Injective oriented colourings

Gary MacGillivray
Mathematics and Statistics
University of Victoria
Victoria, British Columbia, Canada
gmacgill@math.uvic.ca

Joint work with Andre Raspaud (Bordeaux)
and Jacobus Swarts (Victoria)

Motivation

- Bring together
  - injective colourings
    [Fiala, Hahn, Hell, Fertin, Kratochvil, Por, Rizzi, Siran, Sotteau, Stacho, Vialette, …]
  - oriented colourings
    [Klostermeyer, Kostochka, M, Nesetril, Ochem, Pinlou, Raspaud, Sopena, Wood, …]
- Build on earlier work Courcelle, and Raspaud and Sopena

Some issues in the study of a colouring parameter

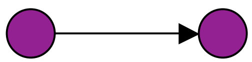
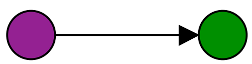
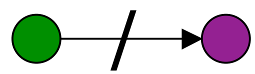
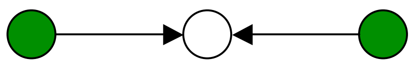
- Is there a homomorphism model? Where does it lead?
- What is the complexity of the decision problem?
- Can obstructions be found algorithmically in polynomial cases? What are the critical graphs?
- What bounds are available? Is there a Brooks' Theorem?
- Are there special bounds and / or algorithms for restricted graph classes?
- Can the colourings that use *k* colours be counted?

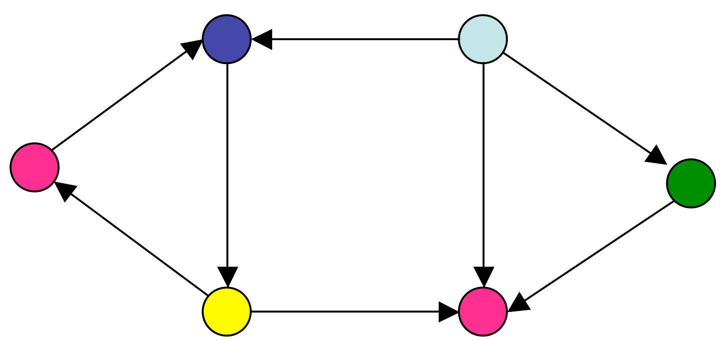
Some issues in the study of a colouring parameter

- Is there a homomorphism model? Where does it lead?
- What is the complexity of the decision problem?
- Can obstructions be found algorithmically in polynomial cases? What are the critical graphs?
- What bounds are available? Is there a Brooks' Theorem?
- Are there special bounds and / or algorithms for restricted graph classes? (oriented trees)
- Can the colourings that use *k* colours be counted?

Injective oriented colourings

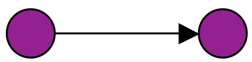
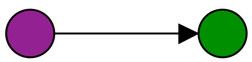
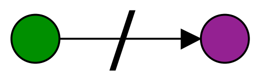
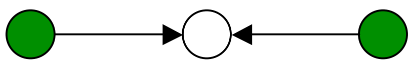
An injective oriented *k*-colouring of a digraph *D* is an assignment of *k* colours to the vertices of *D* so that:

- no 
- if  then 
- no 

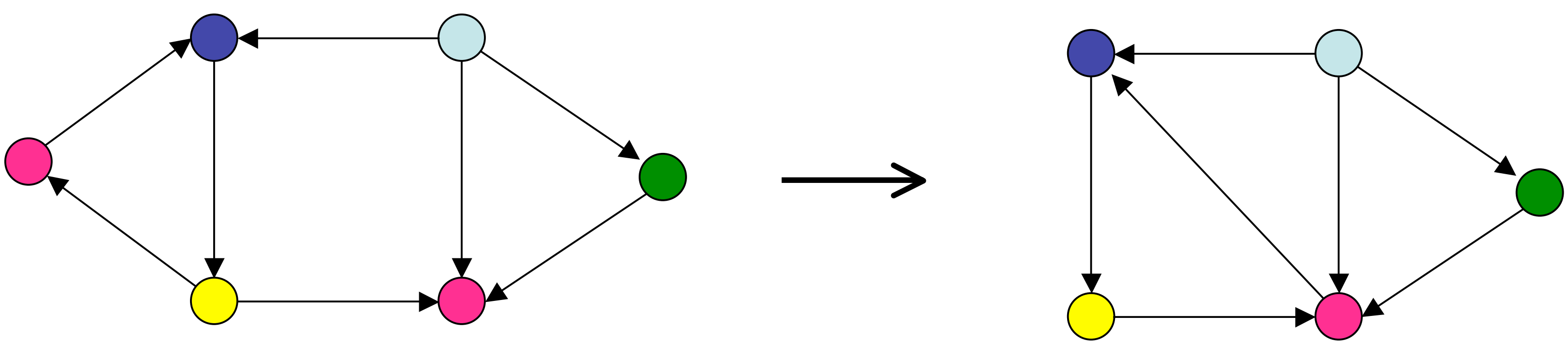


Injective oriented colourings

An injective oriented k-colouring of a digraph D is an assignment of k colours to the vertices of D so that:

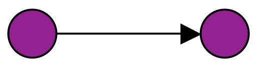
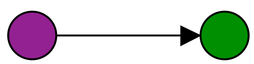
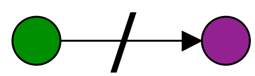
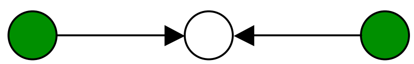
- no 
- if  then 
- no 

Equivalently, an injective oriented k-colouring is an injective homomorphism to an oriented graph on k vertices (where injective is "on in-neighbourhoods").

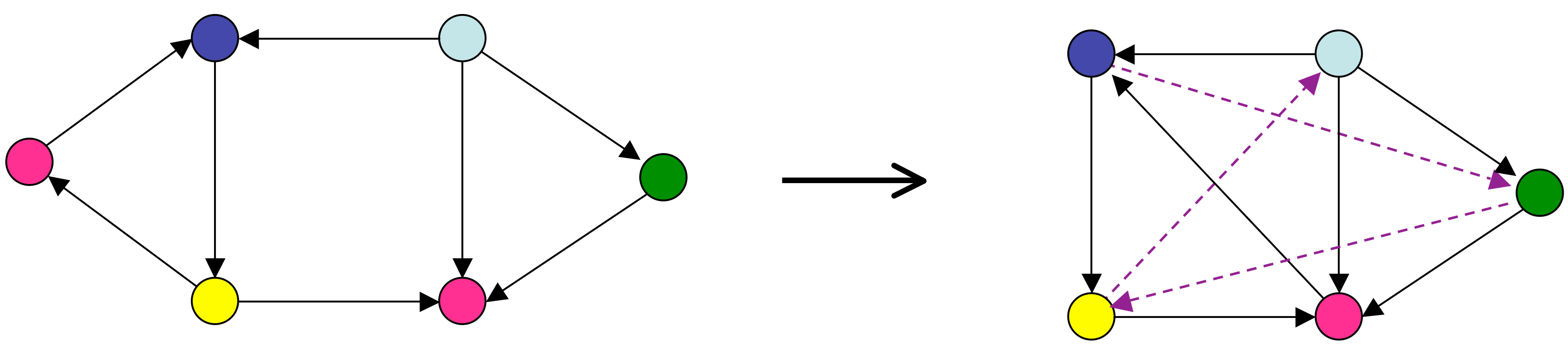


Injective oriented colourings

An injective oriented k-colouring of a digraph D is an assignment of k colours to the vertices of D so that:

- no 
- if  then 
- no 

Equivalently, an injective oriented k-colouring is an injective homomorphism to a **tournament** on k vertices (where injective is "on in-neighbourhoods").



Complexity of Injective Oriented k-Colouring

For $k = 1, 2$ and 3 injective oriented k-colouring is Polynomial.

(reduction to 2-SAT, or direct colouring algorithm)

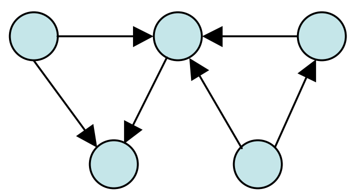
For each fixed $k \geq 4$, injective oriented k-colouring is NP-complete.

(ultimately a transformation from 3-colouring)

The **injective oriented chromatic number of *D* …**

… is the smallest k for which there exists an injective oriented k-colouring of D. (Notation $\chi_{io}(D)$.)

There exist io-cliques: oriented graphs D for which $\chi_{io}(D) = |V|$.



