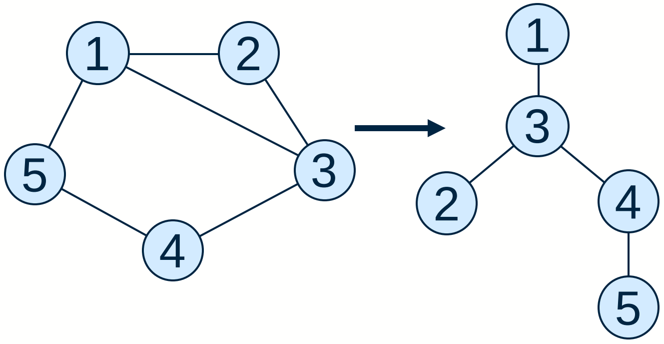
University of Glasgow

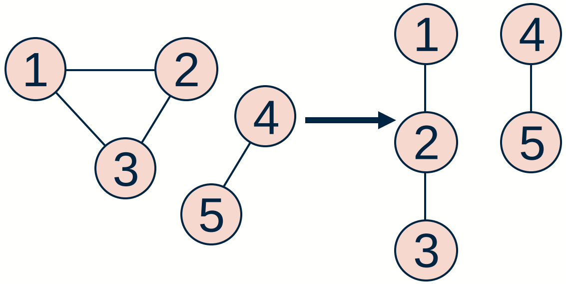
An Algorithm for the Exact Treedepth Problem

James Trimble

The Treedepth Problem

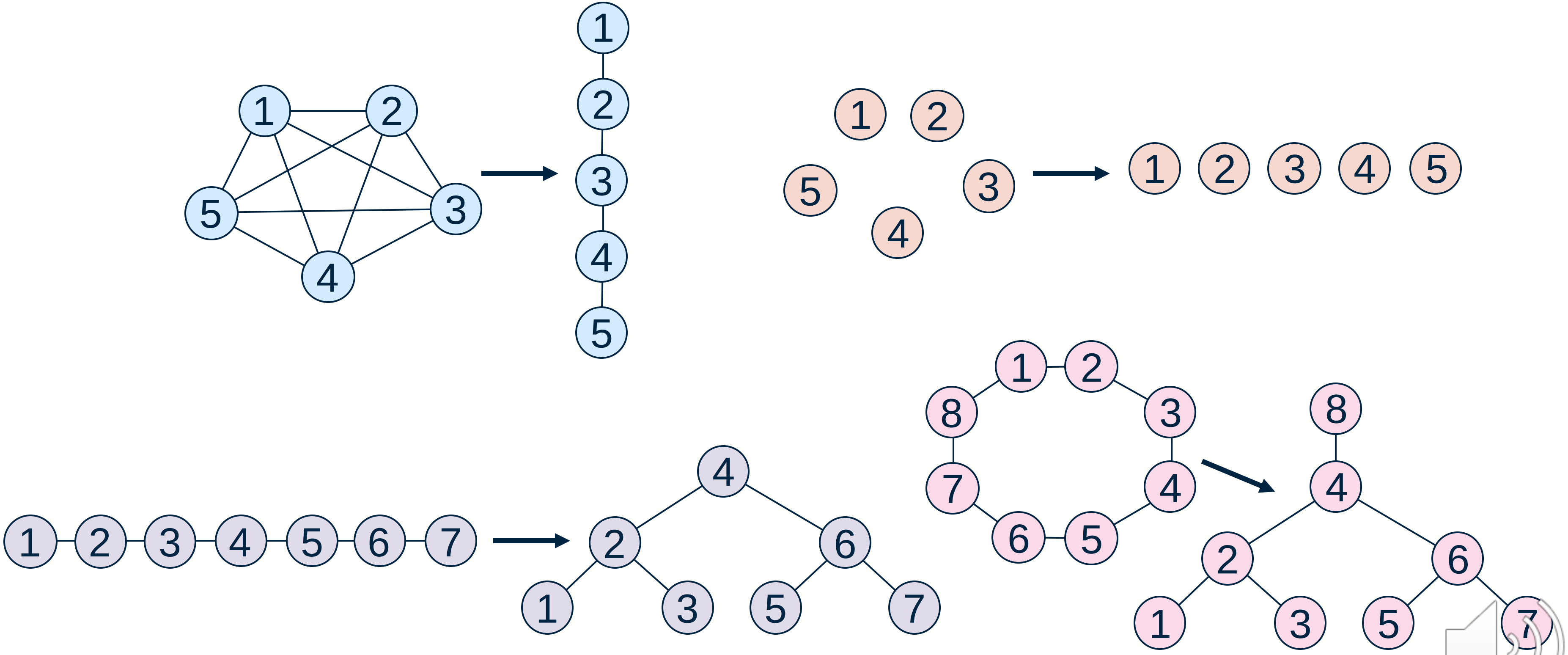
A ***treedepth decomposition*** of graph *G* is a rooted forest with vertex set *V(G)*, such that for each edge *(v,w)* in *G*, *v* is an ancestor of *w* or vice versa.





