

# *Parameterized Approximation Schemes Using Graph Widths*

Michael Lampis  
Research Institute for Mathematical Sciences  
Kyoto University



May 13, 2014

Topic of this talk:

## Randomized Parameterized Approximation Algorithms

- **Approximation:** Ratio of  $(1 + \epsilon)$
- **Parameterized:** Parameter is tree/cliq-ue-width
- **Randomized:** Probabilistic rounding



Message: A generic technique for dealing with problems which are:

- W-hard: need time  $n^k$  to solve exactly
- APX-hard: cannot be  $(1 + \epsilon)$  approximated in poly time

Result: A natural  $(\log n / \epsilon)^{O(k)}$  algorithm with ratio  $(1 + \epsilon)$

# Overview

Topic of this talk:

## Randomized Parameterized Approximation Algorithms

- **Approximation:** Ratio
- **Parameterized:** Para
- **Randomized:** Probab



Message: A generic te

with

with problems which are:

- W-hard: need time  $n^k$
- APX-hard: cannot be (

in poly time

Result: A natural

with ratio  $(1 + \epsilon)$



# Overview

Topic of this talk:

## Randomized Parameterized Approximation Algorithms

- **Approximation:** Ratio
- **Parameterized:** Para
- **Randomized:** Probab



Message: A generic te

with

with problems which are:

- W-hard: need time  $n^k$
- APX-hard: cannot be (

in poly time

Result: A natural

with ratio  $(1 + \epsilon)$

More info: **arxiv** and [ICALP '14]



# What seems to be the problem?

Treewidth/Clique-width are big success stories in parameterized complexity

- Generic DP method solves **everything!**\*
- Meta-theorems: Lots of problems solvable in  $f(k)n$  time.

The problem:

- Often natural DP algorithm runs in  $n^k$ .
- Too slow to solve exactly!
- Problem is hard to approximate.



We extend the standard DP method to obtain FPT running time for many such problems, losing only  $(1 + \epsilon)$  in the solution quality.

# Two concrete problems

- Max Cut parameterized by clique-width
  - Given: Graph  $G(V, E)$  (along with a clique-width expression)
  - Wanted: A partition of  $V$  into  $L, R$  that maximizes edges cut.
  - Parameter: The clique-width of  $G$  ( $k$ ).



# Two concrete problems

- Max Cut parameterized by clique-width
  - Given: Graph  $G(V, E)$  (along with a clique-width expression)
  - Wanted: A partition of  $V$  into  $L, R$  that maximizes edges cut.
  - Parameter: The clique-width of  $G$  ( $k$ ).
  - "Easy"  $n^k$  DP algorithm, known to be essentially optimal [Fomin et al. SODA '10]



## Two concrete problems

- Max Cut parameterized by clique-width
  - Given: Graph  $G(V, E)$  (along with a clique-width expression)
  - Wanted: A partition of  $V$  into  $L, R$  that maximizes edges cut.
  - Parameter: The clique-width of  $G$  ( $k$ ).
  - "Easy"  $n^k$  DP algorithm, known to be essentially optimal [Fomin et al. SODA '10]
- Capacitated Dominating Set parameterized by treewidth
  - Given: Graph  $G(V, E)$ , capacity  $c : V \rightarrow \mathbb{N}$
  - Wanted: Min size dominating set + domination plan
  - ... selected vertex  $u$  can dominate at most  $c(u)$  vertices
  - Parameter: treewidth of  $G$  ( $k$ ).





# Two concrete problems

- Max Cut parameterized by clique-width
  - Given: Graph  $G(V, E)$  (along with a clique-width expression)
  - Wanted: A partition of  $V$  into  $L, R$  that maximizes edges cut.
  - Parameter: The clique-width of  $G$  ( $k$ ).
  - "Easy"  $n^k$  DP algorithm, known to be essentially optimal [Fomin et al. SODA '10]
- Capacitated Dominating Set parameterized by treewidth
  - Given: Graph  $G(V, E)$ , capacity  $c : V \rightarrow \mathbb{N}$
  - Wanted: Min size dominating set + domination plan
  - ... selected vertex  $u$  can dominate at most  $c(u)$  vertices
  - Parameter: treewidth of  $G$  ( $k$ ).
  - "Easy"  $C^k$  algorithm,  $C$  max capacity. Known to be W-hard [Dom et al. IWPEC '08]



# Treewidth - Pathwidth

Gentle definition of pathwidth  $k$ :

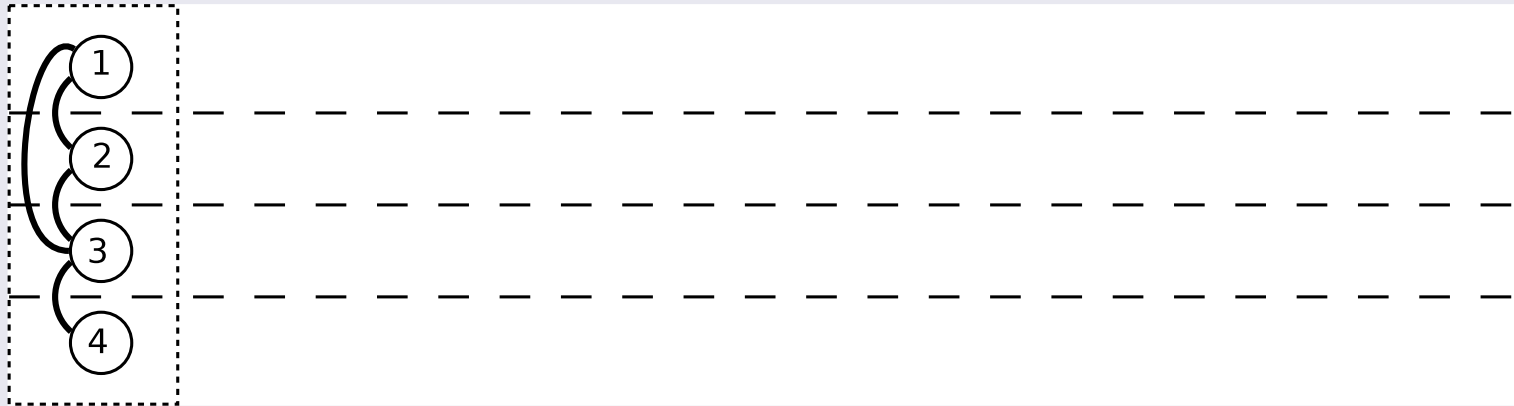
- We have  $k$  stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



# Treewidth - Pathwidth

Gentle definition of pathwidth  $k$ :

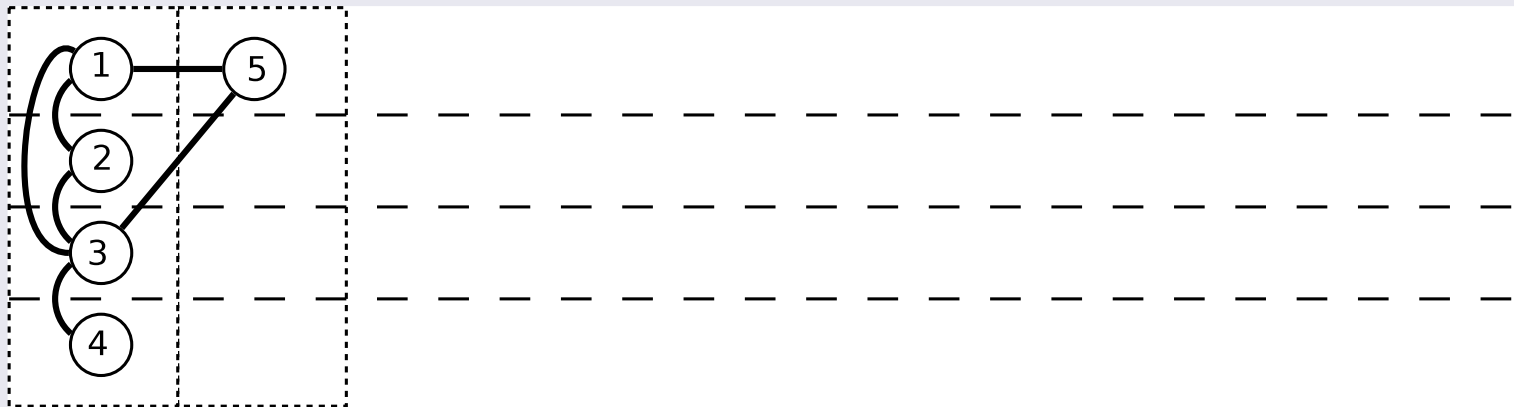
- We have  $k$  stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



# Treewidth - Pathwidth

Gentle definition of pathwidth  $k$ :

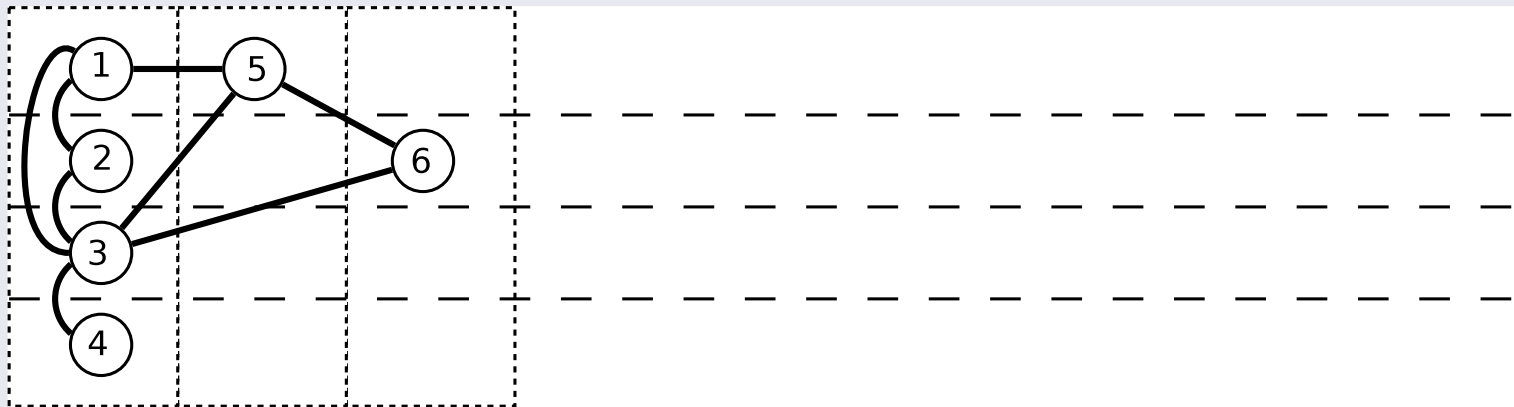
- We have  $k$  stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



# Treewidth - Pathwidth

Gentle definition of pathwidth  $k$ :

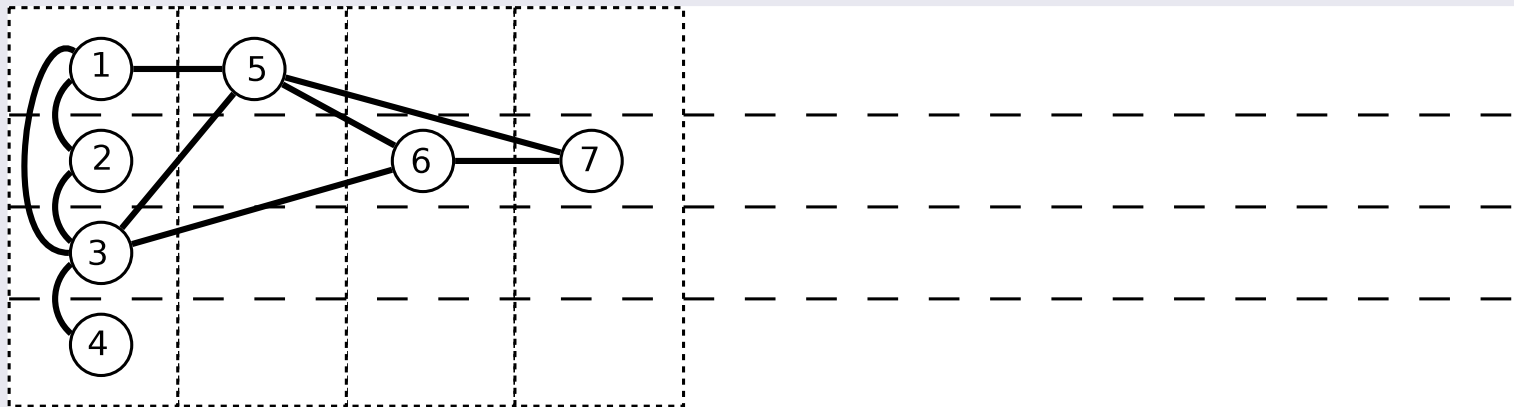
- We have  $k$  stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



# Treewidth - Pathwidth

Gentle definition of pathwidth  $k$ :

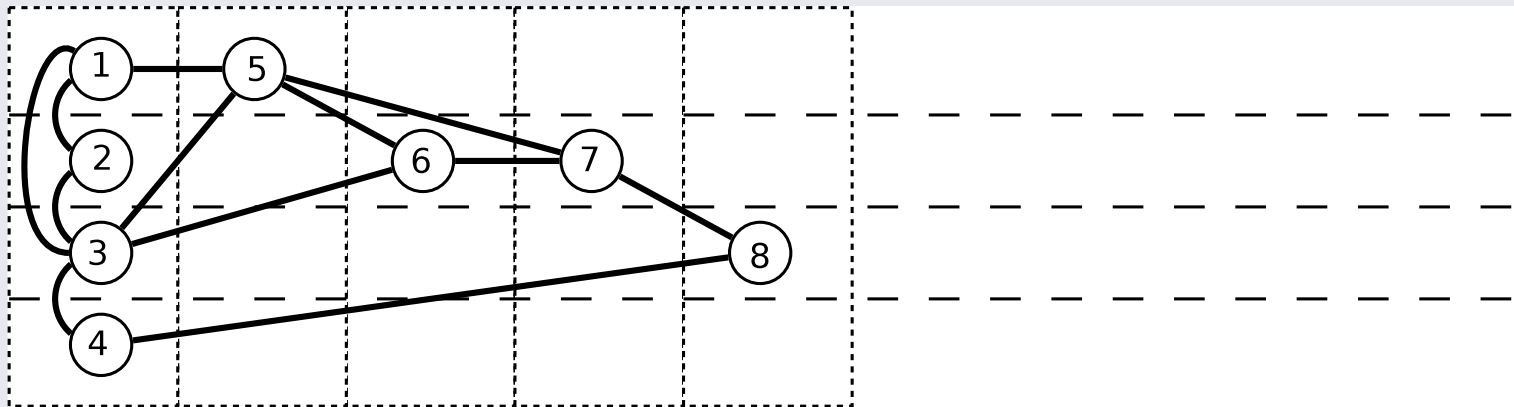
- We have  $k$  stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



# Treewidth - Pathwidth

Gentle definition of pathwidth  $k$ :

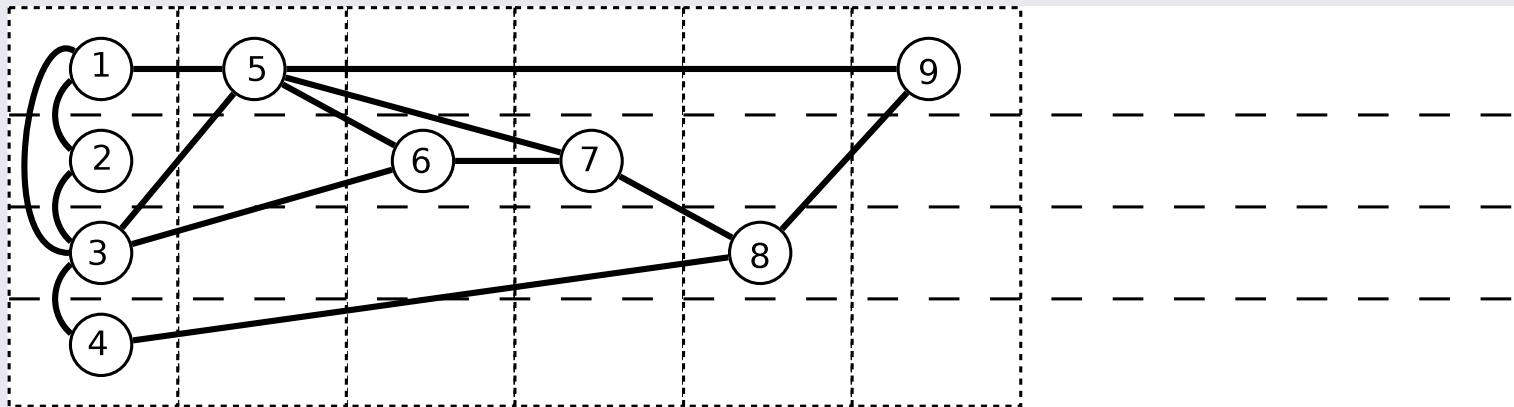
- We have  $k$  stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



