Graph Isomorphisms & Automorphisms

University of Twente & Nedap University
Nedap University 2016–2018 graduation

A. G. van Herwijnen   C. A. M. A. Reuvers   F. T. F. Meijer Cluwen
M. H. B. Banierink   M. M. Slot   E. Huizinga

23-04-2018

Graphs for dummies

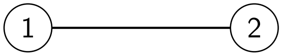
- Vertex

Graphs for dummies

- Vertex

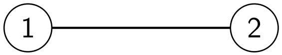
Graphs for dummies

- Vertex
- Edge



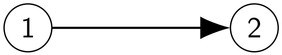
Graphs for dummies

- Vertex
- Edge
- Graph



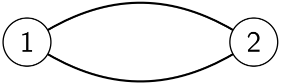
Graphs for dummies

- Vertex
- Edge
- Graph
  - Undirected vs. directed



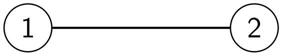
Graphs for dummies

- Vertex
- Edge
- Graph
  - Undirected vs. directed
  - Simple vs. multi-edge



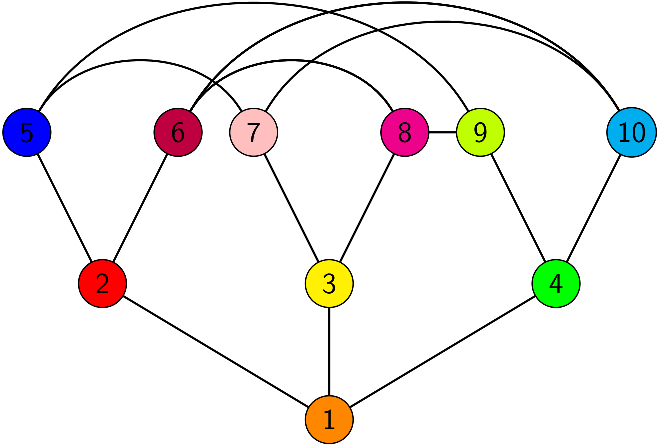
Graphs for dummies

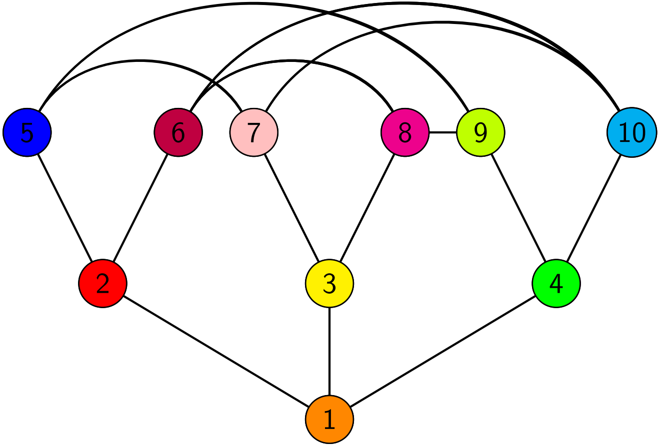
- Vertex
- Edge
- Graph
  - Undirected vs. directed
  - Simple vs. multi-edge

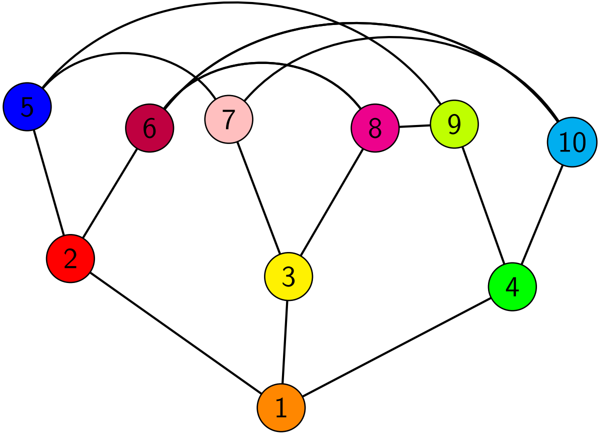


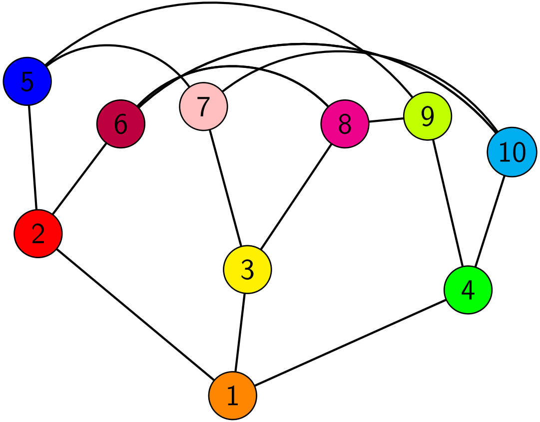
We only consider simple, undirected graphs

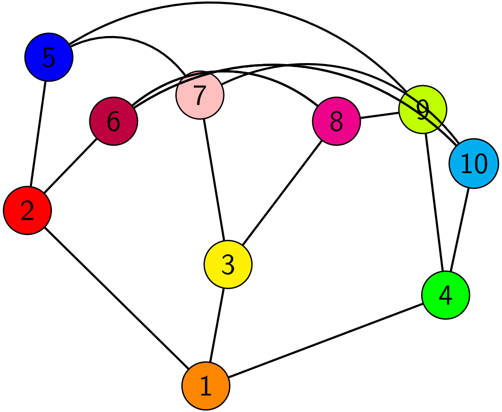
What is graph isomorphism?

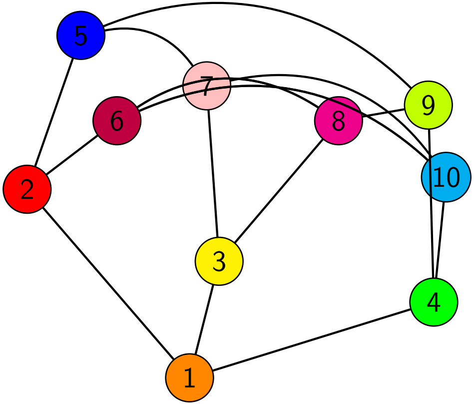


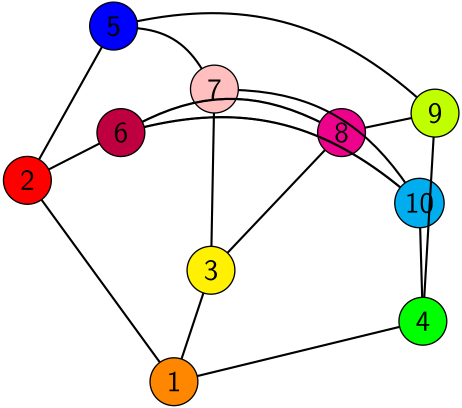

5
6
7
8
9
10
2
3
4
1

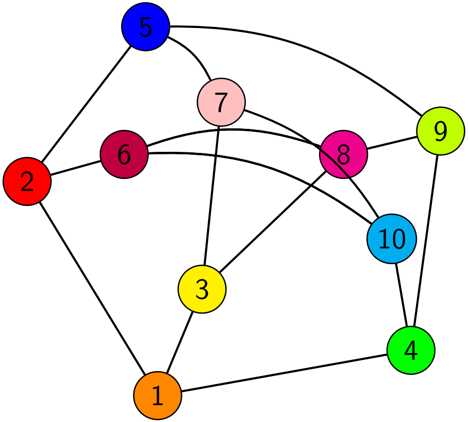

5
6
7
8
9
10
2
3
4
1

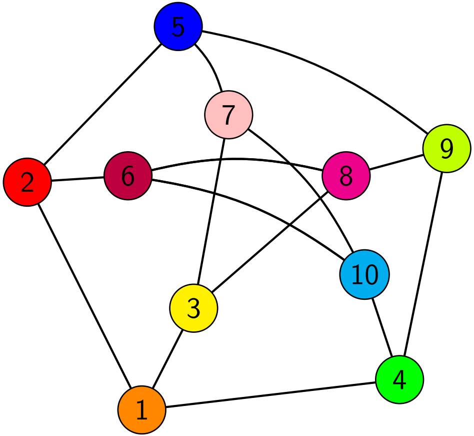

5
6
7
8
9
10
2
3
4
1

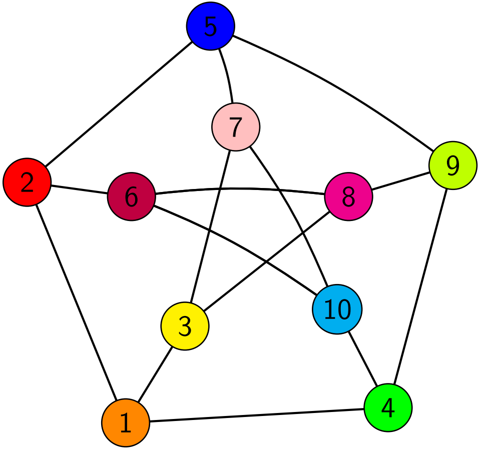

5
7
6
8
9
10
2
3
4
1

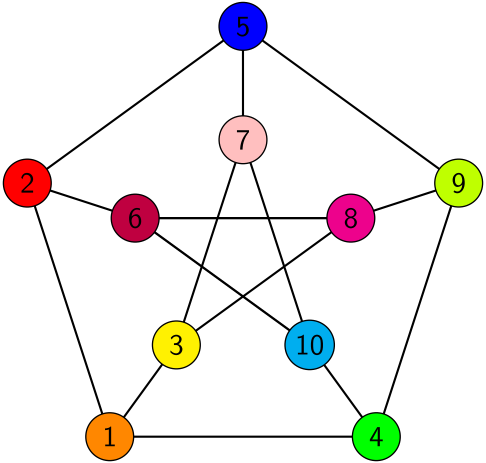

5
7
6
8
9
2
10
3
4
1

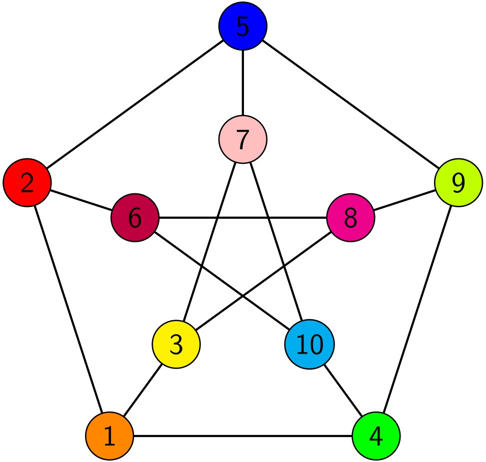

5
7
6
8
9
2
10
3
4
1


5
7
9
6
8
2
10
3
4
1


5
7
2
6
8
9
10
3
4
1


5
7
2
6
8
9
3
10
1
4


5
7
2
6
8
9
3
10
1
4


5
7
2
9
6
8
3
10
1
4

Introduction

1. Graph Isomorphism problem

Introduction

1. Graph Isomorphism problem
   - Why do we need it?

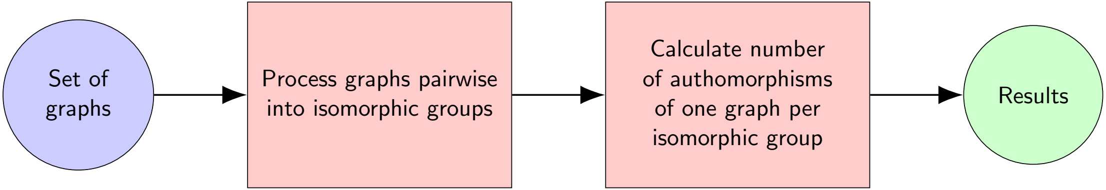
Introduction

1. Graph Isomorphism problem
   - Why do we need it?
   - Solvable in polynomial time?

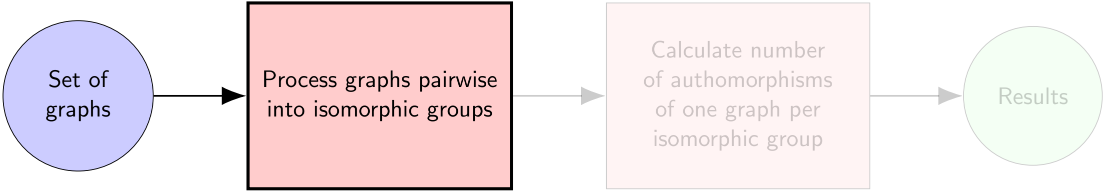
Introduction

1. Graph Isomorphism problem
   - Why do we need it?
   - Solvable in polynomial time?
2. Number of automorphisms problem