Proposition. An oriented graph D is an io-clique if and only if any two non-adjacent vertices have a common out-neighbour or are joined by a directed path of length two.

Any oriented graph is an induced subgraph of an io-clique.

A Brooks' Theorem for χ_{io}

Theorem. If D is not an io-clique then $\chi_{io} \leq 2^t$-1, where $t = \Delta + (\Delta^- - 1)\Delta^+ + (\Delta^+)^2$, and Δ is the maximum degree of the underlying simple graph.

Exponential is best possible

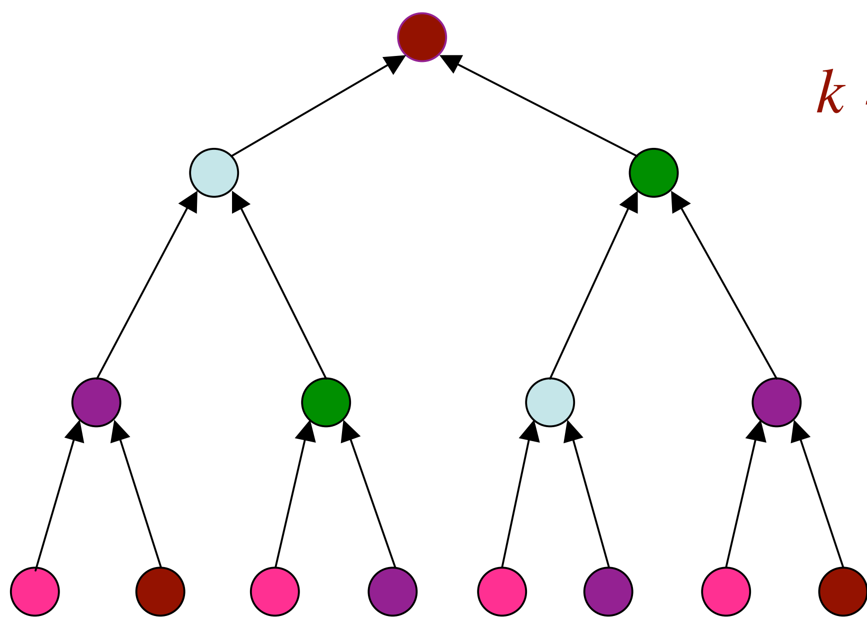
- *D:* disjoint union of all tournaments on *(d+1)* vertices.
- $\Delta^+ = \Delta^- = \Delta = d$, and
- χ_{io} is at least the size of a *(d+1)*-universal tournament (at least $2^{d/2}$, [Moon, 1968])

A better bound for trees

$\chi_{io} \leq 2k + 1$, where $k = \max\{\Delta^+, \Delta^-\}$.

Idea: T has an injective homomorphism to vertex-transitive k-regular tournament.

Best possible:



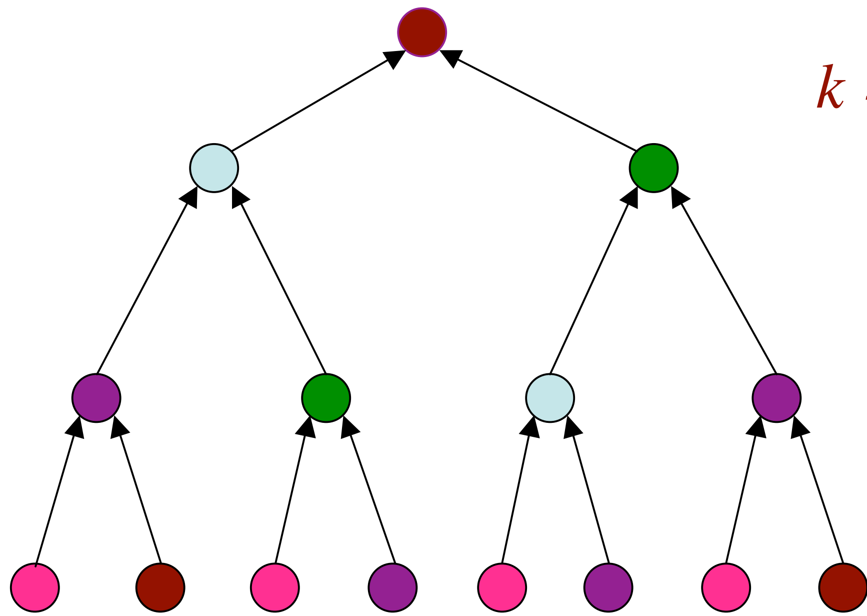
$k = 2 \cdot 2 + 1 = 5$

A better bound for trees

$\chi_{io} \leq 2k + 1$, where $k = \max\{\Delta^+, \Delta^-\}$.

