Finding Branch-decompositions and Rank-decompositions

Petr Hliněný

Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Rep.

e-mail: `hlineny@fi.muni.cz`
`http://www.fi.muni.cz/~hlineny`

joint work with **Sang-il Oum**

Department of Combinatorics and Optimization,
University of Waterloo, Canada.

e-mail: `sangil@math.uwaterloo.ca`.

Contents

1 Brief Overview of “Widths”

Traditional tree-width

- name given by Robertson & Seymour (1983), but

  some ideas (“k-trees”) date back to Beineke & Pippert (1968) and Rose (1974),

1 Brief Overview of "Widths"

Traditional tree-width

- name given by Robertson & Seymour (1983), but

  some ideas ("k-trees") date back to Beineke & Pippert (1968) and Rose (1974),

- the same parameter having various descriptions, e.g. using vertex bags, k-trees, simplicial (elimination) vertex ordering, etc...

1 Brief Overview of "Widths"

Traditional tree-width

- name given by Robertson & Seymour (1983), but

  some ideas ("k-trees") date back to Beineke & Pippert (1968) and Rose (1974),

- the same parameter having various descriptions, e.g. using vertex bags, k-trees, simplicial (elimination) vertex ordering, etc...

- recent matroid tree-width ("vertex-free") by PH & Whittle (2003).

1 Brief Overview of "Widths"

Traditional tree-width

- name given by Robertson & Seymour (1983), but

  some ideas ("k-trees") date back to Beineke & Pippert (1968) and Rose (1974),

- the same parameter having various descriptions, e.g. using vertex bags, k-trees, simplicial (elimination) vertex ordering, etc...

- recent matroid tree-width ("vertex-free") by PH & Whittle (2003).

Branch-width -based

- graph branch-width by Robertson & Seymour (1983),

  withing a factor of $3/2$ of tree-width,

1 Brief Overview of "Widths"

Traditional tree-width

- name given by Robertson & Seymour (1983), but

  some ideas ("k-trees") date back to Beineke & Pippert (1968) and Rose (1974),

- the same parameter having various descriptions, e.g. using vertex bags, k-trees, simplicial (elimination) vertex ordering, etc...

- recent matroid tree-width ("vertex-free") by PH & Whittle (2003).

Branch-width -based

- graph branch-width by Robertson & Seymour (1983),

  withing a factor of $3/2$ of tree-width,

- definition quite abstract, so having an immediate extension to, e.g. hypergraph or matroid branch-width,

1 Brief Overview of "Widths"

Traditional tree-width

- name given by Robertson & Seymour (1983), but

  some ideas ("k-trees") date back to Beineke & Pippert (1968) and Rose (1974),

- the same parameter having various descriptions, e.g. using vertex bags, k-trees, simplicial (elimination) vertex ordering, etc...

- recent matroid tree-width ("vertex-free") by PH & Whittle (2003).

Branch-width -based

- graph branch-width by Robertson & Seymour (1983),

  withing a factor of $3/2$ of tree-width,

- definition quite abstract, so having an immediate extension to, e.g. hypergraph or matroid branch-width,

- and then to graph rank-width, by Oum & Seymour (2003).

1 Brief Overview of "Widths"

Traditional tree-width

- name given by Robertson & Seymour (1983), but

  some ideas ("k-trees") date back to Beineke & Pippert (1968) and Rose (1974),
- the same parameter having various descriptions, e.g. using vertex bags, k-trees, simplicial (elimination) vertex ordering, etc. . .
- recent matroid tree-width ("vertex-free") by PH & Whittle (2003).

Branch-width -based

- graph branch-width by Robertson & Seymour (1983),

  withing a factor of $3/2$ of tree-width,
- definition quite abstract, so having an immediate extension to, e.g. hypergraph or matroid branch-width,
- and then to graph rank-width, by Oum & Seymour (2003).

Other parameters

- hypertree-width variants for hypergraphs, Gottlob, Leone & Scarcello (2002),

Other parameters

- hypertree-width variants for hypergraphs, Gottlob, Leone & Scarcello (2002),
- directed tree-width by Johnson, Robertson, Seymour & Thomas (2001), followed by DAG-width or Kelly-width, etc. . .

