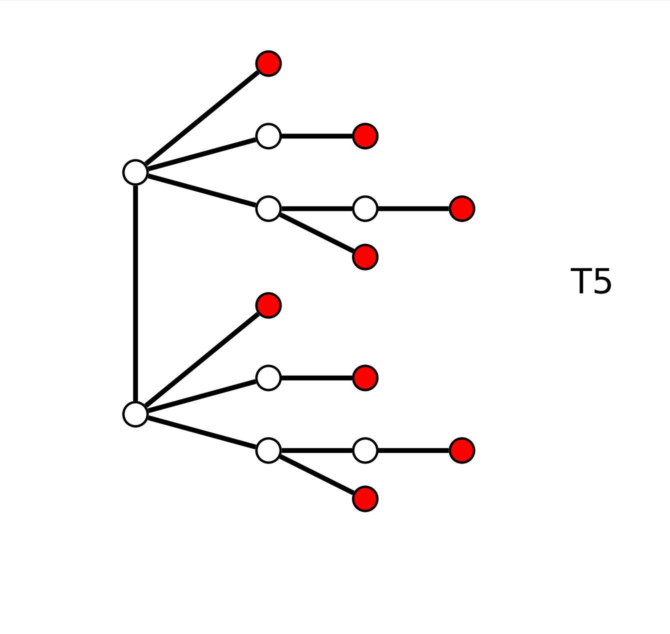
- Two recursive constructions
- T_1 is a vertex.
- T_k is a new root connected to $T_{k-1}, T_{k-2}, \ldots, T_1$.

  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



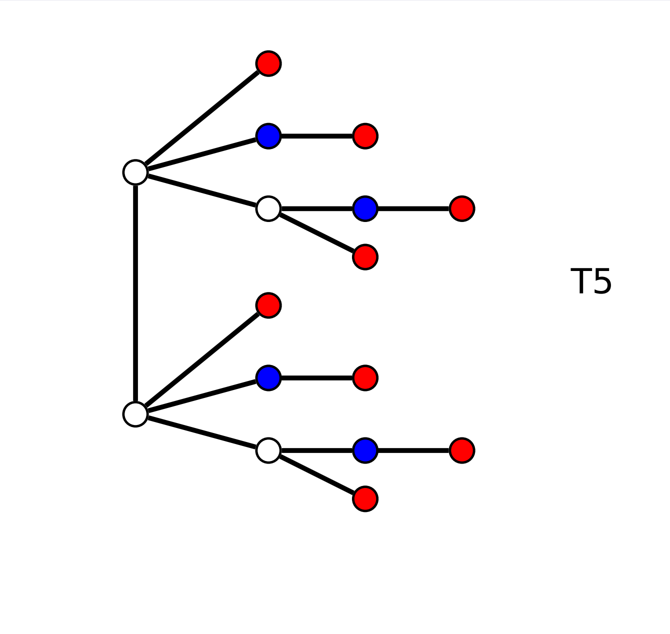
- Two recursive constructions
- T_1 is a vertex.
- T_k is a new root connected to $T_{k-1}, T_{k-2}, \ldots, T_1$.

  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

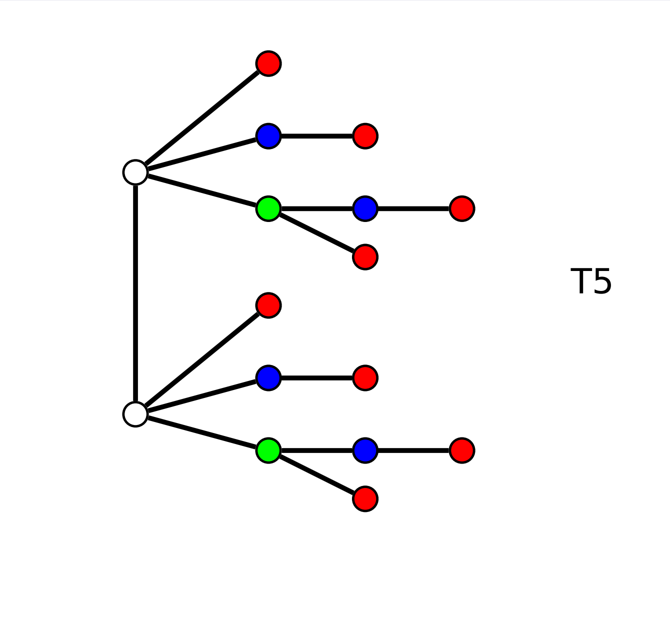


- Two recursive constructions
- T_1 is a vertex.
- T_k is a new root connected to $T_{k-1}, T_{k-2}, \ldots, T_1$.

  Or

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

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



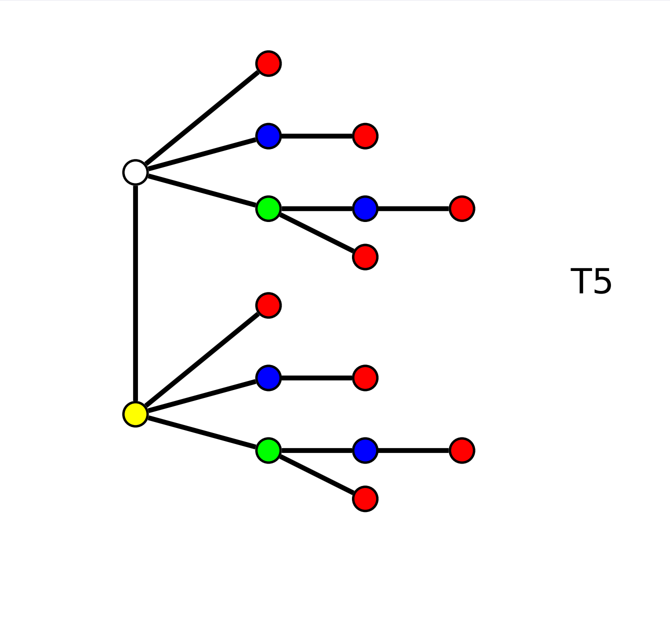
- Two recursive constructions
- T_1 is a vertex.
- T_k is a new root connected to $T_{k-1}, T_{k-2}, \ldots, T_1$.

  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



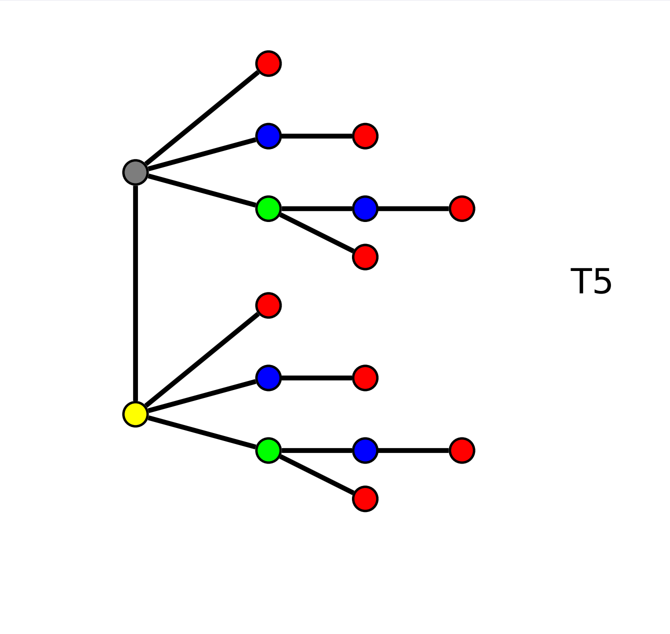
- Two recursive constructions
- T_1 is a vertex.
- T_k is a new root connected to $T_{k-1}, T_{k-2}, \ldots, T_1$.

  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



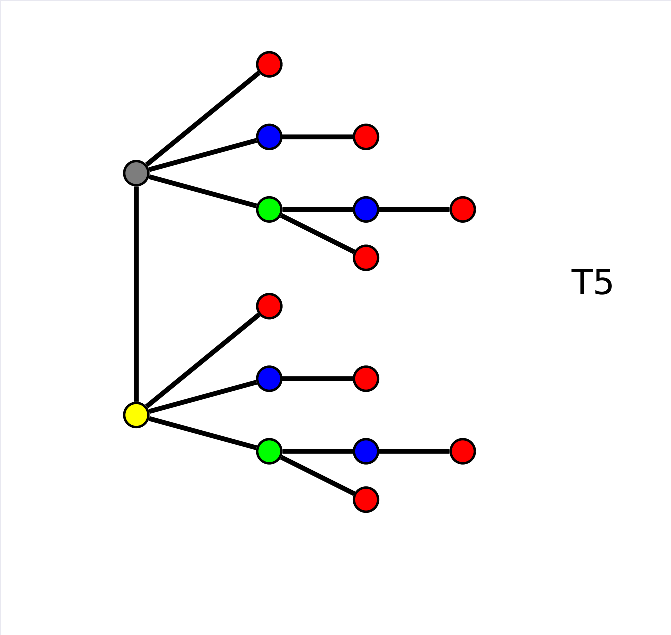
- Two recursive constructions
- T_1 is a vertex.
- T_k is a new root connected to $T_{k-1}, T_{k-2}, \ldots, T_1$.

  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



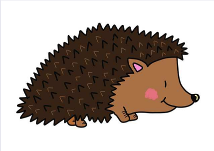
- Two recursive constructions
- T_1 is a vertex.
- T_k is a new root connected to $T_{k-1}, T_{k-2}, \ldots, T_1$.

  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 $\Gamma(G) \leq tw(G) \log n$.



Background on Grundy Coloring

- Grundy Coloring is NP-hard (already in Garey&Johnson)
  - Even on chordal graphs. . .
- Hard to approximate [Kortsarz DMTCS ’07]
- Solvable in XP time parameterized by $\Gamma(G)$ [Zaker DAM ’06]
- But W-hard and not solvable in $n^{2^{o(k)}}$ [Aboulker et al. STACS ’20]

Background on Grundy Coloring

- Grundy Coloring is NP-hard (already in Garey&Johnson)
  - Even on chordal graphs...
- Hard to approximate [Kortsarz DMTCS '07]
- Solvable in XP time parameterized by $\Gamma(G)$ [Zaker DAM '06]
- But W-hard and not solvable in $n^{2^{o(k)}}$ [Aboulker et al. STACS '20]
  - The n^{2^k} algorithm is based on the existence of a "witness"
  - Witness = minimal induced subgraph of $\Gamma = k$.
  - Worst case: witness is binomial tree $\rightarrow$ has size 2^k.
  - We exhaustively look for a witness...
  - This is optimal!

