Precoloring extension on chordal graphs

Dániel Marx

Budapest University of Technology and Economics

dmarx@cs.bme.hu

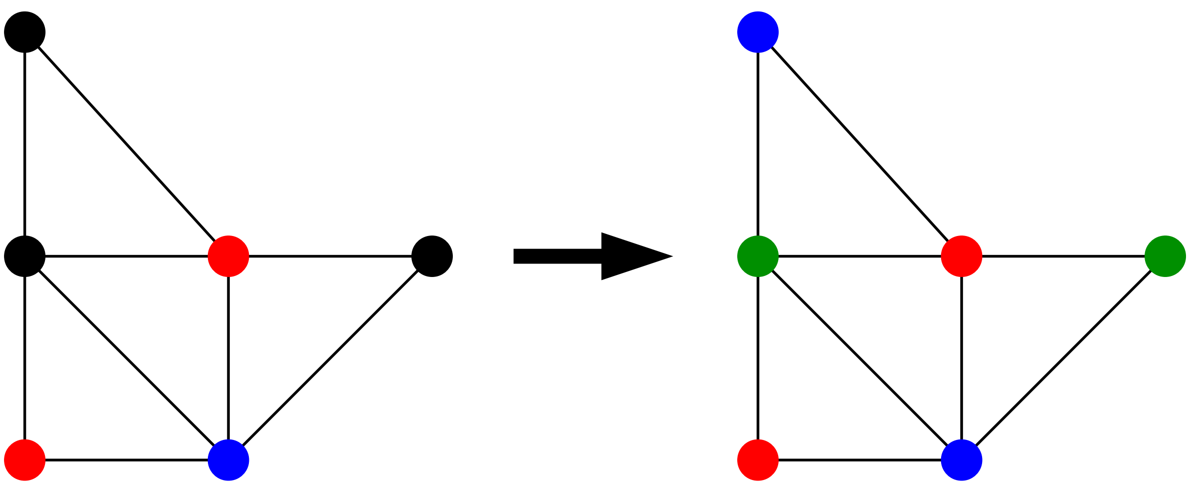
Graph Theory 2004,

July 5–9, Paris

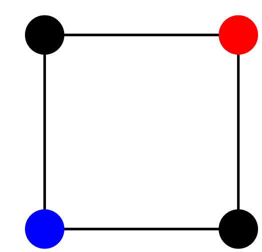
Precoloring extension

Generalization of vertex coloring: given a partial coloring, extend it to the whole graph using k colors.

Example for $k = 3$:



Example for $k = 2$:



Cannot be extended!

Precoloring extension (cont.)

Vertex coloring is a special case of precoloring extension (PREXT).

PREXT is polynomial time solvable for

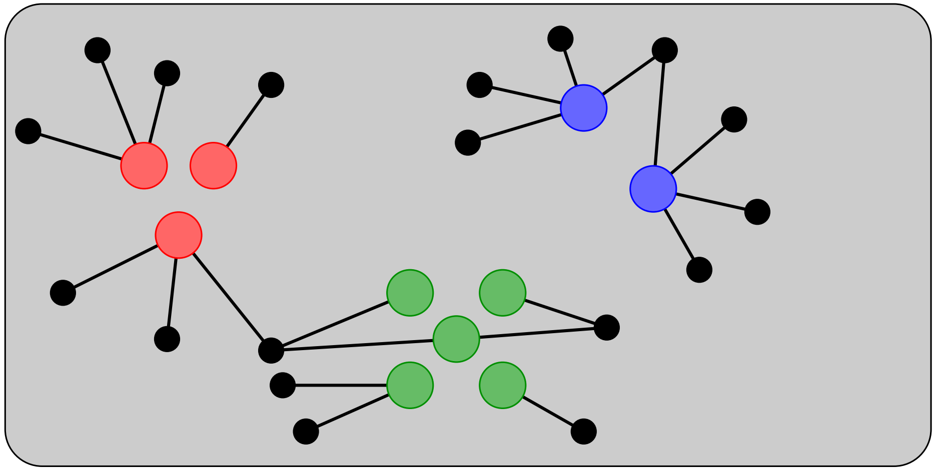
- complements of bipartite graphs
- cographs
- split graphs
- trees
- partial k-trees

PREXT is **NP**-complete for

- bipartite graphs
- line graphs of bipartite graphs
- line graphs of planar graphs
- interval graphs

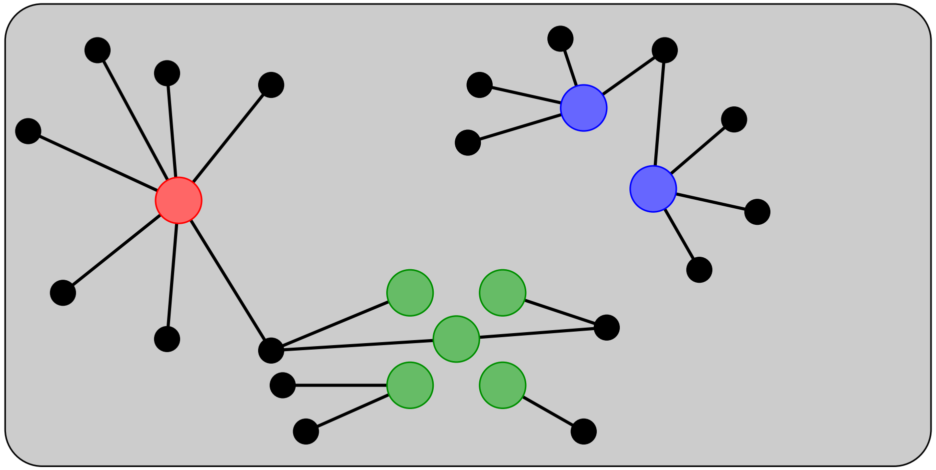
*The special case 1-*PREXT

1-PREXT: every color occurs at most once in the precoloring. In general, not easier than PREXT: the vertices precolored with the same color can be identified.