Idea: T has an injective homomorphism to vertex-transitive k-regular tournament.

Best possible:

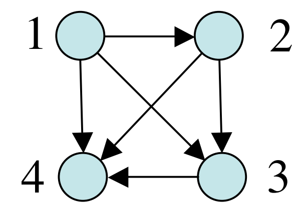


$k = 2 \cdot 2 + 1 = 5$

Examples are complete k-ary trees like this.

Simple linear algorithm for trees

- For the decision problem only. Test for the existence of a homomorphism to each tournament on k vertices, ex.



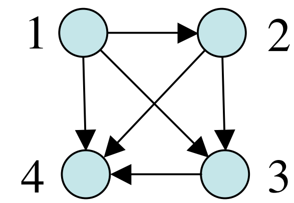
k fixed $\Rightarrow$ number of tournaments on k vertices is a constant

- Main ideas:
  - give each vertex a list of available colours
  - start at the lowest level and work upwards at first
  - reduce lists using consistency checks
  - colour downwards

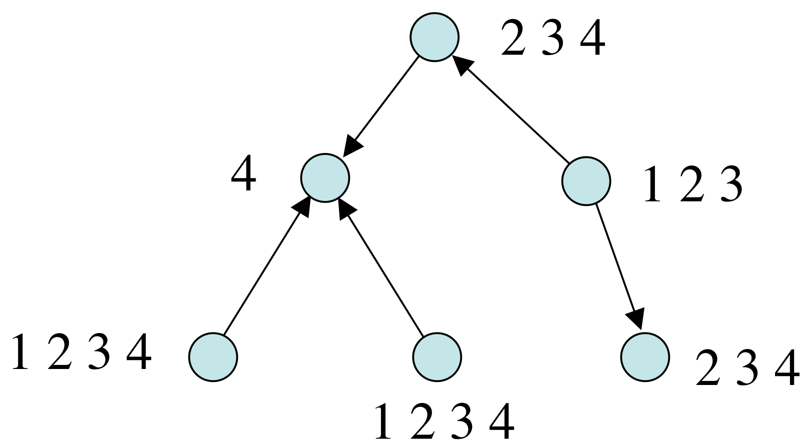
- same algorithm works for any target digraph

- Courcelle's theorem also gives an algorithm

Example: simple linear algorithm for trees

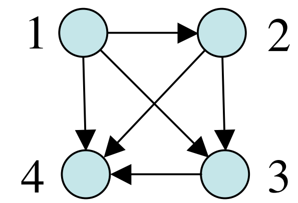
Test for the existence of a homomorphism to 

- give each vertex a list of available colours (respect in-degrees)
- start at the lowest level and work upwards at first
- reduce lists using consistency checks
- colour downwards

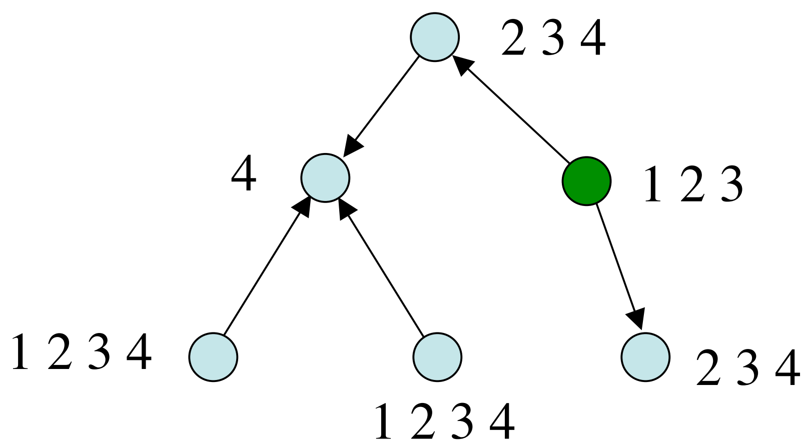


Example: simple linear algorithm for trees

Test for the existence of a homomorphism to



- give each vertex a list of available colours
- start at the lowest level and work upwards at first
- reduce lists using consistency checks
- colour downwards

