A linear-time parameterized algorithm for computing the width of a DAG

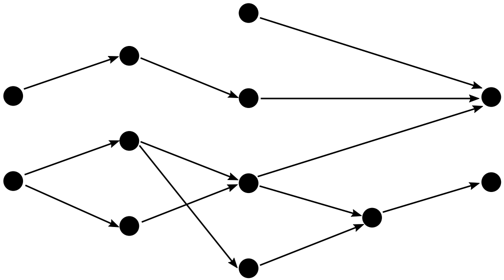
Manuel Cáceres, Massimo Cairo, Brendan Mumey, Romeo Rizzi and Alexandru I. Tomescu

24.06.2021, WG

Basics

Basics

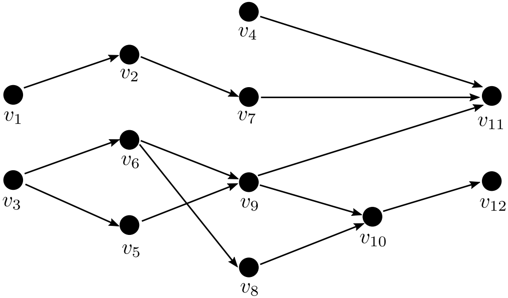
- Directed acyclic graph (DAG) $G = (V, E)$



Basics

- Topological ordering: $O(|V| + |E|)$ [15, 18]

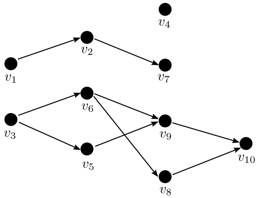
$v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}$