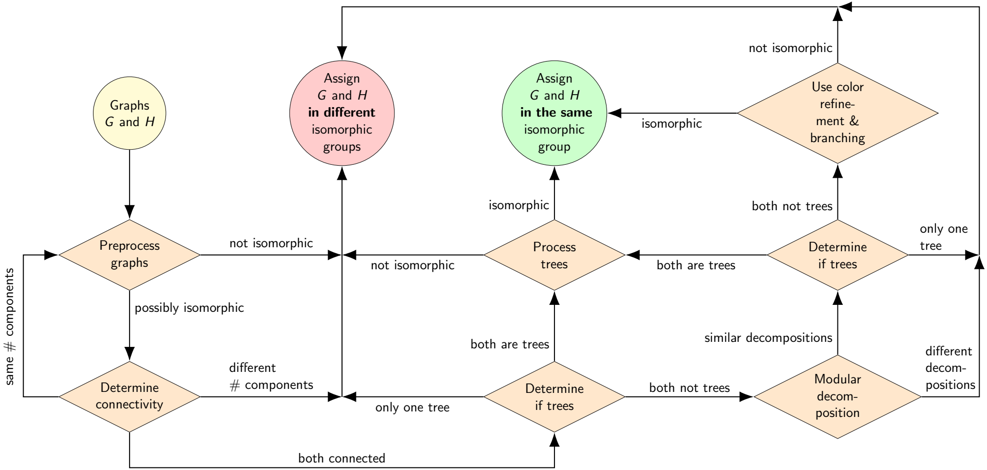
Processing pipeline overview



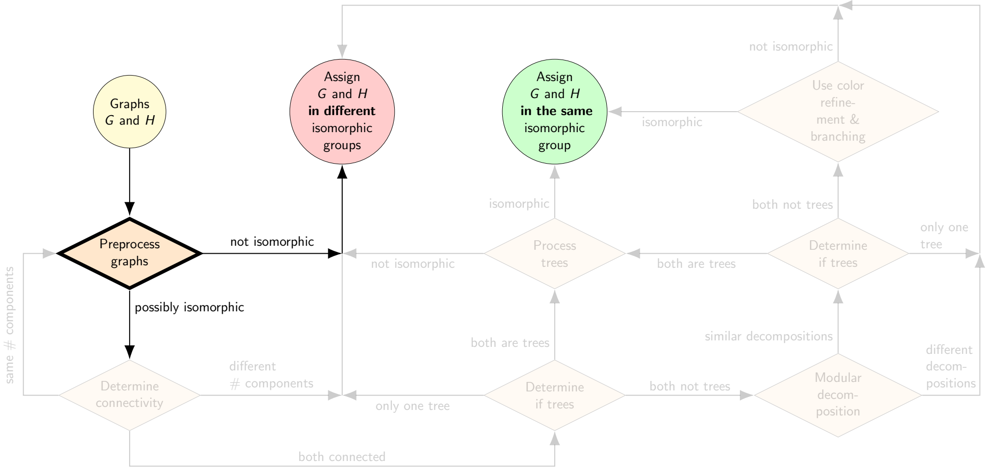
Processing pipeline overview



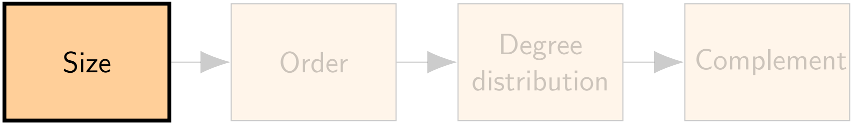
GI processing pipeline

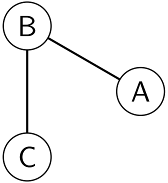


GI processing pipeline



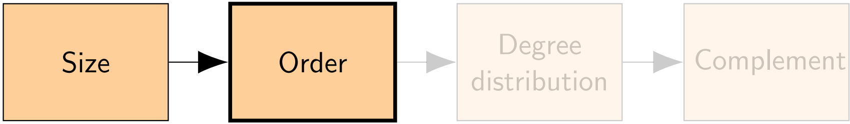
Preprocessing

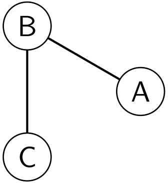




Size $= 2$

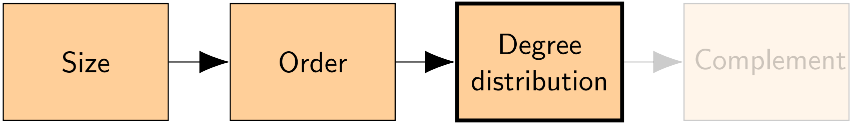
Preprocessing

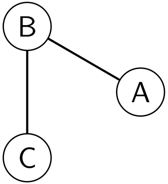




Order $= 3$

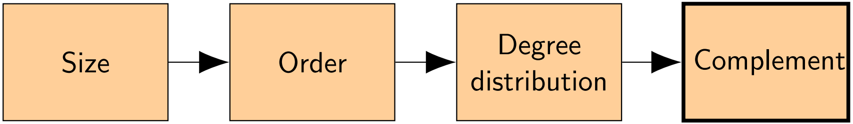
Preprocessing

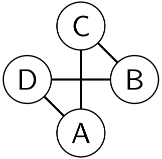




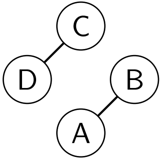
deg $=$ 1: (A,C), deg $=$ 2: (B)

Preprocessing



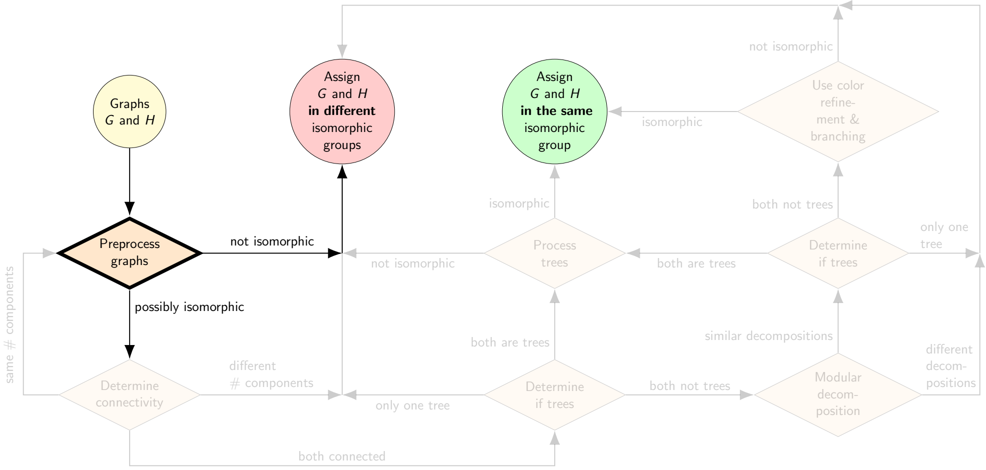


Original graph

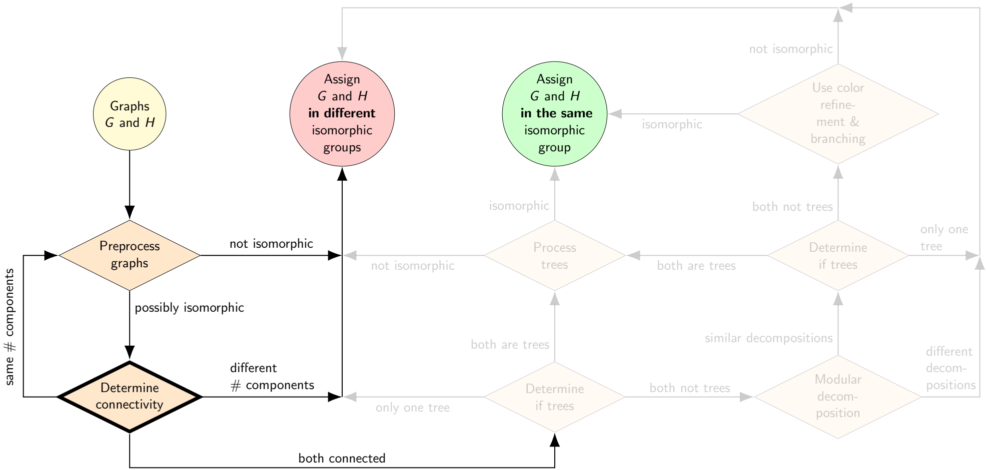


Complement

GI processing pipeline

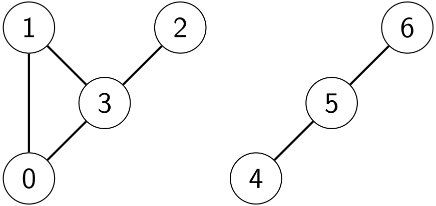


GI processing pipeline



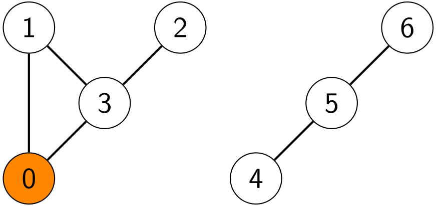
Connectivity

1. Keep track of visited vertices
2. Select any unvisited vertex
3. Start Breadth First Search from this vertex



Connectivity

1. Keep track of visited vertices
2. Select any unvisited vertex
3. Start Breadth First Search from this vertex

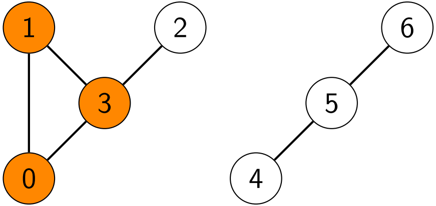


Visited = [0]

Number of components = 1

Connectivity

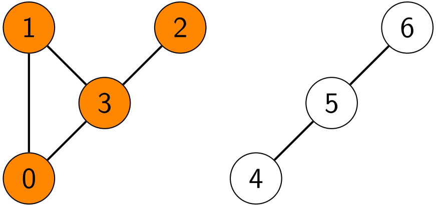
1. Keep track of visited vertices
2. Select any unvisited vertex
3. Start Breadth First Search from this vertex



Visited = [0, 1, 3]
Number of components = 1

Connectivity

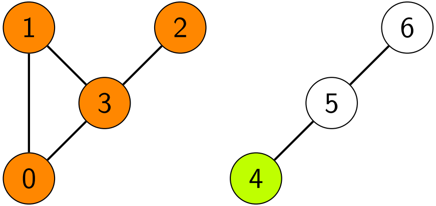
1. Keep track of visited vertices
2. Select any unvisited vertex
3. Start Breadth First Search from this vertex
4. If there are still unvisited vertices, there is another component



Visited = [0, 1, 3, 2]
Number of components = 1

Connectivity

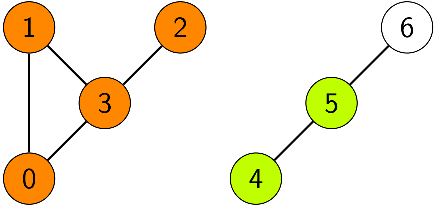
1. Keep track of visited vertices
2. Select any unvisited vertex
3. Start Breadth First Search from this vertex
4. If there are still unvisited vertices, there is another component



Visited = [0, 1, 3, 2, 4]
Number of components = 2

Connectivity

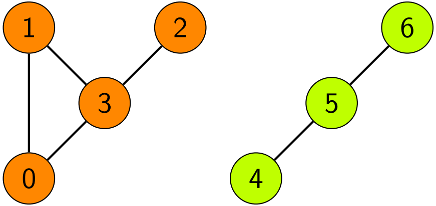
1. Keep track of visited vertices
2. Select any unvisited vertex
3. Start Breadth First Search from this vertex
4. If there are still unvisited vertices, there is another component



