Practical computation of pathwidth and vertex separation

Dorian Mazauric
LIF, Univ. Aix-Marseille

David Coudert, Nicolas Nisse
COATI, INRIA, I3S, CNRS, Univ. Nice Sophia

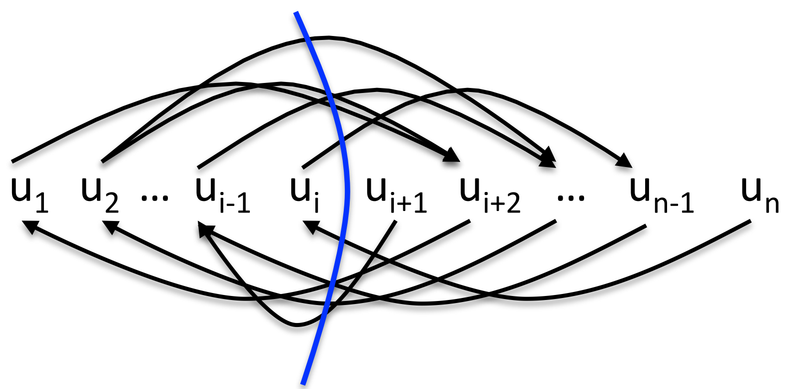
Layout, vertex separation, pathwidth

Layout

- $L = u_1 u_2 \ldots u_n$ is an **ordering** of the elements of a set V
- $\mathcal{L}(V)$, set of all orderings of V

Vertex-separation of a digraph D = (V,A)

- $\omega(L,i) = |\ N_D^+ (u_1 u_2 \ldots u_i)\ |$
- $\omega(L) = \max_{i \leq |L|} \omega(L,i)$
- **$vs(D) = \min_{L \text{ in } \mathcal{L}(V)} \omega(L)$**



Pathwidth of a graph G = (V,E)

- Symmetric digraph D, pw(G) = vs(D)

Existing implementations

Dynamic Programming:

- [Bodlaender, Fomin, Koster, Kratsch, Thilikos - ToCS 2012]
- Worse case time and space complexity in $O(2^n)$
- $O(n^{vs(D)})$ algorithm implemented by N. Cohen in Sage (http://www.sagemath.org)
- (di)graphs with up to 32 nodes in 1 sec.
  - 32 nodes: use up to 4GB of RAM

ILP and SAT formulations:

- [Biedl, Bläsius, Niedermann, Nöllenburg, Prutkin, Rutter - Graph Drawing 2013]
- 20-30 nodes (minutes to hours)
- Another MILP formulation in Sage

Branch-and-Bound:

- [Solano & Pioro – IEEE/OSA JOCN 2010]
- 30 nodes (sec. to hours)

Outline

1. **Pre-processing / reduction rules**
   1. Pathwidth
   2. Vertex separation

2. **Generic branch-and-bound**

3. **Improvements**
   1. Greedy steps
   2. Storing prefixes

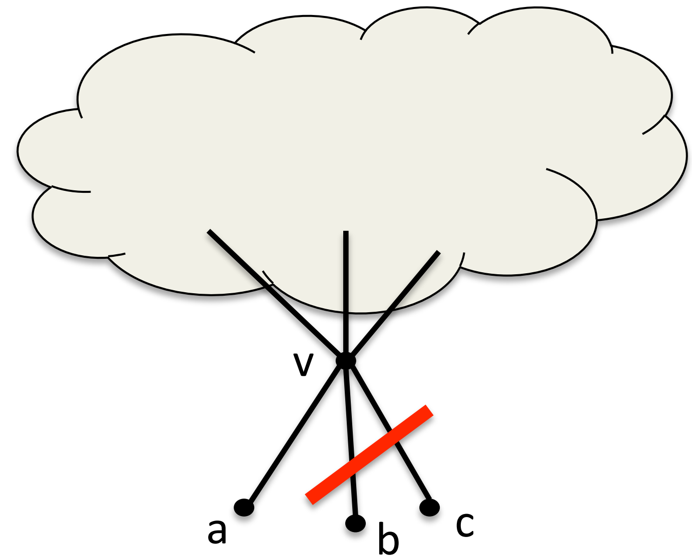
4. **Numerical results**

(Simple) Reduction rules for pathwidth

Simple: **easy to implement** **and** **small computation time**

Safe: pw(G) = pw(G')

Rule 1: If two degree-one vertices share their neighbor, then remove one of them.

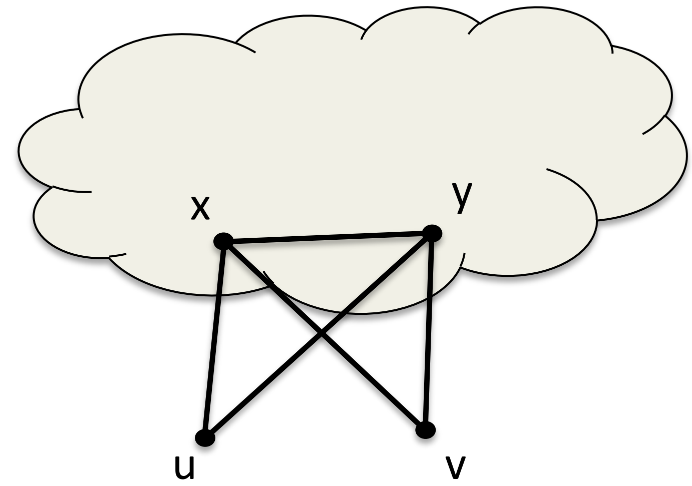


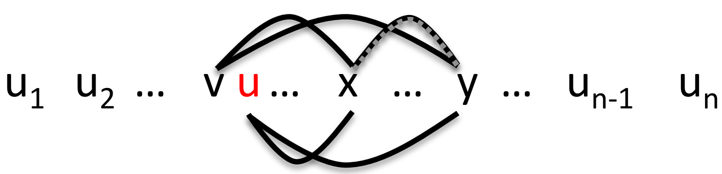
u_1 u_2 … u_{i-1} u_i a … b c v … u_{n-1} u_n