The ***treedepth*** of *G* is the smallest depth of any treedepth decomposition of *G*.

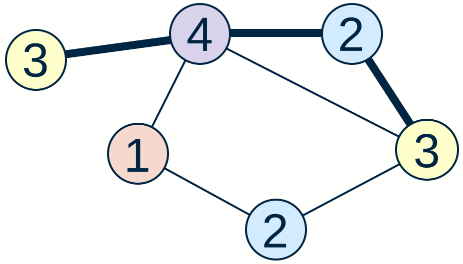
The Treedepth Problem: examples



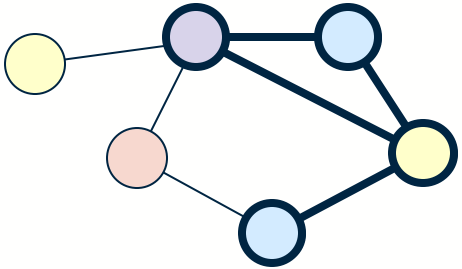
The Treedepth Problem

The treedepth of a connected graph G equals many things, including:

The vertex ranking number of G



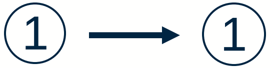
The centred chromatic number of G



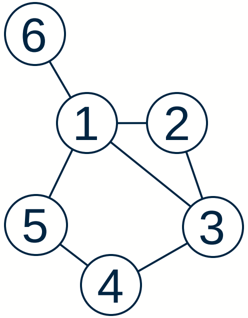
The minimum height of an elimination tree for G

Elimination trees

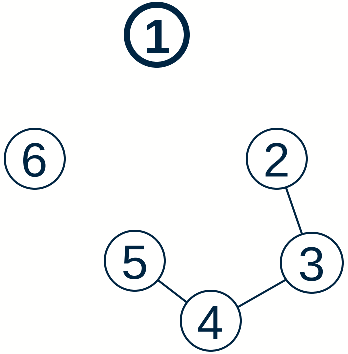
The elimination tree of a single-vertex graph is a single-vertex tree



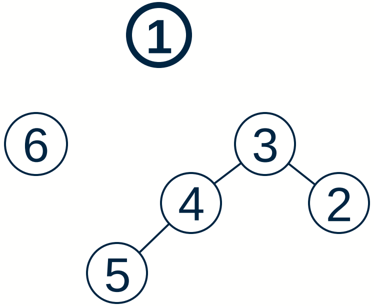
To form an elimination tree of a larger connected graph:



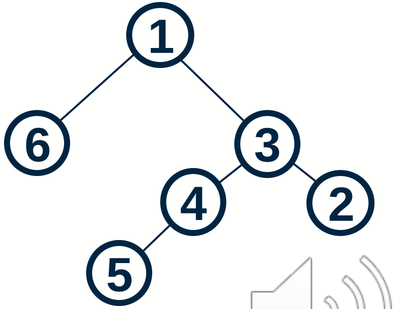
1. Remove a vertex; make this the root



2. Form an elimination tree of each remaining component



3. Make the root of each of these trees a child of the root



Elimination trees (2)

An *Elimination forest* of *G* is the union of elimination trees of the components of *G*.