Background on Grundy Coloring

- Grundy Coloring is NP-hard (already in Garey&Johnson)
  - Even on chordal graphs...
- Hard to approximate [Kortsarz DMTCS '07]
- Solvable in XP time parameterized by $\Gamma(G)$ [Zaker DAM '06]
- But W-hard and not solvable in $n^{2^{o(k)}}$ [Aboulker et al. STACS '20]

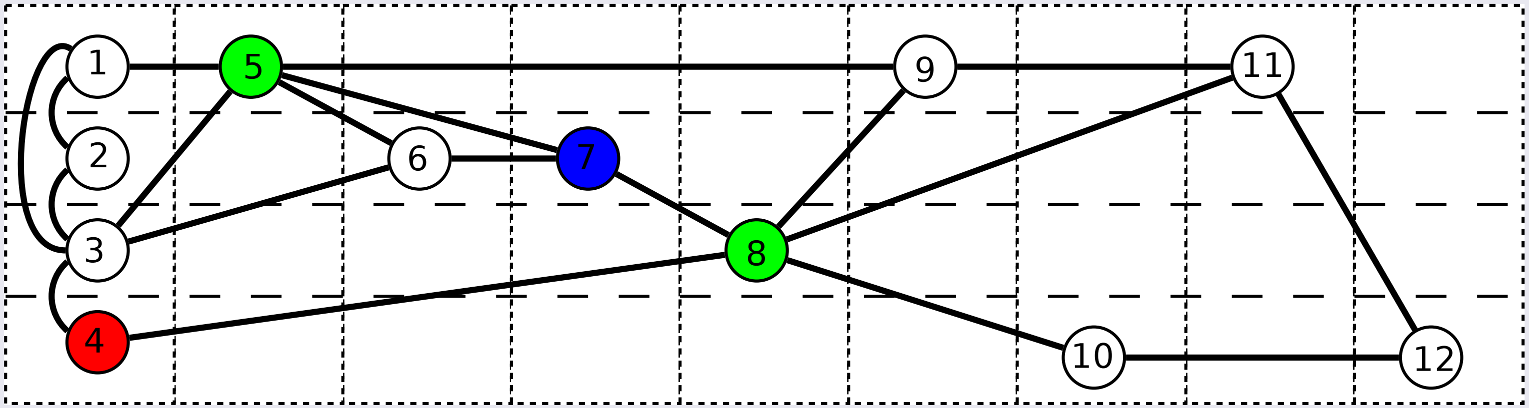
What about treewidth/pathwidth?

  - Problem solvable in $2^{\Gamma tw}$ (next slide)
  - Note: not obviously FPT, or even XP!
  - On interval graphs, $\Gamma(G) \leq 8\chi(G) = 8\omega(G)$ [Narayanaswamy & Babu, Order '08]

  - Recall connection interval graphs $\leftrightarrow$ pathwidth

Algorithm for Grundy and Treewidth

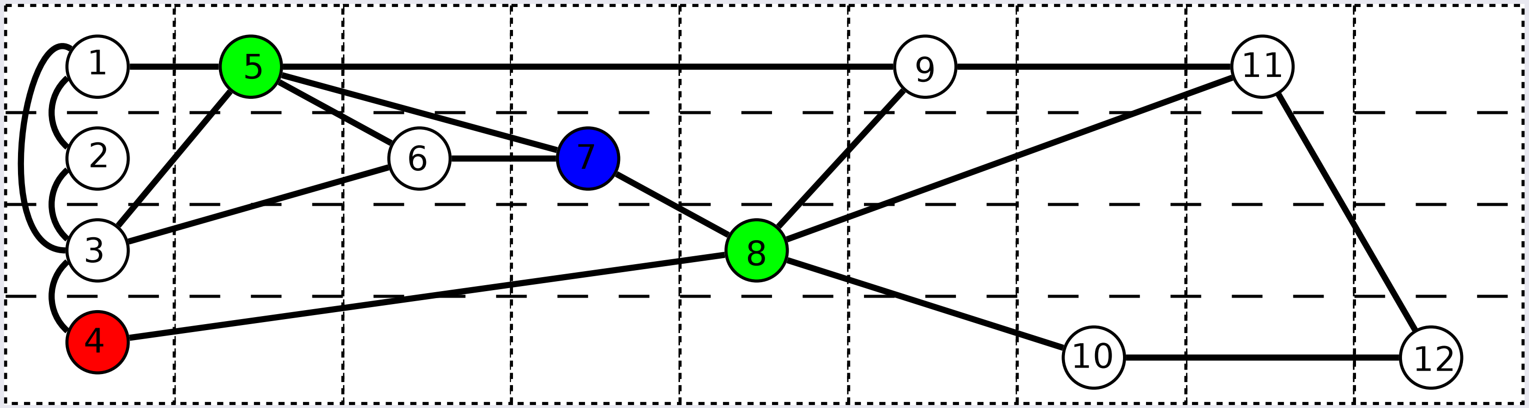
- XP algorithm due to [Telle&Proskurowski SIDMA'97]



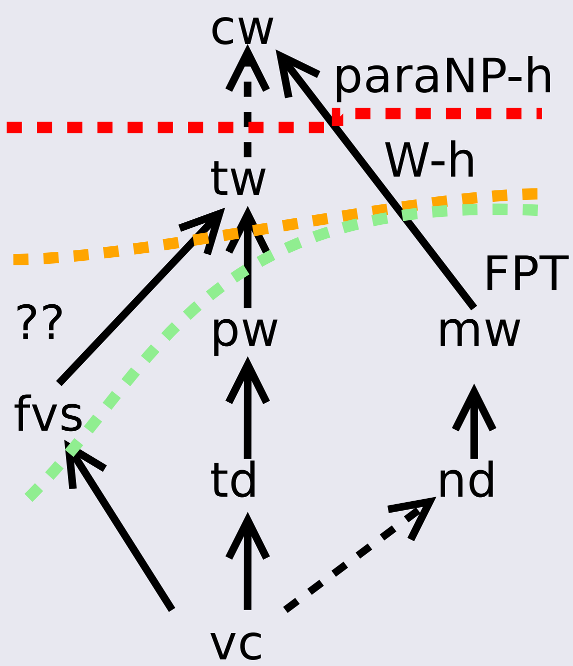
- Standard Coloring DP: recall color of each vertex in bag
  - $\rightarrow k^{tw}$
- Problem: for each vertex we need to make sure that it is dominated by **all** lower colors
  - In this example, this coloring is only valid if 6 takes color Red
- Need to remember for each vertex **the subset** of colors 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: $n^{O(tw^2)}$



Main results:

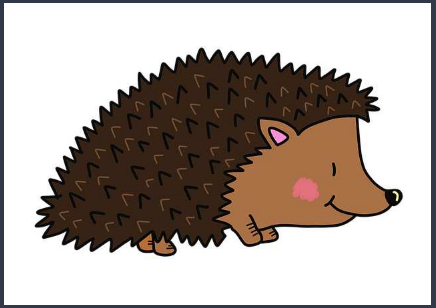
- Grundy Coloring is W[1]-hard by treewidth
- Grundy Coloring is FPT by pathwidth

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

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

Some observations:

- Must produce a Grundy instance where $tw = f(k)$ (specifically $tw = O(k^2)$)
- Furthermore, $\Gamma(G) \leq tw \log(|V(G)|) = O(k^2 \log n)$.
- However, the new instance must have $\Gamma(G)$ unbounded as function of k (otherwise we would get FPT algorithm). So $\Gamma(G) = \Theta(k^2 \log n)$.

Grundy with Supports and Targets

Definition:

- Given graph $G = (V, E)$
- For some vertices $T \subseteq V$ given “target” values $t : T \to \mathbb{N}$.
- For some vertices $S \subseteq V$ given “support“ sets $s : S \to 2^{\mathbb{N}}$.

We are looking for:

- A proper coloring $c : V \to \mathbb{N}$ of G
- Such that all $v \in T$ have $c(v) \geq t(T)$ (target achieving)
- For each $v \in V$, $s(v) \cup c^{-1}(N(v)) \supseteq \{1, \dots, c(v) - 1\}$.

Grundy with Supports and Targets

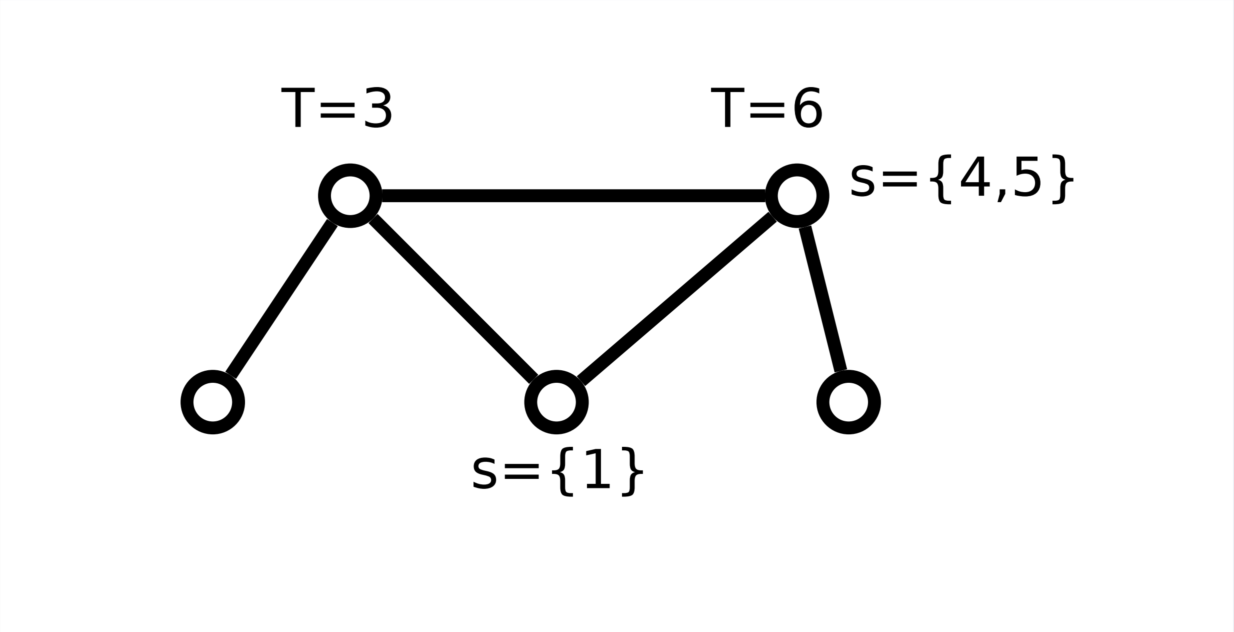
Definition:

- Given graph $G = (V, E)$
- For some vertices $T \subseteq V$ given “target” values $t : T \to \mathbb{N}$.
- For some vertices $S \subseteq V$ given “support“ sets $s : S \to 2^{\mathbb{N}}$.