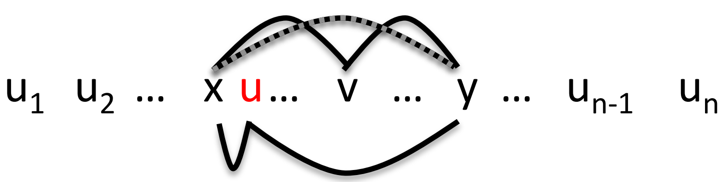
u_1 u_2 … u_{i-1} u_i b c v … a … u_{n-1} u_n

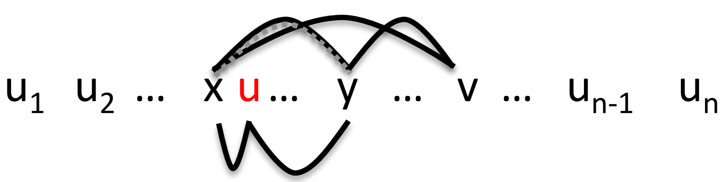
Reduction rules for pathwidth (2)

Rule 2: If two degree-two vertices u and v have the same neighbors x and y, then contract edge ux.









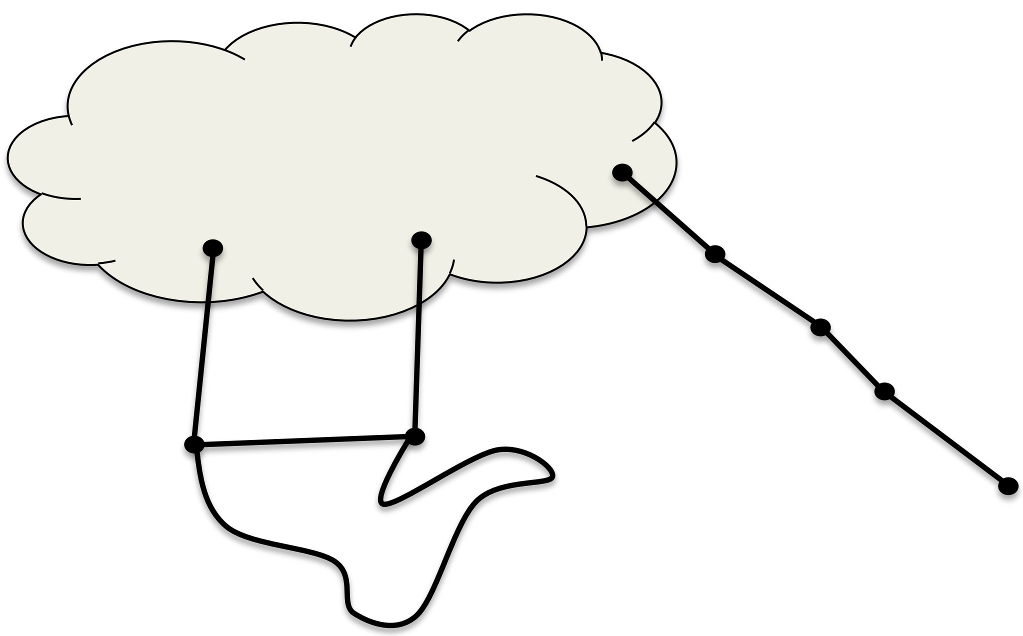
Insert u after leftmost of v, x, y

Value of $\omega(L, i)$ unchanged

Reduction rules for pathwidth (3)

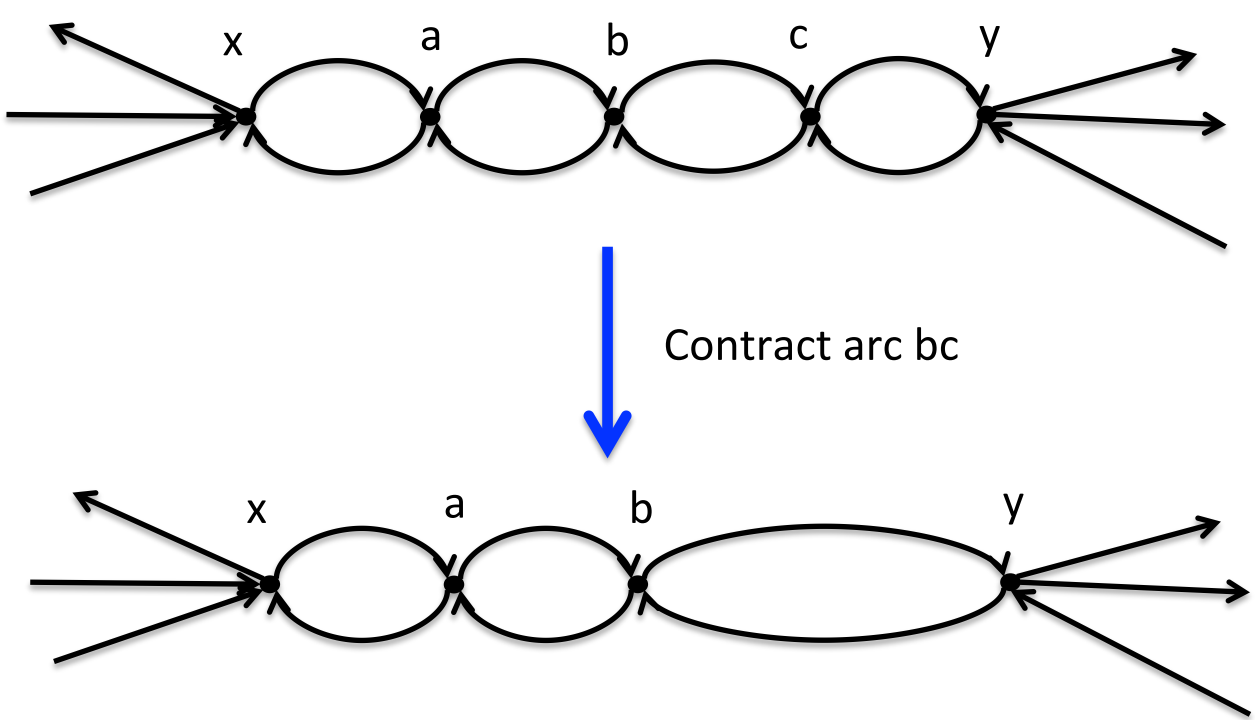
Rule 3: Contract long chains

- Contract long chains of degree-two vertices to chains of length 3 (edges).
- Contract pending chains of degree-two vertices to chains of length 2.