# 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.

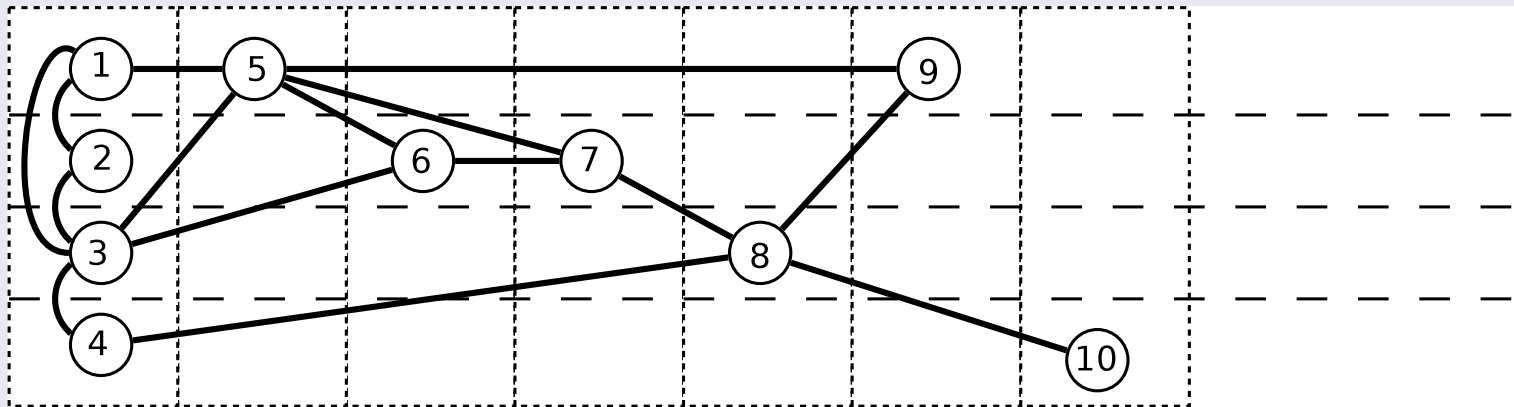




# Treewidth - Pathwidth

Gentle definition of pathwidth  $k$ :

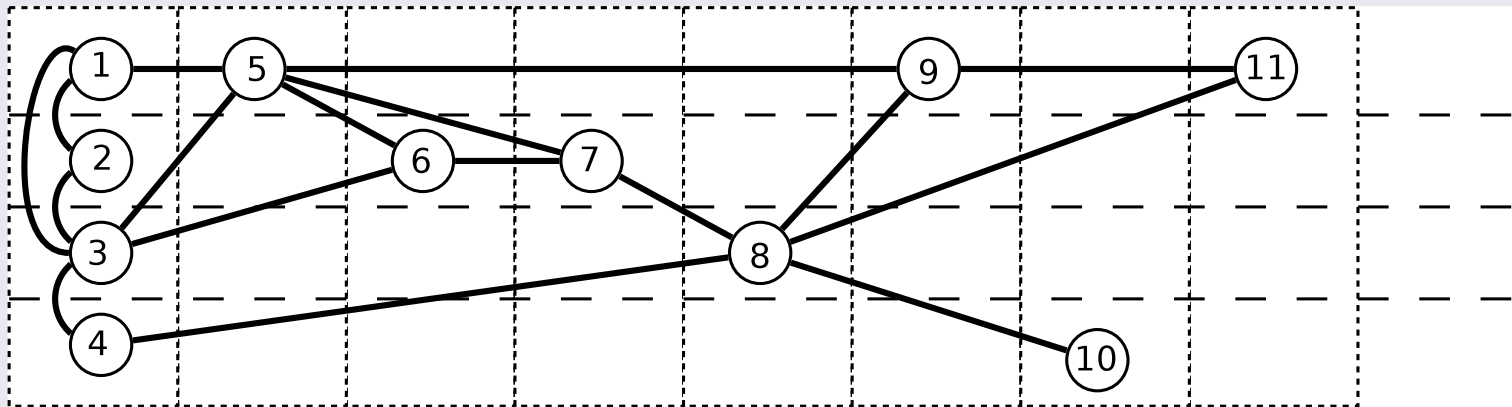
- We have  $k$  stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



# Treewidth - Pathwidth

Gentle definition of pathwidth  $k$ :

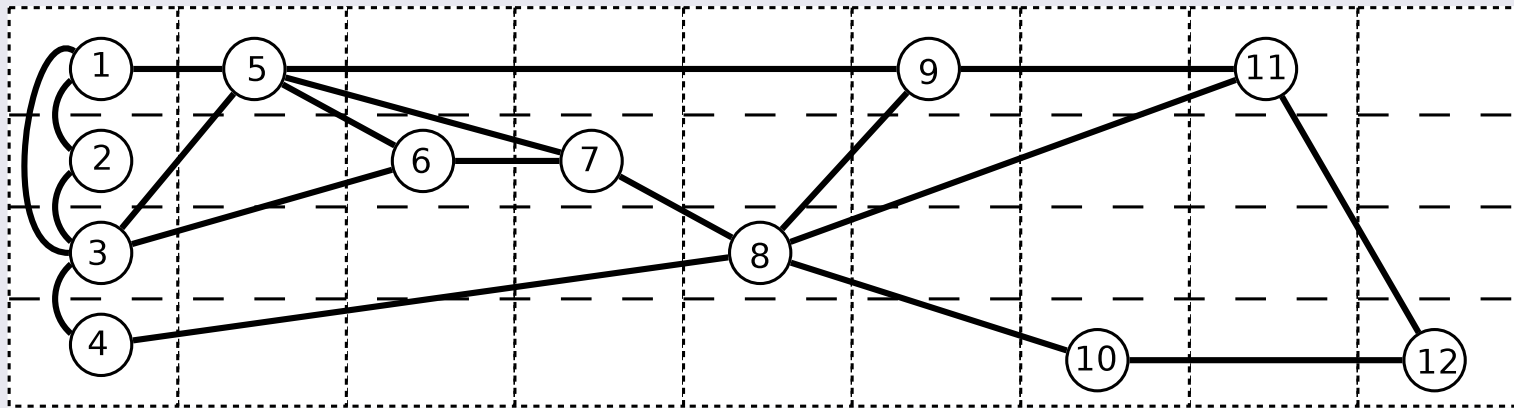
- We have  $k$  stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



# Treewidth - Pathwidth

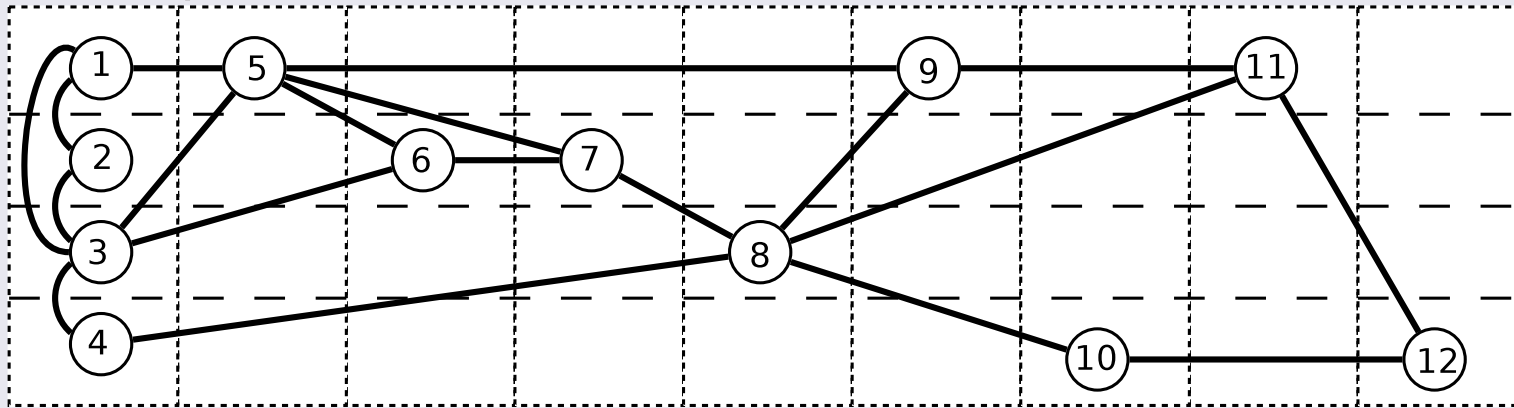
Gentle definition of pathwidth  $k$ :

- 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.



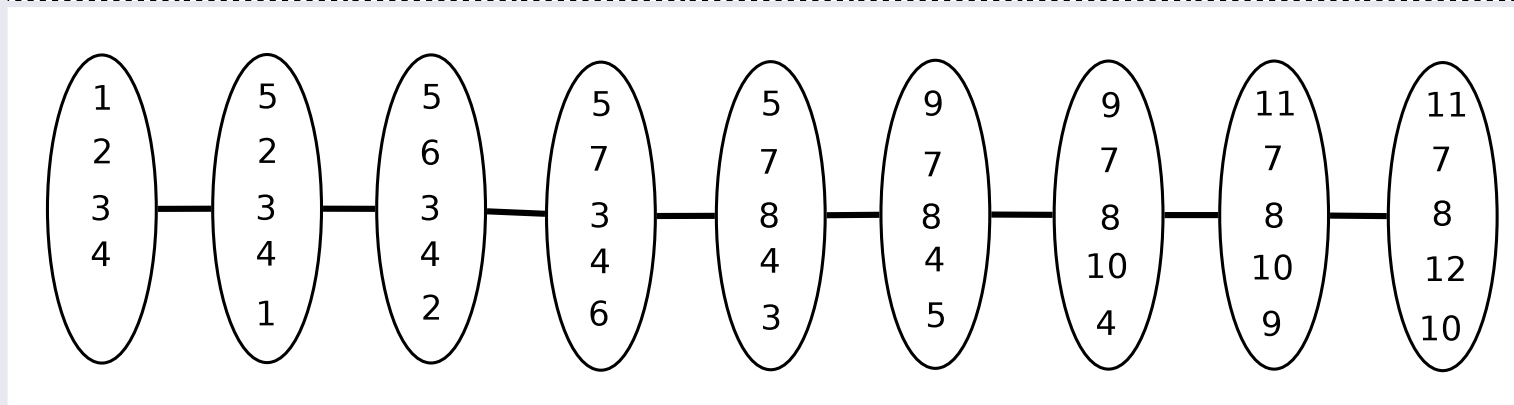
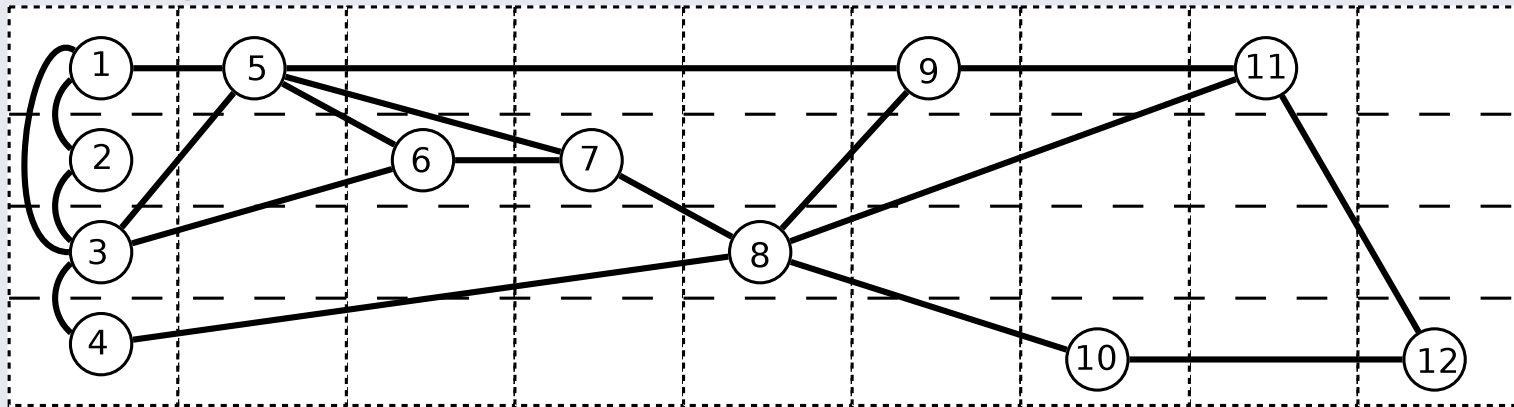
# Treewidth - Pathwidth

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.



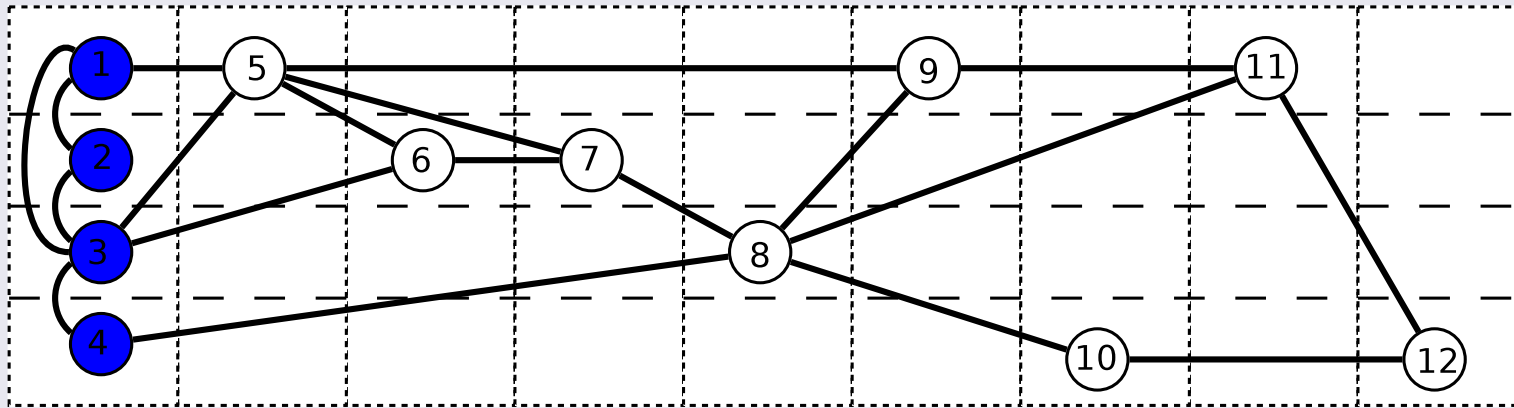
# Algorithmic view

The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.



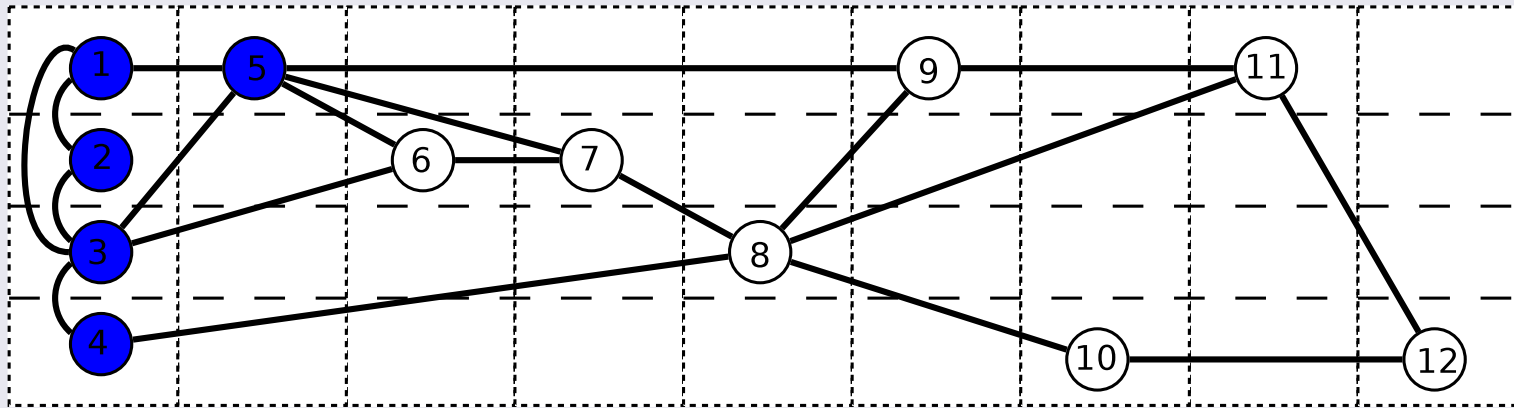
# Algorithmic view

The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.