Other parameters

- hypertree-width variants for hypergraphs, Gottlob, Leone & Scarcello (2002),
- directed tree-width by Johnson, Robertson, Seymour & Thomas (2001),

  followed by DAG-width or Kelly-width, etc...
- graph clique-width defined by Courcelle & Olariu (2000),

  which is a rather different, logic-motivated concept; it interests us since clique-width directly motivated graph rank-width,

Other parameters

- hypertree-width variants for hypergraphs, Gottlob, Leone & Scarcello (2002),
- directed tree-width by Johnson, Robertson, Seymour & Thomas (2001),

  followed by DAG-width or Kelly-width, etc...
- graph clique-width defined by Courcelle & Olariu (2000),

  which is a rather different, logic-motivated concept; it interests us since clique-width directly motivated graph rank-width,
- and few more notions (path-width, bandwidth, cut-width, * * * * *).

Algorithmic aspects of "width" parameters

Tree-width

- is NP-hard to compute on graphs, but

Algorithmic aspects of “width” parameters

Tree-width

- is NP-hard to compute on graphs, but
- there is a linear-time FPT algorithm for it by Bodlaender (1996).

Algorithmic aspects of “width” parameters

Tree-width

- is NP-hard to compute on graphs, but
- there is a linear-time FPT algorithm for it by Bodlaender (1996).

Branch-width

- is also NP-hard to compute on graphs, but

Algorithmic aspects of “width” parameters

Tree-width

- is NP-hard to compute on graphs, but
- there is a linear-time FPT algorithm for it by Bodlaender (1996).

Branch-width

- is also NP-hard to compute on graphs, but
- it is fully polynomial on planar graphs, by Seymour & Thomas (1994),

Algorithmic aspects of “width” parameters

Tree-width

- is NP-hard to compute on graphs, but
- there is a linear-time FPT algorithm for it by Bodlaender (1996).

Branch-width

- is also NP-hard to compute on graphs, but
- it is fully polynomial on planar graphs, by Seymour & Thomas (1994),
- and having a linear-time FPT algorithm, Bodlaender & Thilikos (1997).

Algorithmic aspects of “width” parameters

Tree-width

- is NP-hard to compute on graphs, but
- there is a linear-time FPT algorithm for it by Bodlaender (1996).

Branch-width

- is also NP-hard to compute on graphs, but
- it is fully polynomial on planar graphs, by Seymour & Thomas (1994),
- and having a linear-time FPT algorithm, Bodlaender & Thilikos (1997).

The **hardness results**
carry over to matroid branch-width / tree-width, and to graph rank-width.

The **FPT results** here
can be extended to matroids over finite fields, and to graph rank-width. . . .

2 Branch-width, Definition

A ground set E, with a **connectivity function** λ (arbitr. symm. submod.)
$\longrightarrow$ a **branch decomposition:**

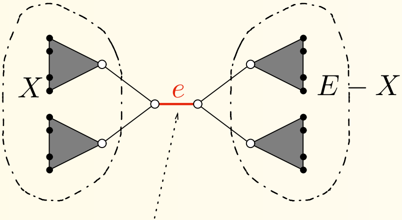
- E decomposed to a *sub-cubic tree* (degrees ≤ 3), and
- the elements mapped one-to-one to the tree leaves.

2 Branch-width, Definition

A ground set E, with a **connectivity function** λ (arbitr. symm. submod.)

$\longrightarrow$ a **branch decomposition:**

- E decomposed to a *sub-cubic tree* (degrees ≤ 3), and
- the elements mapped one-to-one to the tree leaves.
- The tree edges have *widths* as follows:



$$\underline{\text{width}(e) = \lambda(X)} = \lambda(E \setminus X),$$

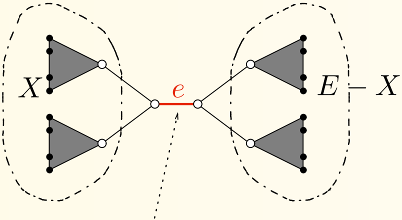
where X is *"displayed" by e* in the tree.

2 Branch-width, Definition

A ground set E, with a **connectivity function** λ (arbitr. symm. submod.)

$\longrightarrow$ a **branch decomposition:**

- E decomposed to a *sub-cubic tree* (degrees ≤ 3), and
- the elements mapped one-to-one to the tree leaves.
- The tree edges have *widths* as follows:



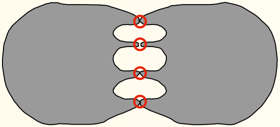
$$\underline{\text{width}(e) = \lambda(X)} = \lambda(E \setminus X),$$

where X is *"displayed" by* e in the tree.