Reduction rules for vertex separation

Rule 4: Contract long **symmetric** chains

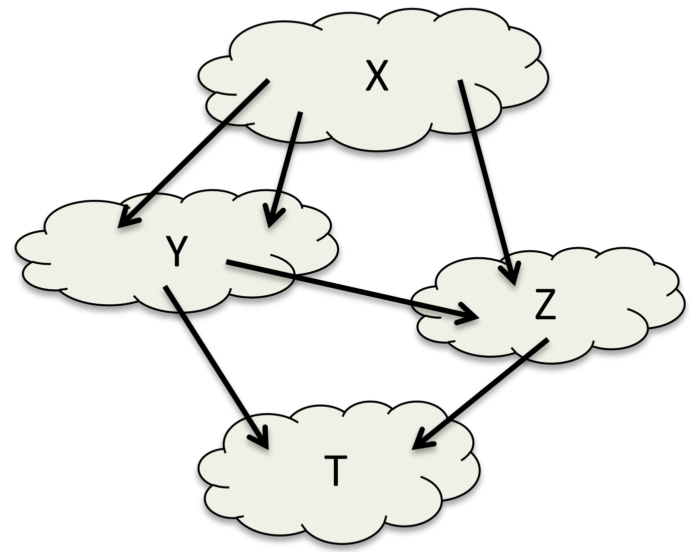


Reduction rules for vertex separation (2)

Rule 5: Decomposition into **strongly connected components**

$L(D) = L(T) \odot L(Z) \odot L(Y) \odot L(X)$

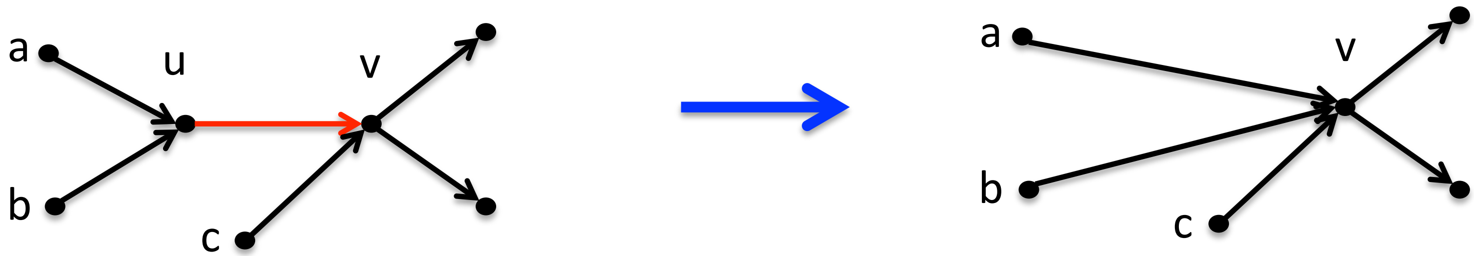
$$\underbrace{t_1\ t_2\ \dots\ t_{|T|}}_{L(T)}\ \underbrace{z_1\ z_2\ \dots\ z_{|Z|}}_{L(Z)}\ \underbrace{y_1\ y_2\ \dots\ y_{|Y|}}_{L(Y)}\ \underbrace{x_1\ x_2\ \dots\ x_{|X|}}_{L(X)}$$

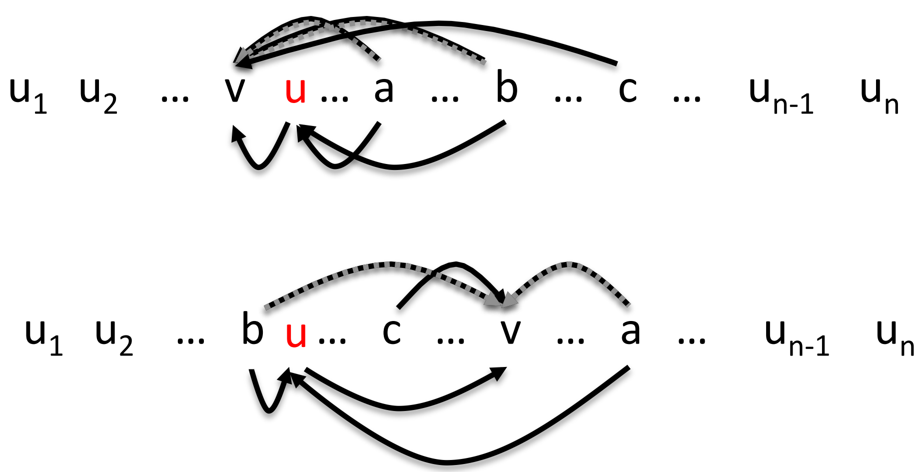


$vs(D) = \max \{ vs(X), vs(Y), vs(Z), vs(T) \}$

Reduction rules for vertex separation (3)

Rule 6: Contract arc *uv* if *u* has out-degree one or *v* has in-degree one unless it creates a loop or parallel arcs