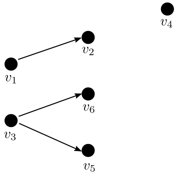
Basics

- Topologically induced subgraph, $G_i := G[\{v_1, \ldots, v_i\}]$

$G_{10} :=$

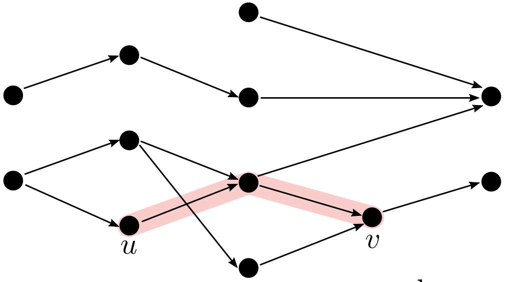


$G_6 :=$



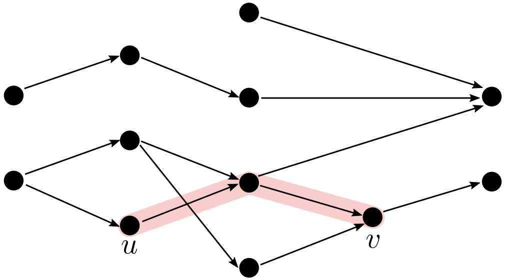
Basics

- Constant-time reachability queries



u reaches v

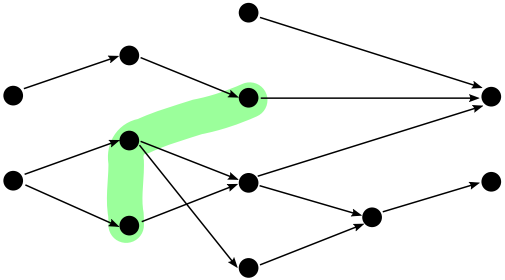
- Constant-time reachability queries



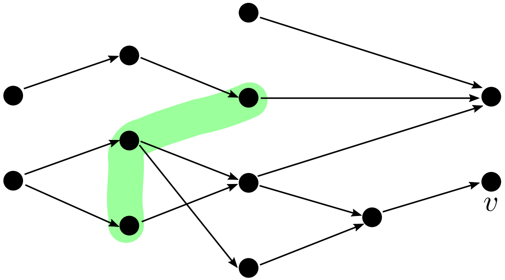
$\mathcal{S}.reaches(u, v)$ in $O(1)$

Basics

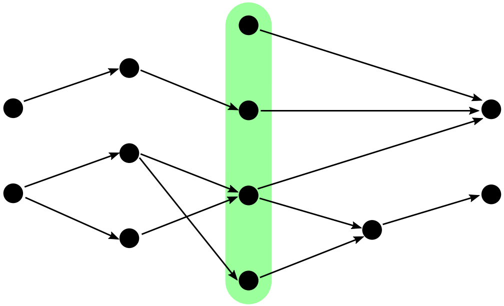
- Antichain



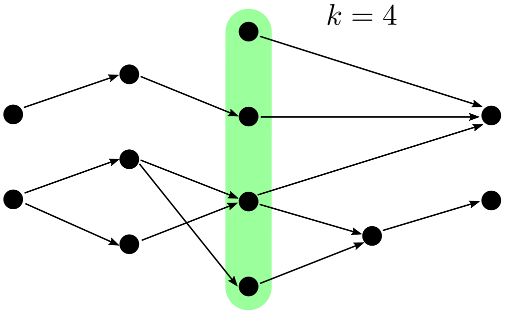
- Antichain reaches v



- Maximum antichain



- Width of DAG



Applications

Applications of computing the width

- Bioinformatics [1, 11]

Applications of computing the width

- Bioinformatics [1, 11]
  - Perfect Phylogeny Haplotyping

Applications of computing the width

- Bioinformatics [1, 11]
  - Perfect Phylogeny Haplotyping

- Evolutionary computation [14]

Applications of computing the width

- Bioinformatics [1, 11]
  - Perfect Phylogeny Haplotyping
- Evolutionary computation [14]
  - Dimension of a game

Applications of computing the width

- Bioinformatics [1, 11]
  - Perfect Phylogeny Haplotyping
- Evolutionary computation [14]
  - Dimension of a game
- Distributed computation [13, 19]

Applications of computing the width

- Bioinformatics [1, 11]
  - Perfect Phylogeny Haplotyping
- Evolutionary computation [14]
  - Dimension of a game
- Distributed computation [13, 19]
  - K mutual exclusion violation

Algorithms parameterized by the width k

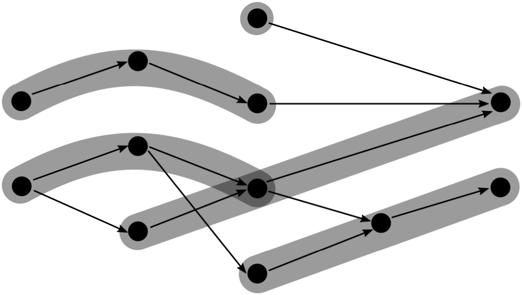
Why algorithms parameterized by k?

- Natural parameter
- Some applications: small k (pan-genomes [16])
- FPT-algorithms [20, 5, 2, 10]

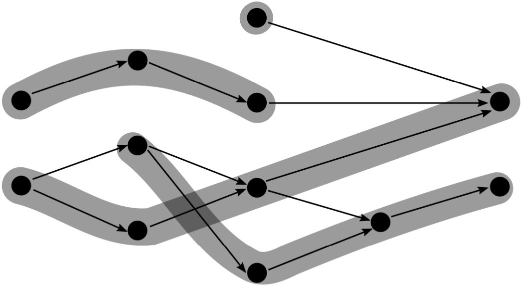
(Most) State-of-the-art: *Minimum Path Cover*

Basics

- Path cover

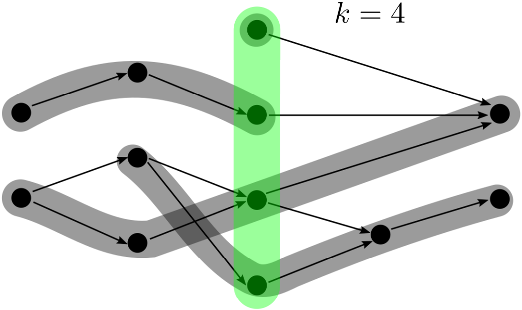


- Minimum Path Cover (MPC)



Basics

- Dilworth's Theorem [6]



MPC algorithms

Maximum Matching	Minimum Flow
– $O(\sqrt{\lvert V\rvert}\lvert E\rvert)$ [9, 12] (posets)	– $O(\lvert V\rvert\lvert E\rvert)$ [17, 8]
– $O(\lvert V\rvert^2 + k\sqrt{k}\lvert V\rvert)$ [3]	– $O(k\lvert E\rvert \log \lvert V\rvert)$ [16]
– $O(\sqrt{\lvert V\rvert}\lvert E\rvert + k\sqrt{k}\lvert V\rvert)$ [4]	

MPC algorithms

Maximum Matching	Minimum Flow
– $O(\sqrt{\lvert V\rvert}\lvert E\rvert)$ [9, 12] (posets)	– $O(\lvert V\rvert\lvert E\rvert)$ [17, 8]
– $O(\lvert V\rvert^2 + k\sqrt{k}\lvert V\rvert)$ [3]	– $O(k\lvert E\rvert \log \lvert V\rvert)$ [16]
– $O(\sqrt{\lvert V\rvert}\lvert E\rvert + k\sqrt{k}\lvert V\rvert)$ [4]	