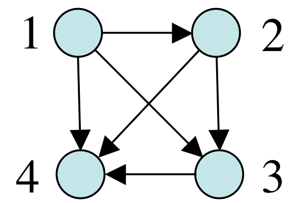


Condition: If *c* is in *L*(*u*) after *u* is processed, then if *u* is coloured with *c*, the colouring extends to the next level

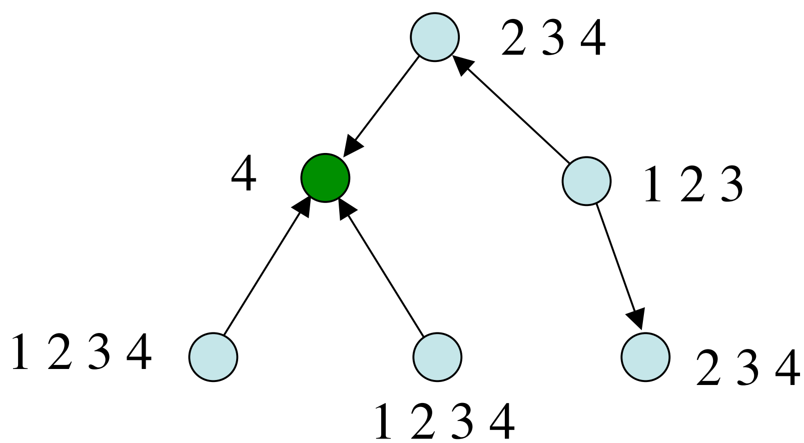
- no further restrictions

Example: simple linear algorithm for trees

Test for the existence of a homomorphism to



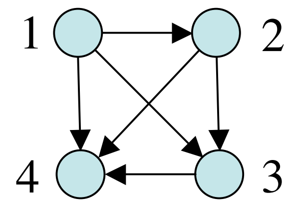
- give each vertex a list of available colours
- start at the lowest level and work upwards at first
- reduce lists using consistency checks
- colour downwards



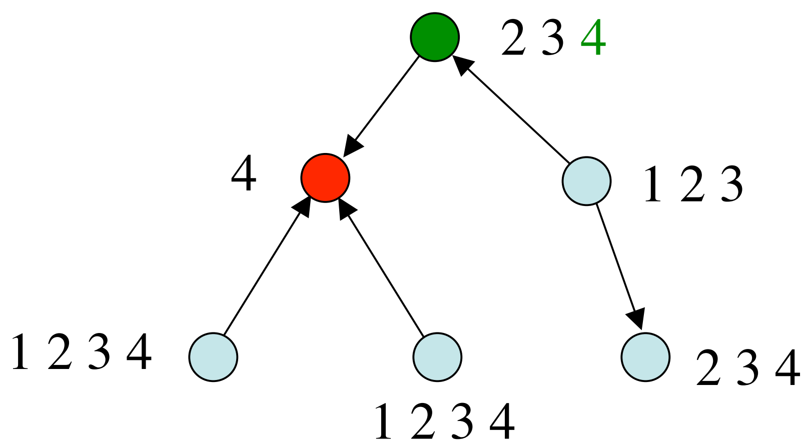
Condition: If *c* is in *L(u)* after *u* is processed, then if *u* is coloured with *c*, the colouring extends to the next level

- orientation of the arc to the lower level $\Rightarrow$ SDR of lists of in-nbrs is computed at a later step

Example: simple linear algorithm for trees

Test for the existence of a homomorphism to 

- give each vertex a list of available colours
- start at the lowest level and work upwards at first
- reduce lists using consistency checks
- colour downwards

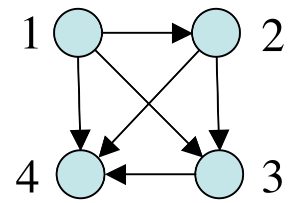


Condition: If c is in $L(u)$ after u is processed, then if u is coloured with c, the colouring extends to the next level