We are looking for:

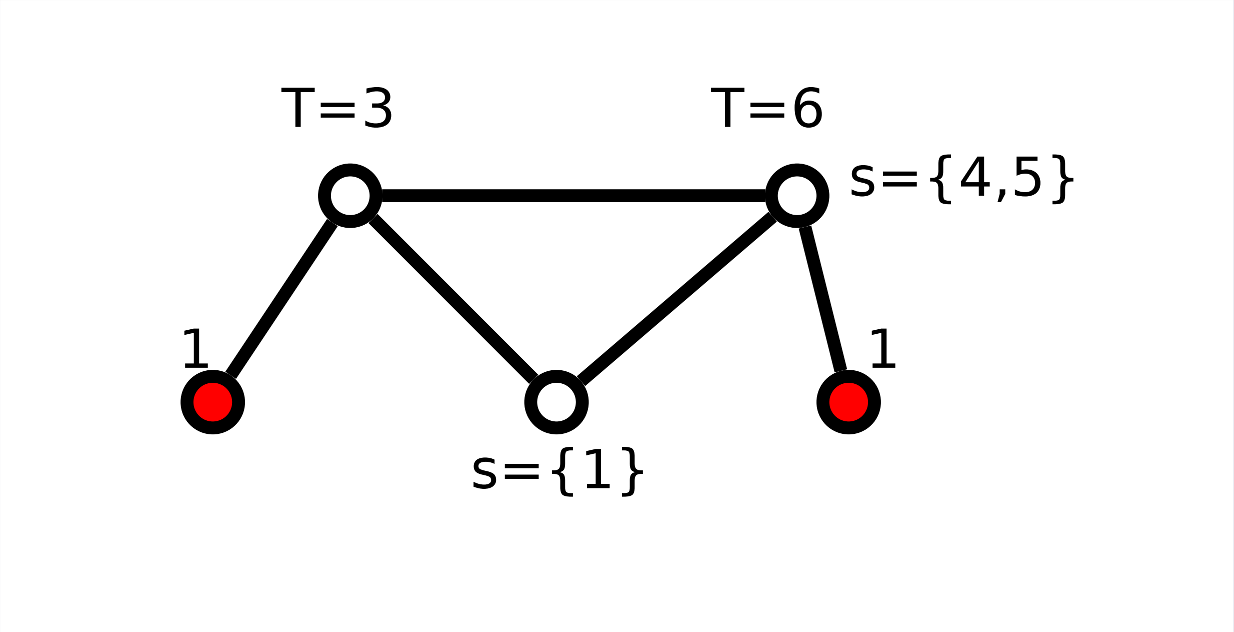
- A proper coloring $c : V \to \mathbb{N}$ of G
- Such that all $v \in T$ have $c(v) \geq t(T)$ (target achieving)
- For each $v \in V$, $s(v) \cup c^{-1}(N(v)) \supseteq \{1, \ldots, c(v) - 1\}$.
  - Explanation: if v has support $s(v)$, we can assume that v has a neighbor “pre-colored” with each color in $s(v)$, so we get these colors “for free”.

Grundy with Supports and Targets – Example



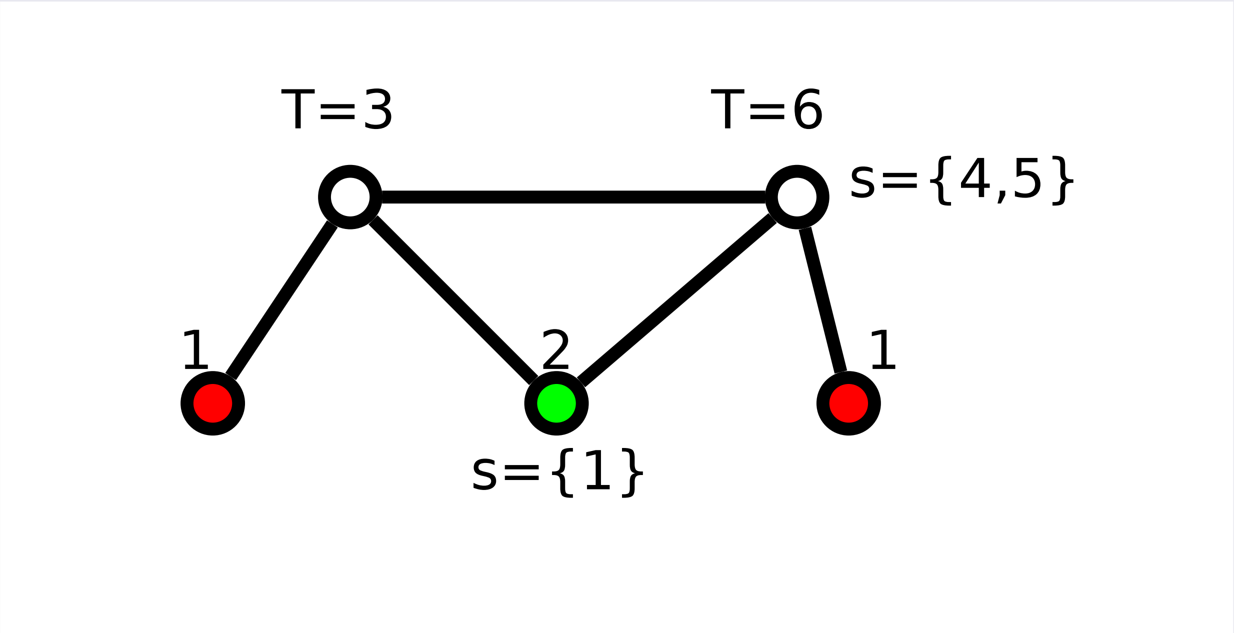
- Example of generalized problem instance.
- Two vertices have a target we want to achieve.
- Some vertices have a support set: we don't need to assign them neighbors of these colors to obtain a higher color.

Grundy with Supports and Targets – Example



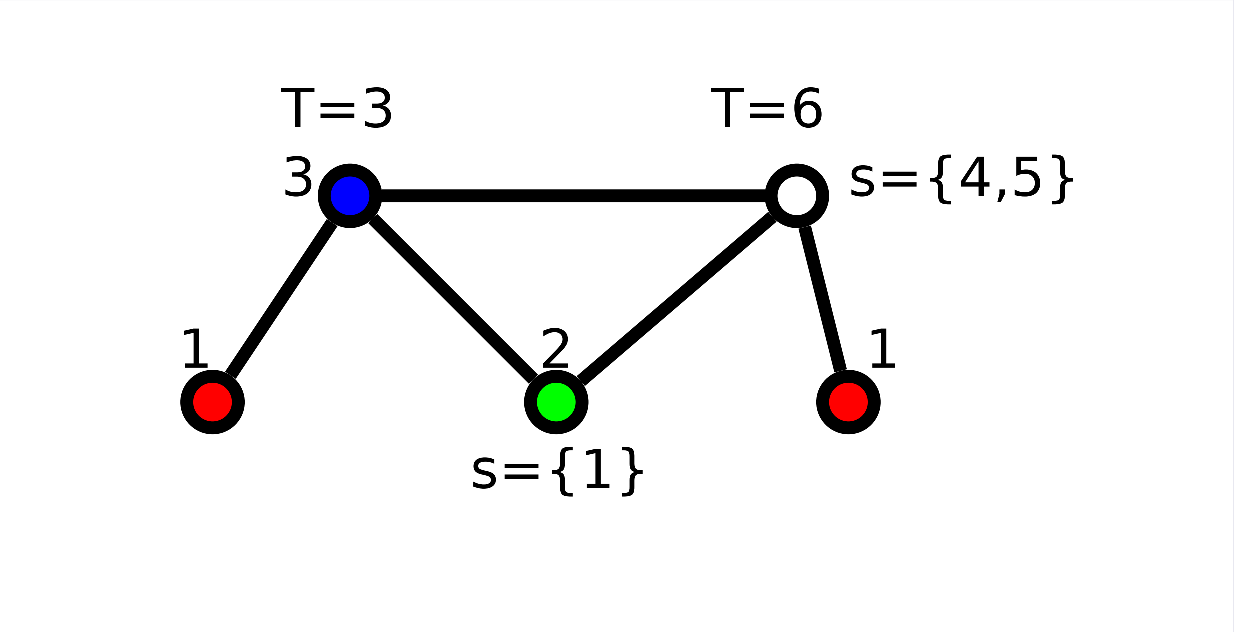
- Example of generalized problem instance.
- Two vertices have a target we want to achieve.
- Some vertices have a support set: we don't need to assign them neighbors of these colors to obtain a higher color.

Grundy with Supports and Targets – Example



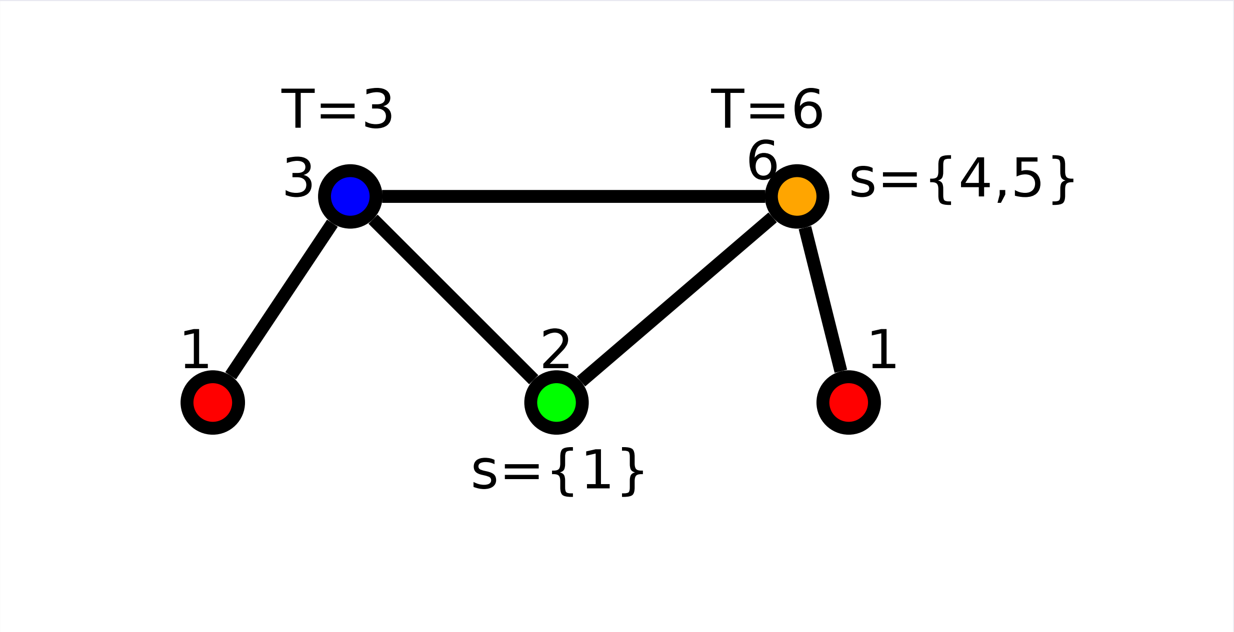
- Example of generalized problem instance.
- Two vertices have a target we want to achieve.
- Some vertices have a support set: we don't need to assign them neighbors of these colors to obtain a higher color.

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

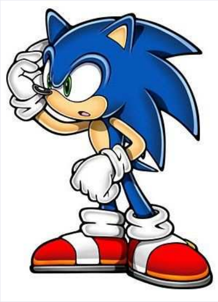


- Example of generalized problem instance.
- Two vertices have a target we want to achieve.
- Some vertices have a support set: we don't need to assign them neighbors of these colors to obtain a higher color.

W-hard by pathwidth?

- Recall: goal is to prove Grundy W-hard by treewidth
- Also: Grundy FPT by pathwidth
- We have an intermediate problem, and we want to prove that it is W-hard by **pathwidth**

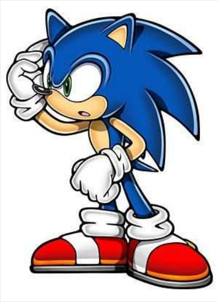
  - Why?
  - If we can reduce this to Grundy, why is Grundy not W-hard by pathwidth?