# Algorithmic view

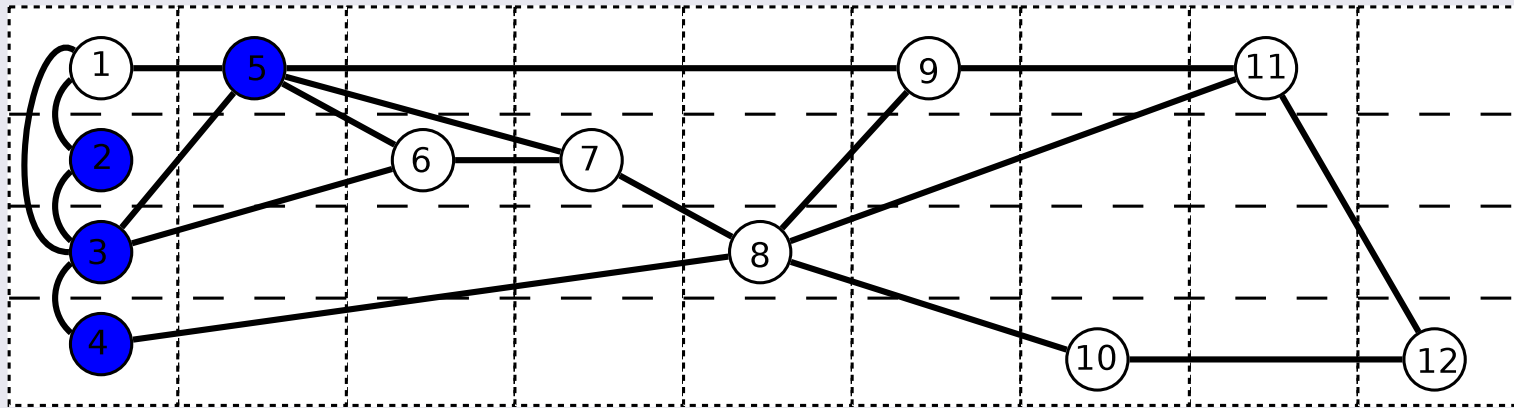
The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.





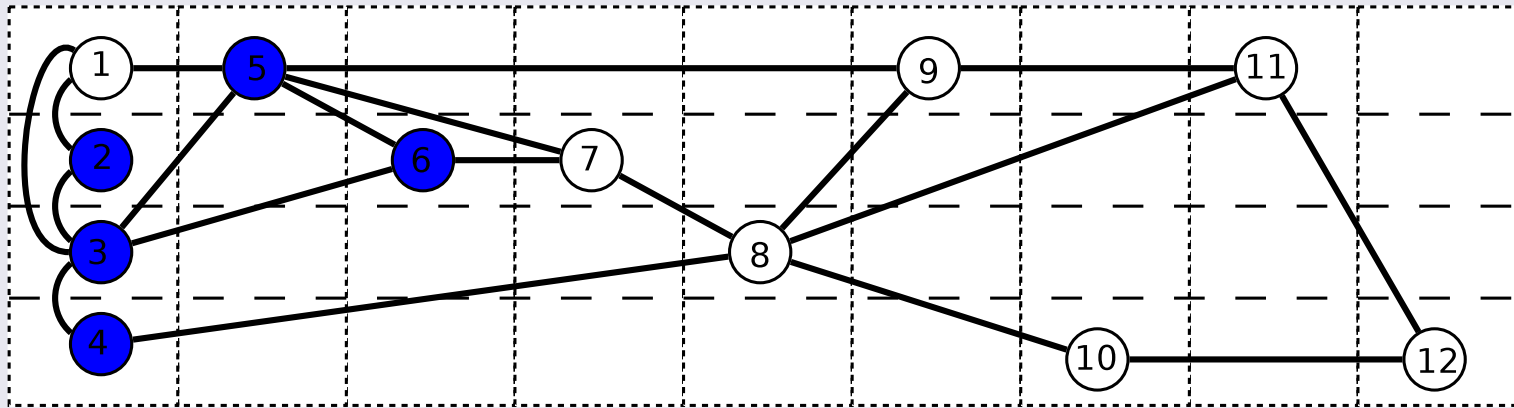
# Algorithmic view

The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.



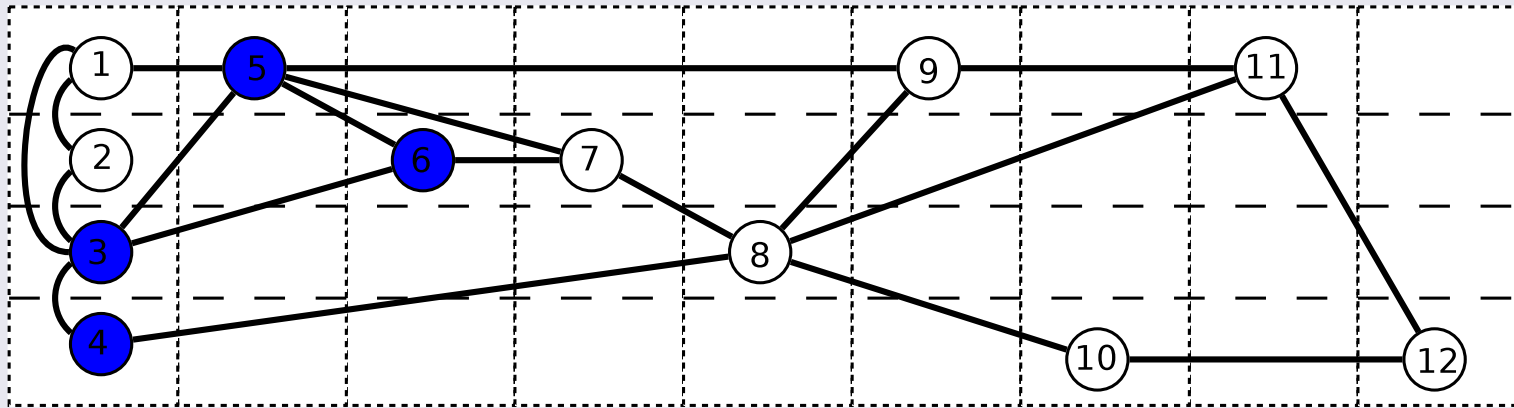
# Algorithmic view

The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.



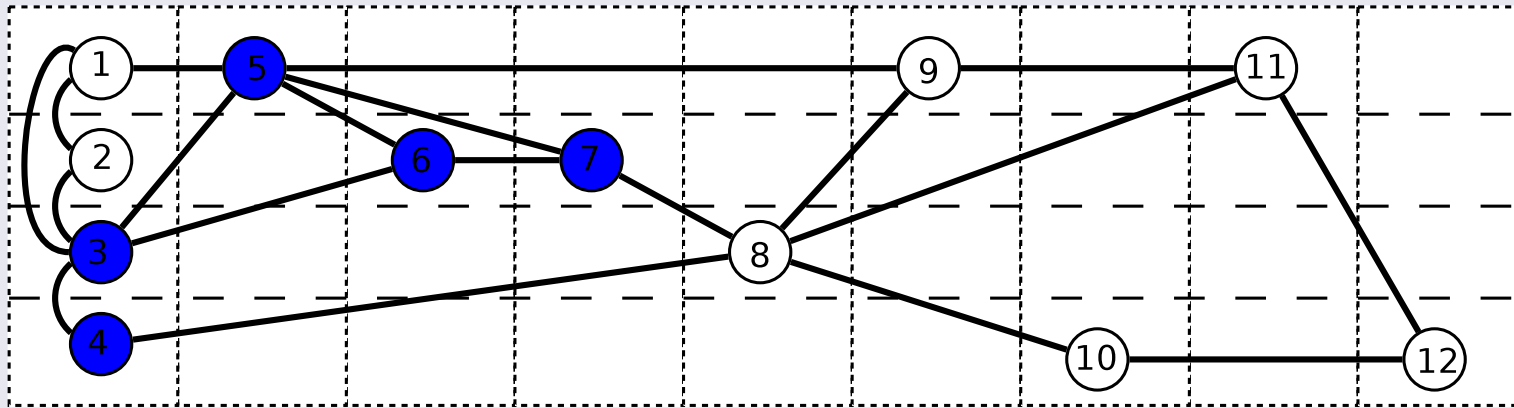
# Algorithmic view

The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.



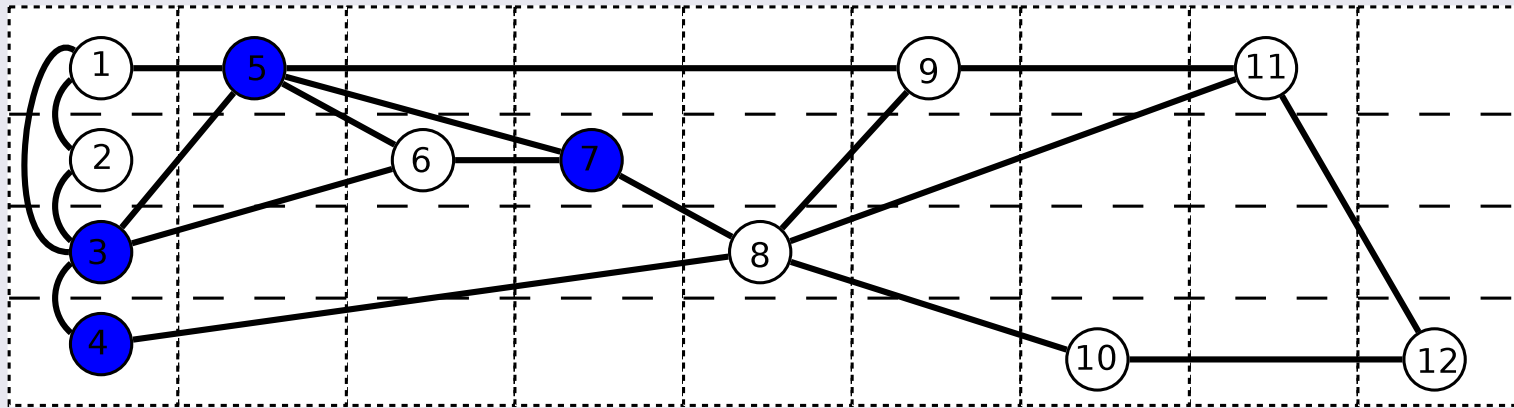
# Algorithmic view

The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.



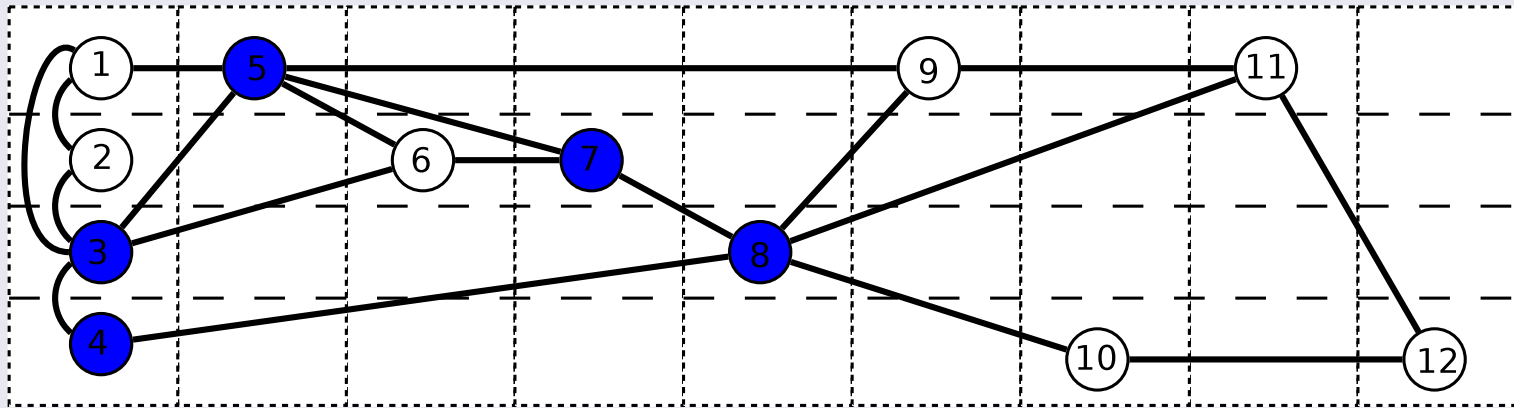
# Algorithmic view

The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.



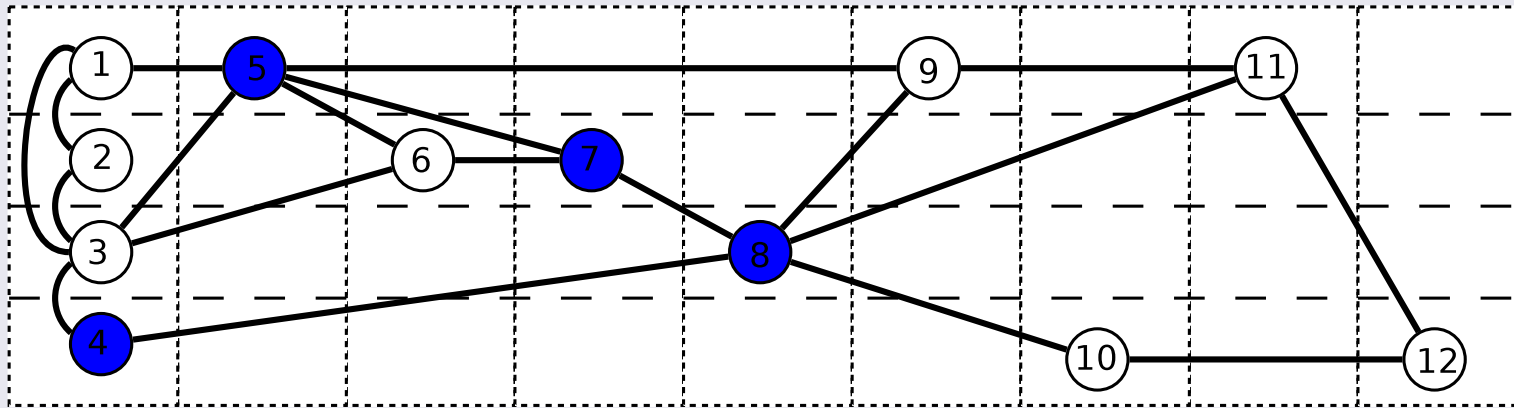
# Algorithmic view

The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.



# Algorithmic view

The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.



# Algorithmic view

The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.

For Dominating Set only need to remember information about boundary

Selected ( <b>Blue</b> )	Not Selected – Already Covered ( <b>Green</b> )
Not Covered ( <b>Red</b> )	Total Cost



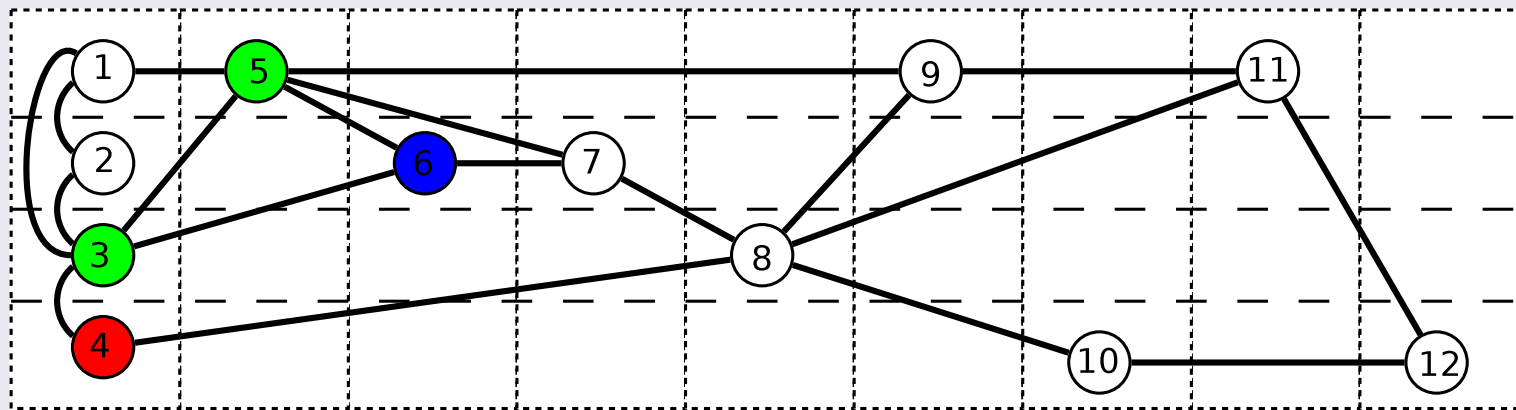


# Algorithmic view

The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.