Visited = [0, 1, 3, 2, 4, 5]
Number of components = 2

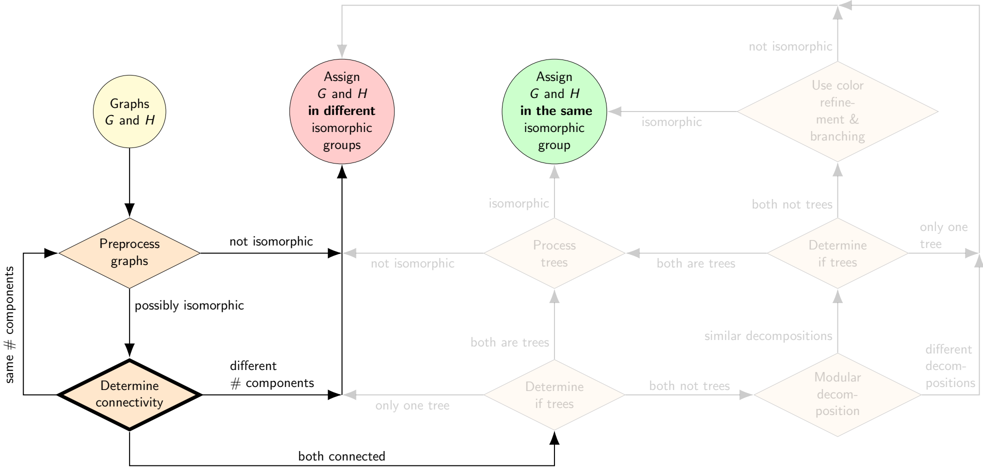
Connectivity

1. Keep track of visited vertices
2. Select any unvisited vertex
3. Start Breadth First Search from this vertex
4. If there are still unvisited vertices, there is another component

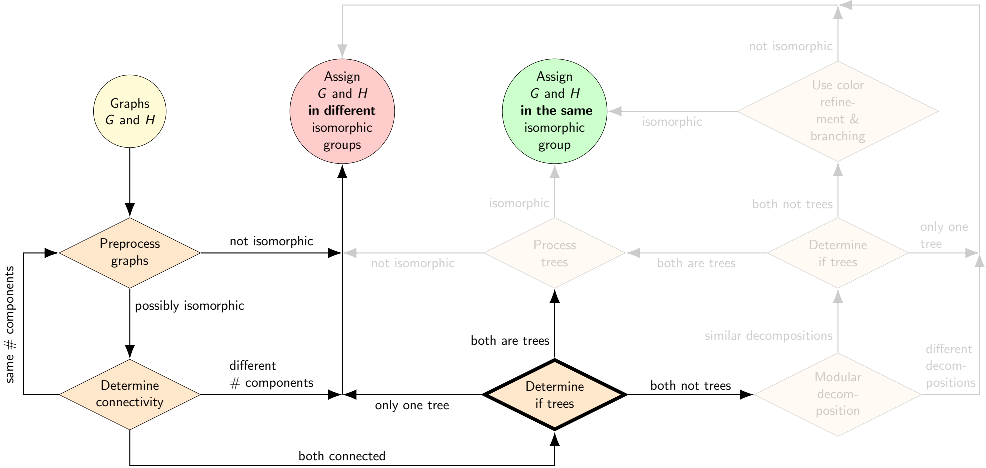


Visited = [0, 1, 3, 2, 4, 5, 6]
Number of components = 2

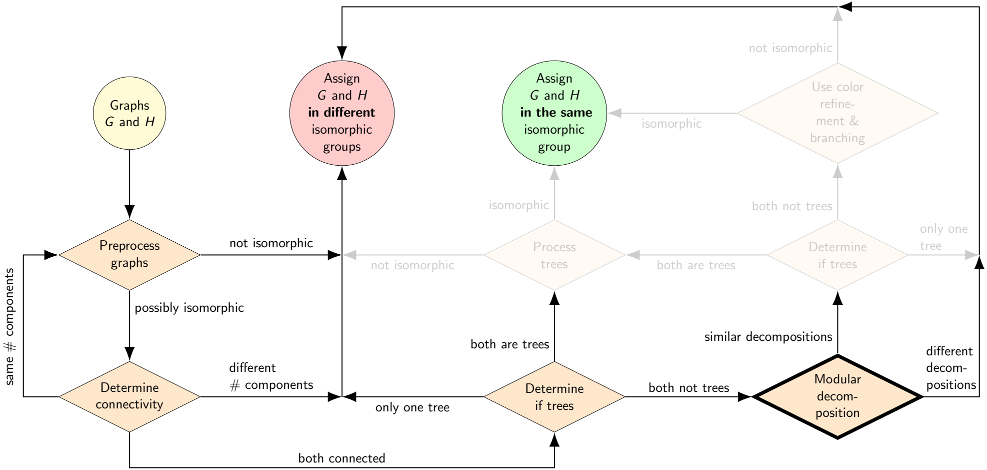
GI processing pipeline



GI processing pipeline



GI processing pipeline



Modular decomposition

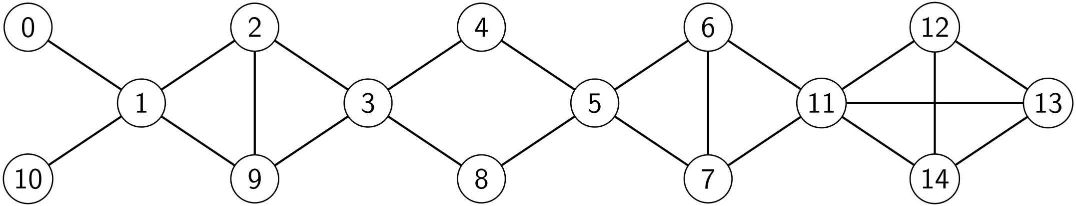
- Simplify graph by grouping similar vertices, i. e. modules, into one vertex

Modular decomposition

- Simplify graph by grouping similar vertices, i. e. modules, into one vertex
- Vertices are grouped by neighbourhood

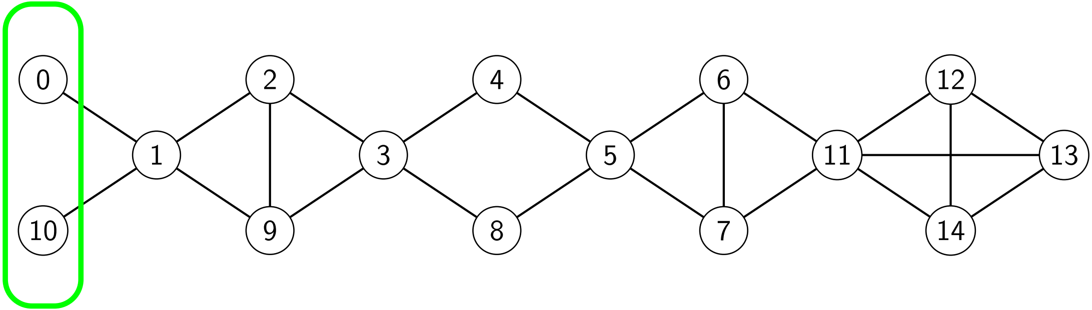
Modular decomposition

- Simplify graph by grouping similar vertices, i. e. modules, into one vertex
- Vertices are grouped by neighbourhood



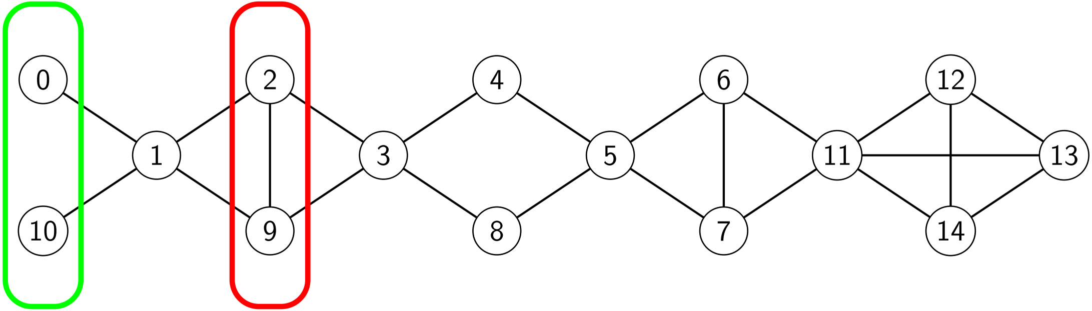
Modular decomposition

- Simplify graph by grouping similar vertices, i. e. modules, into one vertex
- Vertices are grouped by neighbourhood



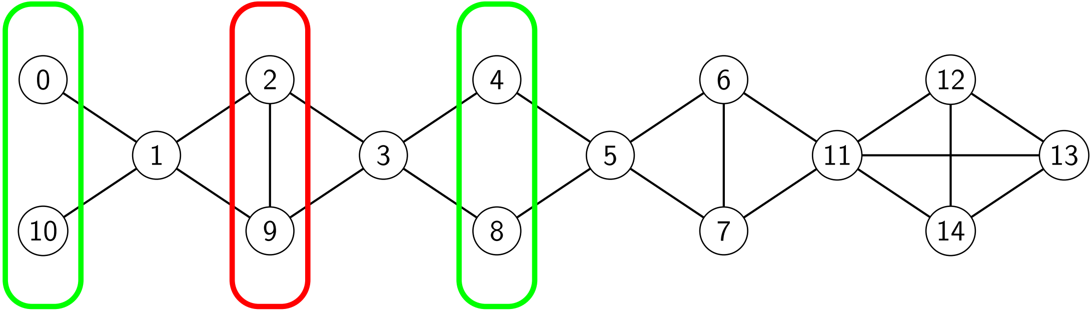
Modular decomposition

- Simplify graph by grouping similar vertices, i. e. modules, into one vertex
- Vertices are grouped by neighbourhood



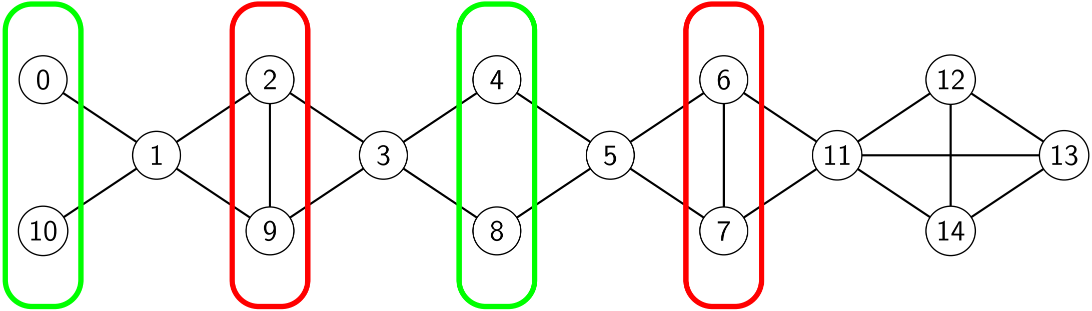
Modular decomposition

- Simplify graph by grouping similar vertices, i. e. modules, into one vertex
- Vertices are grouped by neighbourhood



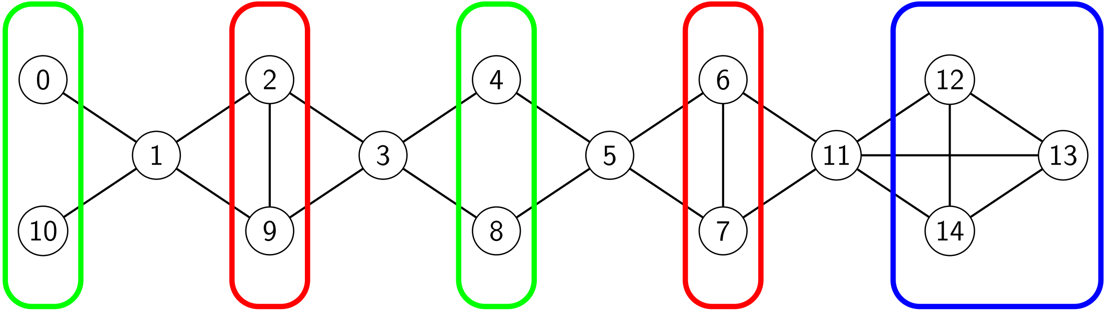
Modular decomposition

- Simplify graph by grouping similar vertices, i. e. modules, into one vertex
- Vertices are grouped by neighbourhood