W-hard by pathwidth?

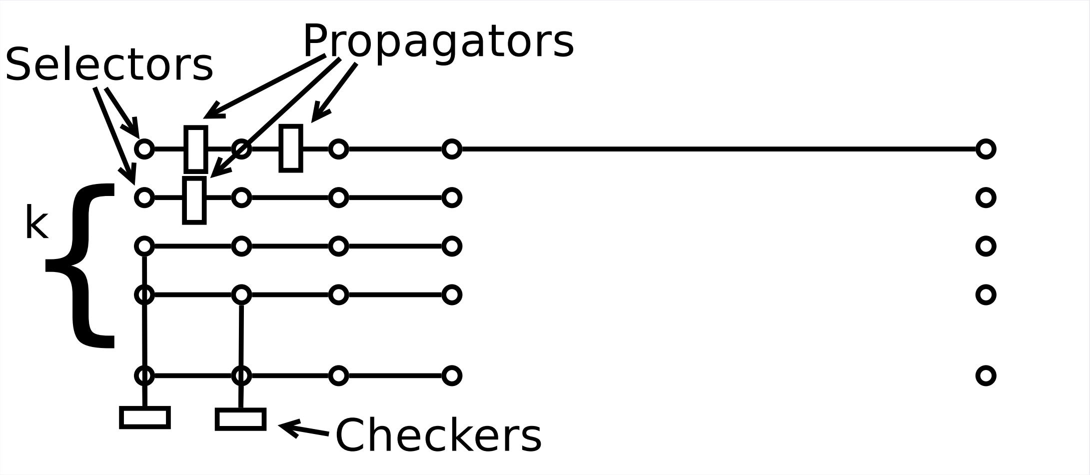
- Recall: goal is to prove Grundy W-hard by treewidth
- Also: Grundy FPT by pathwidth
- We have an intermediate problem, and we want to prove that it is W-hard by **pathwidth**

  - Why?
  - If we can reduce this to Grundy, why is Grundy not W-hard by pathwidth?



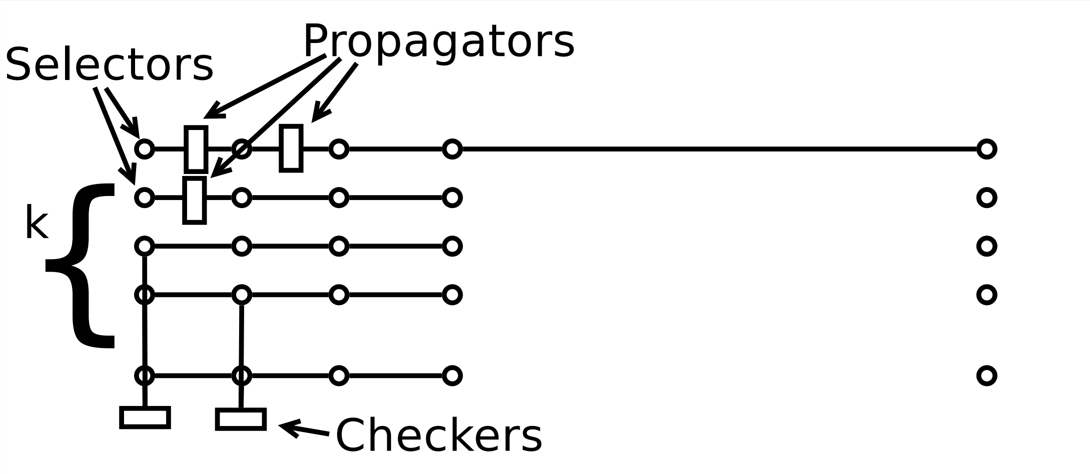
- Reduction will follow standard scheme with $k \times n$ grid
  - Hence, hardness for both pathwidth and treewidth for **Generalized Grundy**
- In GwTS$\rightarrow$Grundy, supports will be implemented using binomial trees
  - Binomial trees have unbounded pathwidth!
  - This breaks the reduction for pathwidth (but not treewidth!)
  - This is necessary (as we will see)!

Outline of hardness for GwTS



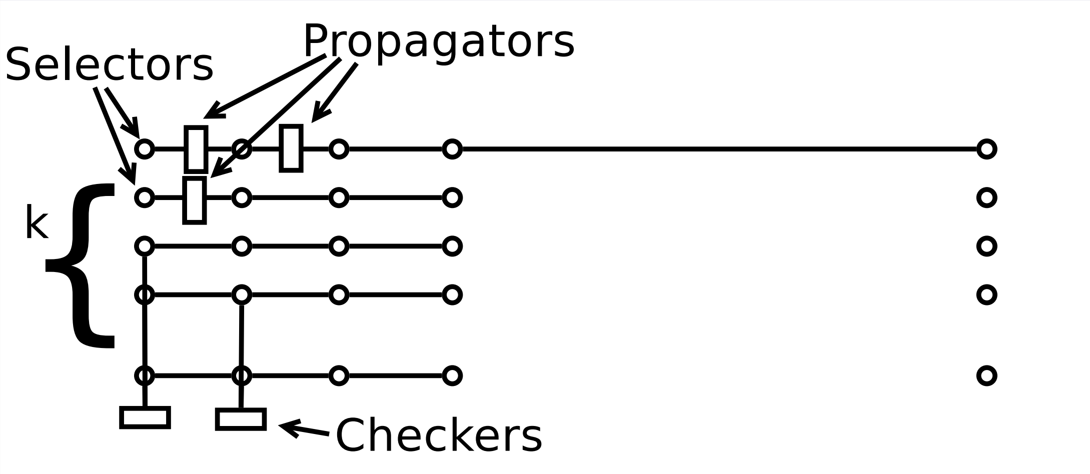
- $k \times m$ "grid" where each row represents a color class

Outline of hardness for GwTS



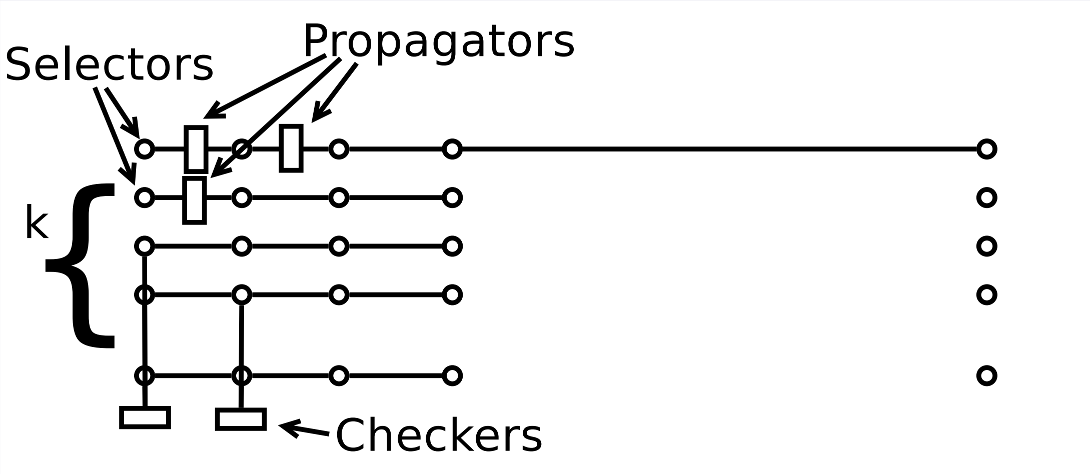
- $k \times m$ “grid” where each row represents a color class
- Selector gadget: has n “reasonable” Grundy colorings. Each encodes a selection of a vertex in original k-MCC instance.

Outline of hardness for GwTS



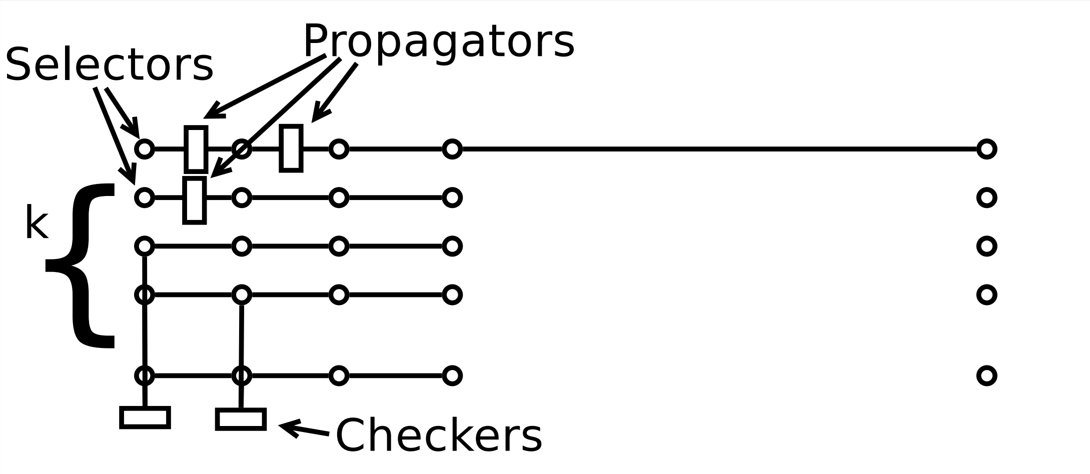
- $k \times m$ "grid" where each row represents a color class
- Selector gadget: has n "reasonable" Grundy colorings. Each encodes a selection of a vertex in original k-MCC instance.
- Propagator gadget: makes sure consecutive selectors encode same vertex.