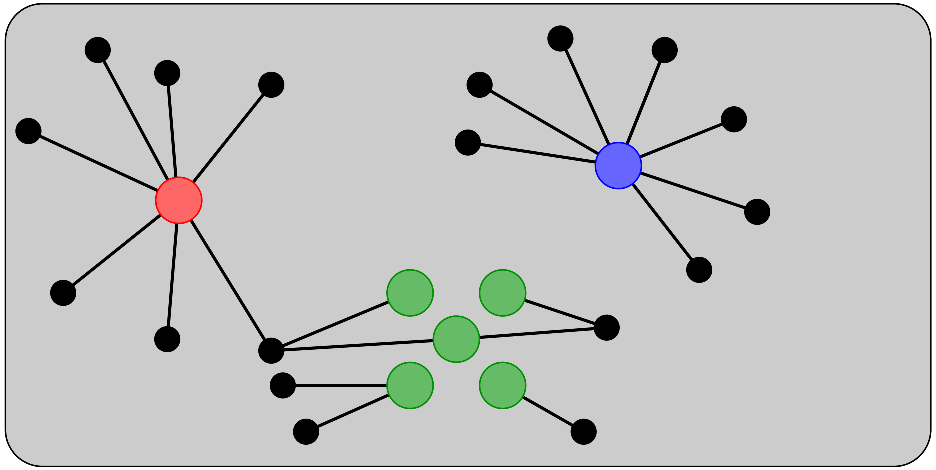
*The special case 1-*PREXT

1-PREXT: every color occurs at most once in the precoloring. In general, not easier than PREXT: the vertices precolored with the same color can be identified.



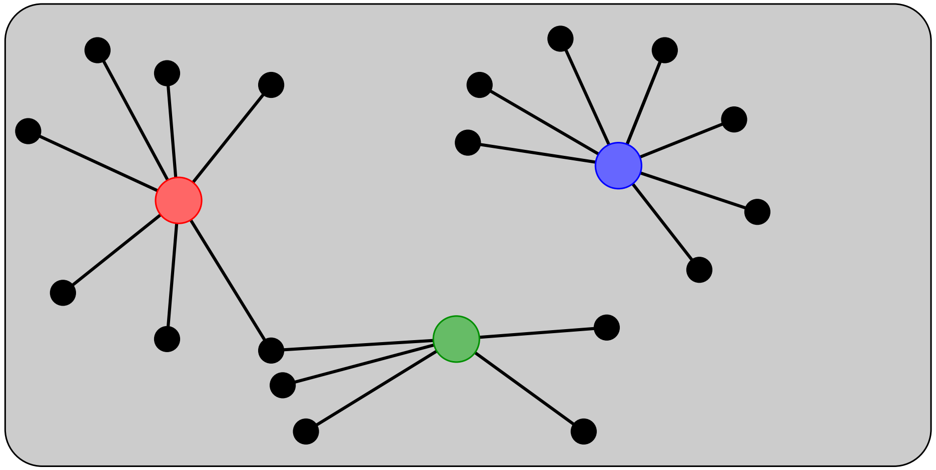
*The special case 1-*PREXT

1-PREXT: every color occurs at most once in the precoloring. In general, not easier than PREXT: the vertices precolored with the same color can be identified.



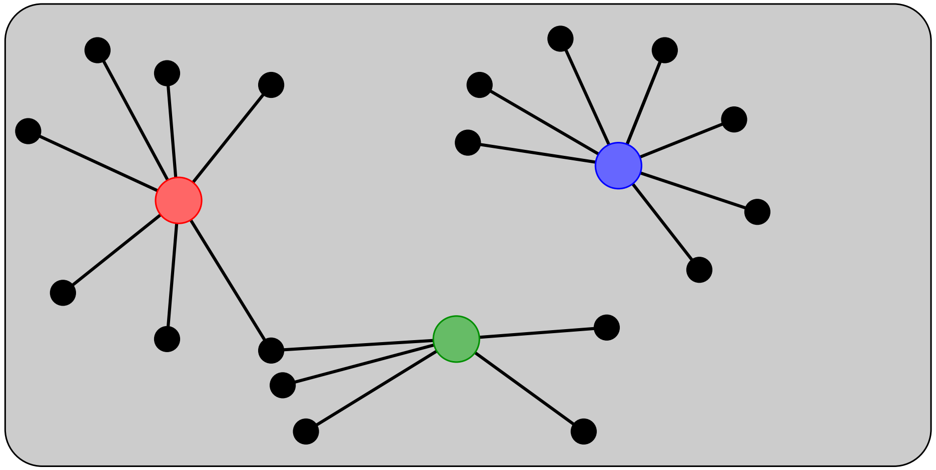
The special case 1-PREXT

1-PREXT: every color occurs at most once in the precoloring. In general, not easier than PREXT: the vertices precolored with the same color can be identified.



The special case 1-PREXT

1-PREXT: every color occurs at most once in the precoloring. In general, not easier than PREXT: the vertices precolored with the same color can be identified.

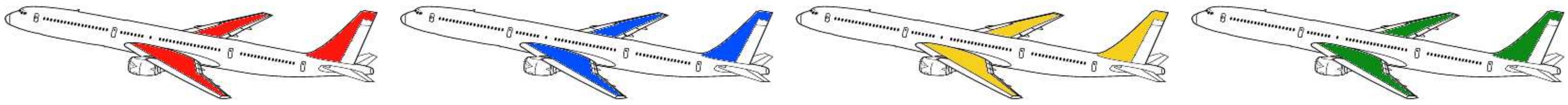


However, 1-PREXT can be easier for a restricted graph class: PREXT for interval graphs is **NP**-hard [Biró, Hujter, Tuza, 1992], even if every interval has the same length [M. 2003].

1-PREXT is polynomial-time solvable for interval graphs [Biró, Hujter, Tuza, 1992].