Felsner et al. [7] recognize posets:

- $O(|V|)$, for $k \leq 3$.
- $O(|V| \log |V|)$, for $k = 4$.
- “the case $k = 5$ already seems to require an unpleasantly involved case analysis“ [7, p. 359]

Our result

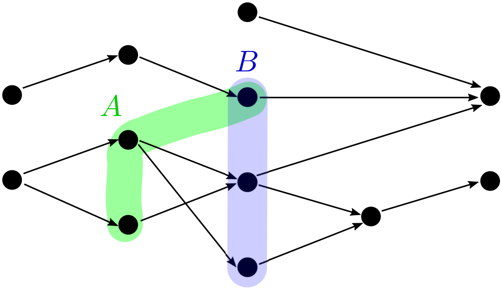
$O(f(k)(|V|| + |E|))$ time algorithm

Maximum Antichain

Antichain domination

Definition 1 (Dominates)

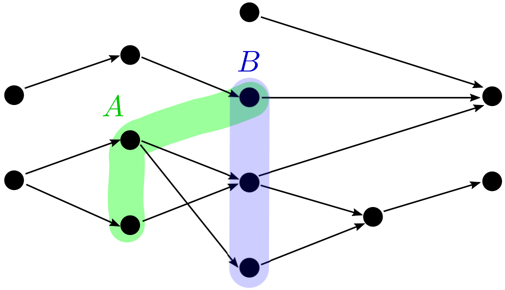
Antichain B *dominates* antichain A if $|A| = |B|$ and for each $b \in B$, A reaches b



Antichain domination

Definition 1 (Dominates)

Antichain B *dominates* antichain A if $|A| = |B|$ and for each $b \in B$, A reaches b



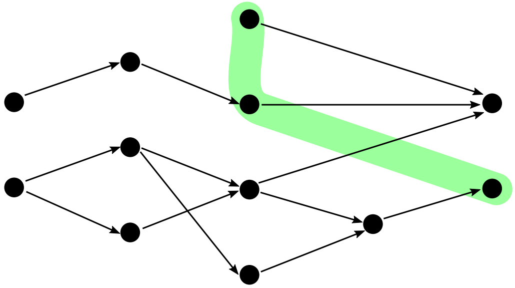
Lemma 1

Domination is a partial order on antichains of G.

Frontier Antichains

Definition 2 (Frontier)

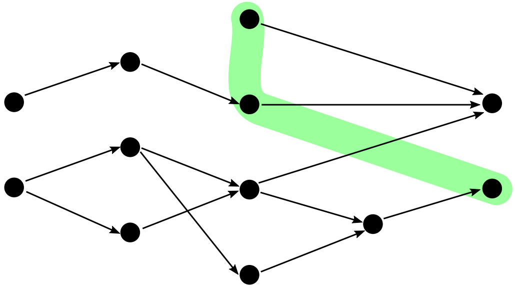
- Maximal elements of domination relation
- Antichains only dominated by themselves



Frontier Antichains

Definition 2 (Frontier)

- Maximal elements of domination relation
- Antichains only dominated by themselves



Lemma 3

G has at most 2^k frontier antichains

If A is a frontier antichain of G we also say that A is G-frontier

Classification of G_i-frontiers

We classify G_i-frontiers A into two categories:

Classification of G_i-frontiers

We classify G_i-frontiers A into two categories:

Type 1 : $v_i \in A$

Classification of G_i-frontiers

We classify G_i-frontiers A into two categories:

Type 1 : $v_i \in A$

Type 2 : $v_i \notin A$

Classification of G_i-frontiers

We classify G_i-frontiers A into two categories:

Type 1 : $v_i \in A$

Lemma 4

If A is type 1 G_i-frontier, then $A \setminus \{v_i\}$ is G_{i-1}-frontier

Type 2 : $v_i \notin A$

Classification of G_i-frontiers

We classify G_i-frontiers A into two categories:

Type 1 : $v_i \in A$

Lemma 4

If A is type 1 G_i-frontier, then $A \setminus \{v_i\}$ is G_{i-1}-frontier

Lemma 6

If B is G_{i-1}-frontier and B does not reach v_i, then $B \cup \{v_i\}$ is type 1 G_i-frontier