Outline of hardness for GwTS



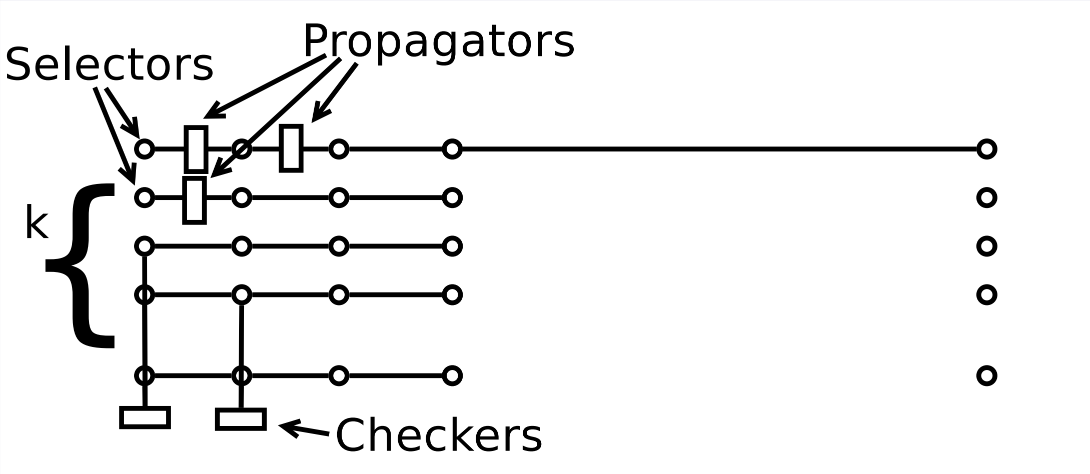
- $k \times m$ "grid" where each row represents a color class
- Selector gadget: has n "reasonable" Grundy colorings. Each encodes a selection of a vertex in original k-MCC instance.
- Propagator gadget: makes sure consecutive selectors encode same vertex.
- Checker gadget: one for each edge of G. Connected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

Outline of hardness for GwTS

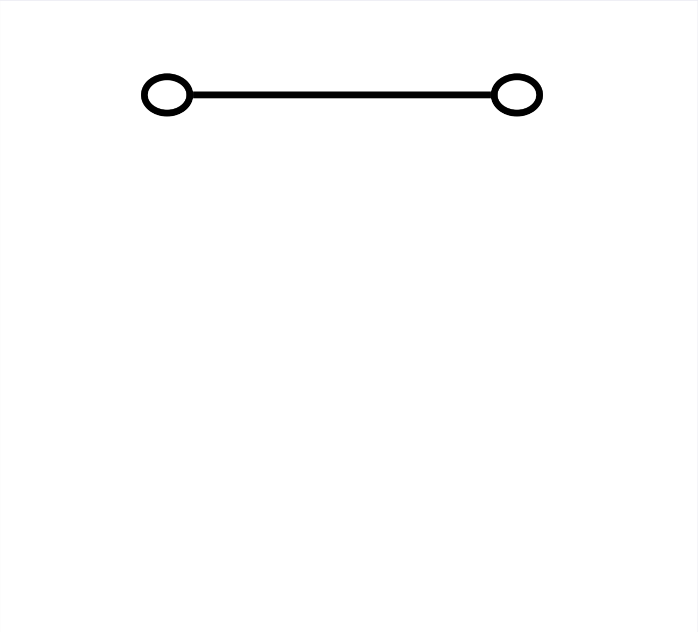


- $k \times m$ “grid” where each row represents a color class
- Selector gadget: has n “reasonable” Grundy colorings. Each encodes a selection of a vertex in original k-MCC instance.
- Propagator gadget: makes sure consecutive selectors encode same vertex.
- Checker gadget: one for each edge of G. Connected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

Outline of hardness for GwTS



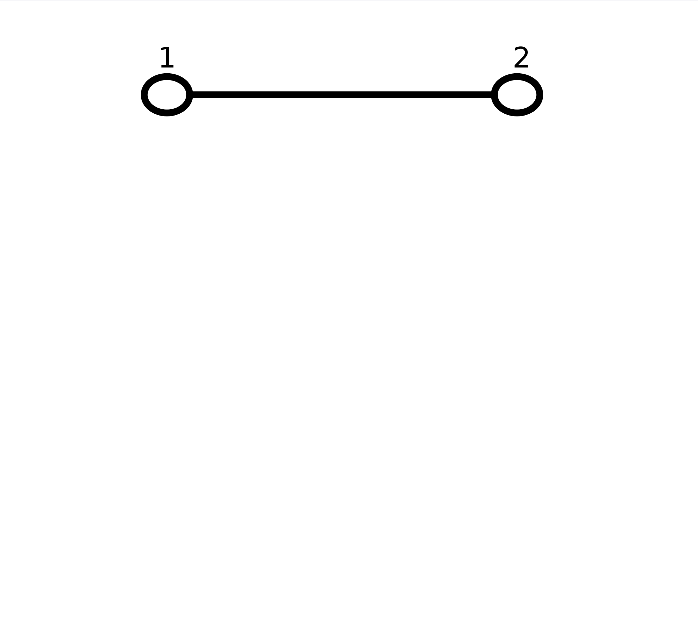
- $k \times m$ "grid" where each row represents a color class
- Selector gadget: has n "reasonable" Grundy colorings. Each encodes a selection of a vertex in original k-MCC instance.
- Propagator gadget: makes sure consecutive selectors encode same vertex.
- Checker gadget: one for each edge of G. Connected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



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.

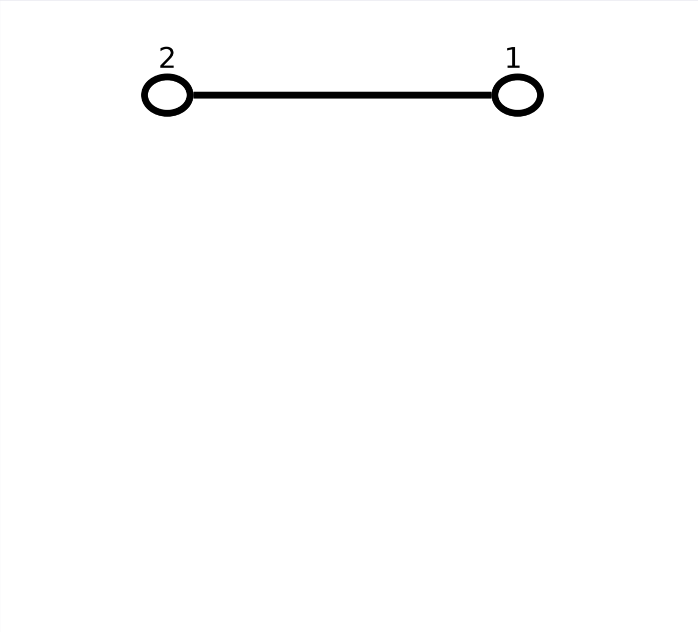
Selector Gadget



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.

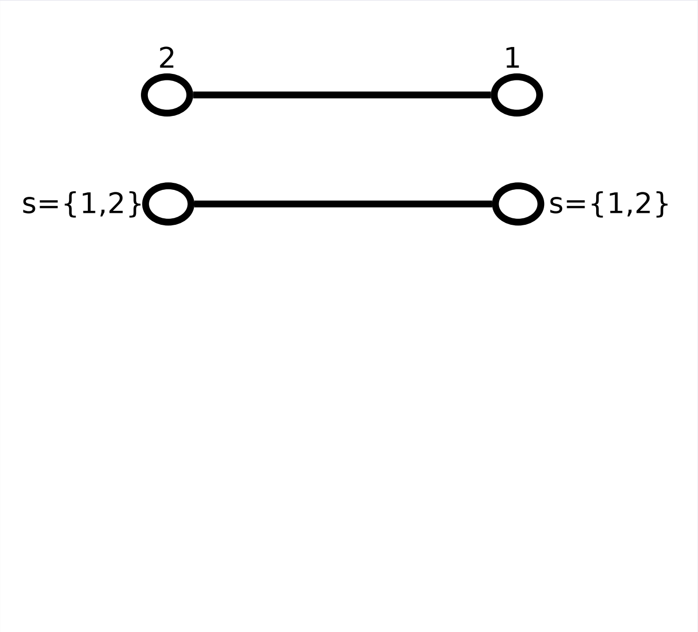
Selector Gadget



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.

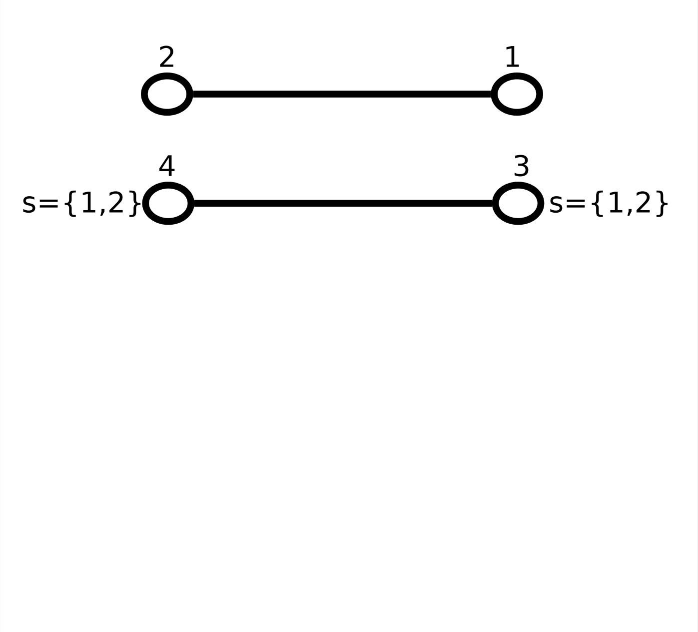
Selector Gadget



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.

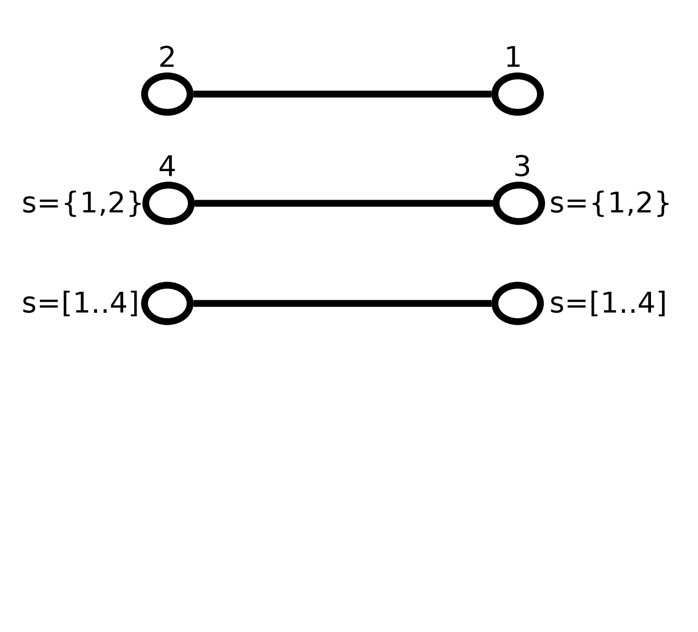
Selector Gadget



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.

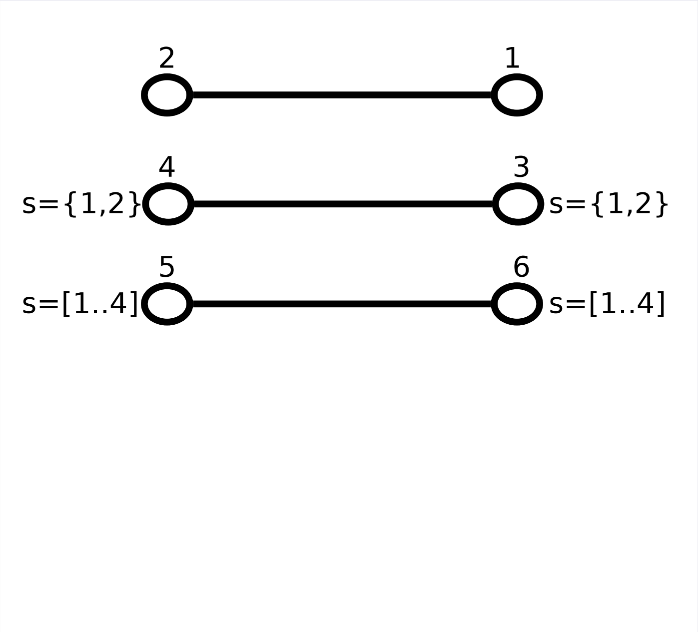
Selector Gadget



Intuition:

- We construct $\log n$ independent edges, numbered $1 \ldots \log n$.
- 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.