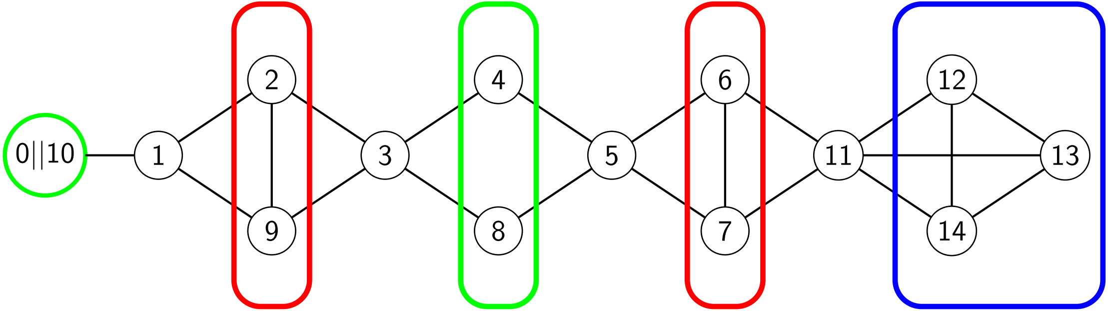
Modular decomposition

- Simplify graph by grouping similar vertices, i. e. modules, into one vertex
- Vertices are grouped by neighbourhood



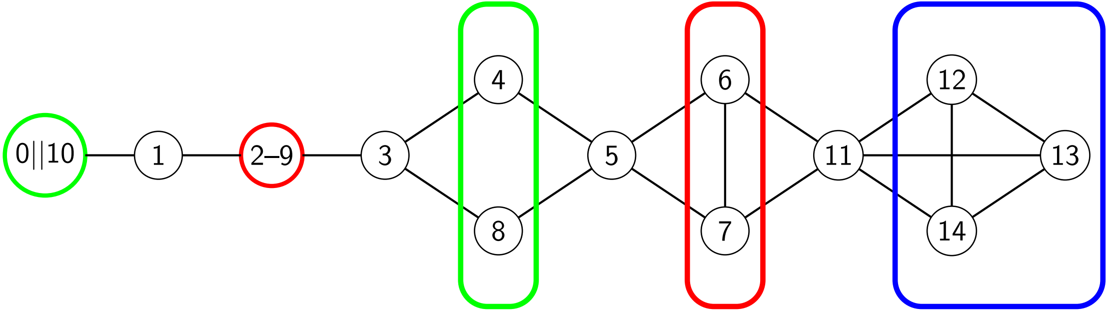
Modular decomposition

- Simplify graph by grouping similar vertices, i. e. modules, into one vertex
- Vertices are grouped by neighbourhood



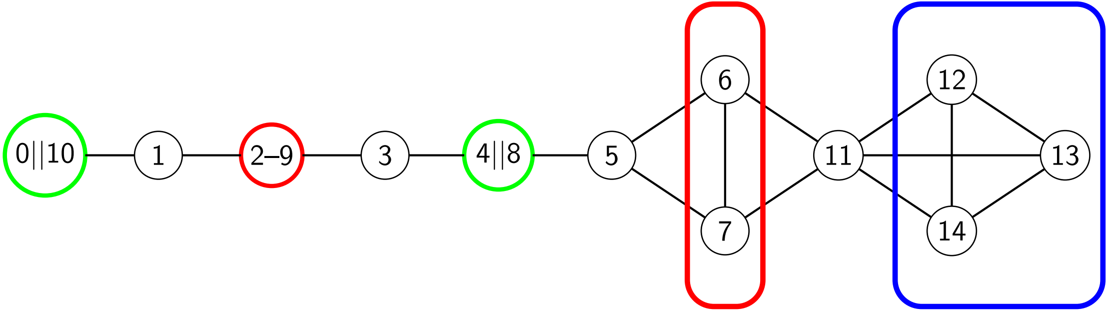
Modular decomposition

- Simplify graph by grouping similar vertices, i. e. modules, into one vertex
- Vertices are grouped by neighbourhood



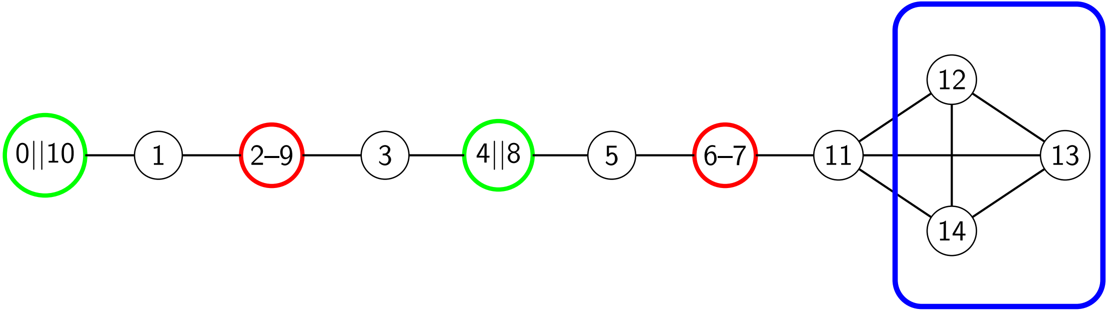
Modular decomposition

- Simplify graph by grouping similar vertices, i. e. modules, into one vertex
- Vertices are grouped by neighbourhood



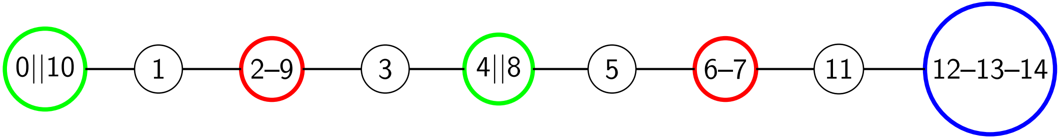
Modular decomposition

- Simplify graph by grouping similar vertices, i. e. modules, into one vertex
- Vertices are grouped by neighbourhood



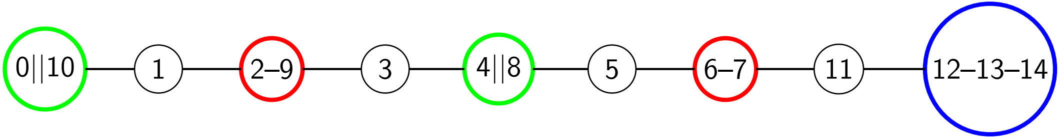
Modular decomposition

- Simplify graph by grouping similar vertices, i. e. modules, into one vertex
- Vertices are grouped by neighbourhood



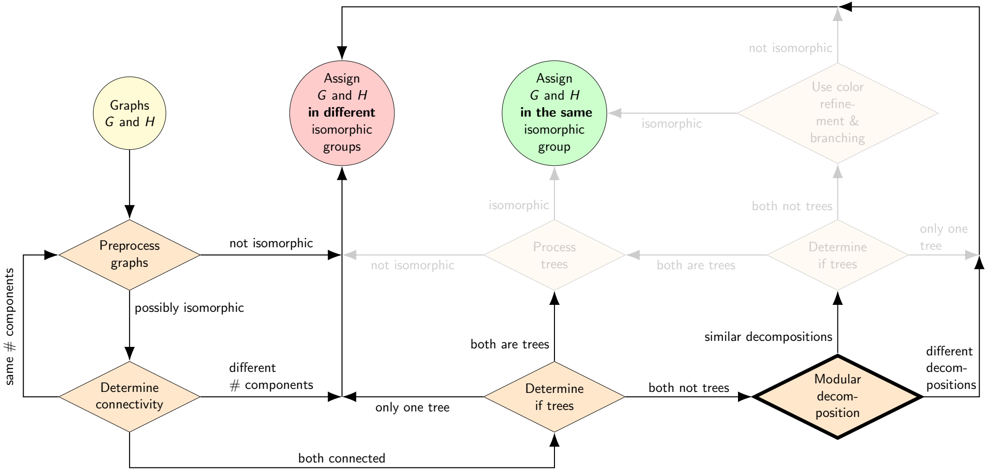
Modular decomposition

- Simplify graph by grouping similar vertices, i. e. modules, into one vertex
- Vertices are grouped by neighbourhood

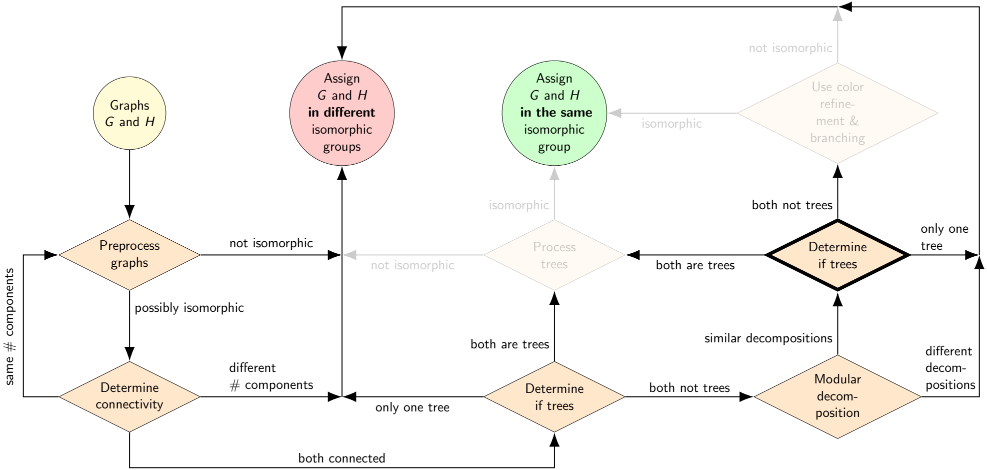


☞ This is a tree, albeit a linear one ☜

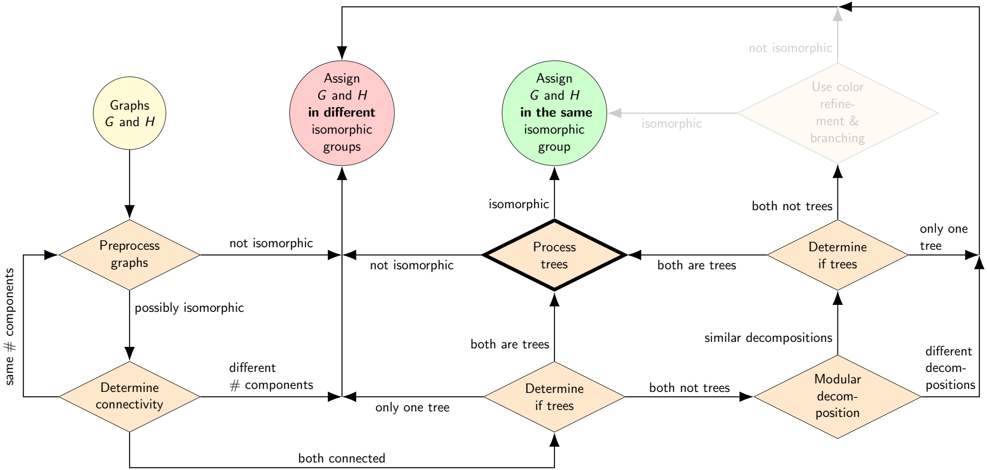
GI processing pipeline



GI processing pipeline



GI processing pipeline



Tree isomorphism

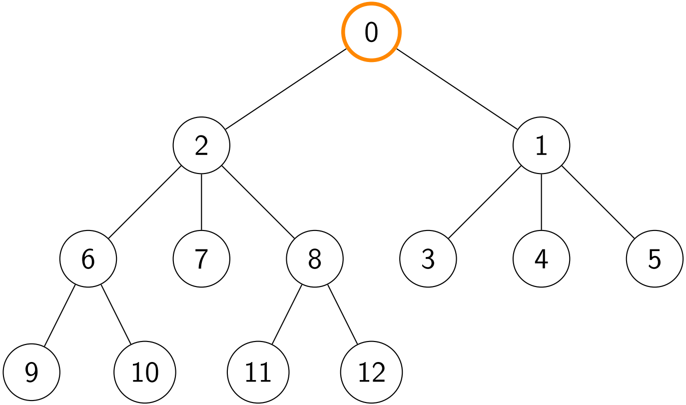
- Trees are non-cyclic graphs
- Solution for the GI problem in linear time
- Good combination with modular decomposition