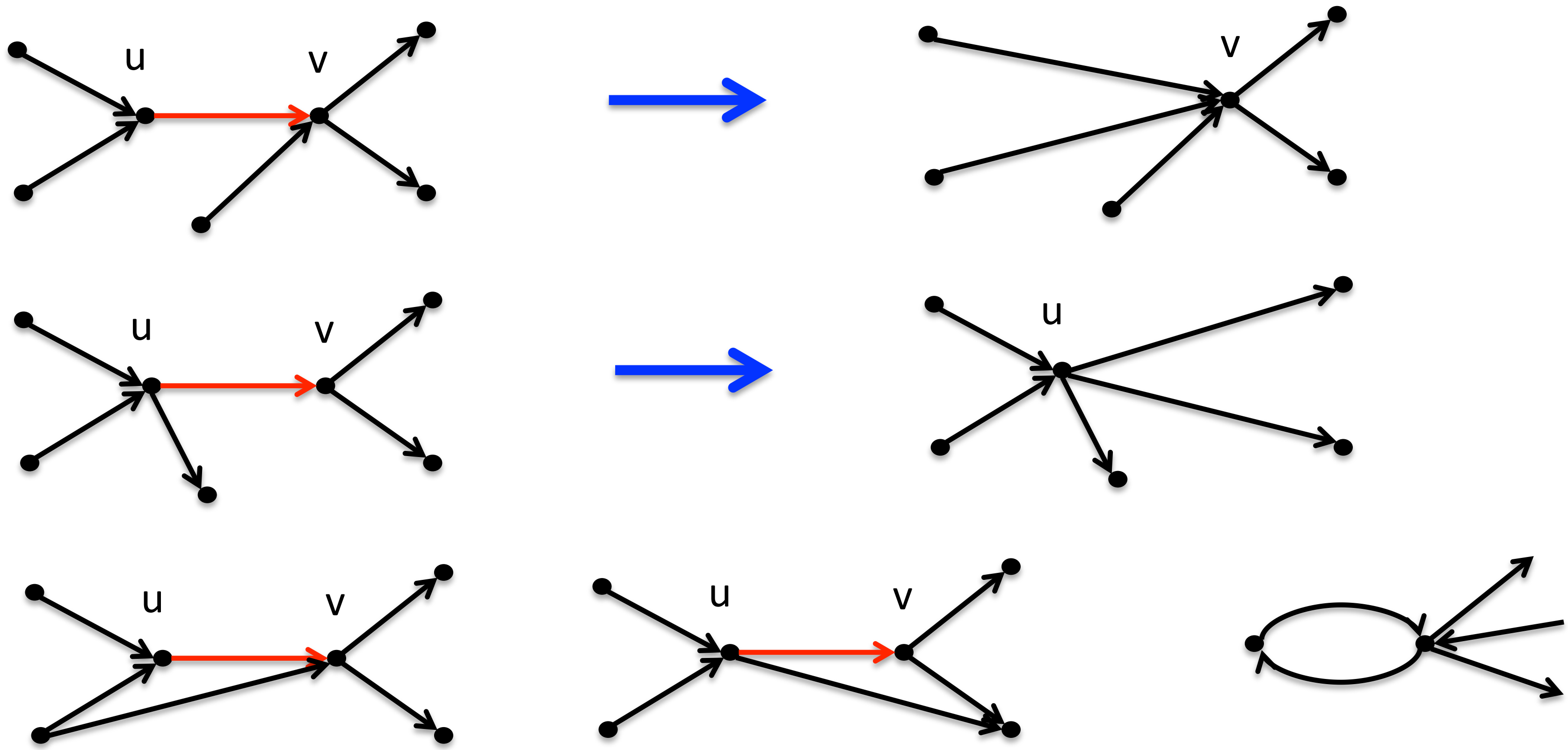




Insert after leftmost of v and the in-neighbors of v

Reduction rules for vertex separation (3)

Rule 6: Contract arc *uv* if *u* has out-degree one or *v* has in-degree one unless it creates a loop or parallel arcs



NO

Generic Branch and Bound

[Solano & Pioro – IEEE/OSA JOCN 2010]

Principle:

- Given current prefix $P = u_1\ u_2 \dots u_k$
- Try all possible extensions $P \odot v$ for v in $V\text{-}V(P)$ using *sort-and-prune*

Sort:

- Try most promising extensions first
- Sort vertices in $V\text{-}V(P)$ by increasing values of $\omega(P \odot v)$

$\omega(P) = \max_{1 \leq i \leq |P|} \omega(L, i)$
for any layout L with prefix P

Prune:

- Given best layout found so far, L^*
- Try only extensions such that $\omega(P \odot v) < \omega(L^*)$

$\omega(P \odot v)$ is a lower bound on the value of any layout with prefix $P \odot v$

Generic Branch and Bound

[Solano & Pioro – IEEE/OSA JOCN 2010]

B&B(D, P, L*):
if V(P) == V and ω(P) < ω(L*) :
    L* := P
else:
    for all v in V-V(P) by increasing value of ω(P ⊙ v) **do**
        if ω(P ⊙ v) < ω(L*) :
            P' := B&B(D, P ⊙ v, L*)
            if ω(P') < ω(L*):
                L* := P'
return L*

sort

prune

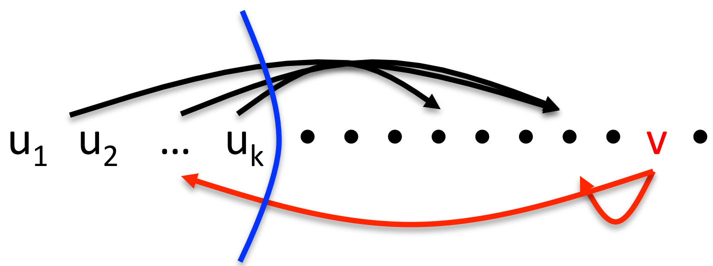
L* := B&B(D, Ø, L)   for some ordering L of V

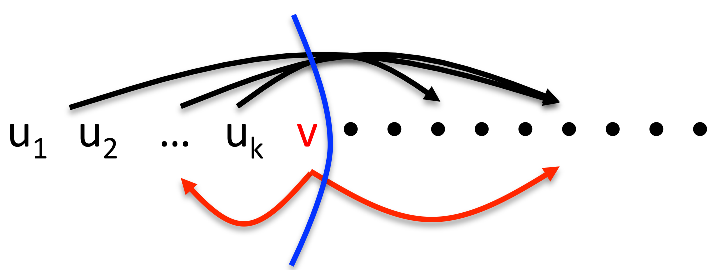
Greedy steps

Lemma 1:

- Given D = (V, A) and prefix $P = u_1\, u_2 \dots u_k$
- Set $\mathcal{L}(V, P)$ of layouts with prefix P
- If there exists v in V-V(P) such that $N^+(v) \subseteq V(P) \cup N^+(V(P))$

  then $\min_{L \text{ in } \mathcal{L}(V,\, P)} \omega(L) = \min_{L \text{ in } \mathcal{L}(V,\, P \odot v)} \omega(L)$



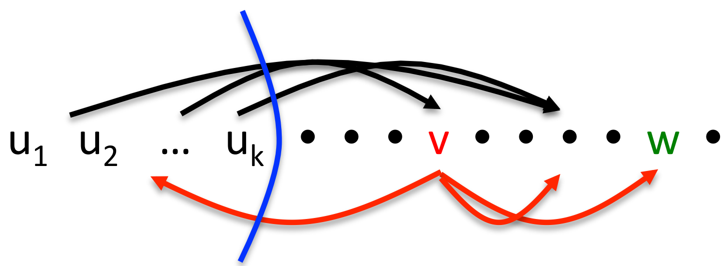


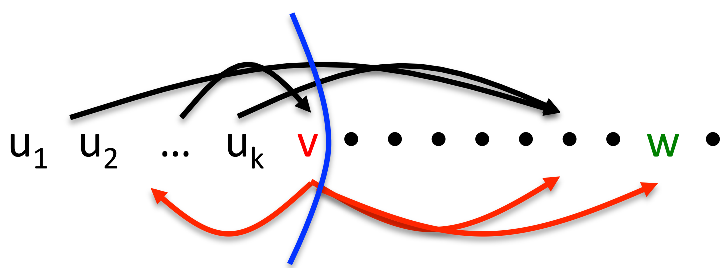
Greedy steps

Lemma 2:

- Given $D = (V, A)$ and prefix $P = u_1\ u_2 \dots u_k$
- Set $\mathcal{L}(V, P)$ of layouts with prefix P
- If there exists v in $N^+(V(P))$ such that $N^+(v) - (V(P) \cup N^+(V(P))) = \{w\}$

  then $\min_{L \text{ in } \mathcal{L}(V,\ P)} \omega(L) = \min_{L \text{ in } \mathcal{L}(V,\ P \odot v)} \omega(L)$





Branch and Bound + Greedy steps

```
B&B(D, P, L* ):
  P := Greedy_steps(D, P)
  if  V(P) == V  and  ω(P) < ω(L*) :
        L* := P
  else:
        for all v in V-V(P) by increasing value of ω(P ⊙ v) do
              if ω(P ⊙ v) < ω(L*) :
                    P' := B&B(D, P ⊙ v, L*)
                    if  ω(P') < ω(L*):
                          L* := P'
  return L*
```

$L^* := B\&B(D, \emptyset, L)$ for some ordering L of V

Observation

L = a b c d *R* best layout with prefix a b c d

1. ω(a b c d) < ω(L) ⟶ the layout of a,b,c,d has no impact on the solution
2. ω(a b c d) = ω(L) ⟶ possible improvement with better layout for a,b,c,d

Lemma 3:

- Given D = (V, A) and prefixes P, P' such that V(P)=V(P')
- Set $\mathcal{L}(V, P)$ of layouts with prefix P
- If $\omega(P) < \min_{L \text{ in } \mathcal{L}(V,\,P)} \omega(L)$ or $\omega(P) \leq \omega(P')$

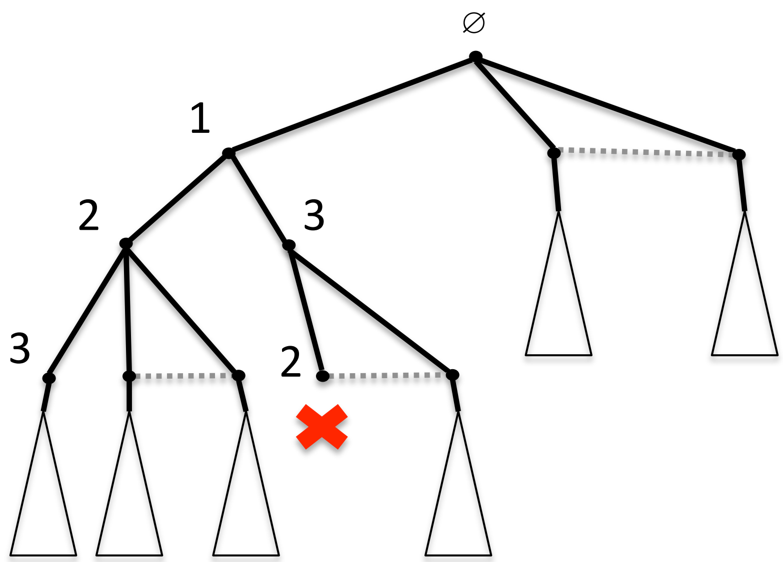
  then $\min_{L \text{ in } \mathcal{L}(V,\,P)} \omega(L) \leq \min_{L \text{ in } \mathcal{L}(V,\,P')} \omega(L)$

Prune branches with prefix P' such that V(P)=V(P') when best possible layout L* with prefix P is such that ω(P) < ω(L*)

Prune branches starting with prefix P' such that ω(P) ≤ ω(P')