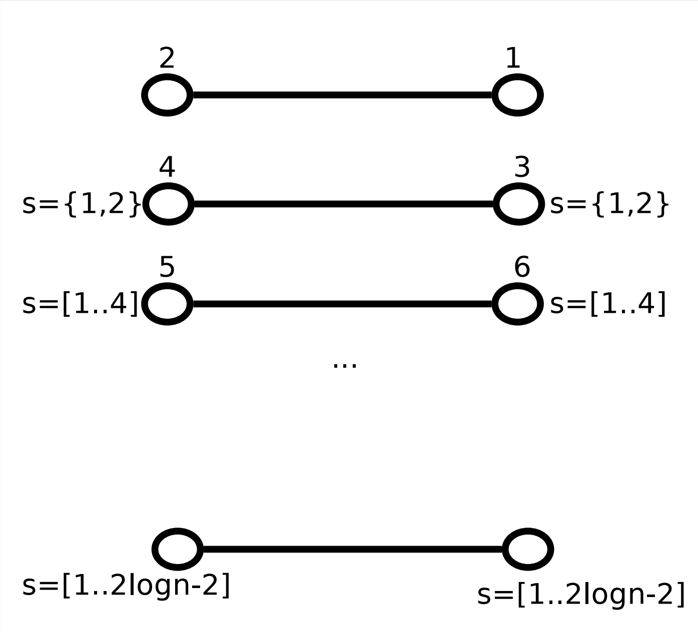
Selector Gadget



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.

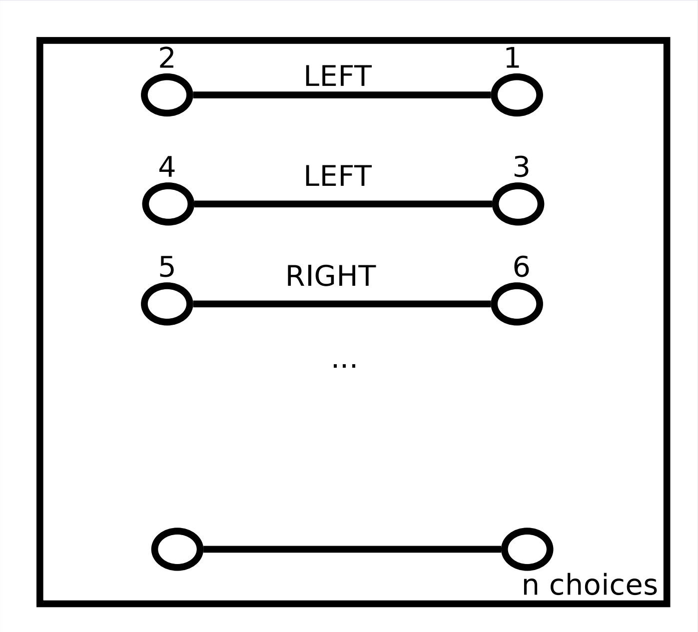
Selector Gadget



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.

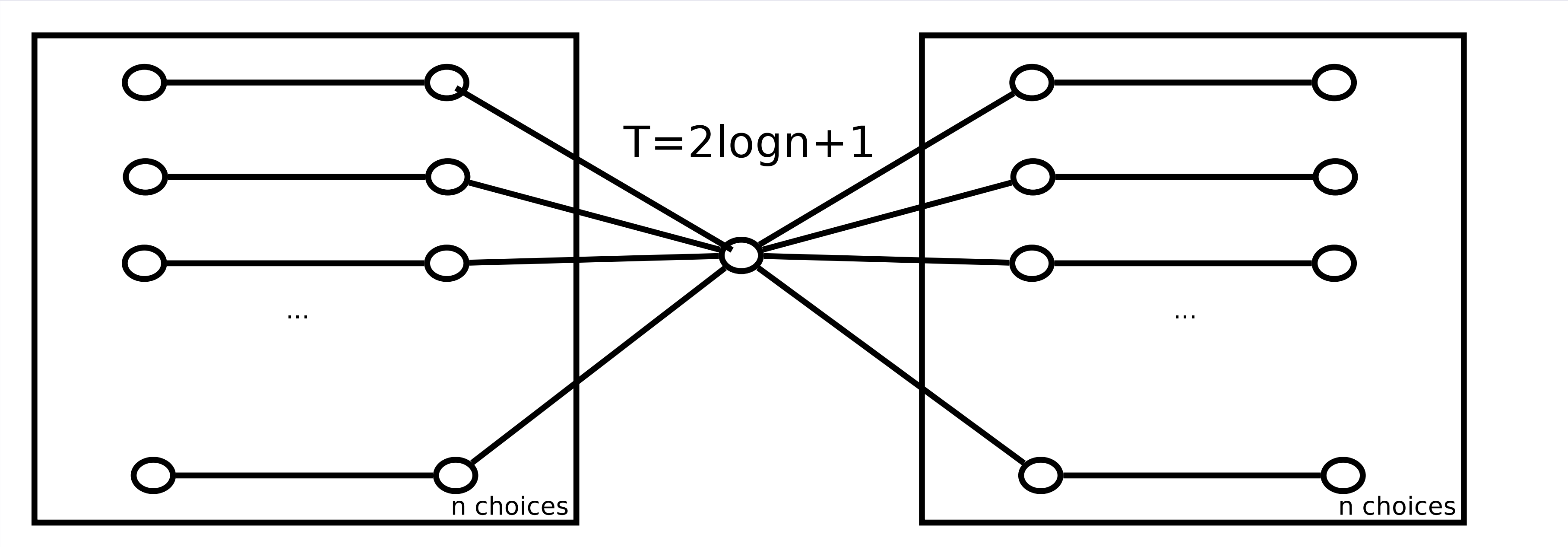
Selector Gadget



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.

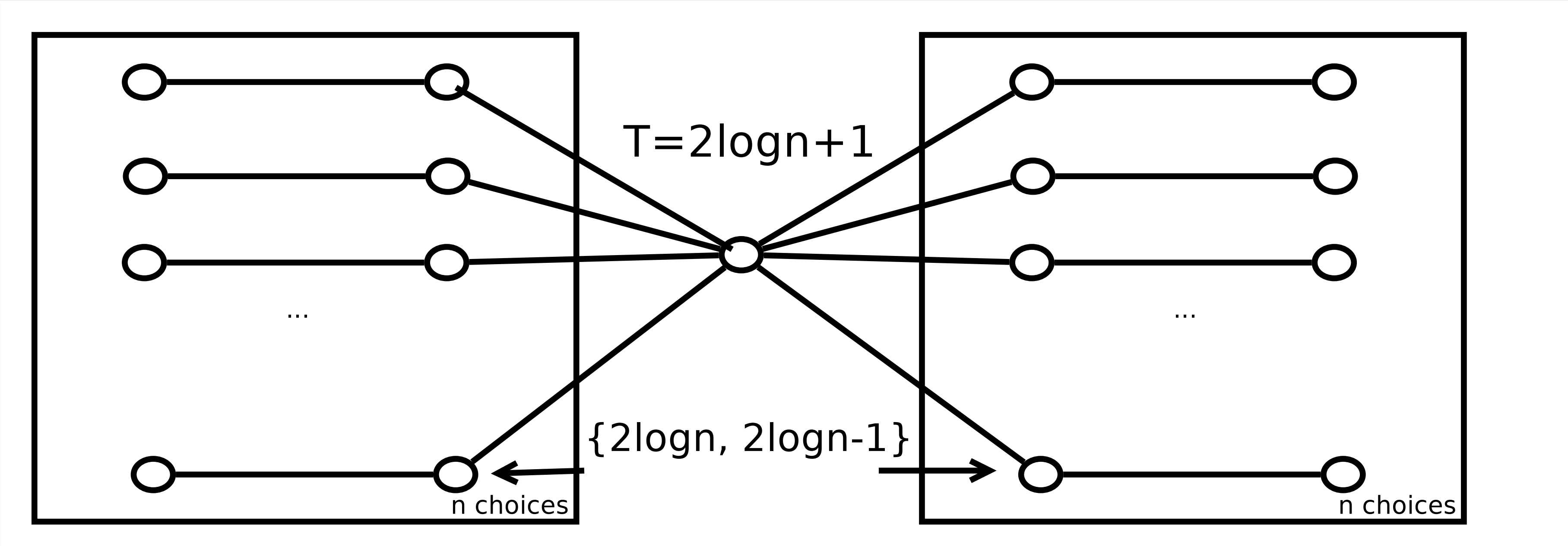
Propagator Gadget



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.

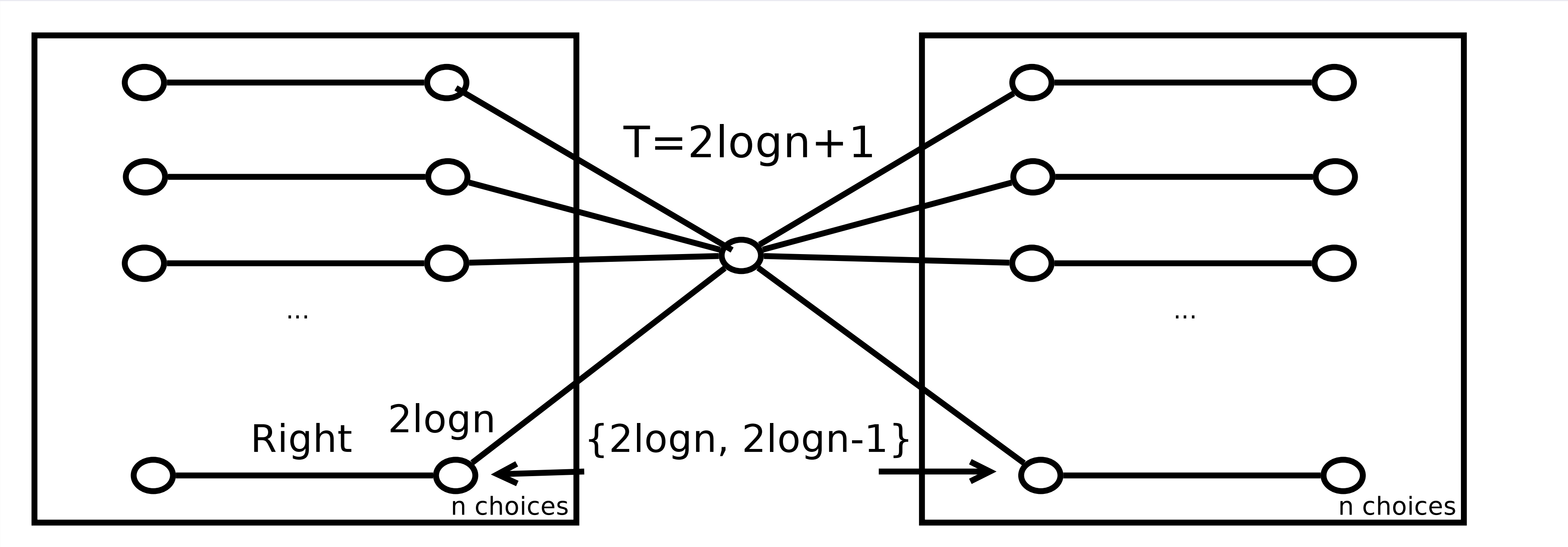
Propagator Gadget



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.