From graph to rooted tree

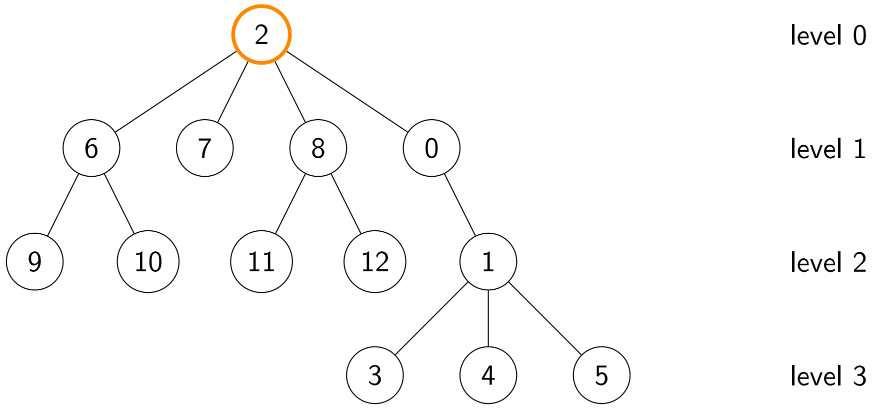
Algorithm by Bonany [2], returns the root that splits the tree in half

1. Choose an arbitrary root r
2. Assign each vertex the weight of its induced subtree
3. Start at r and shift it with the neighbour that has weight $> \frac{n}{2}$

Initial graph

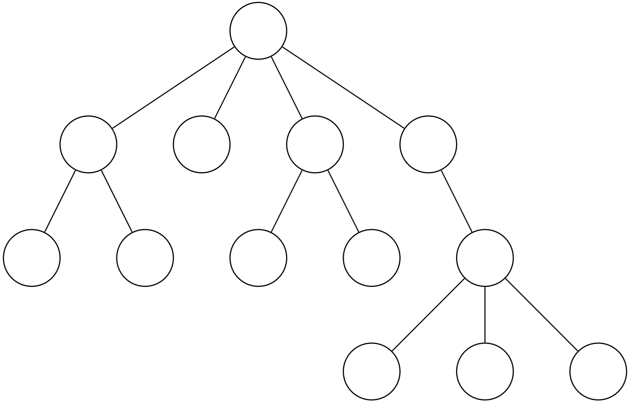


Rooted with assigned levels



Algorithm

In $\mathcal{O}(n)$, by Aho, Hopcroft and Ullman [1].



level 0

level 1

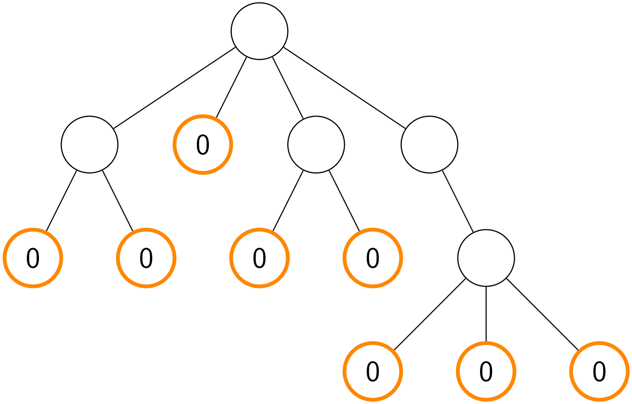
level 2

level 3

Algorithm

Assigned 0 to all leaves

Remark: The orange indicates the final value of that node



level 0

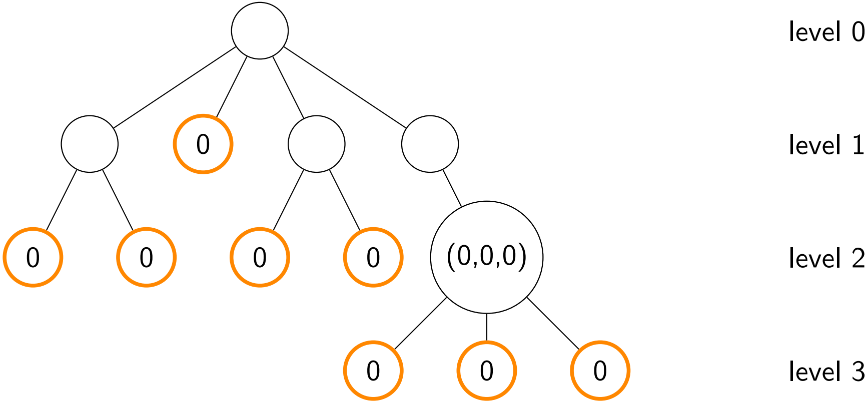
level 1

level 2

level 3

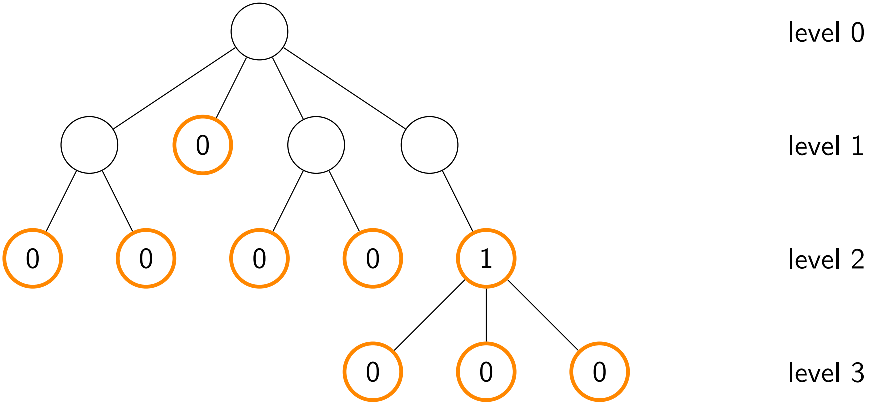
Algorithm

Assign tuples to the vertices at level 2



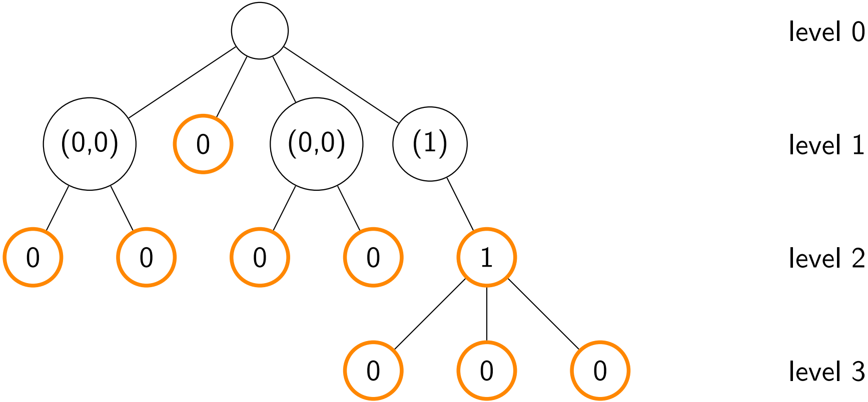
Algorithm

Assign values to the distinct tuples, starting at 1, at level 2



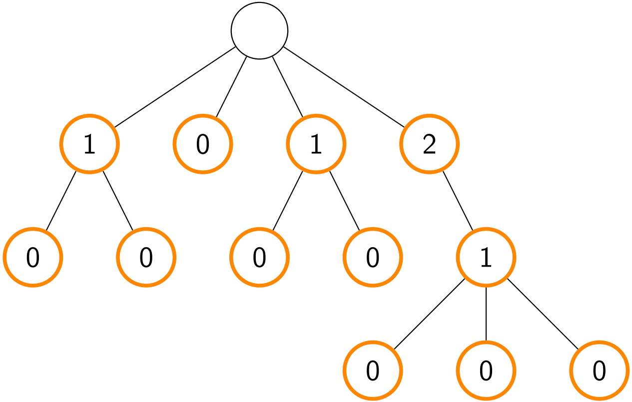
Algorithm

Assign tuples to the vertices at level 1



Algorithm

Assign values to the distinct tuples



level 0

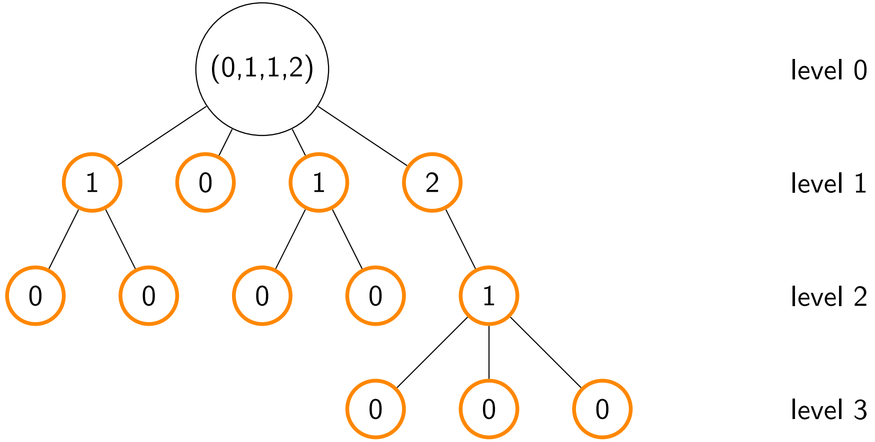
level 1

level 2

level 3

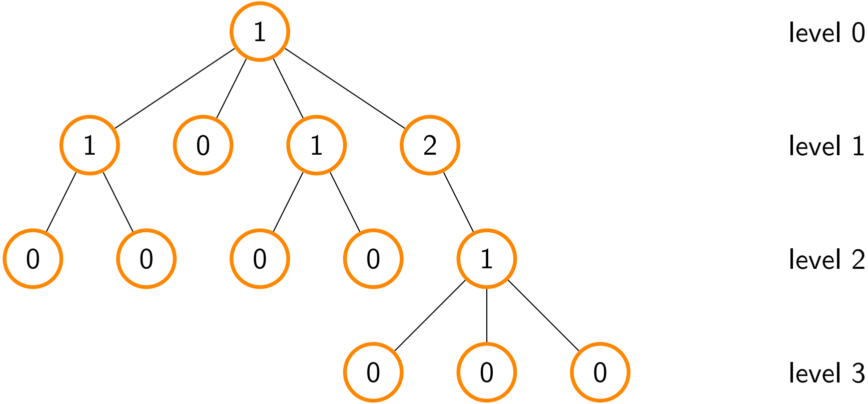
Algorithm

Assign tuples to the roots



Algorithm

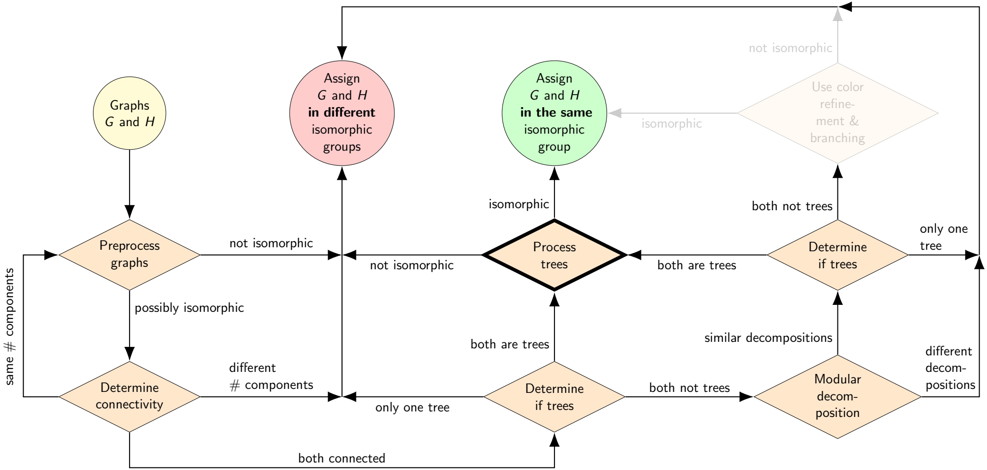
Assign values to the root



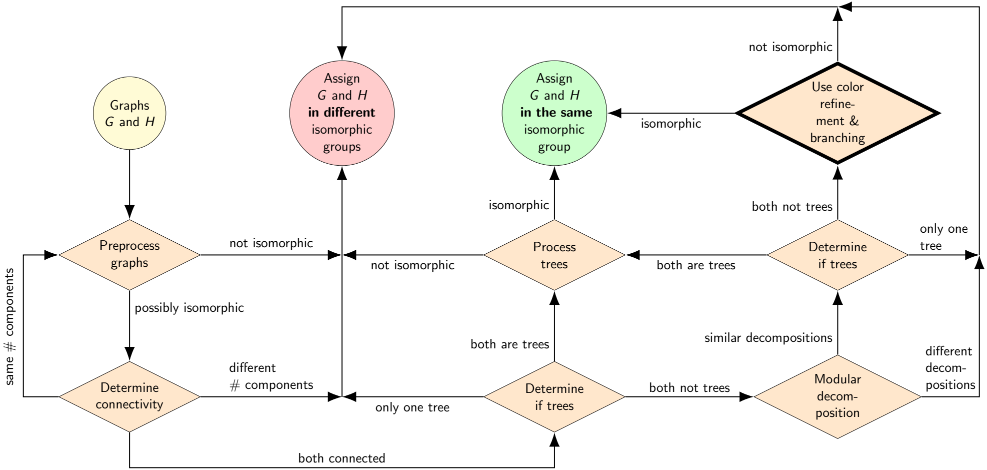
Modular decomposition and trees

- Assign the same value to isomorphic modules, starting at the length of the vertices.
- Check if the modules in both graphs have the tuples; otherwise their induced subtrees are different.