Branch-width $\mathrm{bw}(\lambda)$ = min. of max. edge widths over all decompositions.

Branch-width variants, rank-width

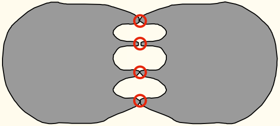
- *Graph branch-width*:



$E = E(G)$ and $\lambda(X) = \#$ of vertices "shared" by X and $E \setminus X$ in G.

Branch-width variants, rank-width

- *Graph branch-width*:



$E = E(G)$ and $\lambda(X) = \#$ of vertices "shared" by X and $E \setminus X$ in G.

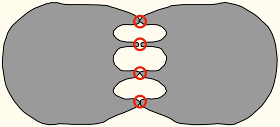
- *Matroid branch-width*:

$E = E(M)$ and $\lambda(X) = \mathrm{r}_M(X) + \mathrm{r}_M(E \setminus X) - \mathrm{r}(M) + 1$

(The "dimension" of the intersection of spans of X and $E \setminus X$ in M.)

Branch-width variants, rank-width

- *Graph branch-width*:



$E = E(G)$ and $\lambda(X) = \#$ of vertices "shared" by X and $E \setminus X$ in G.

- *Matroid branch-width*:

$E = E(M)$ and $\lambda(X) = \mathrm{r}_M(X) + \mathrm{r}_M(E \setminus X) - \mathrm{r}(M) + 1$

(The "dimension" of the intersection of spans of X and $E \setminus X$ in M.)

- *Graph rank-width*:

(as motivated by clique-width)

$$\boldsymbol{A}(G) \;\rightarrow\; \begin{array}{c|ccc} & & V(G)\setminus X & \\ \hline & 0 & 1 & 1 \\ X & 1 & 0 & 1 \\ & 0 & 1 & 0 \\ \hline \end{array}$$

$E = V(G)$ and $\lambda(X) = \mathrm{rank}(\, A(G)[X, E \setminus X]\,)$ over $GF(2)$.

Computing branch-width and rank-width

Theorem 1. (Bodlaender & Thilikos, 1997) *There is an $O(n)$-time FPT algorithm that, for each fixed k, either confirms that the branch-width of a given graph G is $> k$, or it finds a branch-decomposition of G of width $\leq \boldsymbol{k}$.*

Computing branch-width and rank-width

Theorem 1. (Bodlaender & Thilikos, 1997) *There is an $O(n)$-time FPT algorithm that, for each fixed k, either confirms that the branch-width of a given graph G is $> k$, or it finds a branch-decomposition of G of width $\leq \boldsymbol{k}$.*

Theorem 2. (PH, 2005) *There is an $O(n^3)$-time FPT algorithm that, for each fixed k, either confirms that the branch-width of a given matroid M represented over a finite field is $> k$, or it confirms that the branch-width is $\leq k$ and finds a branch-decomposition of M of width $\leq \boldsymbol{3k}$.*

Computing branch-width and rank-width

Theorem 1. (Bodlaender & Thilikos, 1997) *There is an $O(n)$-time FPT algorithm that, for each fixed k, either confirms that the branch-width of a given graph G is $> k$, or it finds a branch-decomposition of G of width $\leq \boldsymbol{k}$.*

Theorem 2. (PH, 2005) *There is an $O(n^3)$-time FPT algorithm that, for each fixed k, either confirms that the branch-width of a given matroid M represented over a finite field is $> k$, or it confirms that the branch-width is $\leq k$ and finds a branch-decomposition of M of width $\leq \boldsymbol{3k}$.*

Theorem 3. (Oum & Seymour / Oum & Courcelle, 2005/6) *There is an $O(n^3)$-time FPT algorithm that, for each fixed k, either confirms that the rank-width of a given graph G is $> k$, or it confirms that the rank-width is $\leq k$ and finds a rank-decomposition of G of width $\leq \boldsymbol{3k}$.*

Computing branch-width and rank-width

Theorem 1. (Bodlaender & Thilikos, 1997) *There is an $O(n)$-time FPT algorithm that, for each fixed k, either confirms that the branch-width of a given graph G is $> k$, or it finds a branch-decomposition of G of width $\leq \boldsymbol{k}$.*

Theorem 2. (PH, 2005) *There is an $O(n^3)$-time FPT algorithm that, for each fixed k, either confirms that the branch-width of a given matroid M represented over a finite field is $> k$, or it confirms that the branch-width is $\leq k$ and finds a branch-decomposition of M of width $\leq \boldsymbol{3k}$.*

Theorem 3. (Oum & Seymour / Oum & Courcelle, 2005/6) *There is an $O(n^3)$-time FPT algorithm that, for each fixed k, either confirms that the rank-width of a given graph G is $> k$, or it confirms that the rank-width is $\leq k$ and finds a rank-decomposition of G of width $\leq \boldsymbol{3k}$.*

. . . . **new** . . . .