For Dominating Set only need to remember information about boundary

Selected (Blue)      Not Selected – Already Covered (Green)  
Not Covered (Red)    Total Cost



Separator:  $\{3, 4, 5, 6\}$  includes tuple  $(3, 4, 5, 6; ?)$

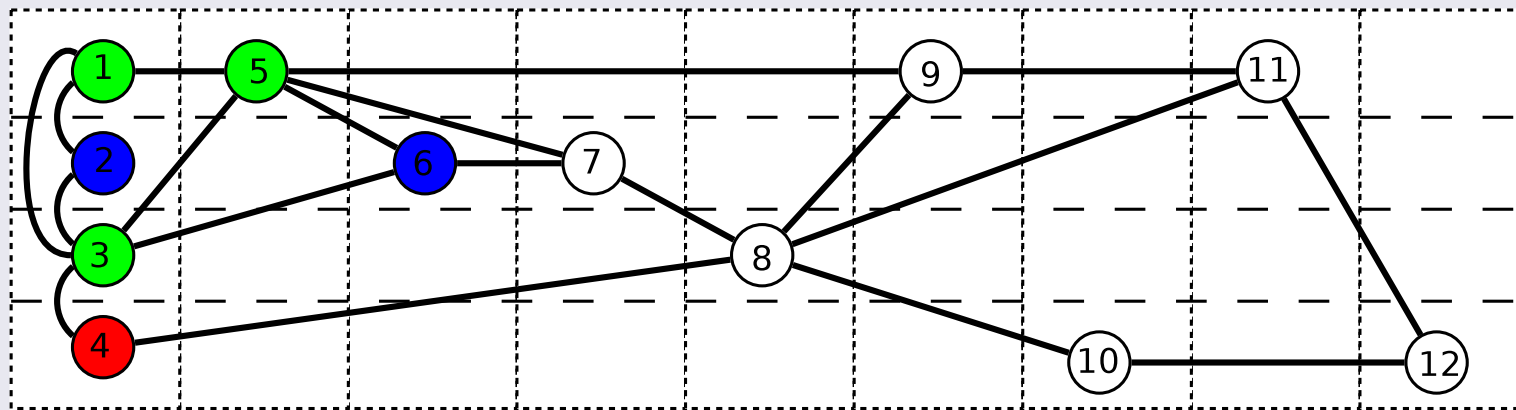


# Algorithmic view

The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.

For Dominating Set only need to remember information about boundary

Selected (Blue)      Not Selected – Already Covered (Green)  
Not Covered (Red)    Total Cost



Separator:  $\{3, 4, 5, 6\}$  includes tuple  $(3, 4, 5, 6; 2)$

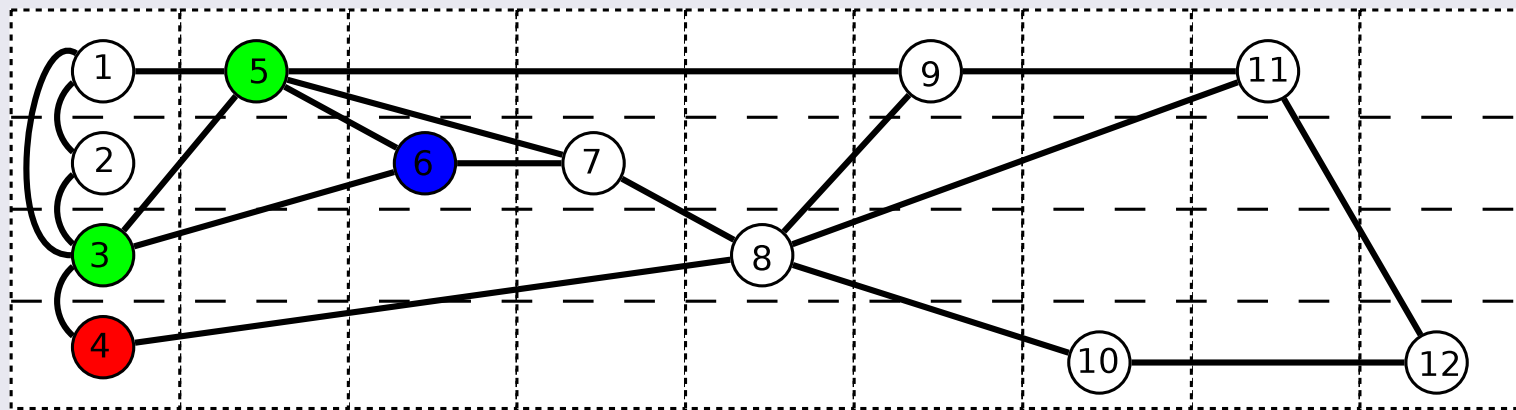


# Algorithmic view

The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.

For Dominating Set only need to remember information about boundary

Selected (Blue)      Not Selected – Already Covered (Green)  
Not Covered (Red)    Total Cost



Separator:  $\{3, 4, 5, 6\}$  includes tuple  $(3, 4, 5, 6; 2)$

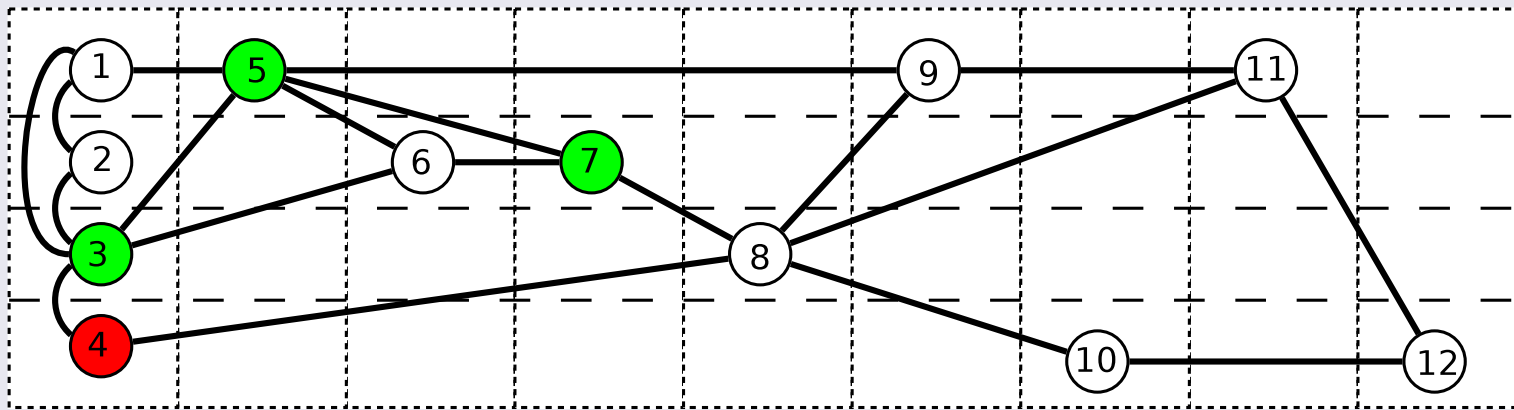


# Algorithmic view

The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.

For Dominating Set only need to remember information about boundary

Selected (Blue)      Not Selected – Already Covered (Green)  
Not Covered (Red)    Total Cost



Separator:  $\{3, 4, 5, 6\}$  includes tuple  $(3, 4, 5, 6; 2)$

Separator:  $\{3, 4, 5, 7\}$  includes tuple  $(3, 4, 5, 7; 2)$

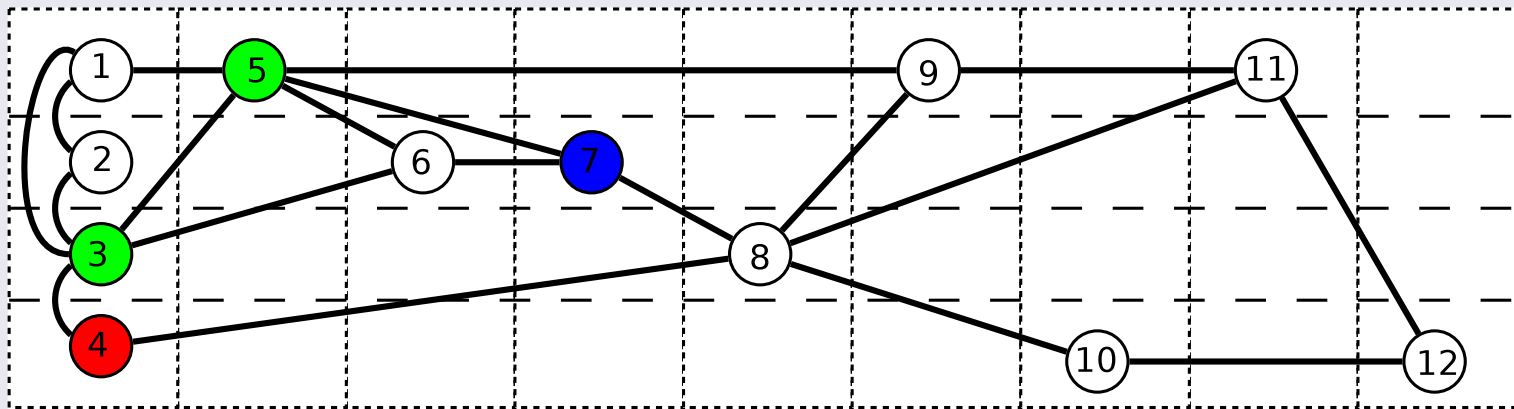


# Algorithmic view

The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.

For Dominating Set only need to remember information about boundary

Selected (Blue)      Not Selected – Already Covered (Green)  
Not Covered (Red)      Total Cost



Separator:  $\{3, 4, 5, 6\}$  includes tuple  $(3, 4, 5, 6; 2)$

Separator:  $\{3, 4, 5, 7\}$  includes tuple  $(3, 4, 5, 7; 3)$

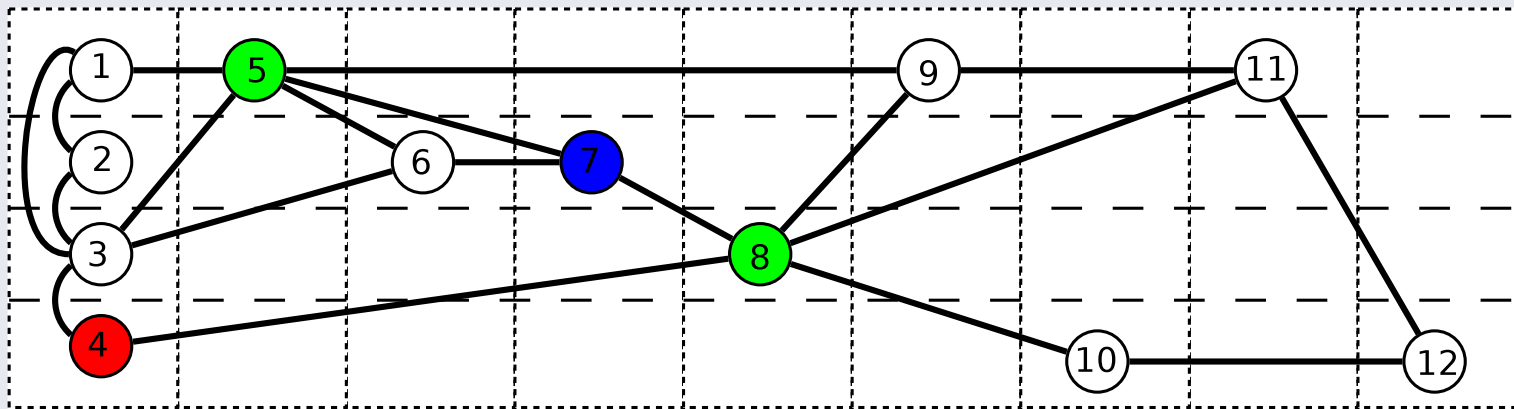


# Algorithmic view

The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.

For Dominating Set only need to remember information about boundary

Selected (Blue)      Not Selected – Already Covered (Green)  
Not Covered (Red)      Total Cost



Separator:  $\{3, 4, 5, 6\}$  includes tuple  $(3, 4, 5, 6; 2)$

Separator:  $\{3, 4, 5, 7\}$  includes tuple  $(3, 4, 5, 7; 3)$

Separator:  $\{4, 5, 7, 8\}$  includes tuple  $(4, 5, 7, 8; 3)$

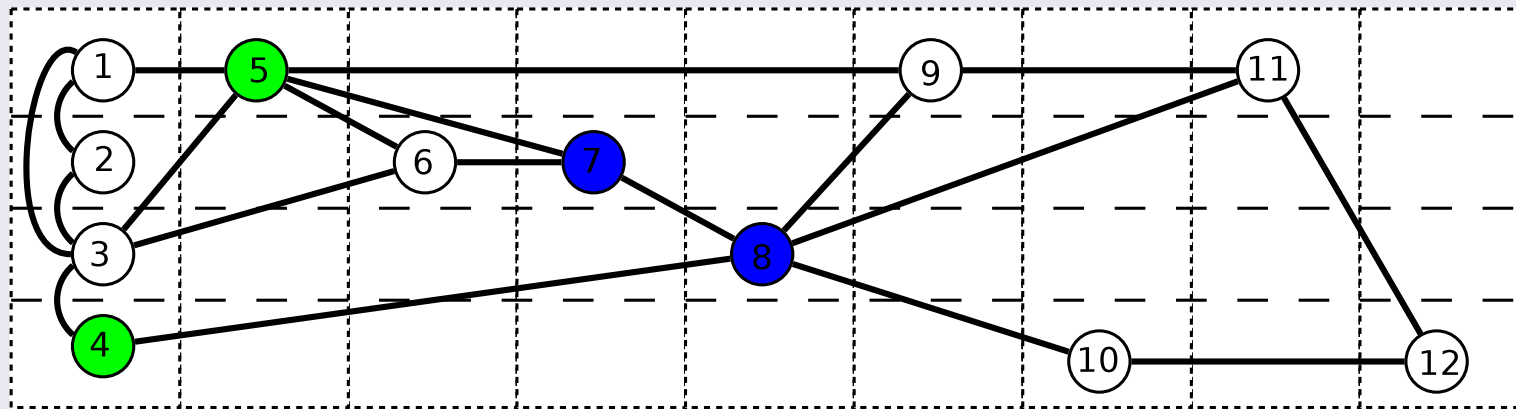


# Algorithmic view

The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.

For Dominating Set only need to remember information about boundary

Selected (Blue)      Not Selected – Already Covered (Green)  
Not Covered (Red)      Total Cost



Separator:  $\{3, 4, 5, 6\}$  includes tuple  $(3, 4, 5, 6; 2)$

Separator:  $\{3, 4, 5, 7\}$  includes tuple  $(3, 4, 5, 7; 3)$

Separator:  $\{4, 5, 7, 8\}$  includes tuple  $(4, 5, 7, 8; 4)$



# Algorithmic view

The reason that this decomposition of the graph is useful is that we have a moving boundary of small separators that “sweeps” the graph.

For Dominating Set only need to remember information about boundary

Selected (Blue)	Not Selected – Already Covered (Green)
Not Covered (Red)	Total Cost

- For Dominating Set DP tables have size  $3^k$ .
- For Capacitated Dominating Set must remember capacity info for selected vertices
  - Table Size:  $C^k$
- Note: May remember Capacity left **OR** Capacity used. Same thing?