Type 2 : $v_i \notin A$

Classification of G_i-frontiers

We classify G_i-frontiers A into two categories:

Type 1 : $v_i \in A$

Lemma 4

If A is type 1 G_i-frontier, then $A \setminus \{v_i\}$ is G_{i-1}-frontier

Lemma 6

If B is G_{i-1}-frontier and B does not reach v_i, then $B \cup \{v_i\}$ is type 1 G_i-frontier

Type 2 : $v_i \notin A$

Lemma 5

If A is type 2 G_i-frontier, then A is G_{i-1}-frontier

Classification of G_i-frontiers

We classify G_i-frontiers A into two categories:

Type 1 : $v_i \in A$

Lemma 4

If A is type 1 G_i-frontier, then $A \setminus \{v_i\}$ is G_{i-1}-frontier

Lemma 6

If B is G_{i-1}-frontier and B does not reach v_i, then $B \cup \{v_i\}$ is type 1 G_i-frontier

Type 2 : $v_i \notin A$

Lemma 5

If A is type 2 G_i-frontier, then A is G_{i-1}-frontier

Lemma 2

If B is G_{i-1}-frontier but not G_i-frontier, then B is dominated by type-1 G_i-frontier

The Algorithm

(for posets)

Algorithm (simplified)

for $v_i \in v_1, \ldots, v_{|V|}$ *in topological order* **do**
  for $A \in G_{i-1}$*-frontiers* **do**
    if A *does not reach* v_i **then**
      Store $A \cup \{v_i\}$ as type 1 G_i-frontier
  for $A \in G_{i-1}$*-frontiers* **do**
    if $\forall B \in$ *type 1* G_i*-frontiers,* B *does not dominate* A **then**
      Store A as type 2 G_i-frontier
return Largest frontier

Algorithm (simplified)

for $v_i \in v_1, \ldots, v_{|V|}$ *in topological order* **do**
- **for** $A \in G_{i-1}$*-frontiers* **do**
  - **if** A *does not reach* v_i **then**
    - Store $A \cup \{v_i\}$ as type 1 G_i-frontier
- **for** $A \in G_{i-1}$*-frontiers* **do**
  - **if** $\forall B \in$ *type 1* G_i*-frontiers,* B *does not dominate* A **then**
    - Store A as type 2 G_i-frontier

return Largest frontier

$O(k^2 4^k |V|)$: with constant-time reachability queries (posets)

The Algorithm

(Maintain constant-time reachability queries)

The Support

Observation 1

When computing G_i-frontiers we only need reachability among vertices of G_{i-1}-frontiers and v_i

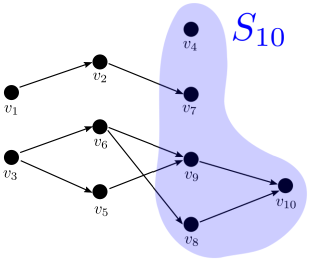
Observation 1

When computing G_i-frontiers we only need reachability among vertices of G_{i-1}-frontiers and v_i

Definition 3 (Support)

$$S_i := \bigcup_{A \in G_i\text{-frontiers}} A$$

$$G_{10} :=$$



The Support

Observation 1

When computing G_i-frontiers we only need reachability among vertices of G_{i-1}-frontiers and v_i

Definition 3 (Support)

$$S_i := \bigcup_{A \in G_i\text{-frontiers}} A$$

Lemma 7 and 8 (Informal)

A vertex v_i only belongs to a topologically adjacent sequence of supports $S_i, \ldots, S_j$

$\Rightarrow$ Theorem 2 and Theorem 3

Algorithm

Observation 1

When computing G_i-frontiers we only need reachability among vertices of $S_{i-1} \cup \{v_i\}$

Algorithm

Observation 1

When computing G_i-frontiers we only need reachability among vertices of $S_{i-1} \cup \{v_i\}$

- Reduced to maintain reachability **from** vertices in S_{j-1} **to** v_j for each $j \leq i$ **(Theorem 2)**

Observation 1

When computing G_i-frontiers we only need reachability among vertices of $S_{i-1} \cup \{v_i\}$

- Reduced to maintain reachability **from** vertices in S_{j-1} **to** v_j for each $j \leq i$ **(Theorem 2)**
- Compute inductively reachability **from** vertices in S_{i-1} **to** v_i, in $O(k2^k)$ per edge incoming to v_i **(Theorem 3)**

Algorithm

Observation 1

When computing G_i-frontiers we only need reachability among vertices of $S_{i-1} \cup \{v_i\}$