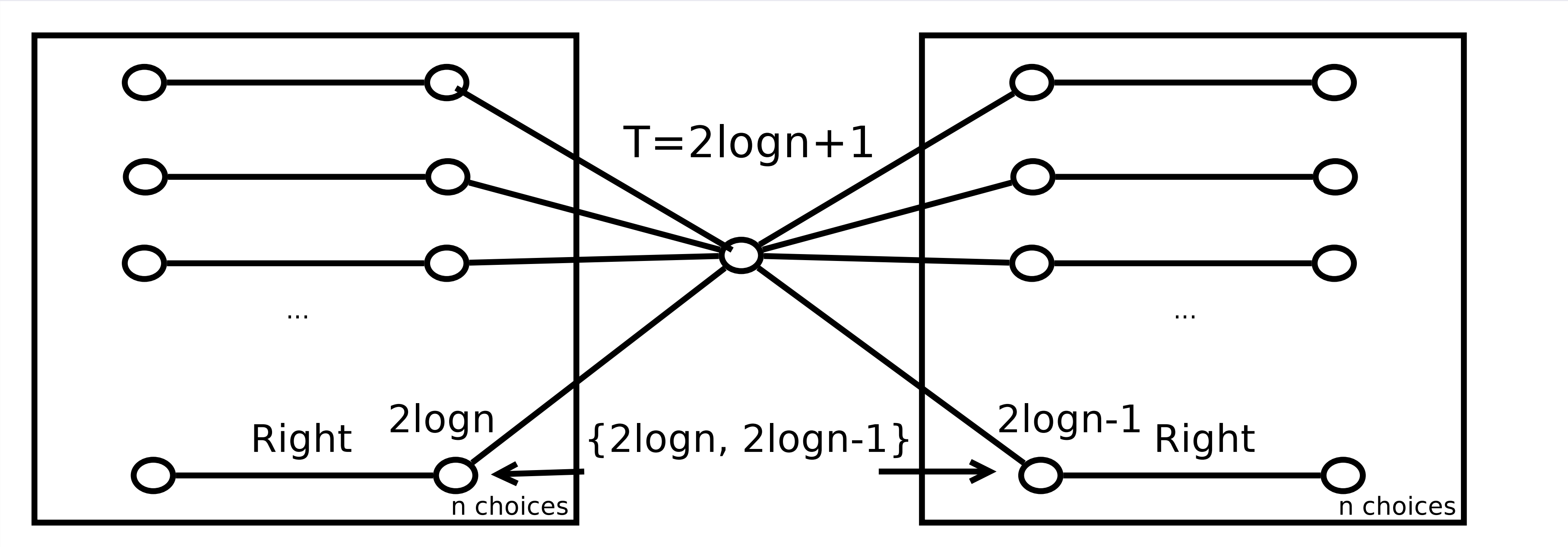
Propagator Gadget



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.

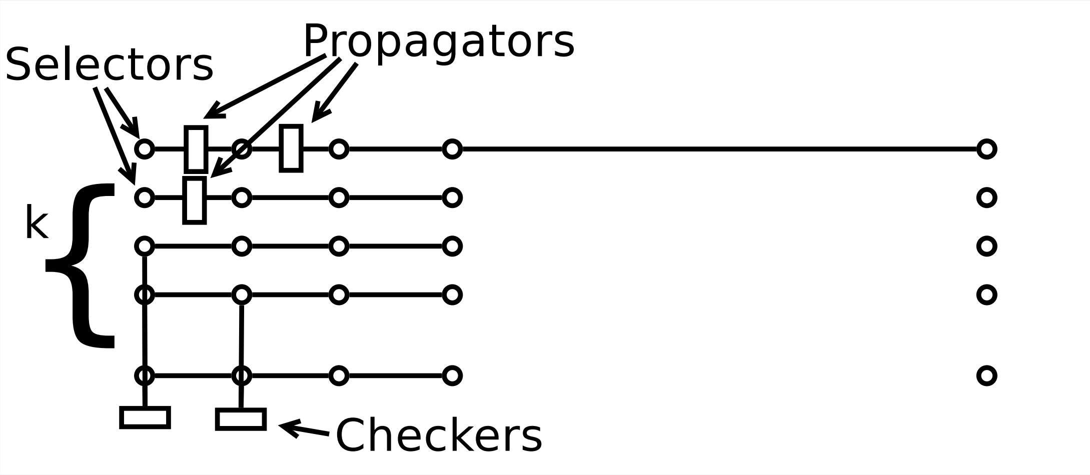
Propagator Gadget



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.

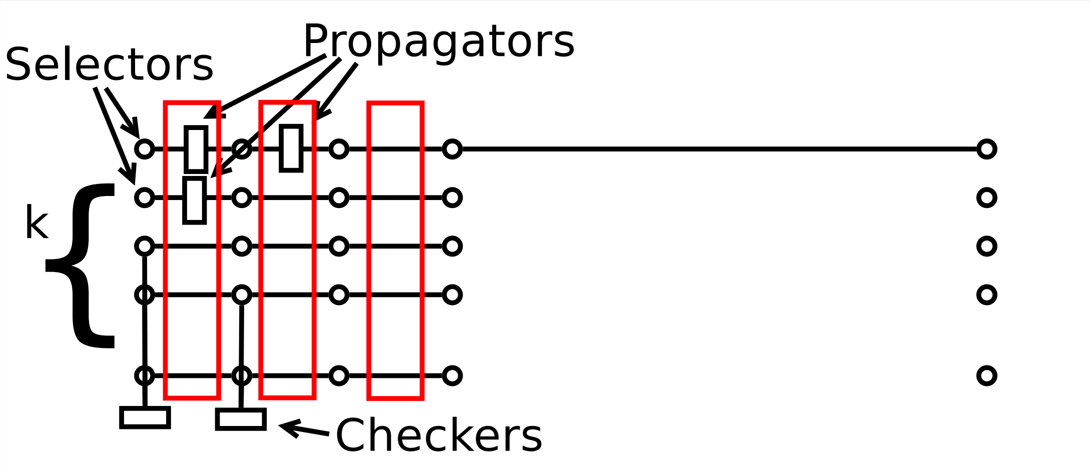
We're on the right track!



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!

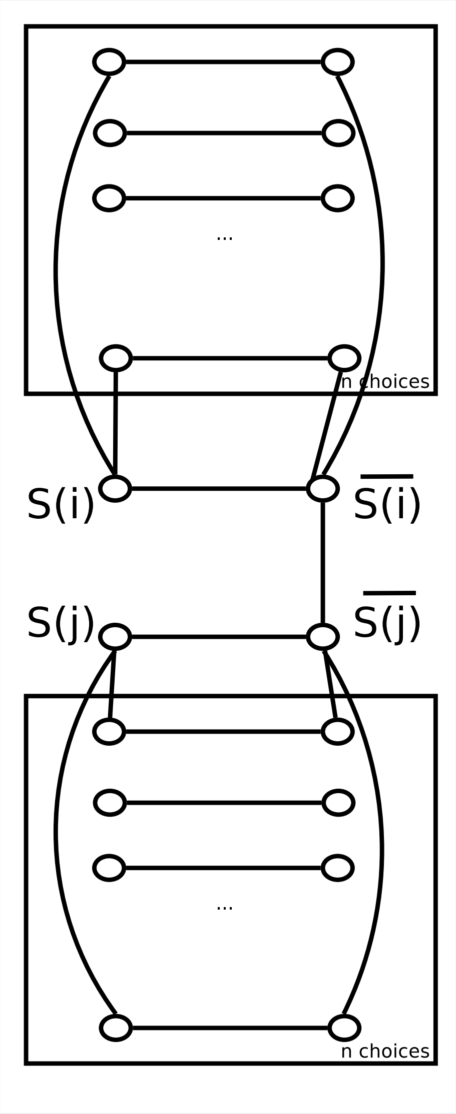
We're on the right track!



How is this reduction going?

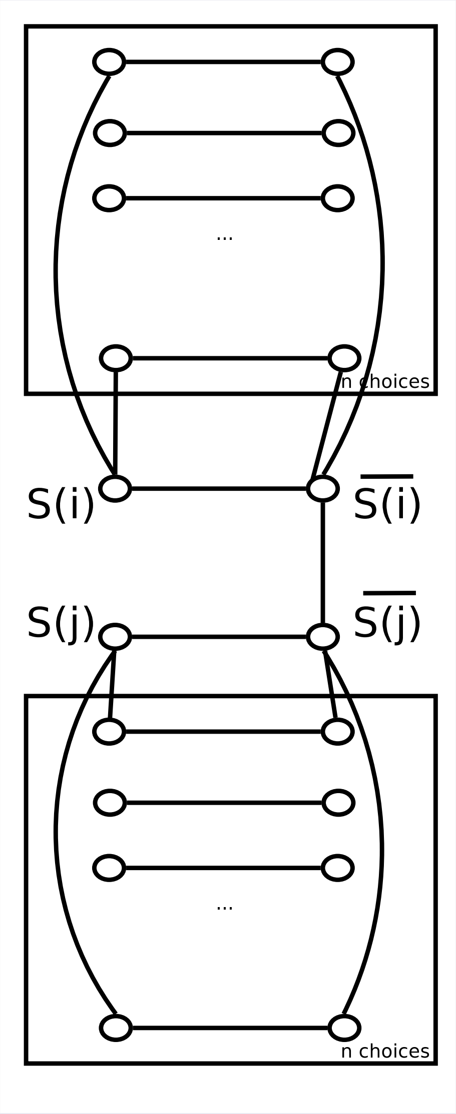
- Graph will have pathwidth $\approx k$
  - Propagators are vertices, form separators, bags of decomposition
- Information encoded?
  - Bottleneck of DP: must remember set of colors seen
  - Encoding of selection: set of colors seen by propagator to its left
  - Makes sense!

Checkers



- Checker is a path on 4 vertices connected to two selectors (one on each side).
- Goal: checker represents edge (i, j). A vertex will receive color $2 \log n + 3$ if and only if we have selected i, j on selectors.
- $S(i)$: support of all colors in $\{1, \dots, 2 \log n\}$ missing from left if we encode i.
- To complete the check, we make a super-checker for each pair (i, j) of color classes and connect it to all checkers of this pair.
- Super-checker has target $2 \log n + 4$ and support $\{1, \dots, 2 \log n + 2\}$.
  - Will achieve target if and only if we selected an edge from this pair.