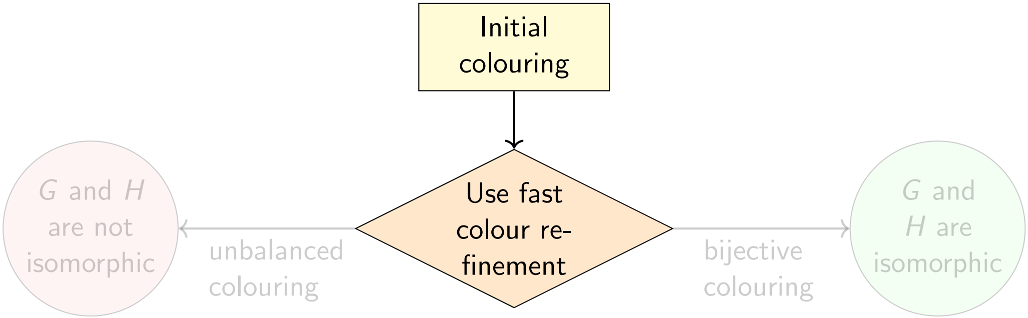
GI processing pipeline



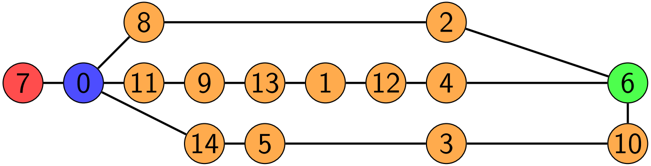
GI processing pipeline

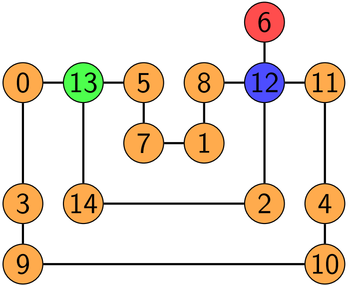


Fast colour refinement and branching

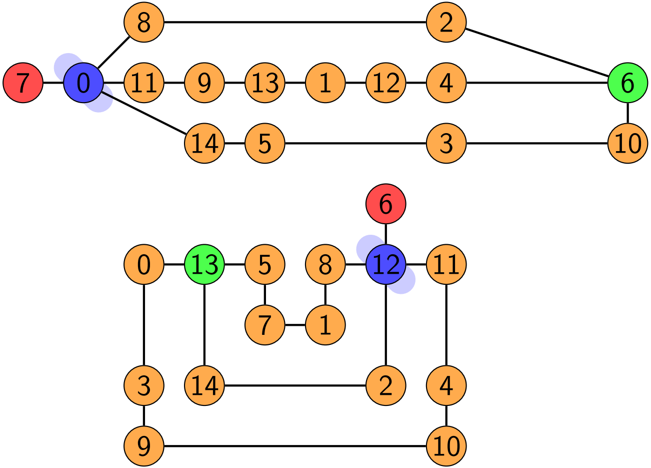


- Start with an initial colouring, eg. by degree
- Queue = [red, orange, green, blue]

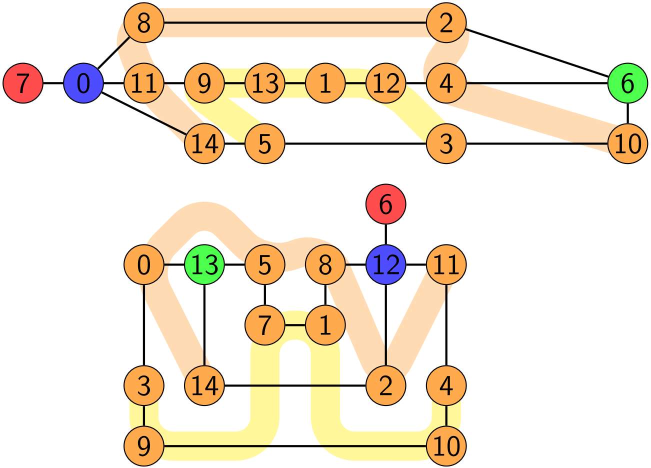




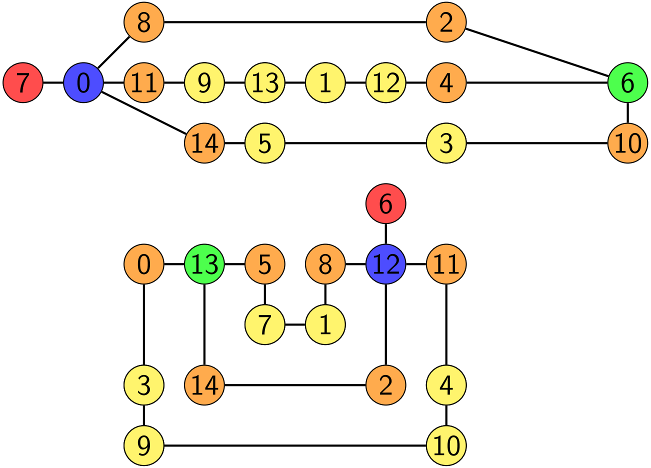
- Refine vertices on **amount of red neighbours**?
- Queue = [~~red~~, orange, green, blue]



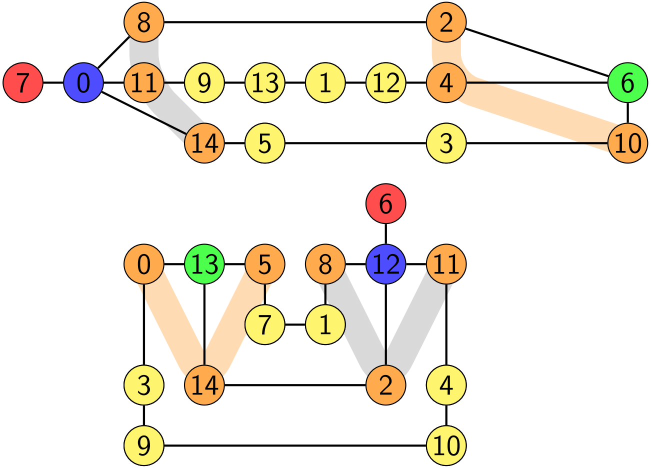
- Refine orange vertices on **amount of orange neighbours**
- Queue = [~~orange~~, green, blue]



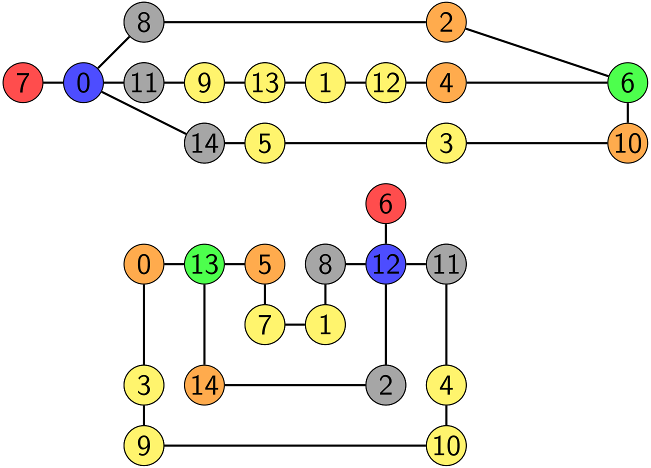
- Recolour orange vertices with 2 orange neighbours
- Queue = [green, blue] $\leftarrow$ orange



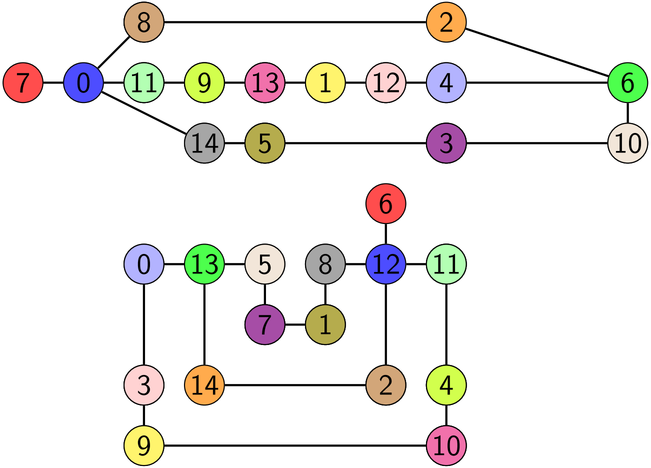
- Refine orange vertices on **amount of green neighbours**
- Queue = [~~green~~, blue, orange]



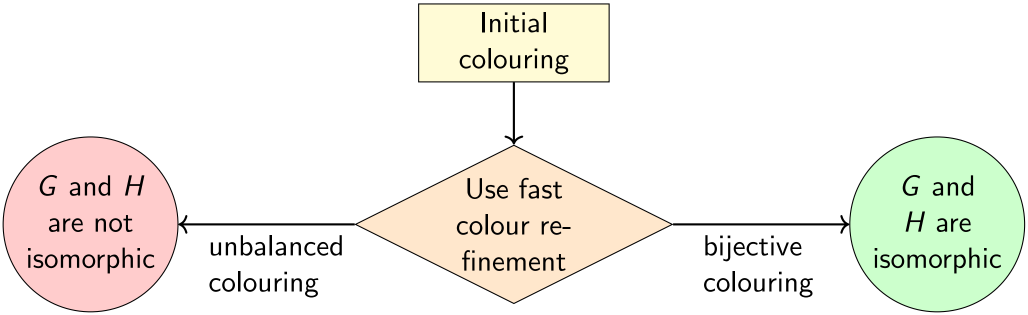
- Recolour orange vertices with no green neighbours
- Queue = [blue, orange] ← gray



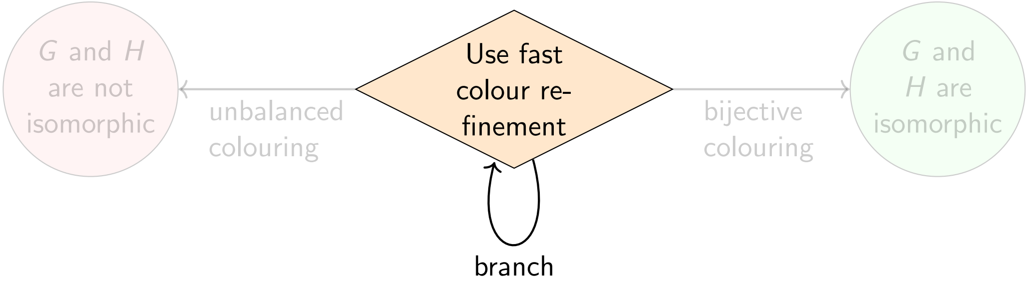
- Repeat these actions until the queue is empty
- Same colours → vertices map onto each other



Colour refinement flow



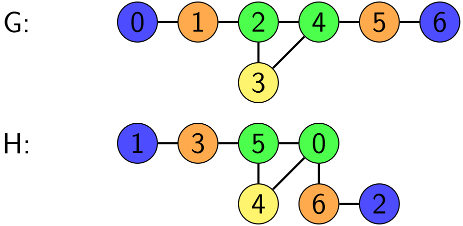
Colour refinement flow



Branching

Sometimes fast colour refinement results in a balanced non-bijective colouring.