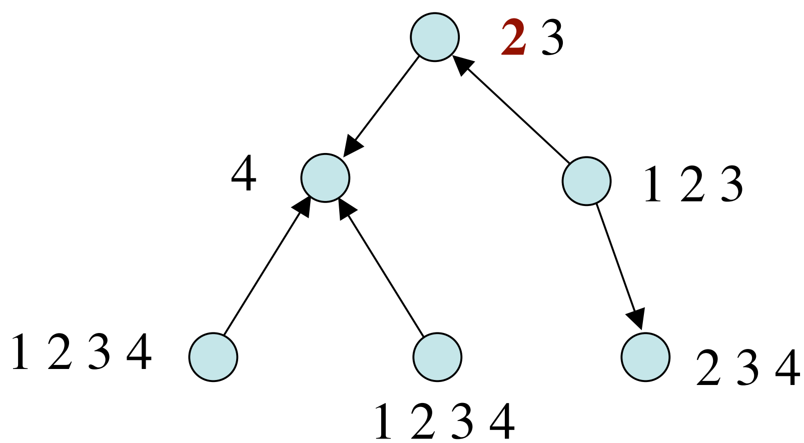
- 4 can't be used

- keep 2 (3) if there is a SDR of lists of in-nbrs of in which 2 (3) represents

Example: simple linear algorithm for trees

Test for the existence of a homomorphism to 

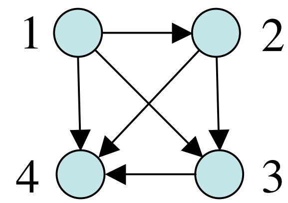
- give each vertex a list of available colours
- start at the lowest level and work upwards at first
- reduce lists using consistency checks
- colour downwards



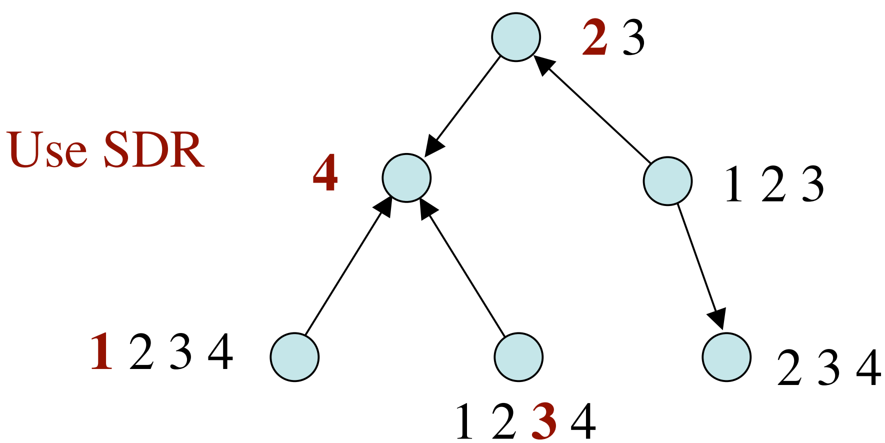
Condition: If c is in $L(u)$ after u is processed, then if u is coloured with c, the colouring extends to the next level

Example: simple linear algorithm for trees

Test for the existence of a homomorphism to



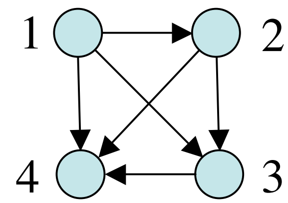
- give each vertex a list of available colours
- start at the lowest level and work upwards at first
- reduce lists using consistency checks
- colour downwards



Condition: If c is in $L(u)$ after u is processed, then if u is coloured with c, the colouring extends to the next level

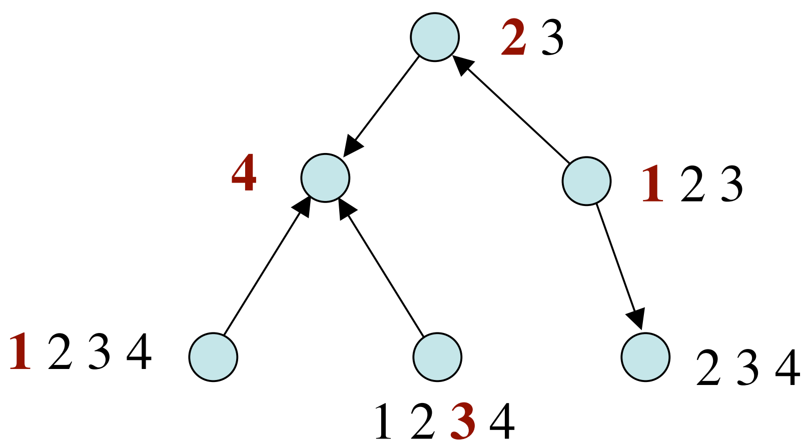
Example: simple linear algorithm for trees