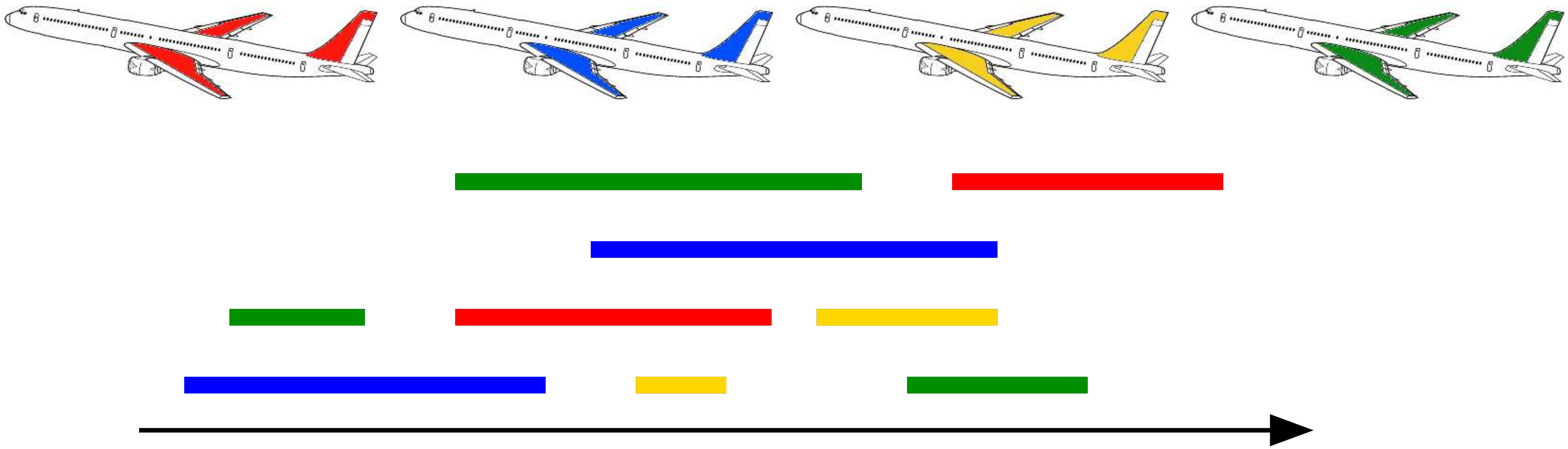
An application

Example: Assign aircrafts (colors) to the different flights (time intervals).



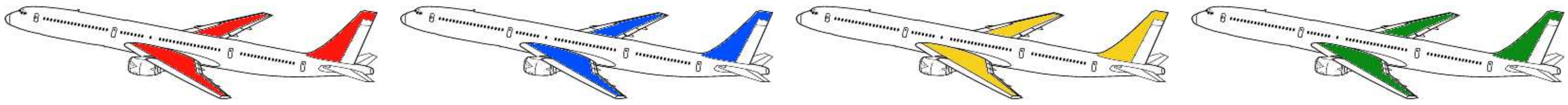
An application

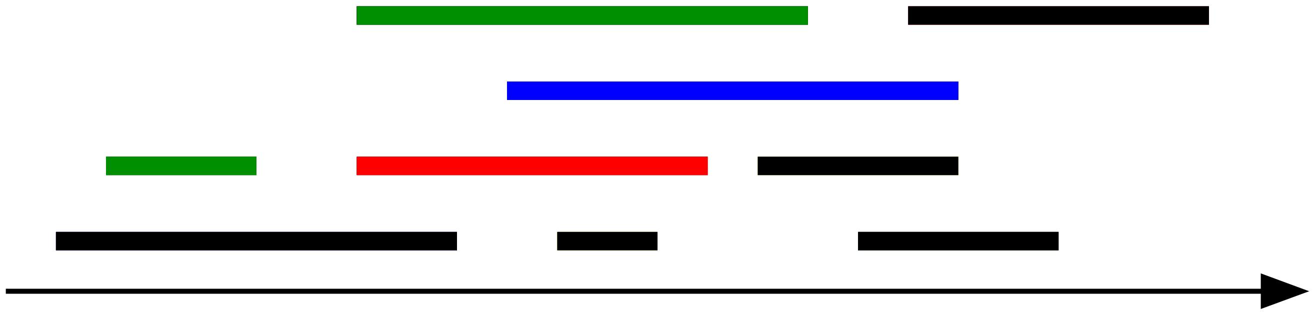
Example: Assign aircrafts (colors) to the different flights (time intervals).



An application

Example: Assign aircrafts (colors) to the different flights (time intervals).





Interval coloring is linear time solvable $\Rightarrow$ linear time algorithm for scheduling.

The problem is **NP**-hard if there are preassigned flights (PREXT).

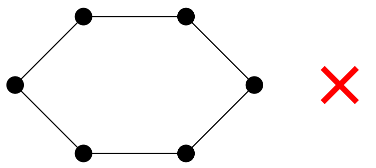
If each aircraft has a preassigned maintenance interval, then the problem can be solved in polynomial time (1-PREXT).

Chordal graphs

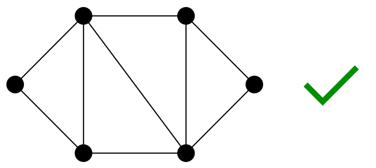
Question: [Biró, Hujter, Tuza] Is it possible to generalize the 1-PREXT algorithm for chordal graphs?

Our main result: 1-PREXT is polynomial-time solvable for **chordal graphs**.

A graph is **chordal** if it does not contain induced cycles longer than 3.



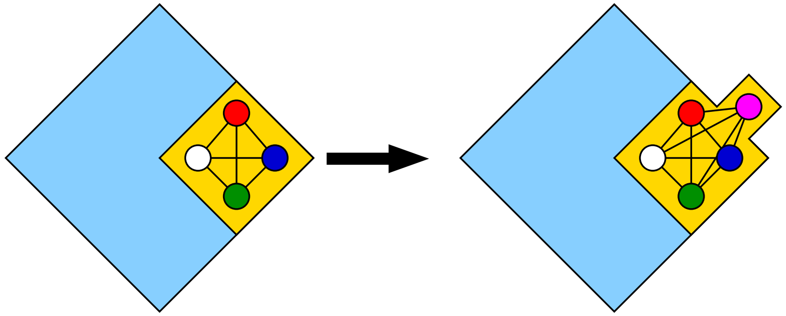
- Interval graphs are chordal.
- Intersection graphs of intervals on a line ⇔ interval graphs.
- Intersection graphs of subtrees in a tree ⇔ chordal graphs.
- Chordal graphs are perfect.



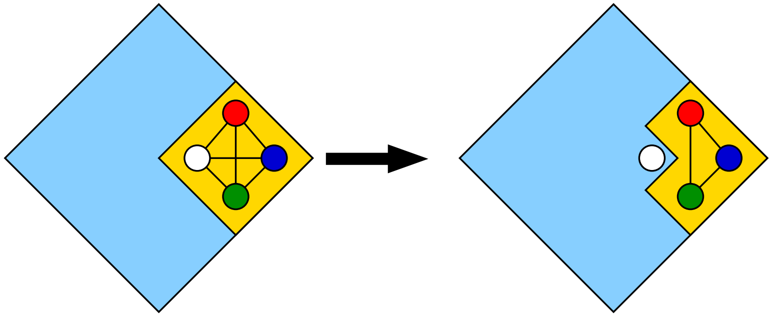
Tree decomposition

Every chordal graph can be built using the following operations. We consider chordal graphs with a distinguished clique.