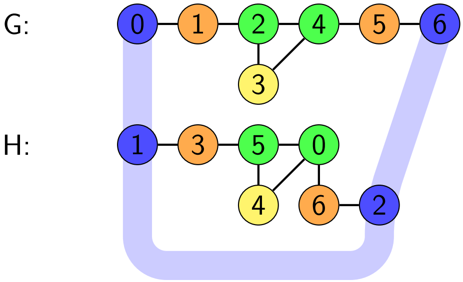
- There are multiple possible mappings



Branching

Sometimes fast colour refinement results in a balanced non-bijective colouring.

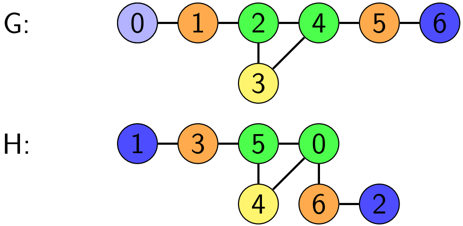
- There are multiple possible mappings
- Pick a colour class with at least 4 vertices



Branching

Sometimes fast colour refinement results in a balanced non-bijective colouring.

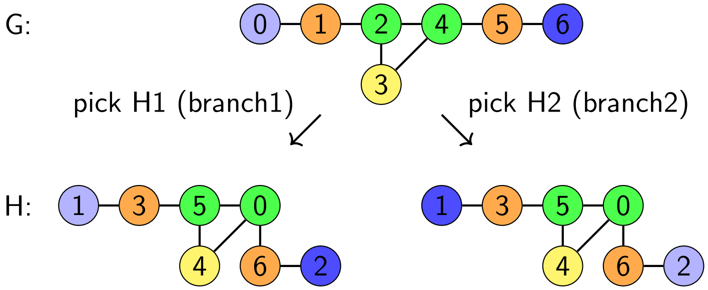
- There are multiple possible mappings
- Pick a colour class with at least 4 vertices
- Pick a vertex from G (eg. G0)



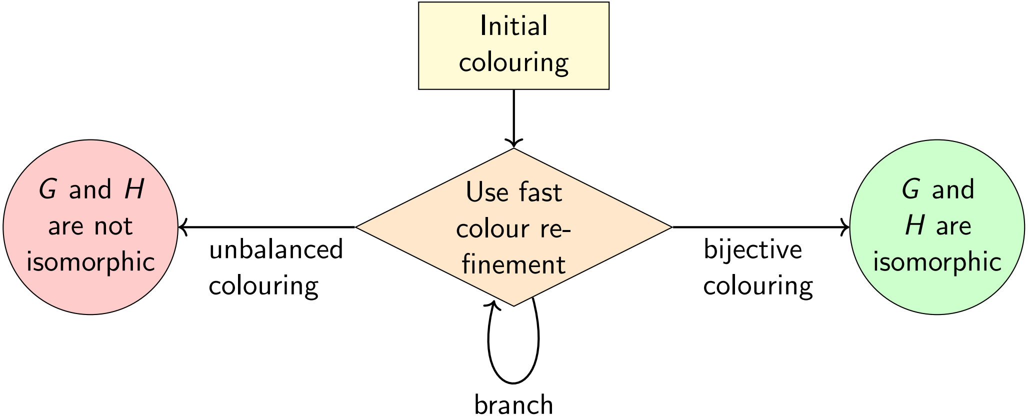
Branching

Sometimes fast colour refinement results in a balanced non-bijective colouring.

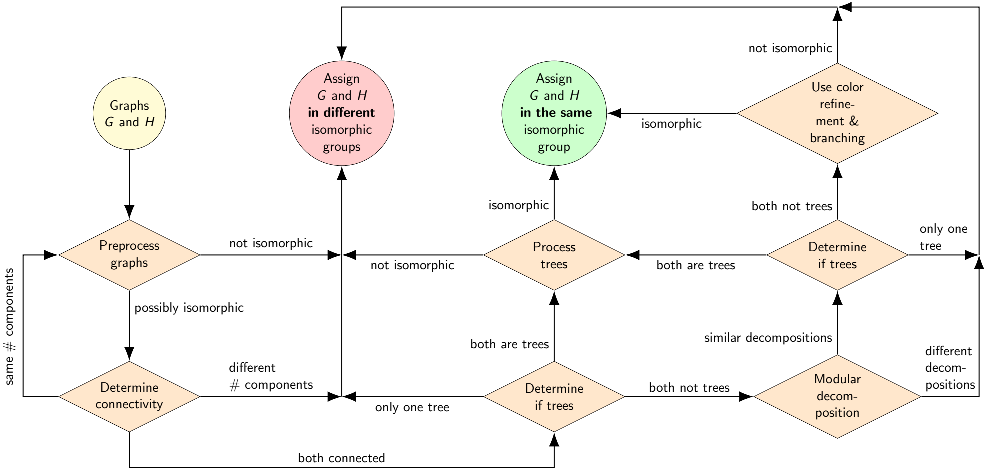
- There are multiple possible mappings
- Pick a colour class with at least 4 vertices
- Pick a vertex from G (eg. G0)
- Branch on vertices of H



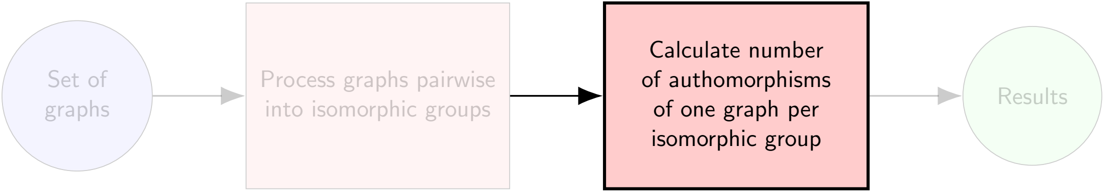
Colour refinement flow



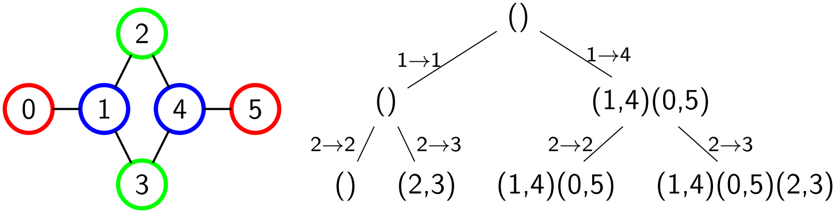
GI processing pipeline



Processing pipeline overview



Counting automorphisms

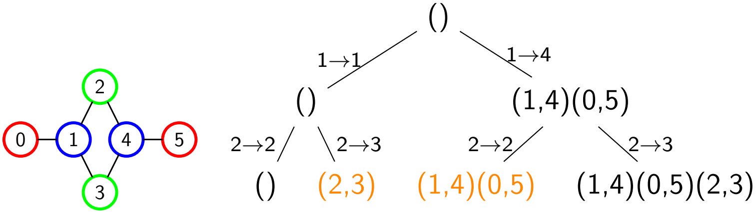


Automorphism groups

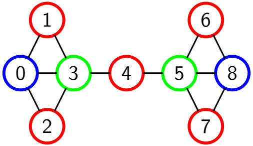
- number automorphisms is a group
- generating set: subgroup of a group which generates all elements of the group
- generating set can be small for large groups

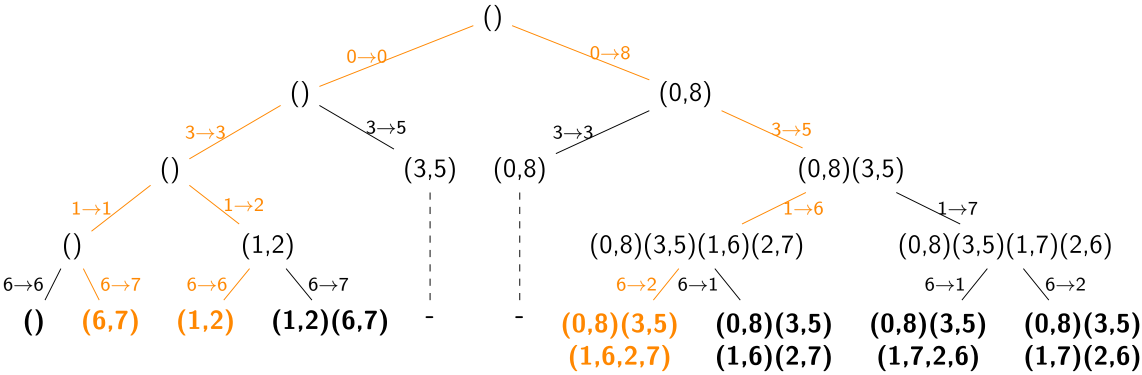
Automorphism groups

- number automorphisms is a group
- generating set: subgroup of a group which generates all elements of the group
- generating set can be small for large groups



Tree pruning



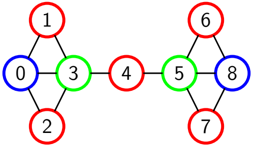


Generating set

[(6,7), (1,2), (0,8)(3,5)(1,6,2,7)]

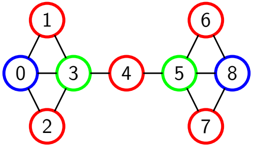
- order: number of automorphisms
- non-trivial mapping: mapping of a node to another node
- orbit: all possible mappings of a node
- stabilizer: permutations in the set where a node maps to itself

Compute number of automorphisms



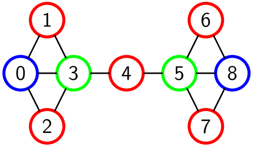
generating set	[(6,7),(1,2),(0,8)(1,6,2,7)(3,5)]
non-trivial mapping	1
orbit	(1,2,6,7)
stabilizer	[(6,7)]
order	length(orbit) $\times$ order of [(6,7)]

Compute number of automorphisms



generating set	[(6,7),(1,2),(0,8)(1,6,2,7)(3,5)]
non-trivial mapping	1
orbit	(1,2,6,7)
stabilizer	[(6,7)]
order	length(orbit) $\times$ order of [(6,7)]
	$=4\times$ order of [(6,7)]

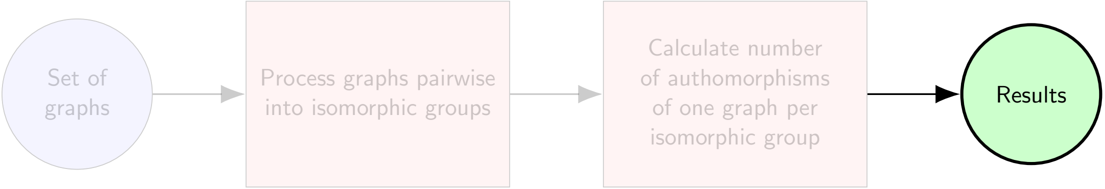
Compute number of automorphisms



generating set	[(6,7),(1,2),(0,8)(1,6,2,7)(3,5)]	[(6,7)]
non-trivial mapping	1	6
orbit	(1,2,6,7)	(6,7)
stabilizer	[(6,7)]	-
order	length(orbit) $\times$ order of [(6,7)]	length(orbit)
	$=$ 4 $\times$ order of [(6,7)]	$=$ 2

number of automorphisms $=$ 8

Processing pipeline overview



Results of the basic graphs

file name	number of graphs	graph order	graph size	modular decom-position factor	isomorphic graphs	number of automorphisms	time to solve (sec)
basicAut1.gr	1	35	34	576	–	2304	0.037
basicAut2.gr	1	69	96	1	–	128	1.554
basicGI1.grl	4	147	857	1	[0, 2] [1, 3]	–	10.678
basicGI2.grl	4	40	60	1	[0, 2] [1, 3]	–	6.179
basicGI3.grl	7	32	80	1	[0, 2, 6] [1, 3, 4, 5]	–	1.813
basicGIAut.grl	9	40	75	1	[0, 5, 7] [1, 6] [2, 3] [4, 8]	10 20 20 10	3.719