Checkers



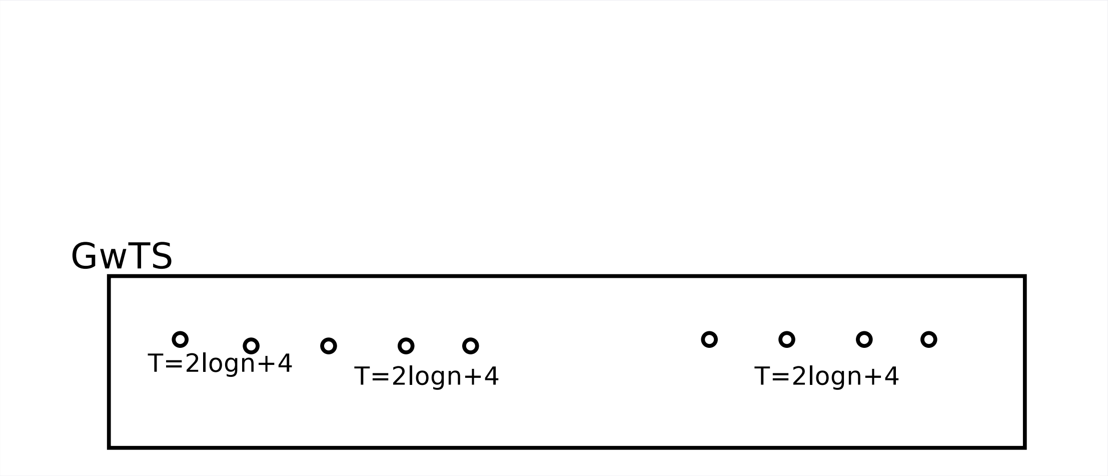
- Checker is a path on 4 vertices connected to two selectors (one on each side).
- Goal: checker represents edge (i, j). A vertex will receive color $2 \log n + 3$ if and only if we have selected i, j on selectors.
- $S(i)$: support of all colors in $\{1, \ldots, 2 \log n\}$ missing from left if we encode i.
- To complete the check, we make a super-checker for each pair (i, j) of color classes and connect it to all checkers of this pair.
- Super-checker has target $2 \log n + 4$ and support $\{1, \ldots, 2 \log n + 2\}$.
  - Will achieve target if and only if we selected an edge from this pair.
- That's it! GwTS is W-hard by pathwidth.

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!

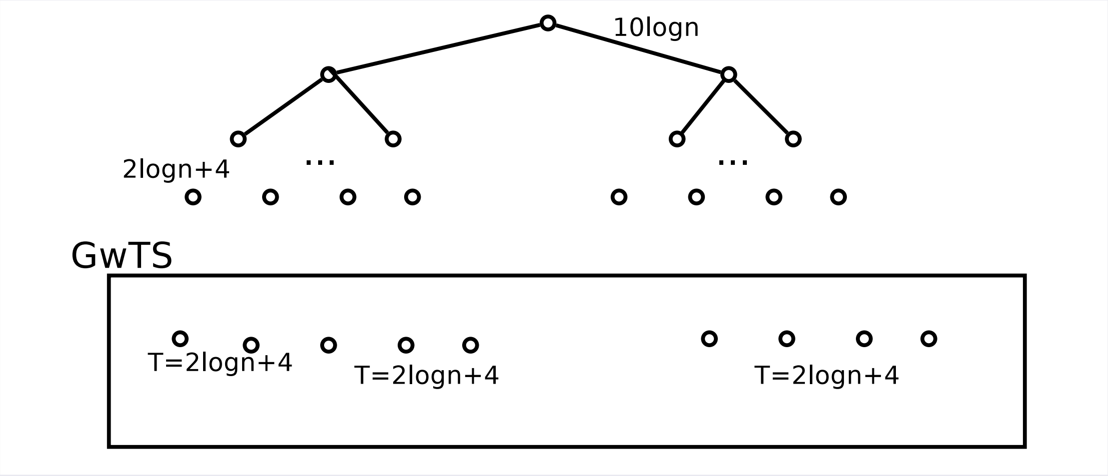
Regular Grundy

- 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!



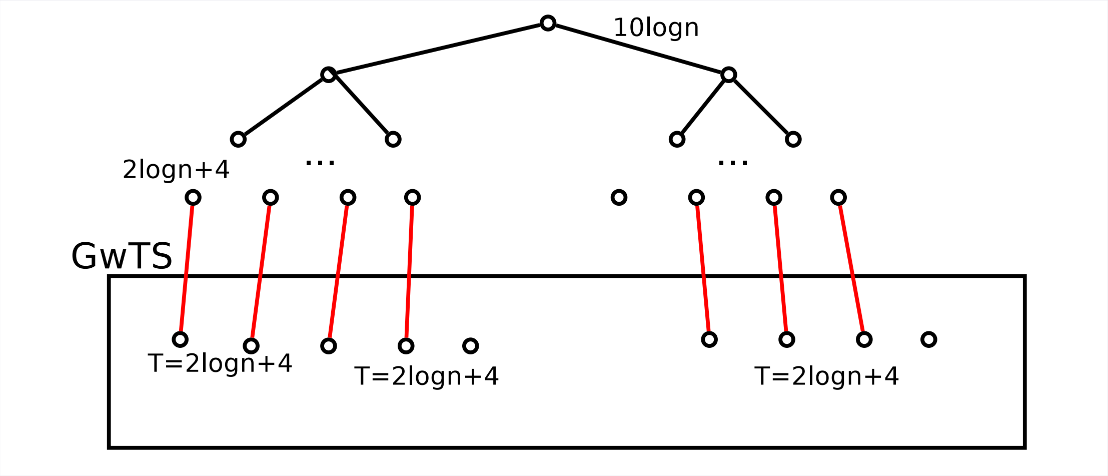
Regular Grundy

- 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

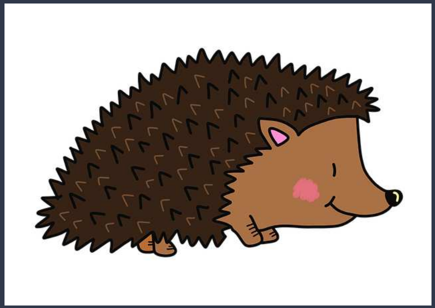
- To implement supports we attach binomial trees to supported vertices.
  - Does not increase treewidth.
  - Crucial: all supports are $O(\log n)$, so binomial trees have polynomial size.
- To implement targets we add a huge binomial tree $T_{10\log n}$.
- For each vertex with target $\leq 2\log n + 4$ 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!



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 ($O(\log n)$)
- Reduction leaves a gap in run-time
  - Treewidth of final graph: $O(k^2)$
  - $\rightarrow$ no $n^{o(\sqrt{tw})}$ algorithm under ETH
  - Can probably be improved easily to no $n^{o(tw/\log tw)}$ algorithm
  - But best algorithm known runs in n^{tw^2}!

FPT for pathwidth



A combinatorial bound

- We claim: for all G, $\Gamma(G) \leq 8pw(G)$.

A combinatorial bound

- We claim: for all G, $\Gamma(G) \leq 8pw(G)$.
- Recall: Statement is true for interval graphs.

A combinatorial bound

- We claim: for all G, $\Gamma(G) \leq 8pw(G)$.
- Recall: Statement is true for interval graphs.
- If claim is true, we are done:
  - DP algorithm runs in $2^{\Gamma tw}$
  - This becomes $2^{O(pw^2)}$ parameterized by pathwidth.

A combinatorial bound

- We claim: for all G, $\Gamma(G) \leq 8pw(G)$.
- Recall: Statement is true for interval graphs.
- If claim is true, we are done:
  - DP algorithm runs in $2^{\Gamma tw}$
  - This becomes $2^{O(pw^2)}$ parameterized by pathwidth.
- We will use the fact that if all bags of a path decomposition are cliques, then the graph is an interval graph.

- This claim was already proved in [Dujmovic, Joret, Wood SIDMA'12]

Reducing to Interval Graphs

Claim: $\Gamma(G) \leq 8pw(G)$

- Take an optimal Grundy coloring and an optimal path decomposition of G.
- We apply two transformations which may only increase Γ and decrease pw.
- In the end G becomes interval graph, so we get our bound.

Transformations:

1. If u, v in the same bag, have the same color, merge u, v.
2. If u, v in the same bag, have different color, add edge (u, v).

Reducing to Interval Graphs

Claim: $\Gamma(G) \leq 8pw(G)$

- Take an optimal Grundy coloring and an optimal path decomposition of G.
- We apply two transformations which may only increase Γ and decrease pw.
- In the end G becomes interval graph, so we get our bound.

Transformations:

1. If u, v in the same bag, have the same color, merge u, v.
2. If u, v in the same bag, have different color, add edge (u, v).

Rule 1 is safe:

- Coloring remains valid Grundy coloring.
- Path decomposition remains valid, width may only decrease.

Reducing to Interval Graphs

Claim: $\Gamma(G) \leq 8pw(G)$

- Take an optimal Grundy coloring and an optimal path decomposition of G.
- We apply two transformations which may only increase Γ and decrease pw.
- In the end G becomes interval graph, so we get our bound.

Transformations:

1. If u, v in the same bag, have the same color, merge u, v.
2. If u, v in the same bag, have different color, add edge (u, v).

Rule 2 is safe:

- Coloring remains valid Grundy coloring.
- Path decomposition remains valid, width same.

Reducing to Interval Graphs

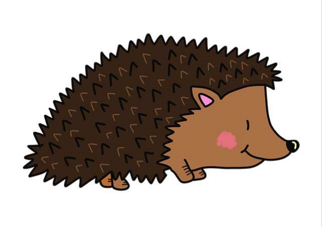
Claim: $\Gamma(G) \leq 8pw(G)$

- Take an optimal Grundy coloring and an optimal path decomposition of G.
- We apply two transformations which may only increase Γ and decrease pw.
- In the end G becomes interval graph, so we get our bound.

Transformations:

1. If u, v in the same bag, have the same color, merge u, v.
2. If u, v in the same bag, have different color, add edge (u, v).

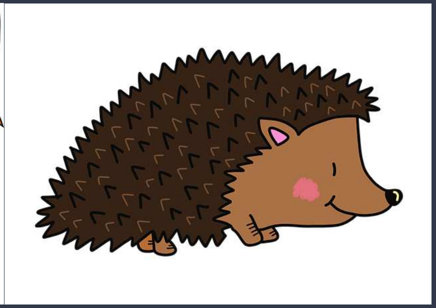
- Final graph G' has $\Gamma(G') \geq \Gamma(G)$ and $pw(G') \leq pw(G)$.
- G' is interval graph, so $\Gamma(G') \leq 8pw(G')$.
- We get $\Gamma(G) \leq 8pw(G)$.



Comparison with treewidth

- Recall: binomial trees "break" reduction for pathwidth.
- Why could we not replace them with something else?
  - Besides the fact that the problem is FPT!?!...
- Binomial trees = graphs with Γ unbounded but treewidth $O(1)$.
- For a pathwidth reduction we need Γ unbounded but **pathwidth** $O(1)$.
  - Such graphs do not exist!
- This is "why" Grundy is FPT for pathwidth but W-hard for treewidth.

Conclusions

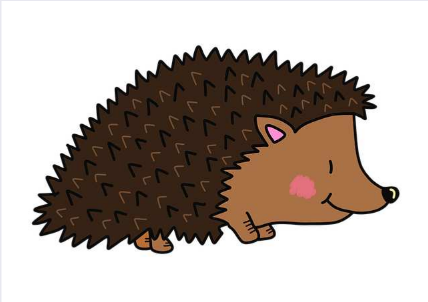



Conclusions – Open Questions

- 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!
Questions?