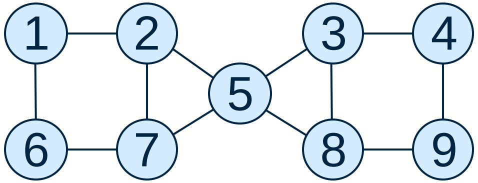
- Every elimination forest is a treedepth decomposition
- Every graph *G* has an elimination forest whose depth equals the treedepth of *G*
- So to solve the treedepth problem (and find a witness), it suffices to find an elimination tree of minimum depth
- (Nesetril and de Mendez (2012). *Sparsity - Graphs, Structures, and Algorithms*)

The algorithm

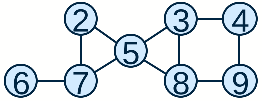
```
elimination_forest(general graph G, int k):
  Result: true if and only if an elimination forest of G with depth no greater than k exists
  if k = 0 and |V(G)| > 0: return false
  for each connected component C of G:
    if not elimination_tree(C,k): return false
  return true

elimination_tree(connected graph G, int k):
  Require: k >= 1
  Result: true if and only if an elimination tree of G with depth no greater than k exists
  if |V(G)| = 1: return true
  for each v in V(G):
    if elimination_forest(G-v, k-1): return true
  return false
```

The algorithm

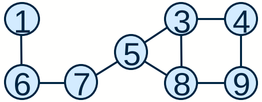
```
elimination_forest(general graph G, int k):
  Result: true iff an EF of G with depth <= k exists
  if k = 0 and |V(G)| > 0: return false
  for each connected component C of G:
    if not elimination_tree(C,k): return false
  return true

elimination_tree(connected graph G, int k):
  Require: k >= 1
  Result: true iff an ET of G with depth <= k exists
  if |V(G)| = 1: return true
  for each v in V(G):
    if elimination_forest(G-v, k-1): return true
  return false
```



Try removing v. 1

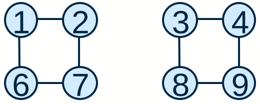


Try removing v. 2



…

Try removing v. 5

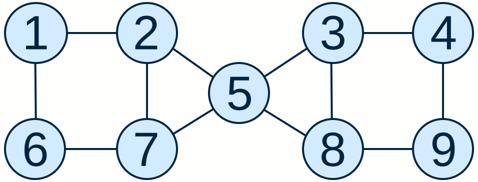


Symmetry breaking and domination rules

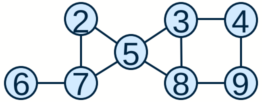
```
elimination_forest(general graph G, int k):
  Result: true iff an EF of G with depth <= k exists
  if k = 0 and |V(G)| > 0: return false
  for each connected component C of G:
    if not elimination_tree(C,k): return false
  return true

elimination_tree(connected graph G, int k):
  Require: k >= 1
  Result: true iff an ET of G with depth <= k exists
  if |V(G)| = 1: return true
  for each v in V(G):
    if elimination_forest(G-v, k-1): return true
  return false
```

General principle:
There's no need to try a vertex v if we can show that there's a lower-numbered vertex v' that produces an elimination tree that is just as good.

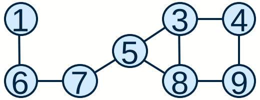
Symmetry breaking



Try removing v. 1

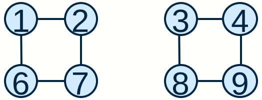


Try removing v. 2

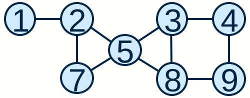


…

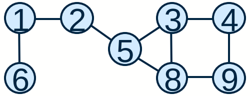
Try removing v. 5



No need to try v .6…



… or v. 7



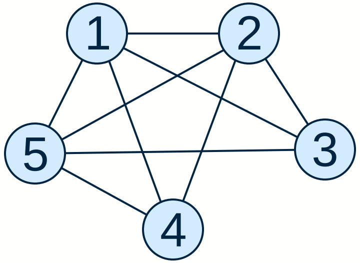
Domination

If G is a subgraph of H, then treedepth(G) <= treedepth(H)

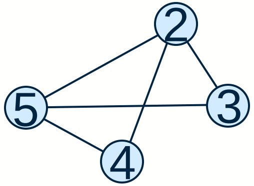
Idea:
If G-v’ is isomorphic to a subgraph of G-v, then we don’t need to try using v as the root of an elimination tree.