Idea: Use a table to store prefixes

Store prefixes



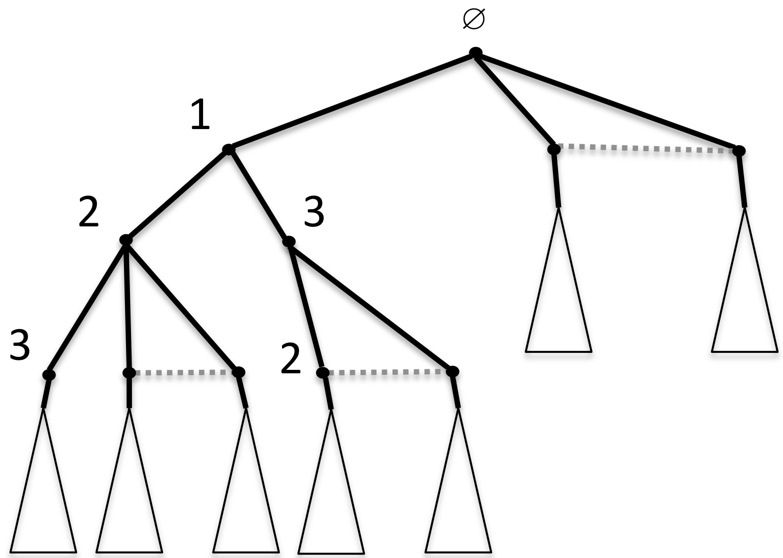
V(P)	ω(P)	b
1	ω(1)	0
1,2	ω(1,2)	0
1,2,3	ω(1,2,3)	0 1
…	…	…

P = (1,2,3)

If $\omega(P) < \min_{L \text{ in } \mathcal{L}(V,\, P)} \omega(L)$

then no possible improvement with other layouts of V(P)

Store prefixes



$V(P)$	$\omega(P)$	b
1	$\omega(1)$	0
1,2	$\omega(1,2)$	0
1,2,3	$\omega(1,3,2)$	0
…	…	…

$P = (1,2,3)$

If $\omega(P) = \min_{L \text{ in } \mathcal{L}(V,\ P)} \omega(L)$

then do nothing

We must try with other layouts of V(P)

$P' = (1,3,2)$

If $\omega(P') < \omega(P)$ then update table

Final branch and Bound

Prefix table

```
B&B(D, 𝒫, P, L* ):
 If not (V(P), ω(P), 1) in 𝒫:
     P := Greedy_steps(D, P)
     if V(P) == V and ω(P) < ω(L*) :
         L* := P
     else:
         Orig := ω(L*)
         for all v in V-V(P) by increasing value of ω(P ⊙ v) do
             if ω(P ⊙ v) < ω(L*) :
                 P' := B&B(D, 𝒫, P ⊙ v, L*)
                 if ω(P') < ω(L*):
                     L* := P'
         Update-prefix-table (D, 𝒫, V(P), Orig, ω(L*) )
 return L*
```

L* := B&B(D, ∅, ∅, L)   for some ordering L of V

Implementation

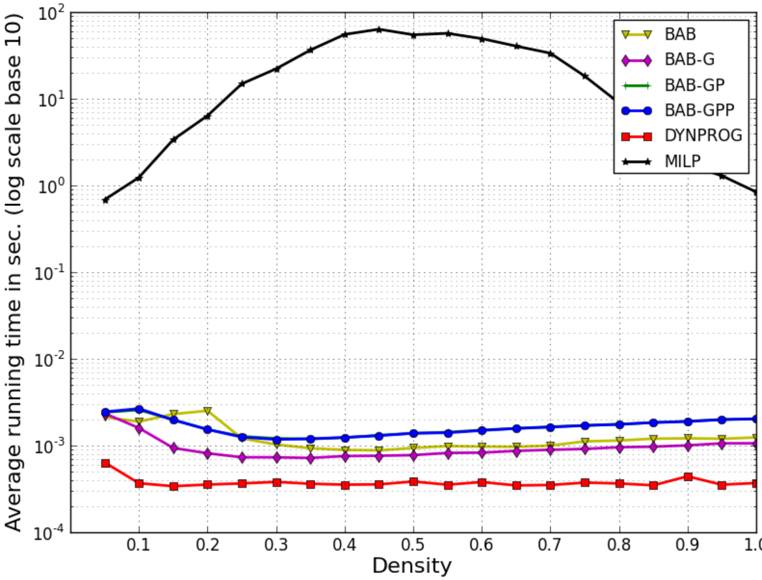
- Sage open source mathematical software (Python/Cython)
  - Several graph algorithms
  - Existing implementation of dyn. prog. and MILP formulation for vertex separation

- Use bitsets for fast operations on neighborhoods
  - Union (or), intersection (and), size (popcount), etc.
  - Use int on 32 bits or 64 bits if n smaller than 64.

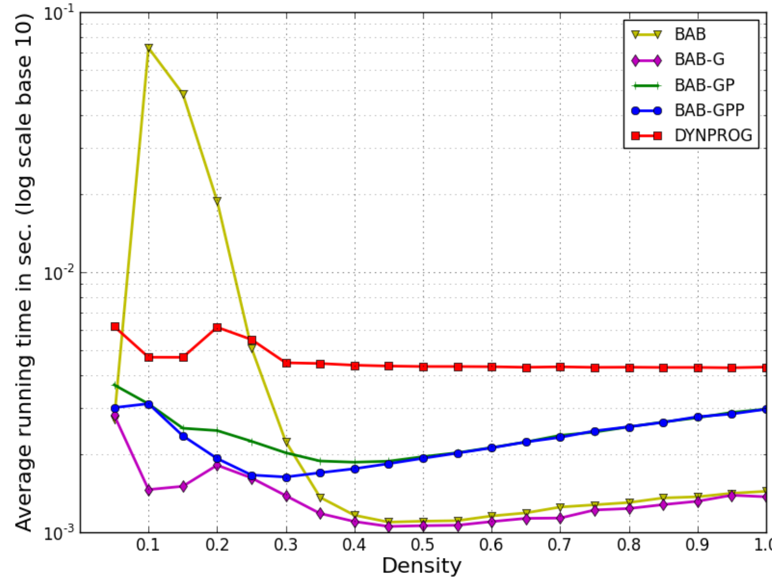
- Prefix table: tree structure
  - Parameterized maximum prefix length (e.g. 10 or n/3)
  - Limit on the number of stored prefixes (e.g. 10 millions)