# Why $n^k$ for Max Cut? (1/2)

A labelled graph  $G$  has clique-width at most  $k$  if

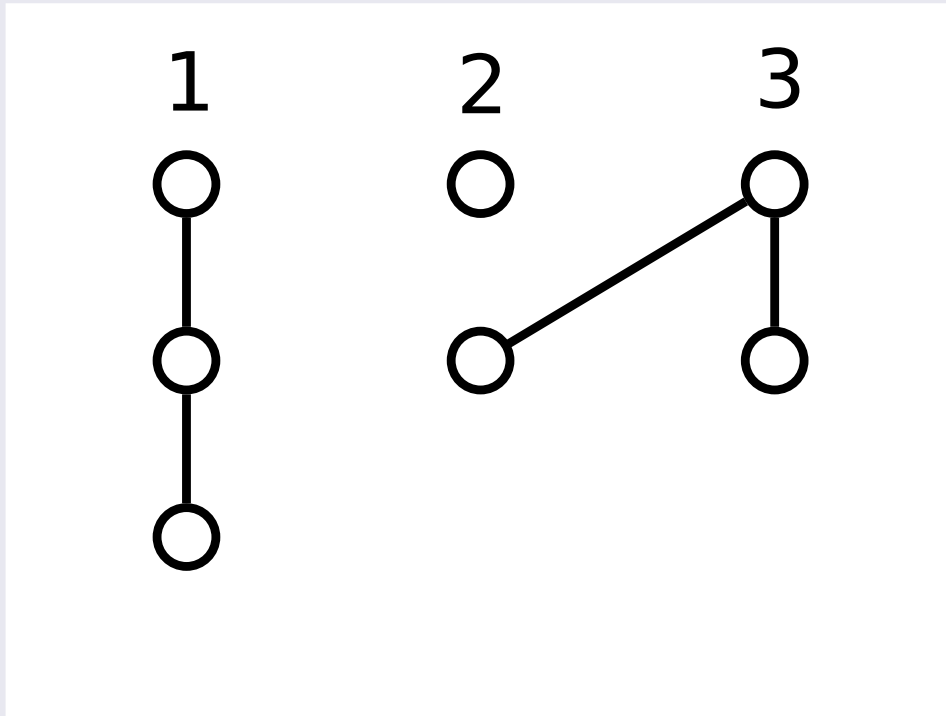
- $G$  is  $K_1$  with some label in  $\{1, \dots, k\}$
- Union:  $G = G_1 \cup G_2$ , with cw  $k$
- Join:  $G = \text{Join}(i, j, G')$ ,  $i, j \in \{1, \dots, k\}$  and  $G'$  has cw  $k$
- Rename:  $G = \text{Rename}(i \rightarrow j, G')$ ,  $i, j \in \{1, \dots, k\}$  and  $G'$  has cw  $k$



# Why $n^k$ for Max Cut? (1/2)

A labelled graph  $G$  has clique-width at most  $k$  if

- $G$  is  $K_1$  with some label in  $\{1, \dots, k\}$
- Union:  $G = G_1 \cup G_2$ , with cw  $k$
- Join:  $G = \text{Join}(i, j, G')$ ,  $i, j \in \{1, \dots, k\}$  and  $G'$  has cw  $k$
- Rename:  $G = \text{Rename}(i \rightarrow j, G')$ ,  $i, j \in \{1, \dots, k\}$  and  $G'$  has cw  $k$

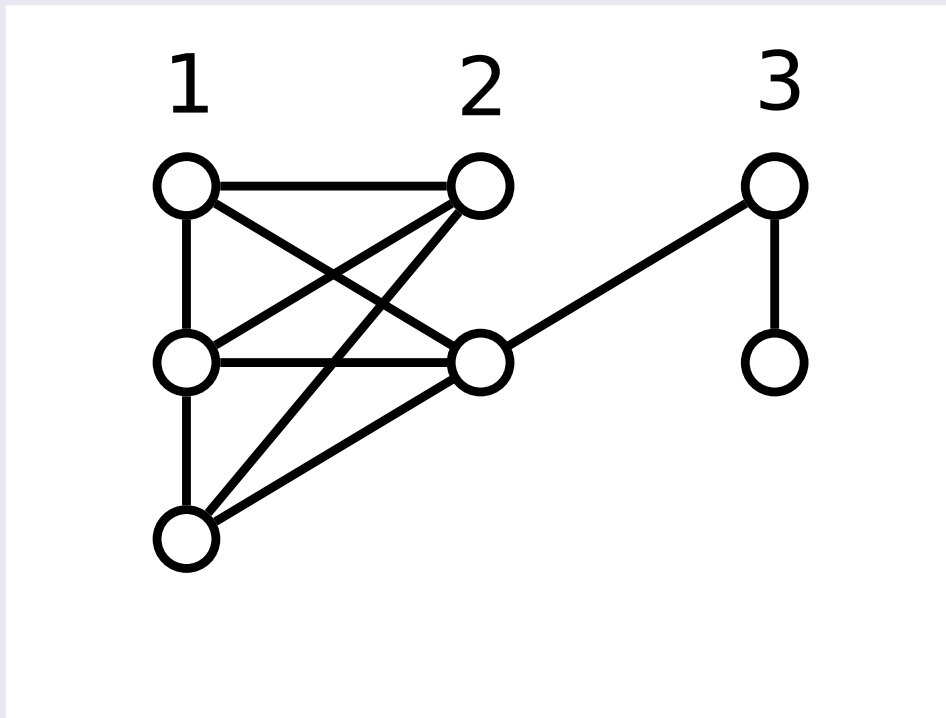


Example:  $\text{Join}(1,2)$   
 $\text{Rename}(3 \rightarrow 2)$

# Why $n^k$ for Max Cut? (1/2)

A labelled graph  $G$  has clique-width at most  $k$  if

- $G$  is  $K_1$  with some label in  $\{1, \dots, k\}$
- Union:  $G = G_1 \cup G_2$ , with cw  $k$
- Join:  $G = \text{Join}(i, j, G')$ ,  $i, j \in \{1, \dots, k\}$  and  $G'$  has cw  $k$
- Rename:  $G = \text{Rename}(i \rightarrow j, G')$ ,  $i, j \in \{1, \dots, k\}$  and  $G'$  has cw  $k$

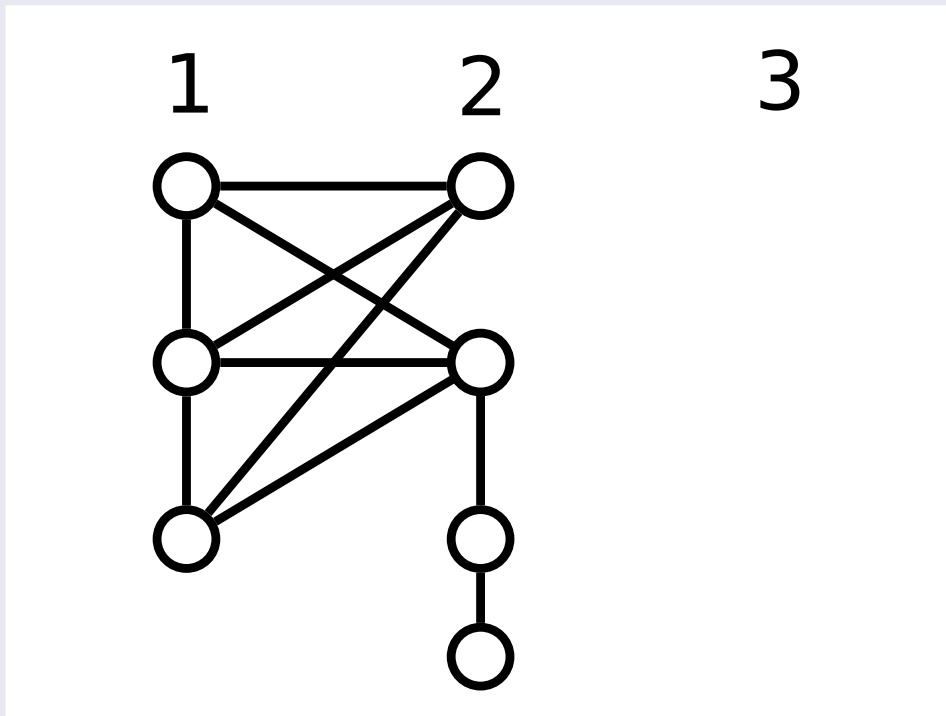


Example: Join(1,2)  
Rename(3→2)

# Why $n^k$ for Max Cut? (1/2)

A labelled graph  $G$  has clique-width at most  $k$  if

- $G$  is  $K_1$  with some label in  $\{1, \dots, k\}$
- Union:  $G = G_1 \cup G_2$ , with cw  $k$
- Join:  $G = \text{Join}(i, j, G')$ ,  $i, j \in \{1, \dots, k\}$  and  $G'$  has cw  $k$
- Rename:  $G = \text{Rename}(i \rightarrow j, G')$ ,  $i, j \in \{1, \dots, k\}$  and  $G'$  has cw  $k$



# Why $n^k$ for Max Cut? (1/2)

A labelled graph  $G$  has clique-width at most  $k$  if

- $G$  is  $K_1$  with some label in  $\{1, \dots, k\}$
  - Union:  $G = G_1 \cup G_2$ , with cw  $k$
  - Join:  $G = \text{Join}(i, j, G')$ ,  $i, j \in \{1, \dots, k\}$  and  $G'$  has cw  $k$
  - Rename:  $G = \text{Rename}(i \rightarrow j, G')$ ,  $i, j \in \{1, \dots, k\}$  and  $G'$  has cw  $k$
- 
- A clique-width expression for  $G$  is a “proof” that  $G$  can be built using these operations and  $k$  labels.
  - Finding an optimal expression is generally hard...
  - We “hope” that such an expression is supplied.
  - We view it as a binary tree and perform dynamic programming.



## Why $n^k$ for Max Cut? (2/2)

### Natural dynamic program for Max Cut

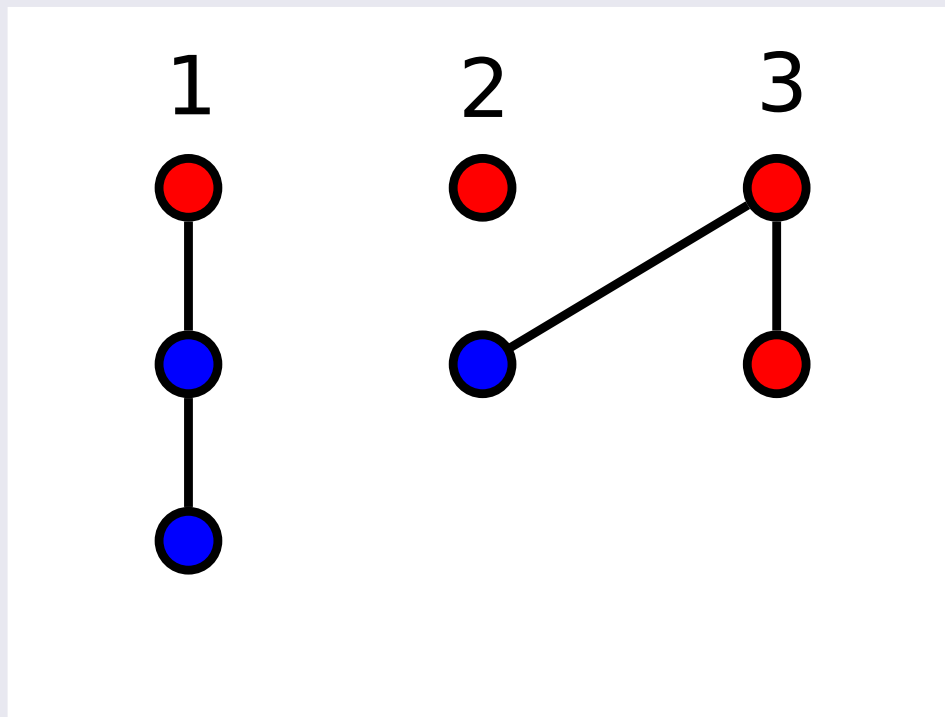
- For each node store a collection of tuples  $(l_1, l_2, \dots, l_k; C)$
- Meaning: There exists a solution that places **exactly**  $l_i$  vertices with label  $i$  in  $L$  and cuts  $C$  edges.



## Why $n^k$ for Max Cut? (2/2)

### Natural dynamic program for Max Cut

- For each node store a collection of tuples  $(l_1, l_2, \dots, l_k; C)$
- Meaning: There exists a solution that places **exactly**  $l_i$  vertices with label  $i$  in  $L$  and cuts  $C$  edges.



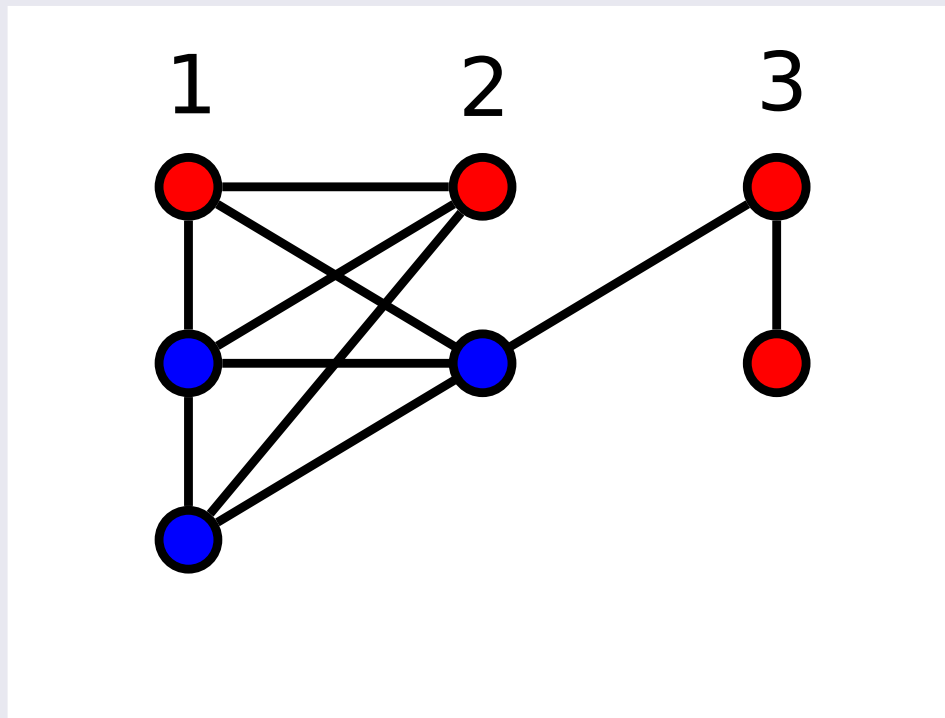
Example tuple: (red = L)  
(1, 1, 2; 2)



## Why $n^k$ for Max Cut? (2/2)

### Natural dynamic program for Max Cut

- For each node store a collection of tuples  $(l_1, l_2, \dots, l_k; C)$
- Meaning: There exists a solution that places **exactly**  $l_i$  vertices with label  $i$  in  $L$  and cuts  $C$  edges.



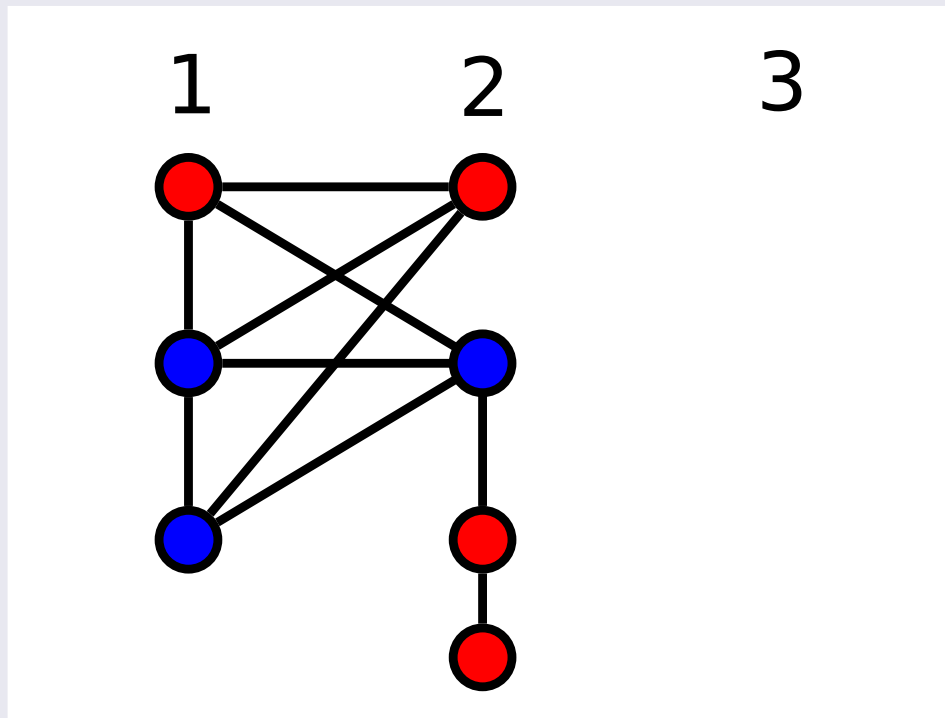
Example tuple: (red =  $L$ )  
(1, 1, 2; 5)



## Why $n^k$ for Max Cut? (2/2)

### Natural dynamic program for Max Cut

- For each node store a collection of tuples  $(l_1, l_2, \dots, l_k; C)$
- Meaning: There exists a solution that places **exactly**  $l_i$  vertices with label  $i$  in  $L$  and cuts  $C$  edges.