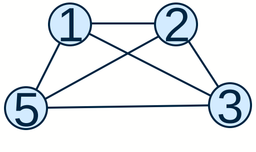
Domination rule:
If v’ < v and N(v’)\{v} is a superset of N(v)\{v’}, there’s no need to try v as root.

This was used for preprocessing by Ganian, Lodha, Ordyniak, Szeider (2019)

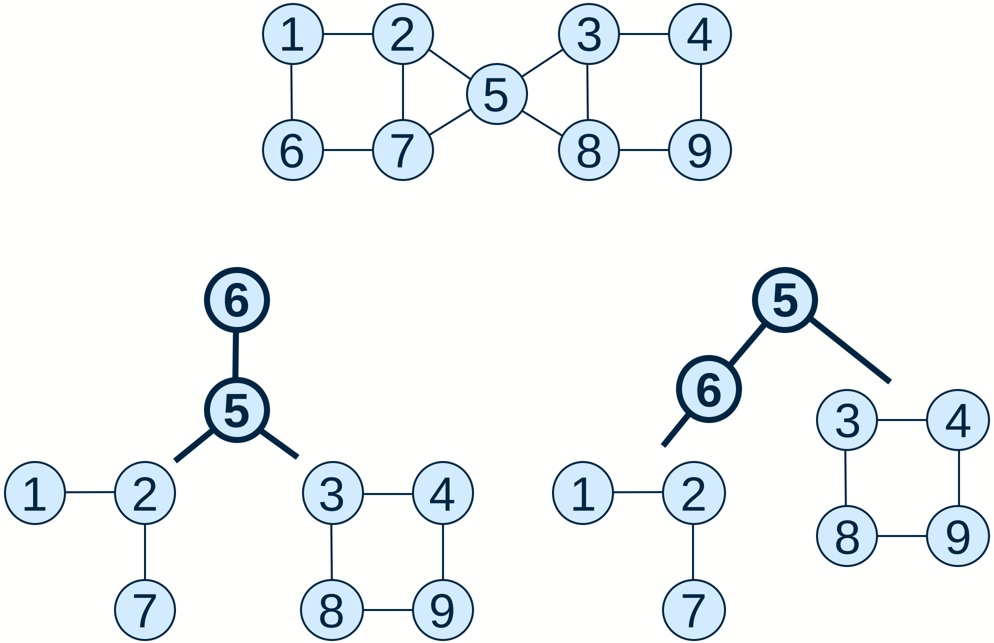


N(1) \ {4} = {2,3,5}
N(4) \ {1} = {2,5}





Only-child rule

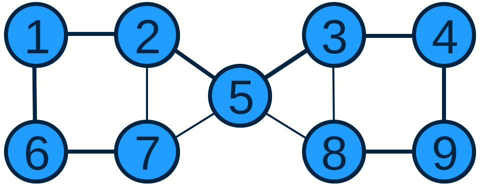


Only child rule:
There’s no need to try v as the root of a subtree if v is an only child in the tree, and has a lower number than its parent.

Path lower bound

A graph containing a k-vertex path has treedepth at least $\log_2(k+1)$.

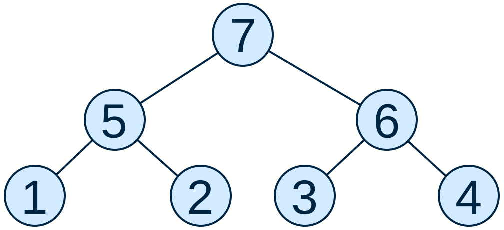
We greedily find a path in G, and use this for a lower bound on treedepth.