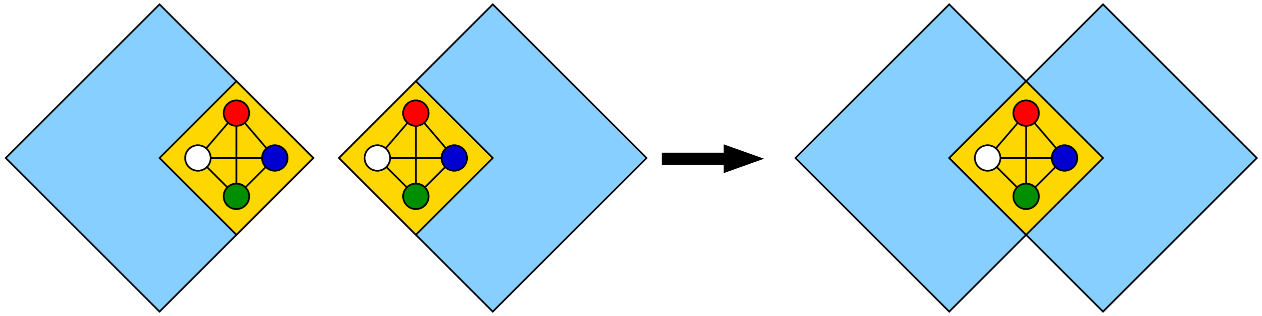
Add: attach a new vertex to the clique.



Forget: remove a vertex from the clique.



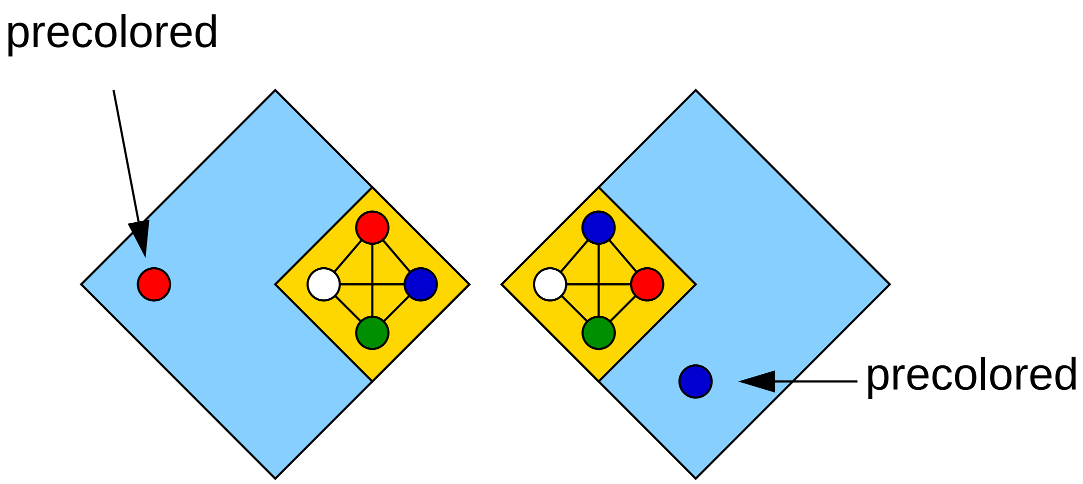
Join: identify the cliques of two chordal graphs.



Coloring chordal graphs

Tree decomposition gives a method of coloring chordal graphs. Main idea: before the **join** operation we can permute the colors such that the clique has the same coloring in both graphs.

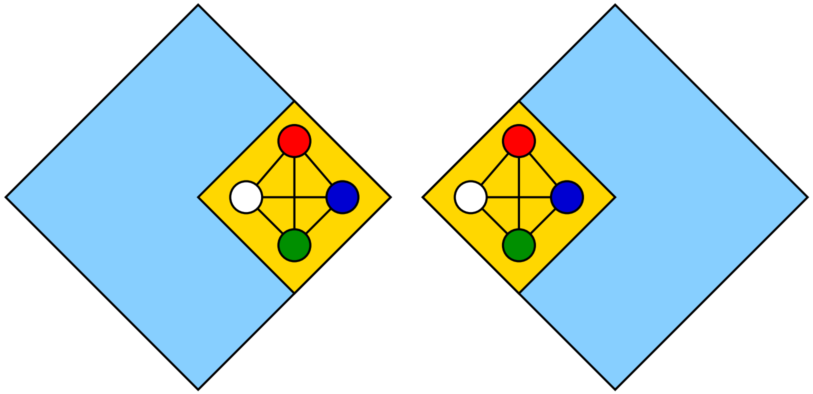
This approach does not work if there are precolored vertices:



The two colorings cannot be joined!

Precoloring extension

Idea 1: For each subgraph appearing in the construction of the chordal graph, list all possible colorings that can appear on the distinguished clique in a precoloring extension.

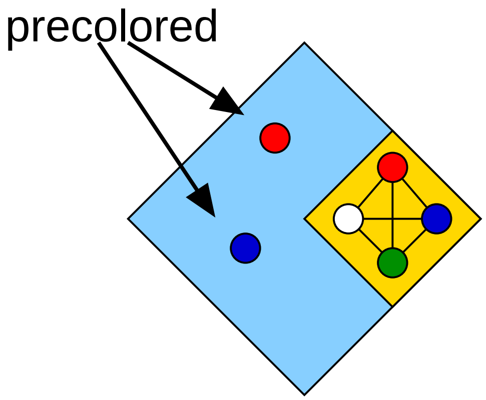


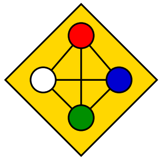
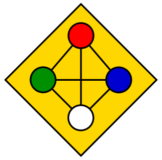
The graphs can be joined if they have colorings that agree on the clique. The colors outside the clique are not important.

Problem: There can be too many (exponentially many) colorings.

Colorings of the clique

For a subgraph H, let C_H be those colors that are used in the precoloring inside H.