Performances on directed GNP

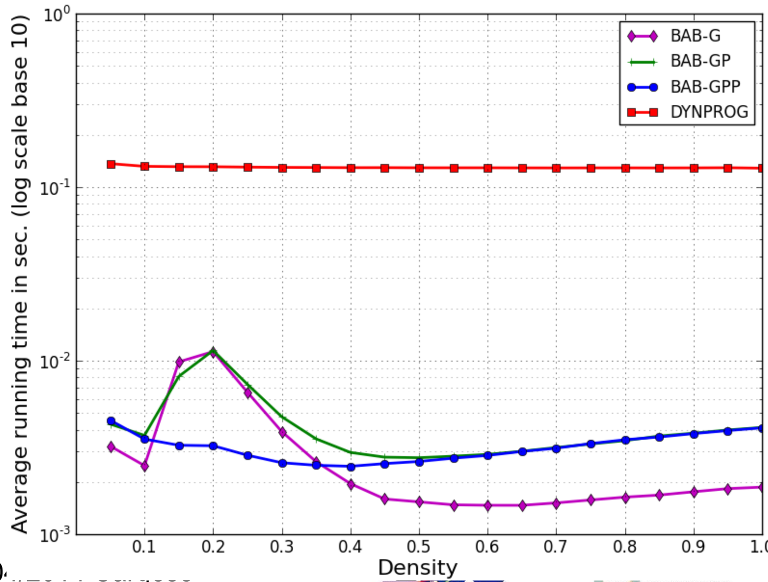
20



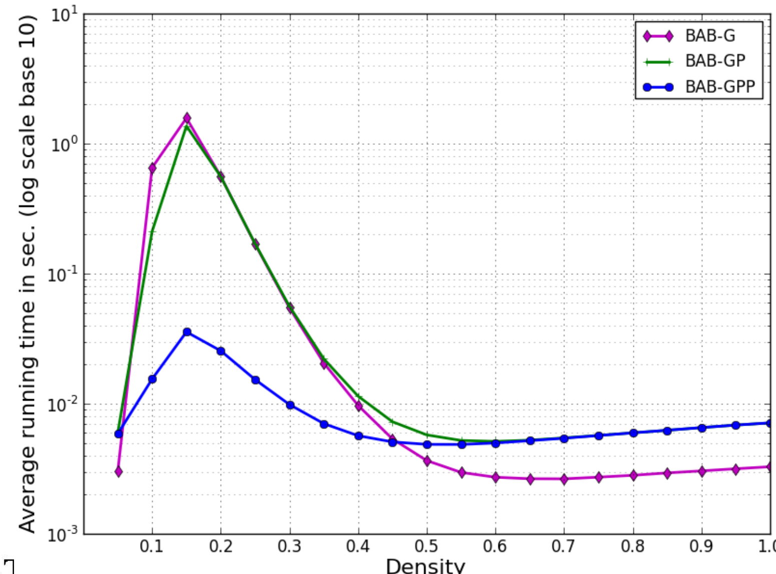
25



30

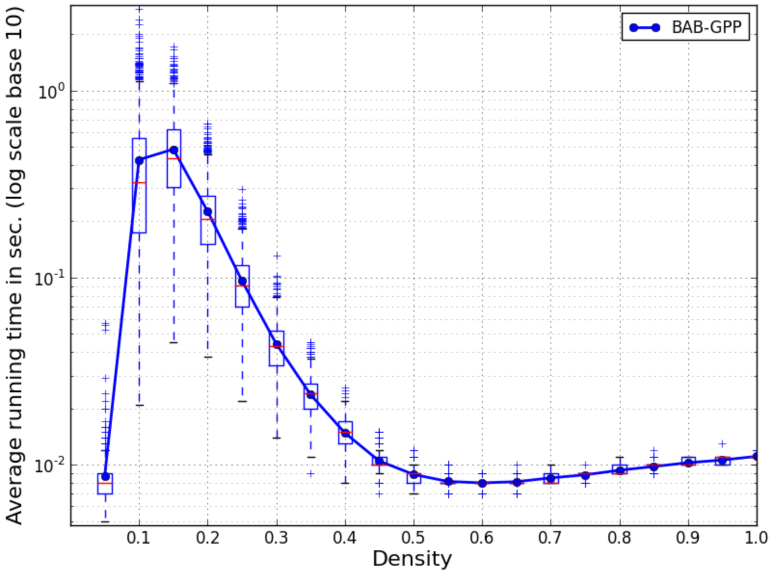


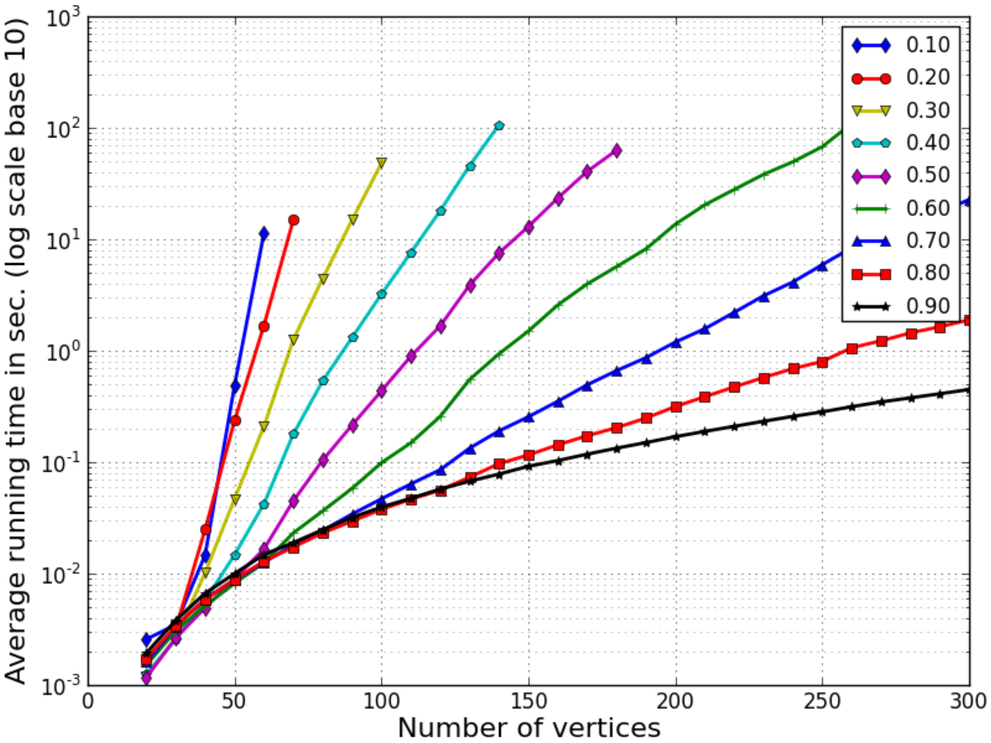
40



Performances on directed GNP

50





Performances on Mycielski graphs

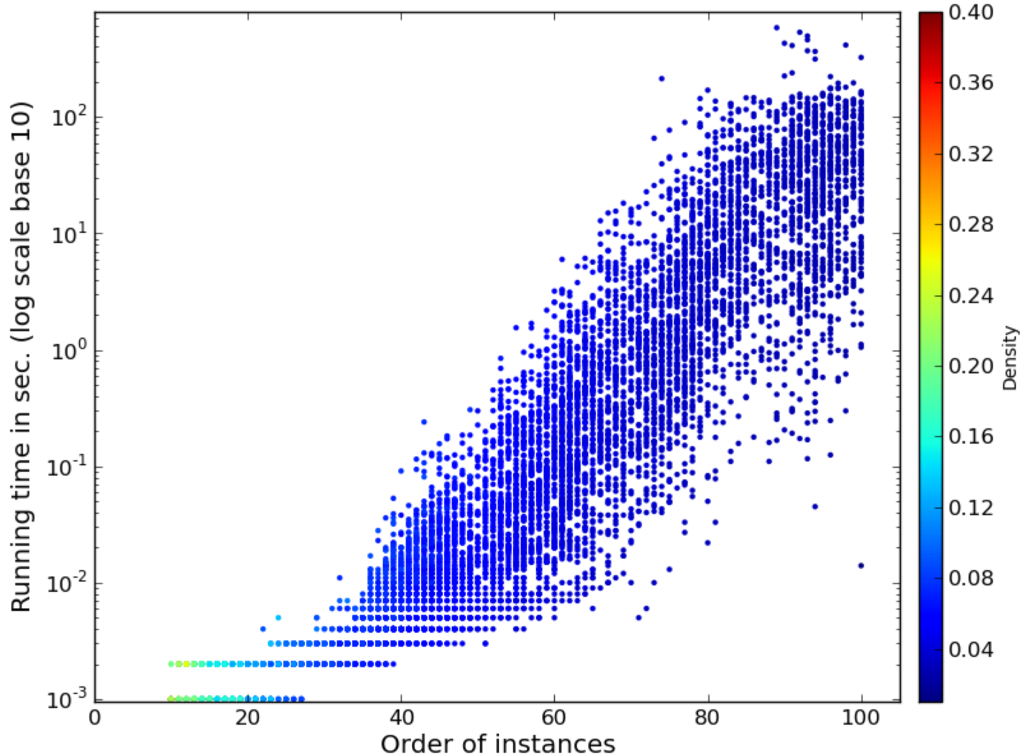
Triangle free graphs with increasing chromatic number (and large pathwidth)

$\mathbf{M_6}$ with 47 nodes and pw(M_6) = 20

Max. prefix length	Time (in sec.)	Visited nodes	Stored prefix
1	2226.45	1 256 780 074	44
5	144.93	82 417 700	16 386
8	5.81	3 319 482	47 756
10	1.07	664 677	61 466
15	0.77	496 482	65 252
20	0.78	496 482	65 253

Performances on Rome dataset

- 11 529 undirected n-node graphs with 10 ≤ n ≤ 100
- ILP/SAT [Biedl *et al.* - Graph Drawing 2013]
  - Time limit per graph: 10 min
  - 17 % graphs solved
  - Almost all such that n+m ≤ 45 and almost no graphs s.t. n+m ≥ 70
- B&B
  - Same time limit
  - 95.6 % graphs solved
  - All graphs with n ≤ 82



Ongoing & future work

Various heuristics:

- First solution of B&B
- Best solution of B&B in x sec.
- Local Search, simulated annealing

Future work:

- Inclusion of B&B in Sage
- Other reduction rules, cuts, new ideas, etc.