- Reduced to maintain reachability **from** vertices in S_{j-1} **to** v_j for each $j \leq i$ **(Theorem 2)**
- Compute inductively reachability **from** vertices in S_{i-1} **to** v_i, in $O(k2^k)$ per edge incoming to v_i **(Theorem 3)**

Theorem 1

Given a DAG $G = (V, E)$ of width k, we can compute a maximum antichain of it in time $O(k^2 4^k |V| + k2^k |E|)$

Acknowledgments

We thank the anonymous reviewers for their useful suggestions, and you for listening until the end 😊

Funding

European Research Council

Established by the European Commission

This work was partially funded by the ERC Starting Grant 851093 (SAFEBIO)

References I

[1] Bonizzoni, P.
A linear-time algorithm for the perfect phylogeny haplotype problem.
Algorithmica 48, 3 (2007), 267–285.

[2] Bova, S., Ganian, R., and Szeider, S.
Model checking existential logic on partially ordered sets.
ACM Transactions on Computational Logic (TOCL) 17, 2 (2015), 1–35.

[3] Chen, Y., and Chen, Y.
An efficient algorithm for answering graph reachability queries.
In *2008 IEEE 24th International Conference on Data Engineering* (2008), IEEE, pp. 893–902.

References II

[4] Chen, Y., and Chen, Y.
On the graph decomposition.
In *2014 IEEE Fourth International Conference on Big Data and Cloud Computing* (2014), IEEE, pp. 777–784.

[5] Colbourn, C. J., and Pulleyblank, W. R.
Minimizing setups in ordered sets of fixed width.
Order 1, 3 (1985), 225–229.

[6] Dilworth, R. P.
A decomposition theorem for partially ordered sets.
Annals of Mathematics 51, 1 (1950), 161–166.

[7] Felsner, S., Raghavan, V., and Spinrad, J.
Recognition algorithms for orders of small width and graphs of small dilworth number.
Order 20, 4 (2003), 351–364.

References III

[8] Ford, L. R., and Fulkerson, D. R.
Maximal flow through a network.
In *Classic papers in combinatorics*. Springer, 2009, pp. 243–248.

[9] Fulkerson, D. R.
Note on dilworth's decomposition theorem for partially ordered sets.
In *Proc. Amer. Math. Soc* (1956), vol. 7, pp. 701–702.

[10] Gajarskỳ, J., Hlinenỳ, P., Lokshtanov, D., Obdralek, J., Ordyniak, S., Ramanujan, M., and Saurabh, S.
Fo model checking on posets of bounded width.
In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science* (2015), IEEE, pp. 963–974.

[11] Gramm, J., Nierhoff, T., Sharan, R., and Tantau, T.
Haplotyping with missing data via perfect path phylogenies.
Discrete Applied Mathematics 155, 6-7 (2007), 788–805.

[12] Hopcroft, J. E., and Karp, R. M.
An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs.
SIAM Journal on computing 2, 4 (1973), 225–231.

[13] Ikiz, S., and Garg, V. K.
Efficient incremental optimal chain partition of distributed program traces.
In *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)* (2006), IEEE, pp. 18–18.

[14] Jaśkowski, W., and Krawiec, K.
Formal analysis, hardness, and algorithms for extracting internal structure of test-based problems.
Evolutionary computation 19, 4 (2011), 639–671.

[15] Kahn, A. B.
Topological sorting of large networks.
Communications of the ACM 5, 11 (1962), 558–562.

[16] Mäkinen, V., Tomescu, A. I., Kuosmanen, A., Paavilainen, T., Gagie, T., and Chikhi, R.
Sparse Dynamic Programming on DAGs with Small Width.
ACM Transactions on Algorithms (TALG) 15, 2 (2019), 1–21.

[17] Ntafos, S. C., and Hakimi, S. L.
On path cover problems in digraphs and applications to program testing.
IEEE Transactions on Software Engineering, 5 (1979), 520–529.

[18] Tarjan, R. E.
Edge-disjoint spanning trees and depth-first search.
Acta Informatica 6, 2 (1976), 171–185.

[19] Tomlinson, A. I., and Garg, V. K.
Monitoring functions on global states of distributed programs.
Journal of Parallel and Distributed Computing 41, 2 (1997), 173–189.

[20] Van Bevern, R., Bredereck, R., Bulteau, L., Komusiewicz, C., Talmon, N., and Woeginger, G. J.
Precedence-constrained scheduling problems parameterized by partial order width.
In *International conference on discrete optimization and operations research* (2016), Springer, pp. 105–120.