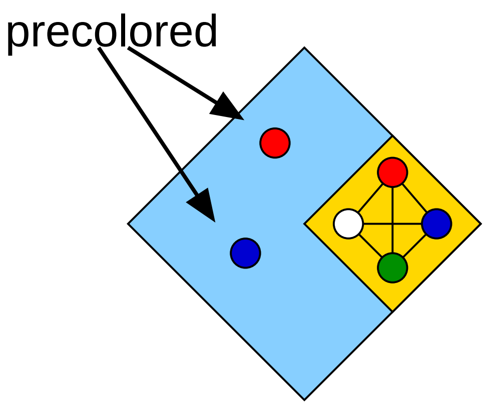


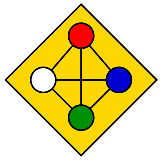
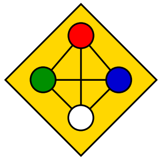
We do not have to distinguish between  and 

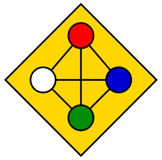
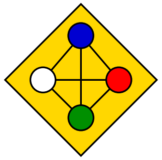
since colors **not** in C_H can be freely permuted.

Colorings of the clique

For a subgraph H, let C_H be those colors that are used in the precoloring inside H.



We do not have to distinguish between  and  since colors **not** in C_H can be freely permuted.

We do not have to distinguish between  and  either: outside H no vertex is precolored with C_H (1-PrExt!), thus the colors can be freely permuted.

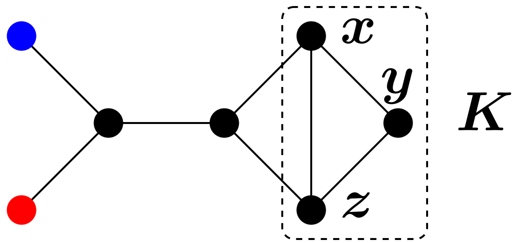
Idea 2: The only important thing in a coloring of the clique is which vertices receive colors from C_H, and which vertices receive colors not in C_H.

The set system

Let H be a graph with a distinguished clique K. Set system $\mathcal{S}(H, K)$ contains $S \subseteq K$ if and only if there is a precoloring extension on H such that

- vertices in S receive colors from C_H,
- vertices not in S receive colors not in C_H.

Example:



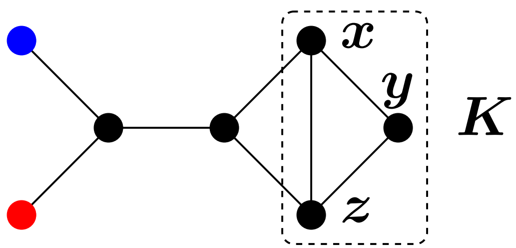
$C_H = \{$●, ●$\}$

The set system

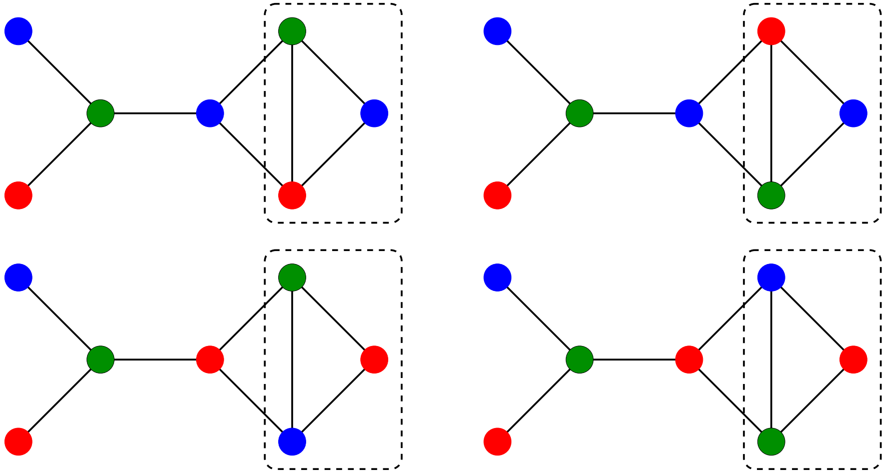
Let H be a graph with a distinguished clique K. Set system $\mathcal{S}(H,K)$ contains $S \subseteq K$ if and only if there is a precoloring extension on H such that

- vertices in S receive colors from C_H,
- vertices not in S receive colors not in C_H.

Example:



$C_H = \{$●, ●$\}$

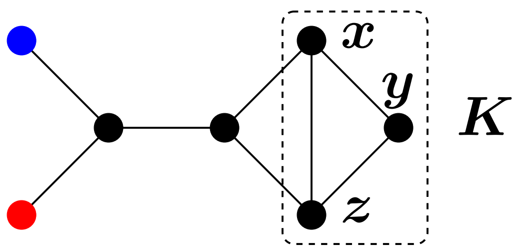


The set system

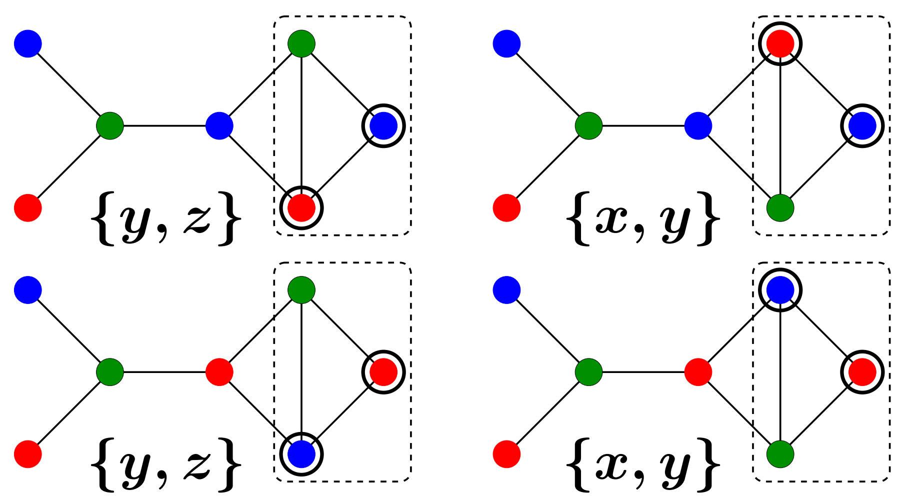
Let H be a graph with a distinguished clique K. Set system $\mathcal{S}(H, K)$ contains $S \subseteq K$ if and only if there is a precoloring extension on H such that

- vertices in S receive colors from C_H,
- vertices not in S receive colors not in C_H.

Example:



$C_H = \{ \bullet, \bullet \}$



$\mathcal{S}(H, K) = \{ \{x, y\}, \{y, z\} \}$

Representing the set systems

For each subgraph H appearing in the tree decomposition, we determine the set system $\mathcal{S}(H, K)$.