Test for the existence of a homomorphism to



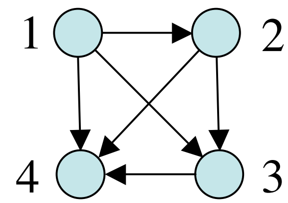
- give each vertex a list of available colours
- start at the lowest level and work upwards at first
- reduce lists using consistency checks
- colour downwards

Condition: If c is in $L(u)$ after u is processed, then if u is coloured with c, the colouring extends to the next level

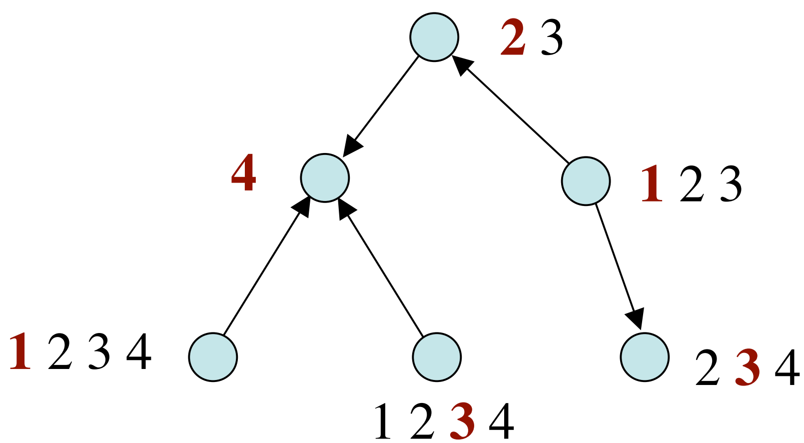


Example: simple linear algorithm for trees

Test for the existence of a homomorphism to



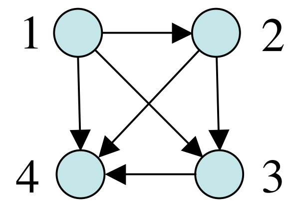
- give each vertex a list of available colours
- start at the lowest level and work upwards at first
- reduce lists using consistency checks
- colour downwards



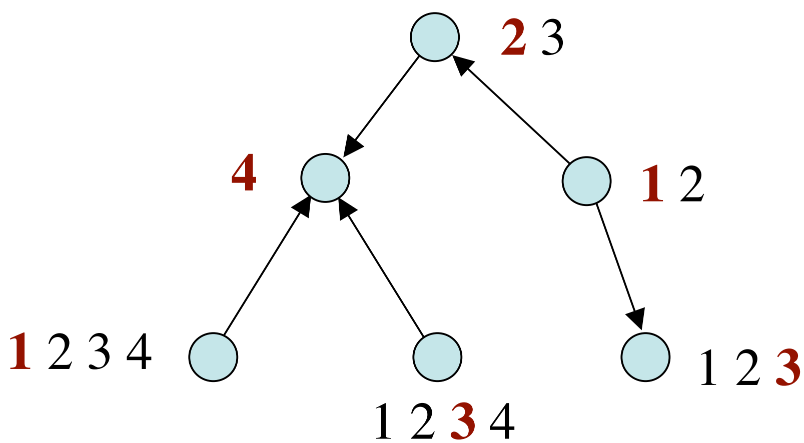
Condition: If c is in $L(u)$ after u is processed, then if u is coloured with c, the colouring extends to the next level

Example: simple linear algorithm for trees

Test for the existence of a homomorphism to



- give each vertex a list of available colours
- start at the lowest level and work upwards at first
- reduce lists using consistency checks
- colour downwards



Linear: Each vertex processed twice; list sizes bounded by $|V(T)|$, as is number of sets in each SDR check

Some things we do / don't know

- For k = 1, 2, 3 there is an algorithm that either produces an injective oriented k-colouring, or produces an obstruction.
- A similar but not identical theory exists when the colourings need not be proper

- We have no results and bounds for most well-studied classes of graphs, ex. *k*-trees, planar.
- We have no results on enumeration of these colourings.


Thank you for listening