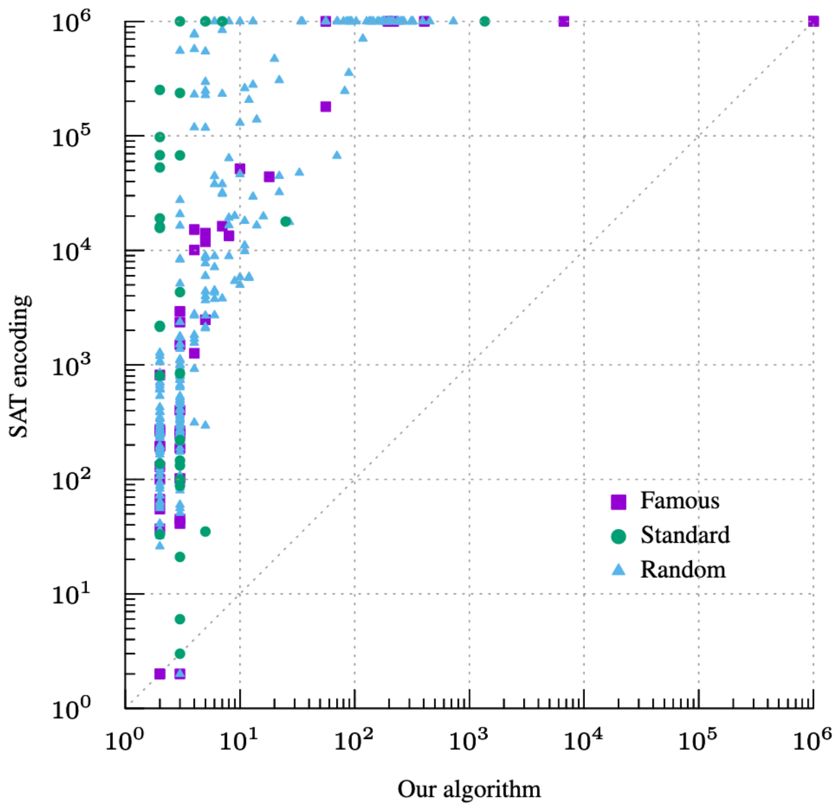
Simple lower bound

Let b be the maximum degree of G

bound(n):
  if n = 0: return 0
  return 1 + bound(ceil((n-1)/b))



Experiments



Comparison with partition-based SAT encoding of Ganian, Lodha, Ordyniak, Szeider (2019)

Experiments

Instance	n	m	td	All	−LB	−Sym	−Dom	SAT
Errera	17	45	10	0.007	0.022	2.486	0.006	16.225
Paley17	17	68	14	0.056	0.072	*	0.052	*
Pappus	18	27	8	0.003	0.029	0.019	0.003	2.363
Robertson	19	38	10	0.018	0.365	3.441	0.021	43.926
Desargues	20	30	9	0.004	0.097	0.323	0.005	15.208
Dodecahedron	20	30	9	0.005	0.104	0.329	0.004	11.871
FlowerSnark	20	30	9	0.008	0.298	0.311	0.007	13.415
Folkman	20	40	9	0.004	0.056	0.118	0.007	10.071
Brinkmann	21	42	11	0.195	3.637	*	0.183	*
Kittell	23	63	12	0.405	3.094	*	0.559	*
McGee	24	36	11	0.219	24.042	344.762	0.175	*
Nauru	24	36	10	0.056	4.914	15.565	0.048	179.968
Holt	27	54	13	6.680	441.213	*	5.623	*
WatkinsSnark	50	75	13	*	*	*	870.345	*
B10Cage	70	105		*	*	*	*	*
Ellingham	78	117		*	*	*	*	*

Conclusion

A simple algorithm for finding an elimination tree of minimum depth.

Added to this:

- Symmetry-breaking rule
- Two domination rules
- Lower bounds

There is scope for improvement in domination rules and lower bounds.

Postscript: PACE Challenge 2020

Thank you

James Trimble

j.trimble.1@research.gla.ac.uk