Example tuple: (red = L)  
(1, 3, 0; 5)

## Why $n^k$ for Max Cut? (2/2)

### Natural dynamic program for Max Cut

- For each node store a collection of tuples  $(l_1, l_2, \dots, l_k; C)$
- Meaning: There exists a solution that places **exactly**  $l_i$  vertices with label  $i$  in  $L$  and cuts  $C$  edges.
- Can prove inductively that all entries corresponding to potential cuts are filled in.
- Algorithm must compute up to  $(n/k)^k$  entries for each node of the clique-width expression.



## Why $n^k$ for Max Cut? (2/2)

### Natural dynamic program for Max Cut

- For each node store a collection of tuples  $(l_1, l_2, \dots, l_k; C)$
- Meaning: There exists a solution that places **exactly**  $l_i$  vertices with label  $i$  in  $L$  and cuts  $C$  edges.
- Can prove inductively that all entries corresponding to potential cuts are filled in.
- Algorithm must compute up to  $(n/k)^k$  entries for each node of the clique-width expression.



Today's idea: keep **rounded** values for the  $l_i$  entries.  
This can make the table smaller.



# What is rounding?

Example rounding scheme:

- Normal table has values  $l_i \in \{0, 1, 2, 3, \dots, n\}$ .
- We can store values  $l_i \in \{0, 1, 2, 4, 8, 16, \dots, n\}$ .
  - Informal meaning: there exists a partition that places **roughly**  $l_i$  vertices with label  $i$  in  $L$
- Running time  $\approx$  table size  $\approx (\log n)^k$
- But approximation ratio  $\geq 2$



# What is rounding?

Example rounding scheme:

- Normal table has values  $l_i \in \{0, 1, 2, 3, \dots, n\}$ .
- Fix some (small) parameter  $\delta > 0$
- We will store values  $l_i \in \{0, (1 + \delta), (1 + \delta)^2, (1 + \delta)^3, \dots\}$ 
  - Informal meaning: there exists a partition that places **roughly**  $l_i$  vertices with label  $i$  in  $L$
- Running time  $\approx$  table size
- For small  $\delta$  we have  $\log_{(1+\delta)} n = O\left(\frac{\log n}{\ln(1+\delta)}\right) = O\left(\frac{\log n}{\delta}\right)$
- Table size  $\rightarrow (\log n / \delta)^k$



# What is rounding?

Example rounding scheme:

- Normal table has values  $l_i \in \{0, 1, 2, 3, \dots, n\}$ .
- Fix some (small) parameter  $\delta > 0$
- We will store values  $l_i \in \{0, (1 + \delta), (1 + \delta)^2, (1 + \delta)^3, \dots\}$ 
  - Informal meaning: there exists a partition that places **roughly**  $l_i$  vertices with label  $i$  in  $L$
- Running time  $\approx$  table size
- For small  $\delta$  we have  $\log_{(1+\delta)} n = O\left(\frac{\log n}{\ln(1+\delta)}\right) = O\left(\frac{\log n}{\delta}\right)$
- Table size  $\rightarrow (\log n / \delta)^k$
- Approximation ratio depends on choice of  $\delta$ , but is at least  $(1 + \delta)$ .
- This is achieved if we have the **correct/best** approximation for each value.



# What is rounding?

Example rounding scheme:

- Normal table has values  $l_i \in \{0, 1, 2, 3, \dots, n\}$ .
- Fix some (small) parameter  $\delta > 0$
- We will store values  $l_i \in \{0, (1 + \delta), (1 + \delta)^2, (1 + \delta)^3, \dots\}$ 
  - Informal meaning: there exists a partition that places **roughly**  $l_i$  vertices with label  $i$  in  $L$
- Running time  $\approx$  table size
- For small  $\delta$  we have  $\log_{(1+\delta)} n = O\left(\frac{\log n}{\ln(1+\delta)}\right) = O\left(\frac{\log n}{\delta}\right)$
- Table size  $\rightarrow (\log n / \delta)^k$
- Approximation ratio depends on choice of  $\delta$ , but is at least  $(1 + \delta)$ .
- This is achieved if we have the **correct/best** approximation for each value.
- This will be hard!



# The problem with rounding

Errors can propagate and pile up!

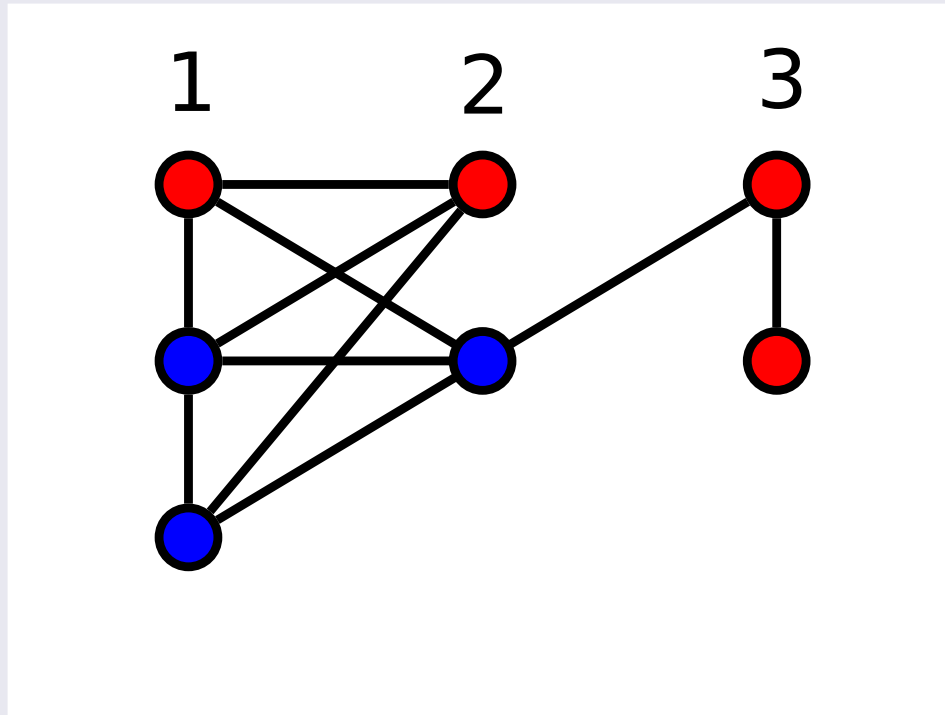




# The problem with rounding

Errors can propagate and pile up!

Concrete example



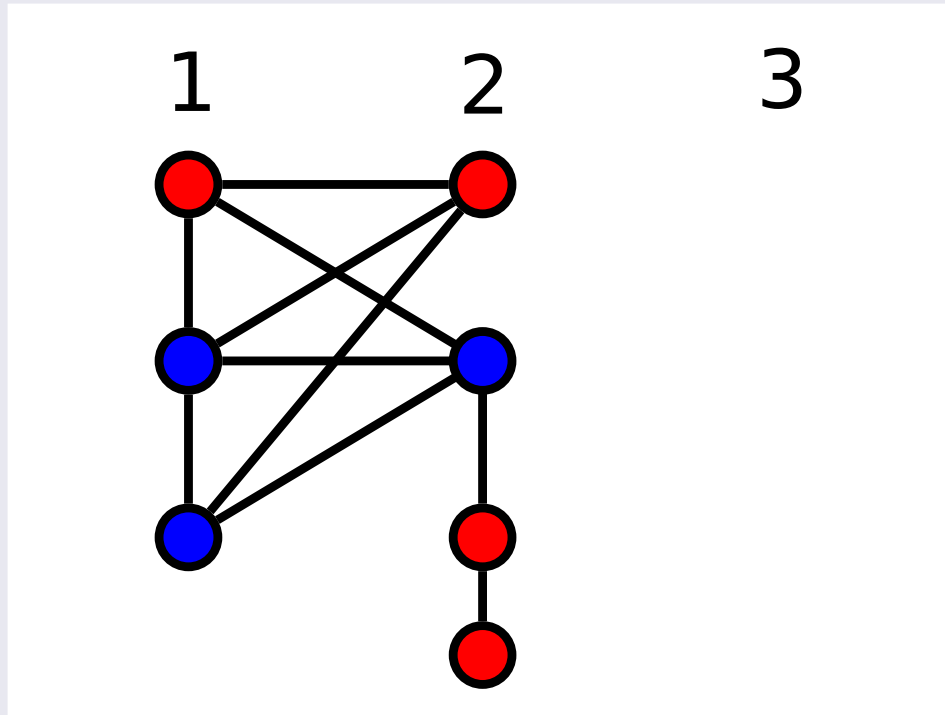
Example tuple: (red = L)  
 $(a_1, a_2, a_3; a_C)$



# The problem with rounding

Errors can propagate and pile up!

Concrete example



Example tuple: (red = L)  
 $(a_1, a_2 + a_3, \mathbf{0}; a_C)$



# The problem with rounding

Errors can propagate and pile up!

- The new value we would like to store  $(a_2 + a_3)$  is not necessarily “round” (integer power of  $(1 + \delta)$ ).
- We must somehow round it to fit the scheme
- This can introduce an additional error of  $(1 + \delta)$



# The problem with rounding

Errors can propagate and pile up!

- The new value we would like to store  $(a_2 + a_3)$  is not necessarily “round” (integer power of  $(1 + \delta)$ ).
- We must somehow round it to fit the scheme
- This can introduce an additional error of  $(1 + \delta)$
- After  $n$  steps this can cause an error of  $(1 + \delta)^n$



# The problem with rounding

Errors can propagate and pile up!

- The new value we would like to store  $(a_2 + a_3)$  is not necessarily “round” (integer power of  $(1 + \delta)$ ).
- We must somehow round it to fit the scheme
- This can introduce an additional error of  $(1 + \delta)$
- After  $n$  steps this can cause an error of  $(1 + \delta)^n$



# The problem with rounding

Errors can propagate and pile up!

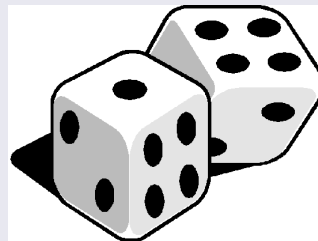
- The new value we would like to store  $(a_2 + a_3)$  is not necessarily “round” (integer power of  $(1 + \delta)$ ).
- We must somehow round it to fit the scheme
- This can introduce an additional error of  $(1 + \delta)$
- After  $n$  steps this can cause an error of  $(1 + \delta)^n$
- Running time:  $(\log n / \delta)^k$ . Want this to be  $(\log n)^{O(k)}$  so  $\delta = 1 / \log^c n$ .
- Then  $(1 + \delta)^n$  is too big! (Certainly not  $1 + \epsilon$ )
- Must round in a way that ensures sometimes rounding **improves** my approximation.



# The problem with rounding

Errors can propagate and pile up!

- The new value we would like to store  $(a_2 + a_3)$  is not necessarily “round” (integer power of  $(1 + \delta)$ ).
- We must somehow round it to fit the scheme
- This can introduce an additional error of  $(1 + \delta)$
- After  $n$  steps this can cause an error of  $(1 + \delta)^n$
- Running time:  $(\log n / \delta)^k$ . Want this to be  $(\log n)^{O(k)}$  so  $\delta = 1 / \log^c n$ .
- Then  $(1 + \delta)^n$  is too big! (Certainly not  $1 + \epsilon$ )
- Must round in a way that ensures sometimes rounding **improves** my approximation.



# How to measure errors

- Plan so far:
  - Start with exact DP. Run it with approximate values.
  - TBD: how to re-round non-round intermediate values.
- There is a value  $x$  calculated by the exact DP
- There is a value  $y$  calculated by approximate DP
- Define

$$Error(x, y) := \log_{(1+\delta)} \left( \max \left\{ \frac{x}{y}, \frac{y}{x} \right\} \right)$$



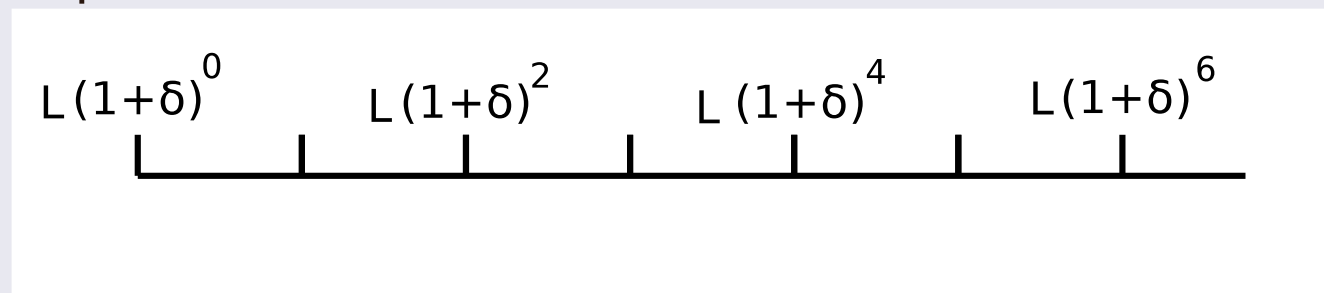


# How to measure errors

- Plan so far:
  - Start with exact DP. Run it with approximate values.
  - TBD: how to re-round non-round intermediate values.
- There is a value  $x$  calculated by the exact DP
- There is a value  $y$  calculated by approximate DP
- Define

$$Error(x, y) := \log_{(1+\delta)} \left( \max \left\{ \frac{x}{y}, \frac{y}{x} \right\} \right)$$

In pictures:

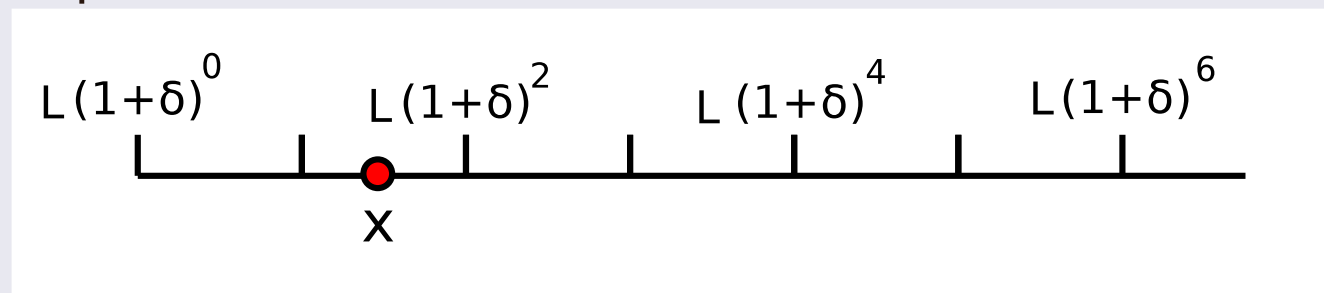


# How to measure errors

- Plan so far:
  - Start with exact DP. Run it with approximate values.
  - TBD: how to re-round non-round intermediate values.
- There is a value  $x$  calculated by the exact DP
- There is a value  $y$  calculated by approximate DP
- Define

$$Error(x, y) := \log_{(1+\delta)} \left( \max \left\{ \frac{x}{y}, \frac{y}{x} \right\} \right)$$

In pictures:

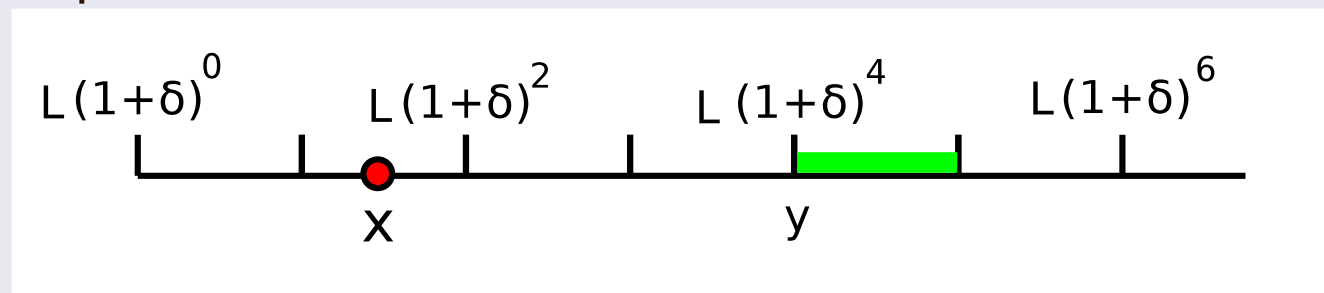


# How to measure errors

- Plan so far:
  - Start with exact DP. Run it with approximate values.
  - TBD: how to re-round non-round intermediate values.
- There is a value  $x$  calculated by the exact DP
- There is a value  $y$  calculated by approximate DP
- Define

$$Error(x, y) := \log_{(1+\delta)} \left( \max \left\{ \frac{x}{y}, \frac{y}{x} \right\} \right)$$

In pictures:

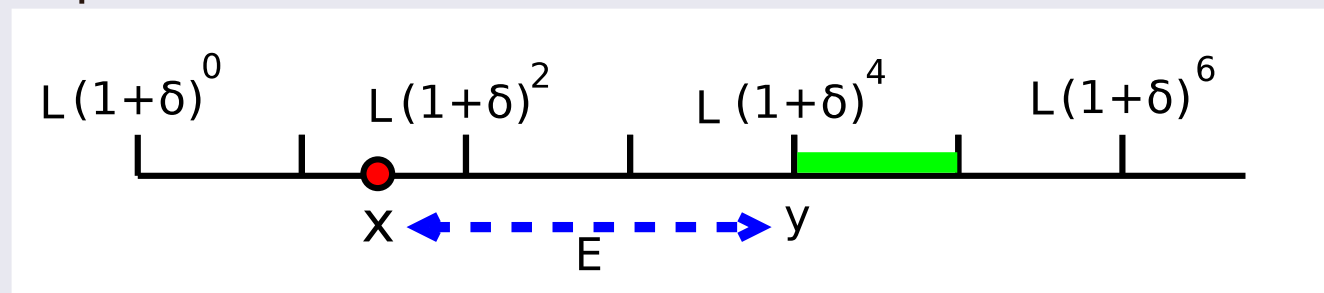


# How to measure errors

- Plan so far:
  - Start with exact DP. Run it with approximate values.
  - TBD: how to re-round non-round intermediate values.
- There is a value  $x$  calculated by the exact DP
- There is a value  $y$  calculated by approximate DP
- Define

$$Error(x, y) := \log_{(1+\delta)} \left( \max \left\{ \frac{x}{y}, \frac{y}{x} \right\} \right)$$

In pictures:



# How to measure errors

- Plan so far:
  - Start with exact DP. Run it with approximate values.
  - TBD: how to re-round non-round intermediate values.
- There is a value  $x$  calculated by the exact DP
- There is a value  $y$  calculated by approximate DP
- Define

$$Error(x, y) := \log_{(1+\delta)} \left( \max \left\{ \frac{x}{y}, \frac{y}{x} \right\} \right)$$

End goal:

- Would like  $Error(x, y) \leq \epsilon/\delta$  for all  $x, y$ .
  - Approximation ratio =  $(1 + \delta)^{Error} \leq (1 + \delta)^{\epsilon/\delta} \approx 1 + \epsilon$



# What we know about errors

- Consider values  $x_1, x_2$  and their approximations  $y_1, y_2$  with Errors  $E_1, E_2$ .



# What we know about errors

- Consider values  $x_1, x_2$  and their approximations  $y_1, y_2$  with Errors  $E_1, E_2$ .
- The (non-round) value  $y_1 + y_2$  has error at most  $\max\{E_1, E_2\}$ .



# What we know about errors

- Consider values  $x_1, x_2$  and their approximations  $y_1, y_2$  with Errors  $E_1, E_2$ .
- The (non-round) value  $y_1 + y_2$  has error at most  $\max\{E_1, E_2\}$ .
- The (non-round) value  $y_1 \cdot y_2$  has error at most  $E_1 + E_2$ .





# What we know about errors

- Consider values  $x_1, x_2$  and their approximations  $y_1, y_2$  with Errors  $E_1, E_2$ .
- The (non-round) value  $y_1 + y_2$  has error at most  $\max\{E_1, E_2\}$ .
- The (non-round) value  $y_1 \cdot y_2$  has error at most  $E_1 + E_2$ .
- The (non-round) value  $y_1 - y_2$  has unbounded error!



# What we know about errors

- Consider values  $x_1, x_2$  and their approximations  $y_1, y_2$  with Errors  $E_1, E_2$ .
  - The (non-round) value  $y_1 + y_2$  has error at most  $\max\{E_1, E_2\}$ .
  - The (non-round) value  $y_1 \cdot y_2$  has error at most  $E_1 + E_2$ .
  - The (non-round) value  $y_1 - y_2$  has unbounded error!
- 
- DPs relying on additions are the “Easiest Target”.

From now on only Additive DPs considered.

- Fortunately, there are plenty...
- E.g. Max Cut, Capacitated Dominating Set



# Two roads to success



Obliviously round in some way.  
Hope for the best!



Probabilistically round. Prove  
that good things happen whp.



# The lucky man's solution

Consider a DP that only uses additions.

- Trivial observation: each level of the given clique-width expression/tree decomposition increases maximum Error by at most 1.
  - Error can only be introduced in re-rounding.
- What if the given decomposition is **balanced**? Then it has logarithmic height!
- Wouldn't this be nice?



# The lucky man's solution

Consider a DP that only uses additions.

- Trivial observation: each level of the given clique-width expression/tree decomposition increases maximum Error by at most 1.
  - Error can only be introduced in re-rounding.
- What if the given decomposition is **balanced**? Then it has logarithmic height!
- Wouldn't this be nice?



# The lucky man's solution

Consider a DP that only uses additions.

- Trivial observation: each level of the given clique-width expression/tree decomposition increases maximum Error by at most 1.
  - Error can only be introduced in re-rounding.
- What if the given decomposition is **balanced**? Then it has logarithmic height!
- Wouldn't this be nice?



Thm [Bodlaender and Hagerup SICOMP '98]: Every graph with treewidth  $w$  has a balanced tree decomposition with width  $3w$ .



# Using our gift

1. Set  $\delta = \epsilon / \log n$ .
2. Balance decomposition.
3. Run approximate DP, rounding arbitrarily.

This works! (As long as we only do additions/comparisons)

- Approximation ratio  $\leq (1 + \delta)^{\log n} \approx (1 + \epsilon)$ .
- Running time  $(\log n / \epsilon)^{O(k)}$ .

Application approximation schemes:

- Capacitated Dom. Set (bi-criteria)
- Capacitated Vertex Cover (bi-criteria)
- Bounded Degree Deletion (bi-criteria)
- Equitable Coloring (bi-criteria)
- Graph Balancing



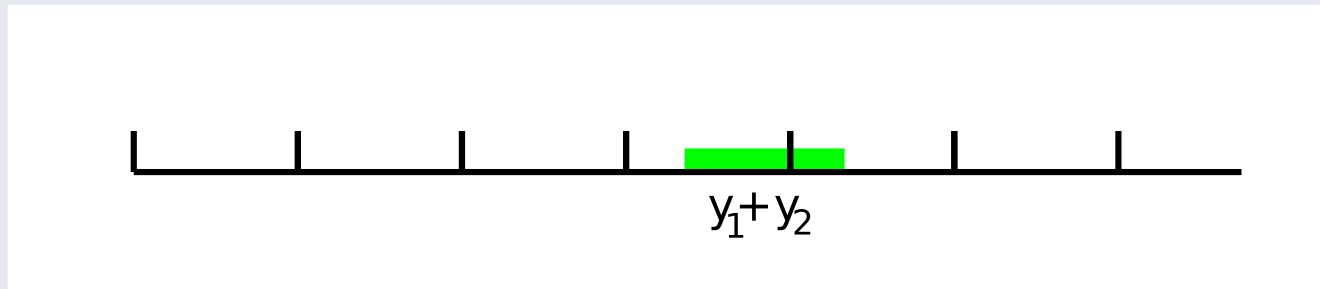
Back to the Interesting Part



## We have to round

- What about Max Cut on clique-width?
  - Best known balancing theorem blows up number of labels to  $2^k$
- Must round in a way that works for  $n$  steps.
- Intuition: randomization “evens out” the errors.

Process:



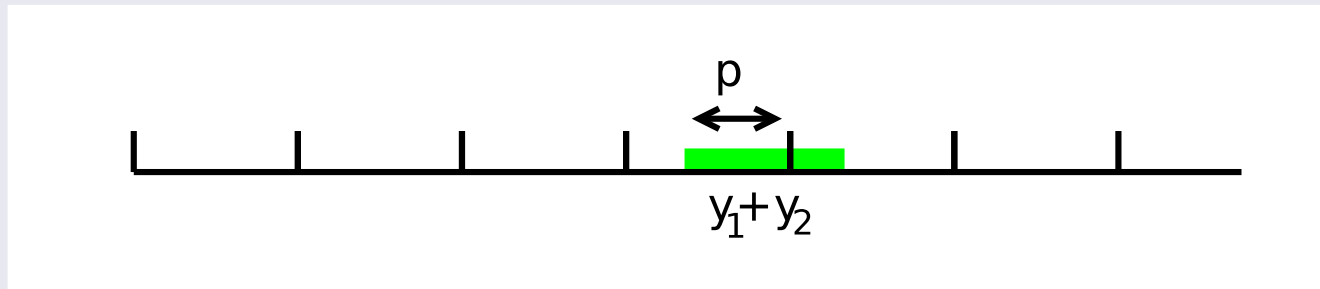
We denote the (random) outcome of this process by  $y_1 \oplus y_2$



## We have to round

- What about Max Cut on clique-width?
  - Best known balancing theorem blows up number of labels to  $2^k$
- Must round in a way that works for  $n$  steps.
- Intuition: randomization “evens out” the errors.

Process:



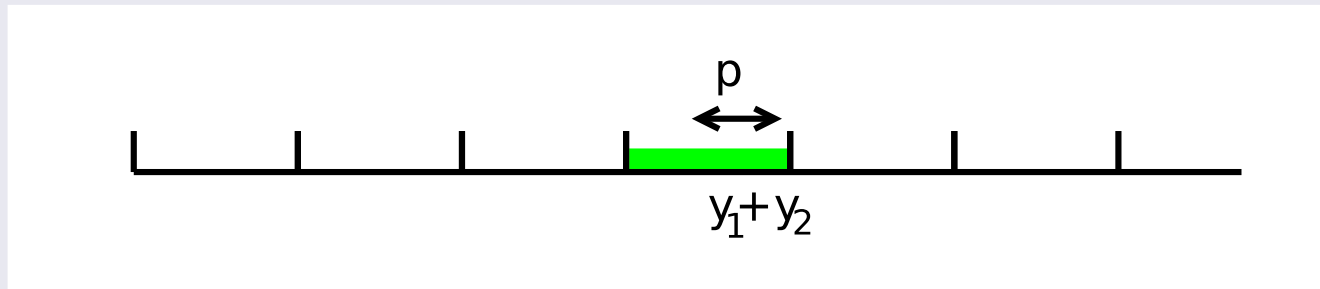
We denote the (random) outcome of this process by  $y_1 \oplus y_2$



# We have to round

- What about Max Cut on clique-width?
  - Best known balancing theorem blows up number of labels to  $2^k$
- Must round in a way that works for  $n$  steps.
- Intuition: randomization “evens out” the errors.

Process:



We denote the (random) outcome of this process by  $y_1 \oplus y_2$



# Addition Trees

- We want this process to work whp for  $\delta = \Omega(1/\text{poly}(\log n))$ .
- This is complicated. So we abstract it out.

Definition: An Addition Tree (AT) is a binary tree with positive integers on the leaves. The value of each node is the sum of its children.

Definition: An Approximate Addition Tree (AAT) is an Addition Tree where additions are replaced by the  $\oplus$  operation.

- Motivation: If AATs are good whp, we can use this as a black box for any DP that only does additions.



# Addition Trees

- We want this process to work whp for  $\delta = \Omega(1/\text{poly}(\log n))$ .
- This is complicated. So we abstract it out.

Definition: An Addition Tree (AT) is a binary tree with positive integers on the leaves. The value of each node is the sum of its children.

Definition: An Approximate Addition Tree (AAT) is an Addition Tree where additions are replaced by the  $\oplus$  operation.

- Motivation: If AATs are good whp, we can use this as a black box for any DP that only does additions.

Theorem: For any  $n$ -vertex AAT  $T$  and any  $\epsilon > 0$ , there exists  $\delta = \Omega(\epsilon^2 / \log^6 n)$  such that:

$$\Pr [\exists v \in T : \text{Error}(v) > 1 + \epsilon] \leq n^{-\log n}$$



# Black Box Applications

Application approximation schemes for clique-width:

- Max Cut
- Edge Dominating Set
  - Is DP additive?
- Capacitated Dom. Set (bi-criteria)
- Bounded Degree Deletion (bi-criteria)
- Equitable Coloring (bi-criteria)
  
- Running times  $(\log n/\epsilon)^{O(k)}$
- Recall: last three are W-hard even for treewidth



# AAT theorem proof sketch

Intuition for main Approximate Addition Tree theorem.

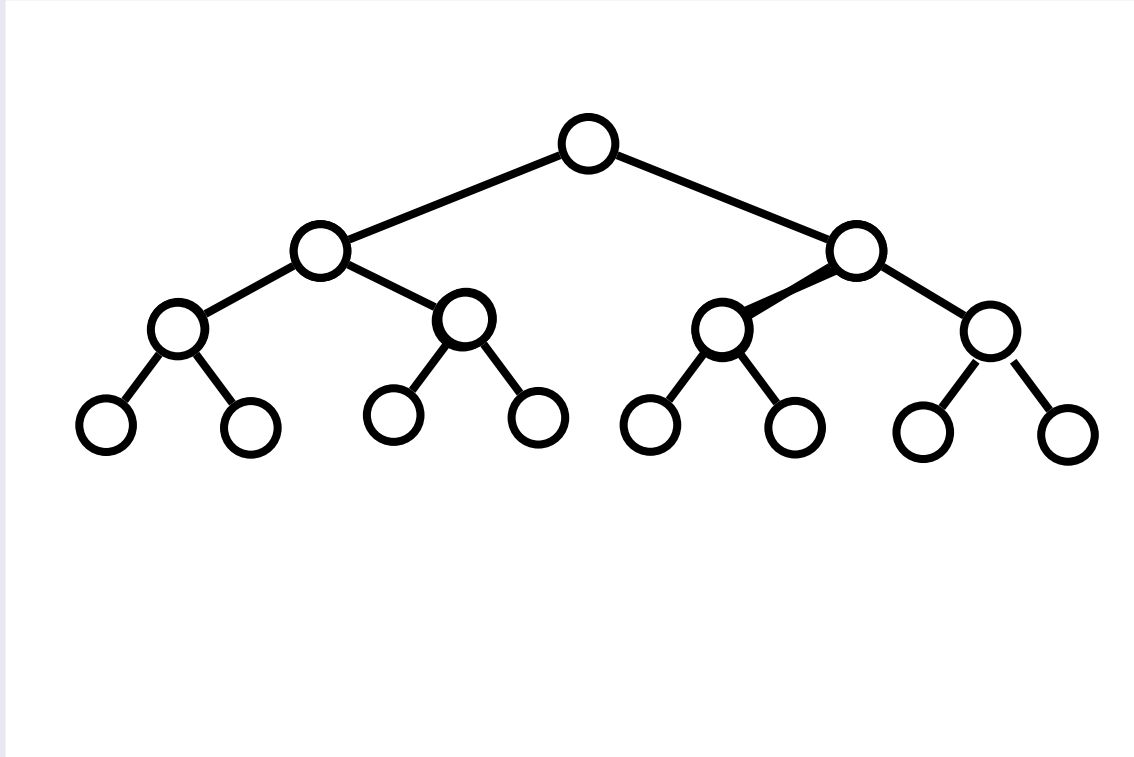
Two main cases:



# AAT theorem proof sketch

Intuition for main Approximate Addition Tree theorem.