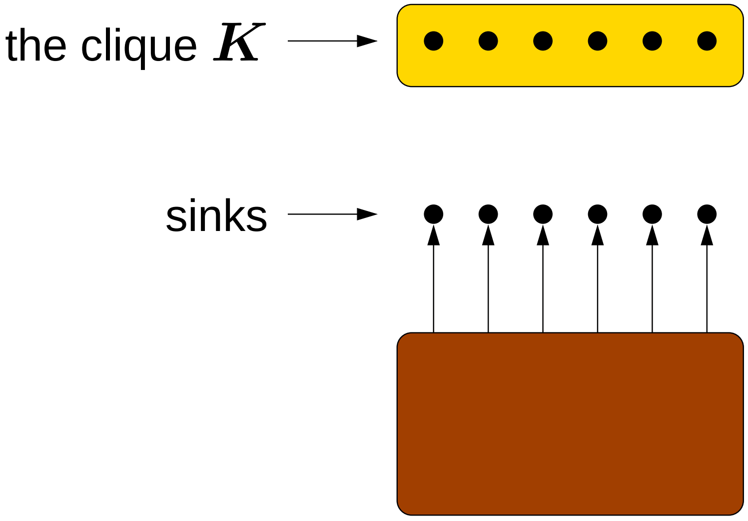
Problem: Size of $\mathcal{S}(H, K)$ can be exponentially large.

Representing the set systems

For each subgraph H appearing in the tree decomposition, we determine the set system $\mathcal{S}(H, K)$.

Problem: Size of $\mathcal{S}(H, K)$ can be exponentially large.

Idea 3: The set system $\mathcal{S}(H, K)$ can be compactly represented by network flows.



Sets in $\mathcal{S}(H, K)$

$\Updownarrow$

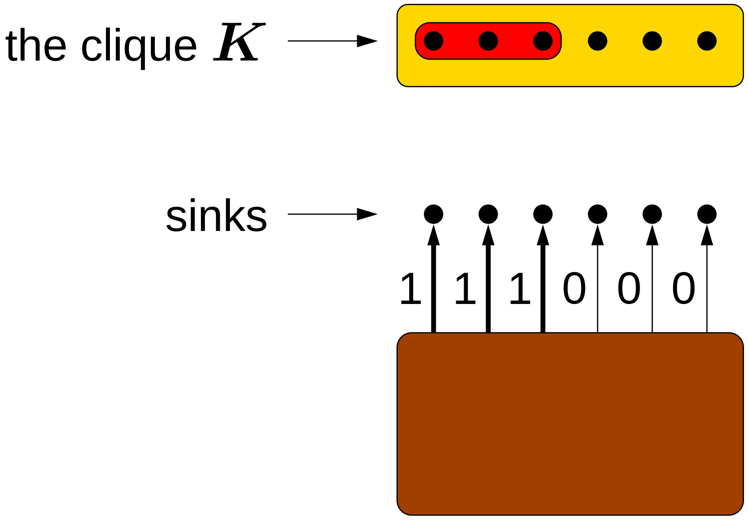
Maximum flows in the network

Representing the set systems

For each subgraph H appearing in the tree decomposition, we determine the set system $\mathcal{S}(H,K)$.

Problem: Size of $\mathcal{S}(H,K)$ can be exponentially large.

Idea 3: The set system $\mathcal{S}(H,K)$ can be compactly represented by network flows.



Sets in $\mathcal{S}(H,K)$

$\Updownarrow$

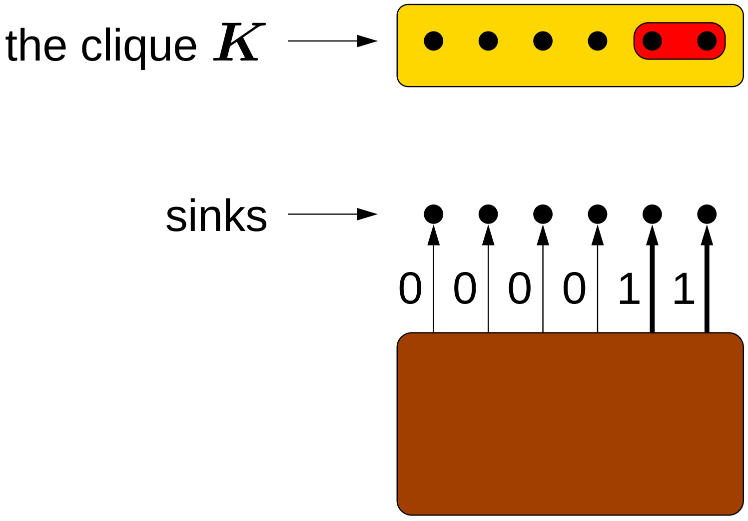
Maximum flows in the network

Representing the set systems

For each subgraph H appearing in the tree decomposition, we determine the set system $\mathcal{S}(H,K)$.

Problem: Size of $\mathcal{S}(H,K)$ can be exponentially large.

Idea 3: The set system $\mathcal{S}(H,K)$ can be compactly represented by network flows.



Sets in $\mathcal{S}(H,K)$

$\Updownarrow$

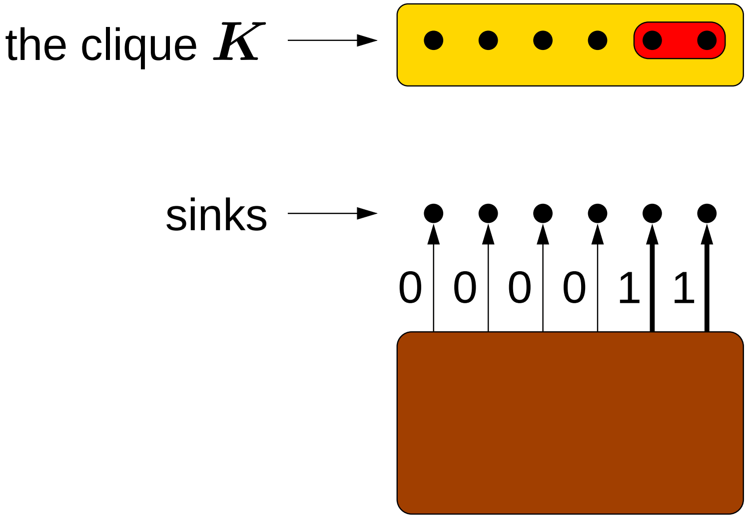
Maximum flows in the network

Representing the set systems

For each subgraph H appearing in the tree decomposition, we determine the set system $\mathcal{S}(H, K)$.

Problem: Size of $\mathcal{S}(H, K)$ can be exponentially large.