- Lower bounds
  - Almost all computation time used to prove optimality of the solution
  - Important for optimality gap

- Extend to other width: cutwidth, bandwidth, etc.

References

1. T. C. Biedl, T. Bläsius, B. Niedermann, M. Nöllenburg, R. Prutkin, and I. Rutter. Using ILP/SAT to determine pathwidth, visibility representations, and other grid-based graph drawings. In Graph Drawing, volume 8242 of LNCS, pages 460–471. Springer, 2013.
2. H. L. Bodlaender, F. V. Fomin, A. M. Koster, D. Kratsch, and D. M. Thilikos. A note on exact algorithms for vertex ordering problems on graphs. Theory Comput. Syst., 50(3):420–432, 2012.
3. H.L. Bodlaender, B. M. P. Jansen, S. Kratsch: Kernel Bounds for Structural Parameterizations of Pathwidth. SWAT 2012: 352-363
4. D. Coudert, D. Mazauric, and N. Nisse. Experimental evaluation of a branch and bound algorithm for computing pathwidth. Tech. Rep. RR-8470, Inria, Feb. 2014.
5. F. Solano and M. Pioro. Lightpath reconfiguration in WDM networks. IEEE/OSA J. Opt. Commun. Netw., 2(12):1010–1021, 2010.

6. Rome graphs. http://www.graphdrawing.org/download/rome-graphml.tgz.
7. W. Stein et al. Sage Mathematics Software (Version 6.0). The Sage Development Team, 2013. http://www.sagemath.org.