Two main cases:



Balanced Tree: easy







# AAT theorem proof sketch

Intuition for main Approximate Addition Tree theorem.

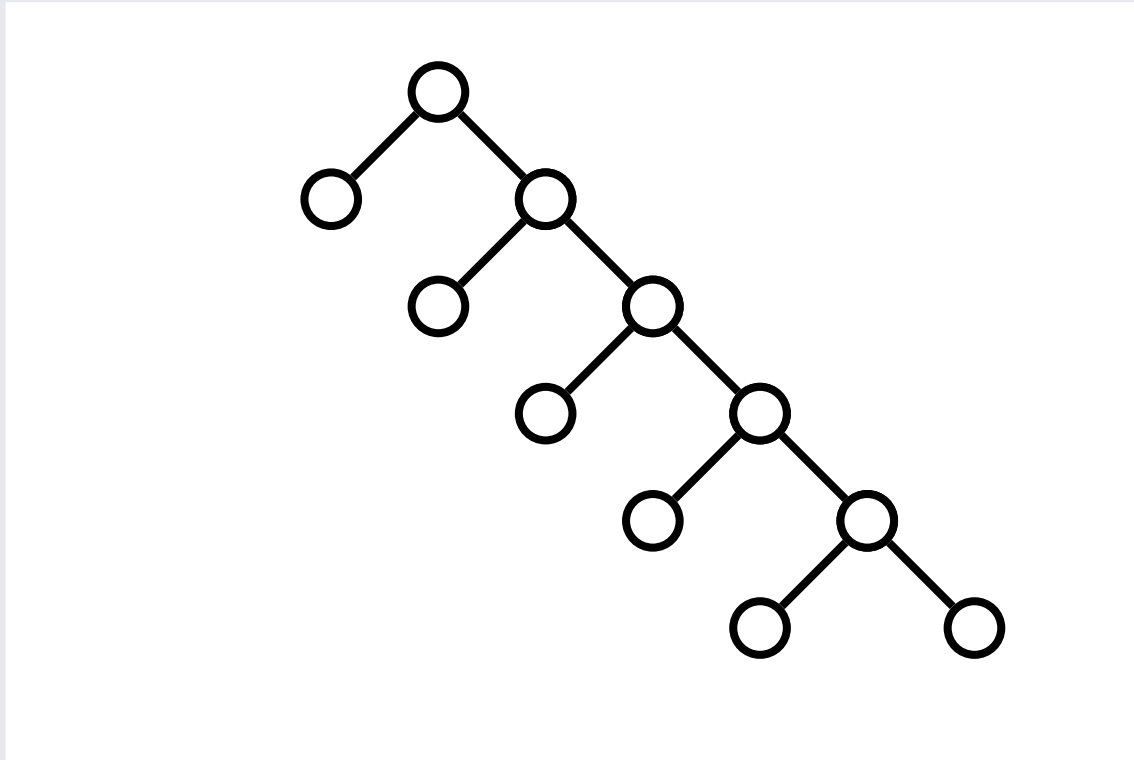
Proof Strategy:

- Prove the theorem for UnBalanced Trees
  - Main part
- Define notion of balanced height
- Use induction
  - Base case: UnBalanced trees
  - Inductive step similar to UnBalanced case



# Unbalanced case

Intuition: self-correcting random walk

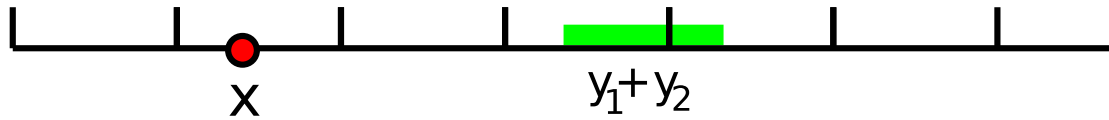


- $n$  addition + rounding, each can increase Error by 1.
- In the end we should have error at most  $\log^c n$

# Unbalanced case

Intuition: self-correcting random walk

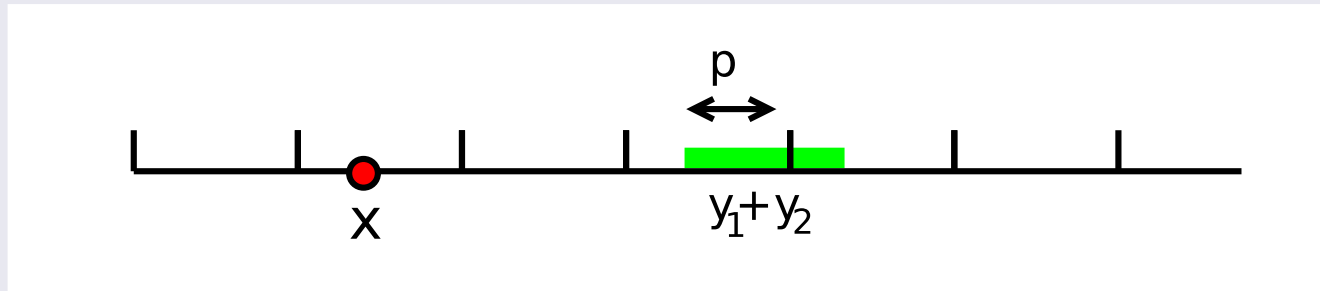
Observation 1: Each rounding step has in expectation **no effect**.



# Unbalanced case

Intuition: self-correcting random walk

Observation 1: Each rounding step has in expectation **no effect**.

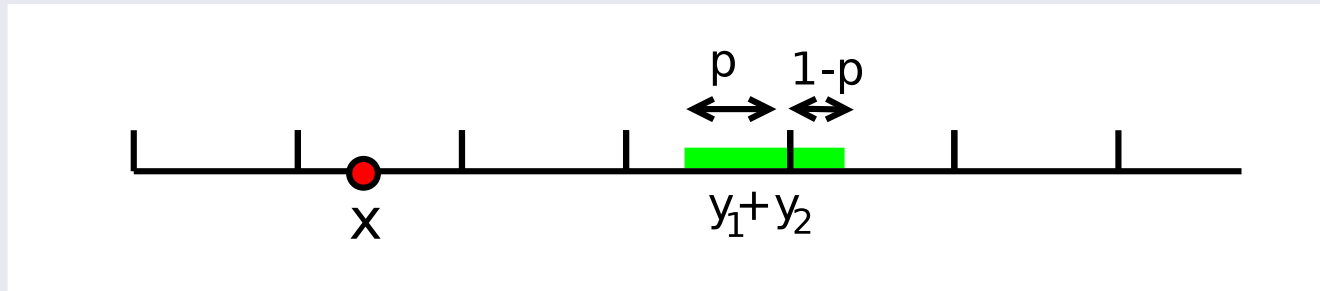


$p$  is the probability of rounding down

# Unbalanced case

Intuition: self-correcting random walk

Observation 1: Each rounding step has in expectation **no effect**.



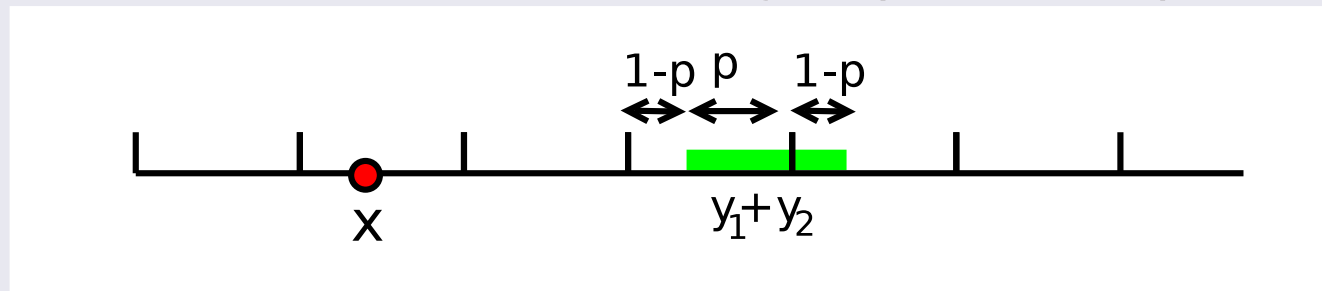
$1 - p$  is the probability of rounding up



# Unbalanced case

Intuition: self-correcting random walk

Observation 1: Each rounding step has in expectation **no effect**.



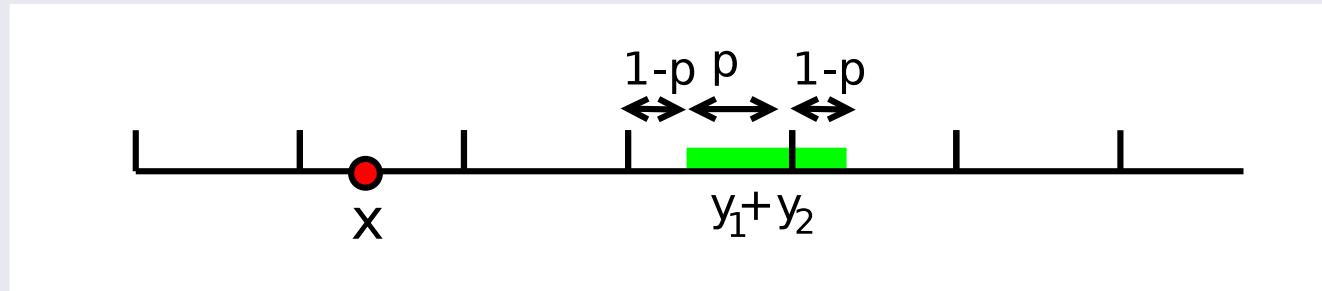
If we round down we **decrease** our error by  $1 - p$



# Unbalanced case

Intuition: self-correcting random walk

Observation 1: Each rounding step has in expectation **no effect**.



If we round down we **decrease** our error by  $1 - p$

If we round up we **increase** our error by  $p$

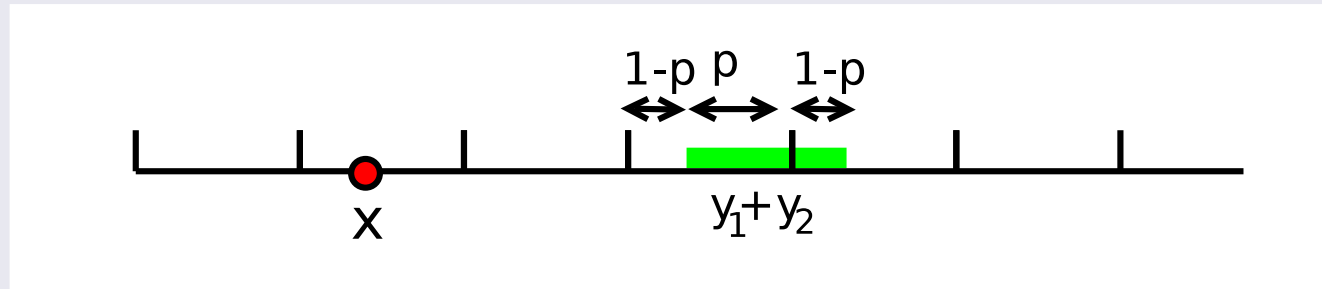




# Unbalanced case

Intuition: self-correcting random walk

Observation 1: Each rounding step has in expectation **no effect**.



If we round down we **decrease** our error by  $1 - p$

If we round up we **increase** our error by  $p$

Expected change:  $-p(1 - p) + (1 - p)p = 0$



# Unbalanced case

Intuition: self-correcting random walk

Observation 1: Each rounding step has in expectation **no effect**.

Unfortunately, this observation is not enough!

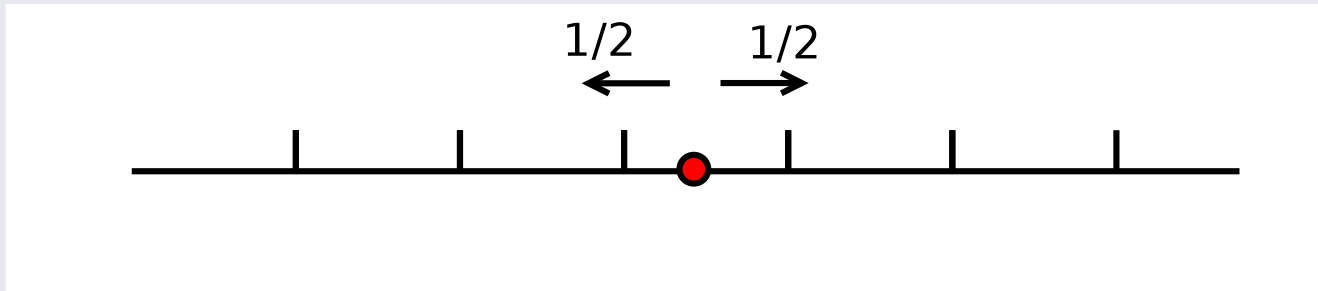


# Unbalanced case

Intuition: self-correcting random walk

Observation 1: Each rounding step has in expectation **no effect**.

Unfortunately, this observation is not enough!



Token will end up at distance  $\sqrt{n}$  whp.

We need distance  $\leq \epsilon/\delta \leq \log^c n$



# Unbalanced case

Intuition: self-correcting random walk

Observation 1: Each rounding step has in expectation **no effect**.

Unfortunately, this observation is not enough!

Observation 2: In UnBalanced tree, initial approximate value  $y_1 + y_2$  always has improved error.

- Informally: one value is known without error
- $y_1 = (1 + \delta)^{E_1} x_1$
- $y_2 = (1 + \delta)^0 x_2$
- $\Rightarrow y_1 + y_2 = (1 + \delta)^{E_1} x_1 + x_2 < (1 + \delta)^{E_1} (x_1 + x_2)$



# Unbalanced case

Intuition: self-correcting random walk

Observation 1: Each rounding step has in expectation **no effect**.

Unfortunately, this observation is not enough!

Observation 2: In UnBalanced tree, initial approximate value  $y_1 + y_2$  always has improved error.

Summary:

- Step 1: Obtain initial approximation  $\Rightarrow$  improves Error
- Step 2: Round  $\Rightarrow$  In expectation does not change Error
- $\Rightarrow$  stronger concentration than just random walk.
- This can be proved with moment-generating function (similar to Chernoff bound/Azuma inequality etc.)



# Unbalanced case

Intuition: self-correcting random walk

Observation 1: Each rounding step has in expectation **no effect**.

Unfortunately, this observation is not enough!

Observation 2: In UnBalanced tree, initial approximate value  $y_1 + y_2$  always has improved error.

Summary:

- Step 1: Obtain initial approximation  $\Rightarrow$  improves Error
- Step 2: Round  $\Rightarrow$  In expectation does not change Error
- $\Rightarrow$  stronger concentration than just random walk.
- This can be proved with moment-generating function (similar to Chernoff bound/Azuma inequality etc.)



UnBalanced Trees are OK



# Balanced height

To generalize the previous argument to any tree, use balanced height.



# Balanced height

To generalize the previous argument to any tree, use balanced height.

Definition:

- A leaf has balanced height 0.
- A node whose children have heights  $h_1 \neq h_2$  has height  $\max\{h_1, h_2\}$
- A node whose children have heights  $h_1 = h_2$  has height  $h_1 + 1$





# Balanced height

To generalize the previous argument to any tree, use balanced height.

Definition:

- A leaf has balanced height 0.
- A node whose children have heights  $h_1 \neq h_2$  has height  $\max\{h_1, h_2\}$
- A node whose children have heights  $h_1 = h_2$  has height  $h_1 + 1$

Fact: All trees have balanced height  $\log n$

Proof idea: Prove bound by induction on balanced height

- Base case: UnBalanced trees
- Inductive step: One child has smaller balanced height  
→ by induction smaller error



# Summary – Further Work

Recap:

- (Randomized) Parameterized Approximation Algorithms for several problems.
- General Approximation Result for AATs.

Further questions:

- Concrete: Hamiltonicity on clique-width
- General: Deal with other operations (subtraction?)
- Soft: Other applications of AATs?
  - Problems  $W$ -hard on trees? (e.g. parameterized by degree)



Thank you!



Questions?