Idea 3: The set system $\mathcal{S}(H, K)$ can be compactly represented by network flows.



Sets in $\mathcal{S}(H, K)$

$\Updownarrow$

Maximum flows in the network

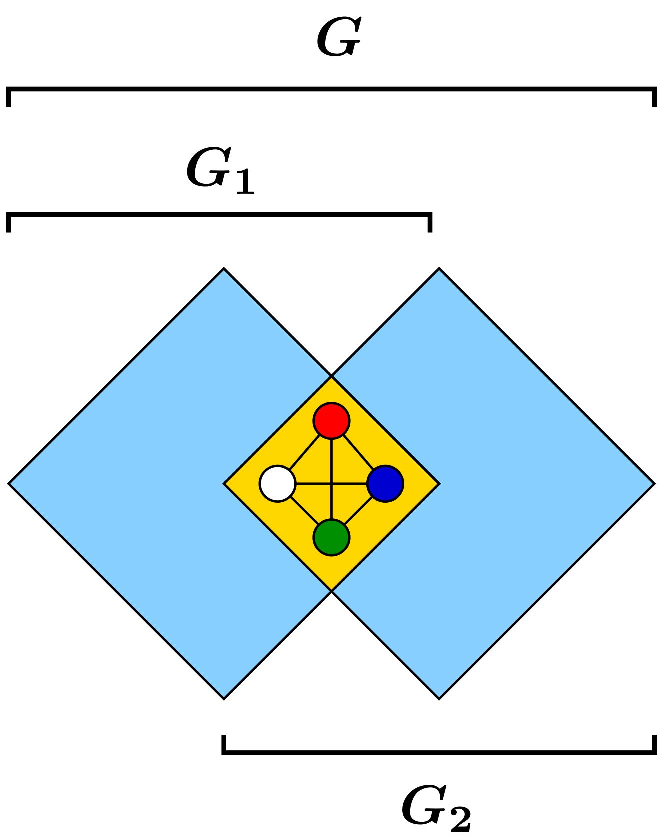
Theorem: $\mathcal{S}(H, K)$ is the projection of the basis set of a matroid.

Building the networks

When we build the graph with the **add**, **forget**, and **join** operations, we can build at the same time the networks representing the set systems.

In the case of **join**, we use the following lemma:

$$S \in \mathcal{S}(G, K)$$
$$\Updownarrow$$
S can be partitioned into S_1 and S_2 such that
$S_1 \in \mathcal{S}(G_1, K)$ and $S_2 \in \mathcal{S}(G_2, K)$
(+ some technical condition holds)



Building the networks

When we build the graph with the **add**, **forget**, and **join** operations, we can build at the same time the networks representing the set systems.

In the case of **join**, we use the following lemma:

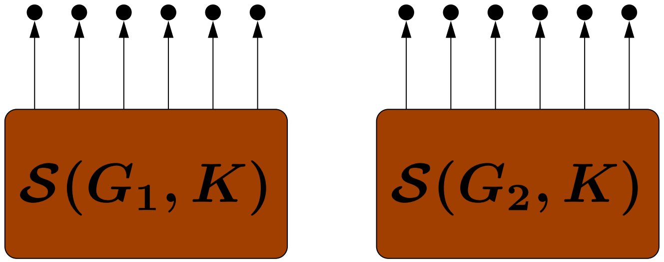
$$S \in \mathcal{S}(G, K)$$

$$\Updownarrow$$

S can be partitioned into S_1 and S_2 such that

$S_1 \in \mathcal{S}(G_1, K)$ and $S_2 \in \mathcal{S}(G_2, K)$

(+ some technical condition holds)



Building the networks

When we build the graph with the **add**, **forget**, and **join** operations, we can build at the same time the networks representing the set systems.

In the case of **join**, we use the following lemma:

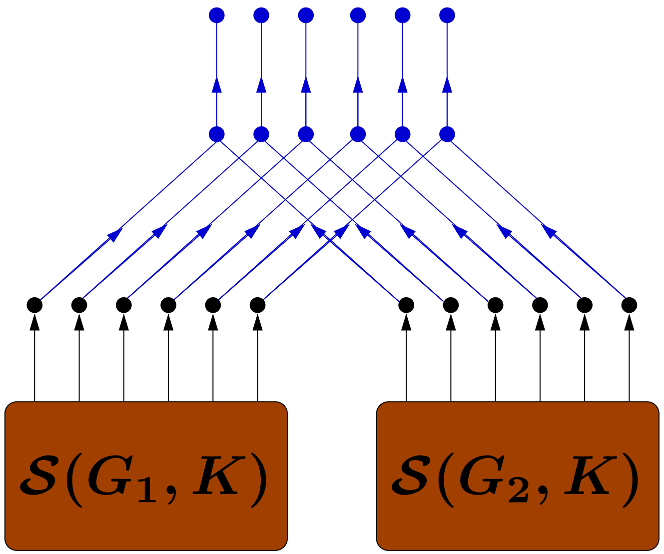
$$\boldsymbol{S} \in \boldsymbol{\mathcal{S}}(\boldsymbol{G}, \boldsymbol{K})$$

$$\Updownarrow$$

$\boldsymbol{S}$ can be partitioned into $\boldsymbol{S_1}$ and $\boldsymbol{S_2}$ such that

$\boldsymbol{S_1} \in \boldsymbol{\mathcal{S}}(\boldsymbol{G_1}, \boldsymbol{K})$ and $\boldsymbol{S_2} \in \boldsymbol{\mathcal{S}}(\boldsymbol{G_2}, \boldsymbol{K})$

(+ some technical condition holds)



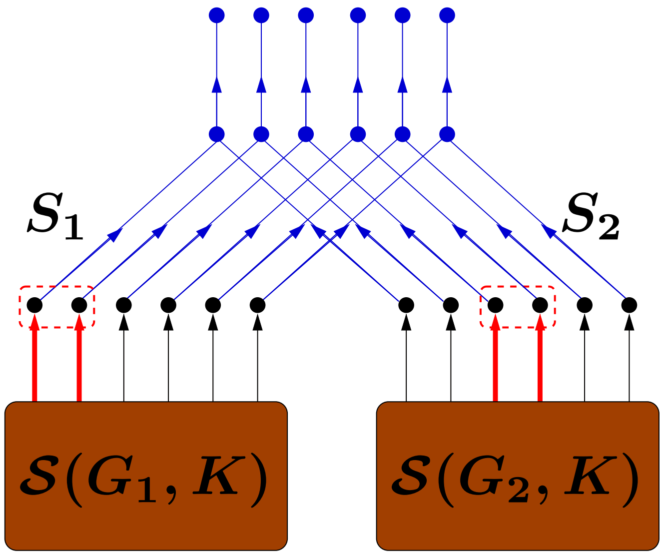
Building the networks

When we build the graph with the **add**, **forget**, and **join** operations, we can build at the same time the networks representing the set systems.