Results of the bonus graph isomorphism problems

file nr	number of graphs	graph order	graph size	modular decomposition factor	isomorphic graphs	time to solve (min:sec)
1	8	52	$\{762, 858\}$	67108864 4403012567040000 543581798400 4403012567040000	[0, 7] [1, 6] [2, 4] [3, 5]	0.9
2	12	64	204	86973087744 67108864 5706304286883840000 4403012567040000 704482010726400 5706304286883840000	[0, 3] [1, 4] [2 ,9] [5, 10] [6, 7] [8, 11]	0.4
3	4	627	6777	1	$[0,3]$, $[1,2]$	15:46.6
4	6	80	120	1	$[0,2,3,4]$, $[1,5]$	6:27.5
5	8	300	300	1	$[0,2]$, $[1,3]$, $[4,6]$, $[5,7]$	1.5

Results of the bonus graph isomorphism problems

file nr	number of graphs	graph order	graph size	modular decom-position factor	isomorphic graphs	time to solve (min:sec)
1	**8**	**52**	**{762, 858}**	**67108864** **4403012567040000** **543581798400** **4403012567040000**	**[0, 7]** **[1, 6]** **[2, 4]** **[3, 5]**	**0.9**
2	12	64	204	86973087744 67108864 5706304286883840000 4403012567040000 704482010726400 5706304286883840000	[0, 3] [1, 4] [2 ,9] [5, 10] [6, 7] [8, 11]	0.4
3	4	627	6777	1	[0, 3], [1, 2]	15:46.6
4	6	80	120	1	[0, 2, 3, 4], [1, 5]	6:27.5
5	8	300	300	1	[0, 2], [1, 3], [4, 6], [5, 7]	1.5

Results of the bonus graph isomorphism problems

file nr	number of graphs	graph order	graph size	modular decom-position factor	isomorphic graphs	time to solve (min:sec)
1	8	52	{762, 858}	67108864 4403012567040000 543581798400 4403012567040000	[0, 7] [1, 6] [2, 4] [3, 5]	0.9
2	**12**	**64**	**204**	**86973087744** **67108864** **5706304286883840000** **4403012567040000** **704482010726400** **5706304286883840000**	**[0, 3]** **[1, 4]** **[2 ,9]** **[5, 10]** **[6, 7]** **[8, 11]**	**0.4**
3	4	627	6777	1	[0, 3], [1, 2]	15:46.6
4	6	80	120	1	[0, 2, 3, 4], [1, 5]	6:27.5
5	8	300	300	1	[0, 2], [1, 3], [4, 6], [5, 7]	1.5

Results of the bonus automorphism problems

file nr	number of graphs	graph order	graph size	modular decom- position factor	number of automorphisms	time to solve (min:sec)
1	1	995	994	1	3538944	27:36.5
2	1	181	180	13589544960000	305267893261020841472415498240000	5:02.7
3	4	36	126	262144	5435817984 32614907904 32614907904 5435817984	2.4
4	3	240	480	1	12779520 17031168 16220160	32:50.6
5	3	86	165	16 4 6	9277129359360 1307993702400 2231764254720	2:44.3

Results of the bonus automorphism problems

file nr	number of graphs	graph order	graph size	modular decom-position factor	number of automorphisms	time to solve (min:sec)
1	**1**	**995**	**994**	**1**	**3538944**	**27:36.5**
2	**1**	**181**	**180**	**13589544960000**	**305267893261020841472415498240000**	**5:02.7**
3	4	36	126	262144	5435817984 32614907904 32614907904 5435817984	2.4
4	3	240	480	1	12779520 17031168 16220160	32:50.6
5	3	86	165	16 4 6	9277129359360 1307993702400 2231764254720	2:44.3

Results of the bonus automorphism problems

file nr	number of graphs	graph order	graph size	modular decom-position factor	number of automorphisms	time to solve (min:sec)
1	1	995	994	1	3538944	27:36.5
2	1	181	180	13589544960000	305267893261020841472415498240000	5:02.7
3	**4**	**36**	**126**	**262144**	**5435817984** **32614907904** **32614907904** **5435817984**	**2.4**
4	3	240	480	1	12779520 17031168 16220160	32:50.6
5	3	86	165	16 4 6	9277129359360 1307993702400 2231764254720	2:44.3

Results of the bonus automorphism problems

file nr	number of graphs	graph order	graph size	modular decom-position factor	number of automorphisms	time to solve (min:sec)
1	1	995	994	1	3538944	27:36.5
2	1	181	180	13589544960000	30526789326102084147241549824 0000	5:02.7
3	4	36	126	262144	5435817984 32614907904 32614907904 5435817984	2.4
4	**3**	**240**	**480**	**1**	**12779520** **17031168** **16220160**	**32:50.6**
5	3	86	165	16 4 6	9277129359360 1307993702400 2231764254720	2:44.3

Results of the bonus automorphism problems

file nr	number of graphs	graph order	graph size	modular decom-position factor	number of automorphisms	time to solve (min:sec)
1	1	995	994	1	3538944	27:36.5
2	1	181	180	13589544960000	30526789326102084147241549824000	5:02.7
3	4	36	126	262144	5435817984 32614907904 32614907904 5435817984	2.4
4	3	240	480	1	12779520 17031168 16220160	32:50.6
5	3	86	165	16 4 6	9277129359360 1307993702400 2231764254720	2:44.3
-	1	298	297	210357201231685877760000	95872928658458861 80... 95416391147200402 95... 528850470522060800 00	53:42.5

Conclusion

- All basic instances were successfully solved
- GI-problem could efficiently be solved for:
  - graphs were the complement is smaller
  - trees
  - graphs with modules
  - disconnected graphs
- Number of automorphisms had only a smart algorithm for graphs with modules
- Bug in the detection of isomorphic graphs with disconnected components

Possible improvements

- Optimisation for fast colour refinement after branching
- Use smart branching
- Use efficient algorithms for different types of graphs for the number of automorphisms

Reflection

- Development of different good ideas in parallel does not guarantee good results
- However, the team effort ultimately resulted in a successful integration

Bibliography

A. V. Aho, J. E. Hopcroft, and J. D. Ullman.
The Design and Analysis of Computer Algorithms.
Addison-Wesley, 1974.

Marthe Bonamy.
A small report on graph and tree isomorphism.
2010.

Results of the basic graphs - improved

file name	number of graphs	graph order	graph size	MD factor	after MD order	after MD size	isomorphic graphs	number of automorphisms	time to solve (sec)	time to solve improved (sec)
basicAut1.gr	1	35	34	576	27	26	–	2304	0.037	0.016
basicAut2.gr	1	69	96	1	-	-	–	128	1.554	0.146
basicGI1.grl	4	147	857	1	- -	- -	[0, 2] [1, 3]	–	10.678	1.843
basicGI2.grl	4	40	60	1	- -	- -	[0, 2] [1, 3]	–	6.179	3.550
basicGI3.grl	7	32	80	1	- -	- -	[0, 2, 6] [1, 3, 4, 5]	–	1.813	1.700
basicGIAut.grl	9	40	75	1	- - - -	- - - -	[0, 5, 7] [1, 6] [2, 3] [4, 8]	10 20 20 10	3.719	2.573

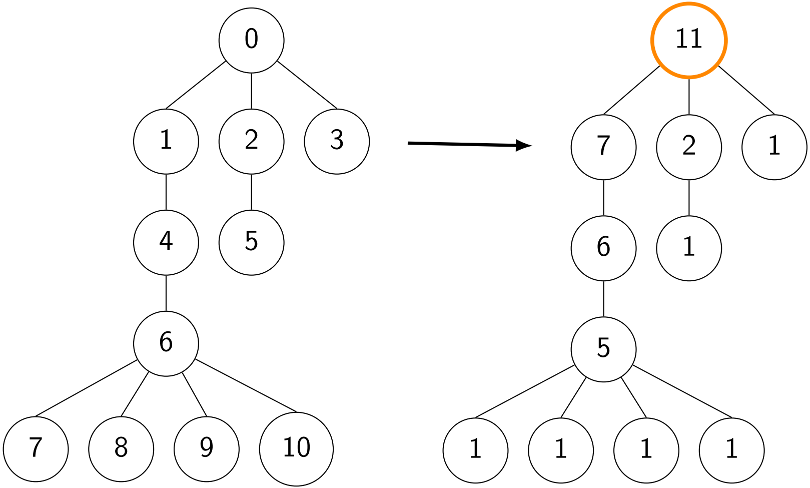
Results of the bonus graph isomorphism problems - improved

file nr	number of graphs	graph order	graph size	MD factor	after MD order	after MD size	isomorphic graphs	time to solve (min:sec)	time to solve improved (min:sec)
1	8	52	$\{762, 858\}$	67108864	26	18	[0, 7]	0.928	0.611
				4403012567040000	18	50^*	[1, 6]		
				543581798400	22	143	[2, 4]		
				4403012567040000	18	103	[3, 5]		
2	12	64	204	86973087744	30	45	[0, 3]	0.438	0.305
				67108864	38	61	[1, 4]		
				5706304286883840000	22	25	[2, 9]		
				4403012567040000	30	41	[5, 10]		
				704482010726400	26	33	[6, 7]		
				5706304286883840000	22	25	[8, 11]		
3	4	627	6777	1	-	-	[0, 3], [1, 2]	15:46.585	58.621
4	6	80	120	1	-	-	[0, 2, 3, 4], [1, 5]	6:27.488	3:01:330
5	8	300	300	1	-	-	[0, 2], [1, 3], [4, 6], [5, 7]	1.549	1.559

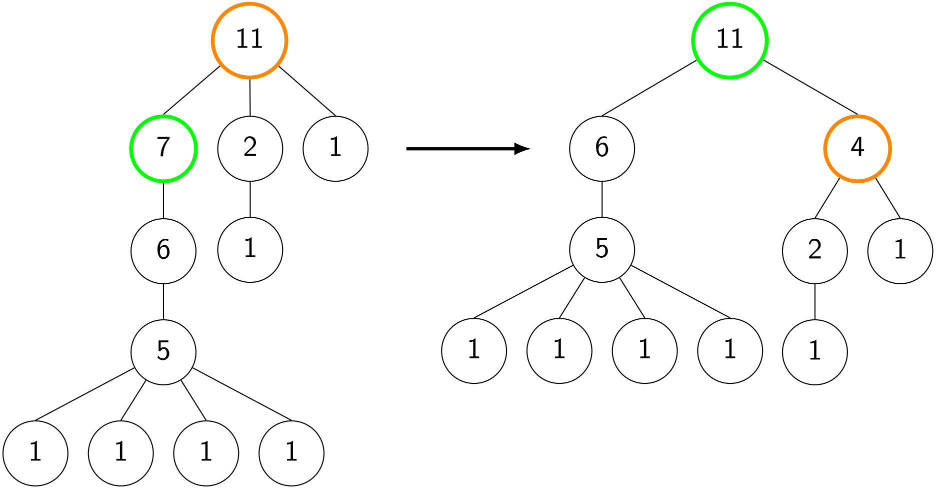
Results of the bonus automorphism problems - improved

file nr	number of graphs	graph order	graph size	MD factor	after MD order	after MD size	number of automorphisms	time to solve (min:sec)	time to solve improved (min:sec)
1	1	995	994	1	-	-	3538944	27:36.511	?
2	1	181	180	13589544960000	149	148	305267893261020841472415498240000	5:02.722	32.111
3	4	36	126	262144	18	27	5435817984 32614907904 32614907904 5435817984	2.421	1.356
4	3	240	480	1	-	-	12779520 17031168 16220160	32:50.569	10:42.775
5	3	86	165	16 4 6	82 84 84	155 160 160	9277129359360 1307993702400 2231764254720	2:44.286	36.878
-	1	298	297	210357201231685877760000	244	243	9587292865845886180... 9541639114720040295... 52885047052206080000	53:42.532	6:29.500

Choose an arbitrary root and assign weights



Shift the weight



Shift the weight, blue is the final root