In the case of **join**, we use the following lemma:

$$S \in \mathcal{S}(G, K)$$
$$\Updownarrow$$
S can be partitioned into S_1 and S_2 such that $S_1 \in \mathcal{S}(G_1, K)$ and $S_2 \in \mathcal{S}(G_2, K)$

(+ some technical condition holds)

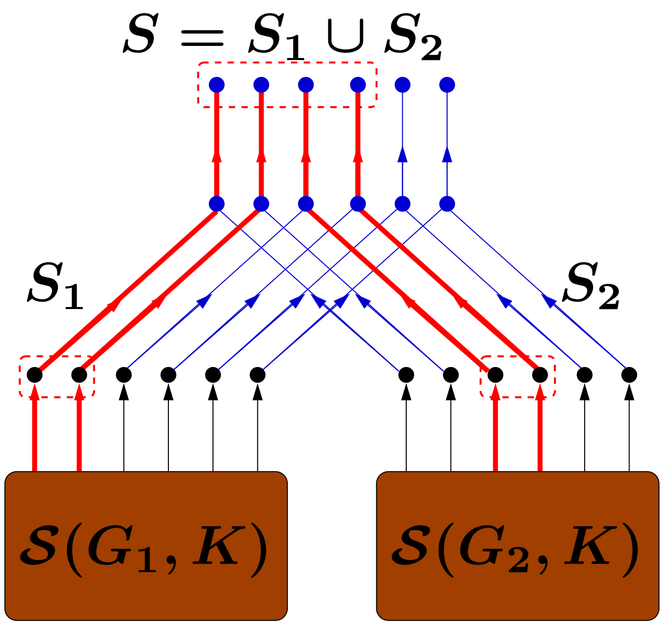


Building the networks

When we build the graph with the **add**, **forget**, and **join** operations, we can build at the same time the networks representing the set systems.

In the case of **join**, we use the following lemma:

$$S \in \mathcal{S}(G, K)$$
$$\Updownarrow$$
S can be partitioned into S_1 and S_2 such that $S_1 \in \mathcal{S}(G_1, K)$ and $S_2 \in \mathcal{S}(G_2, K)$ (+ some technical condition holds)



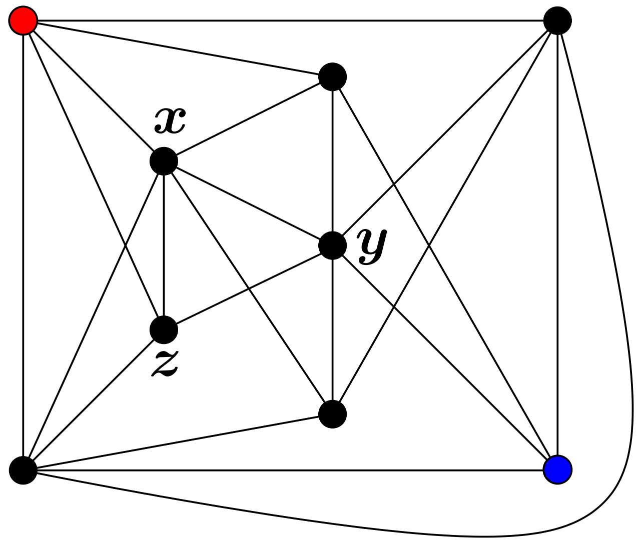
The algorithm

Algorithm for 1-PrExt on chordal graphs:

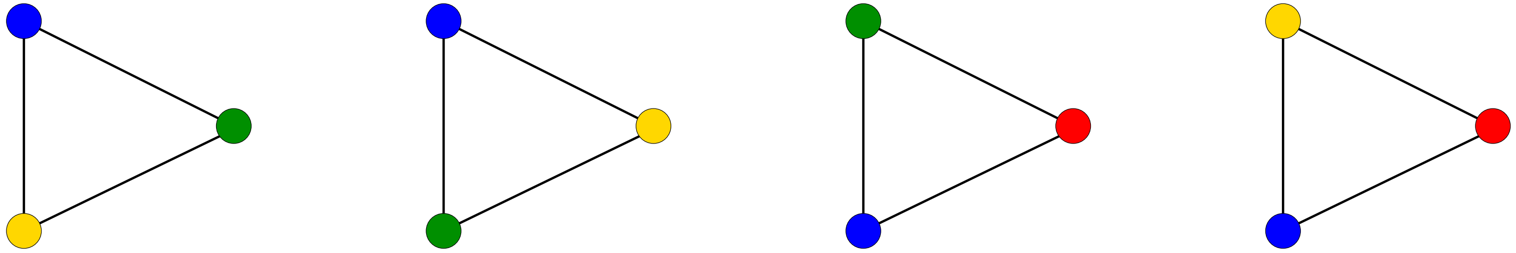
- Find a tree decomposition of G.
- For each subgraph H in the tree decomposition, construct a network representing $\mathcal{S}(H, K)$.
- The network for G can be used to determine whether there is a precoloring extension for G or not.

A non-matroidal example

If the graph is not chordal, then the set system may not be the projection of a matroid:



There are 4 possible colorings:



$$\mathcal{S}(G,K) = \{\ \{x\}, \{y,z\}\ \}$$

Summary

- Special case 1-PREXT of PREXT.
- **Previous result:** 1-PREXT is polynomial-time solvable for **interval** graphs.
- **Our result:** 1-PREXT is polynomial-time solvable for **chordal** graphs.
- Set system $\mathcal{S}(H, K)$ is the projection of a matroid if the graph is chordal.
- If the graph is not chordal, the this set system is not necessarily a matroid.

Summary

- Special case 1-PREXT of PREXT.
- **Previous result:** 1-PREXT is polynomial-time solvable for **interval** graphs.
- **Our result:** 1-PREXT is polynomial-time solvable for **chordal** graphs.
- Set system $\mathcal{S}(H, K)$ is the projection of a matroid if the graph is chordal.
- If the graph is not chordal, the this set system is not necessarily a matroid.

Thank you for your attention!
Questions?