Theorem 4. (PH & Oum, 2007) *There is an $O(n^3)$-time FPT algorithm that, for each fixed k, either confirms that the branch-width of a given matroid M over a finite field (rank-width of a given graph G) is $> k$, or it finds a branch-decomposition of M (a rank-decomposition of G) of width $\leq \boldsymbol{k}$.*

3 Width (a number) → Decomposition

Motivation: There are occasions when getting the branch-width as a number is significantly easier than getting a corresponding decomposition, such as, using "forbiden minor" characterizations.

3 Width (a number) → Decomposition

Motivation: There are occasions when getting the branch-width as a number is significantly easier than getting a corresponding decomposition, such as, using "forbiden minor" characterizations.

Our task: Knowing how to determine the exact branch-width, we want to construct an optimal decomposition for it.

3 Width (a number) → Decomposition

Motivation: There are occasions when getting the branch-width as a number is significantly easier than getting a corresponding decomposition, such as, using "forbiden minor" characterizations.

Our task: Knowing how to determine the exact branch-width, we want to construct an optimal decomposition for it.

Instances to be solved...

- Matroids (over finite fields) — there is a computable finite set of forbidden minors for the matroids of branch-width $\leq k$, and we have a decomposition of width $\leq 3k$ from Theorem 2.

  Hence the branch-width as a number can be determined efficiently, but an optimal decomposition does not follow.

3 Width (a number) → Decomposition

Motivation: There are occasions when getting the branch-width as a number is significantly easier than getting a corresponding decomposition, such as, using "forbiden minor" characterizations.

Our task: Knowing how to determine the exact branch-width, we want to construct an optimal decomposition for it.

Instances to be solved...

- Matroids (over finite fields) — there is a computable finite set of forbidden minors for the matroids of branch-width $\leq k$, and we have a decomposition of width $\leq 3k$ from Theorem 2.

  Hence the branch-width as a number can be determined efficiently, but an optimal decomposition does not follow.

- Same with graph rank-width — there is a computable finite set of forbidden vertex-minors for the graphs of rank-width $\leq k$, and we have a decomposition of width $\leq 3k$ from Theorem 3.

  How can we get an optimal decomposition?

An idea motivated by Geelen [private communication]...

1. Take an arbitrary partition $\mathcal{P}$ of our E, and extend connectivity λ to $\lambda^{\mathcal{P}}$ on $\mathcal{P}$ naturally. Assume the branch-width of $\lambda^{\mathcal{P}}$ is efficiently computable.

An idea motivated by Geelen [private communication]...

1. Take an arbitrary partition $\mathcal{P}$ of our E, and extend connectivity λ to $\lambda^{\mathcal{P}}$ on $\mathcal{P}$ naturally. Assume the branch-width of $\lambda^{\mathcal{P}}$ is efficiently computable.

2. If $|\mathcal{P}| \leq 2$, then we have a decomposition. Done.

An idea motivated by Geelen [private communication]...

1. Take an arbitrary partition $\mathcal{P}$ of our E, and extend connectivity λ to $\lambda^{\mathcal{P}}$ on $\mathcal{P}$ naturally. Assume the branch-width of $\lambda^{\mathcal{P}}$ is efficiently computable.

2. If $|\mathcal{P}| \leq 2$, then we have a decomposition. Done.

3. Ranging over all pairs $P_1, P_2 \in \mathcal{P}$, and $\mathcal{P}' = \mathcal{P} \setminus \{P_1, P_2\} \cup \{P_1 \cup P_2\}$; whenever $\mathrm{bw}(\lambda^{\mathcal{P}'}) \leq \mathrm{bw}(\lambda^{\mathcal{P}})$ happens, go to the next step.

An idea motivated by Geelen [private communication]...

1. Take an arbitrary partition $\mathcal{P}$ of our E, and extend connectivity λ to $\lambda^{\mathcal{P}}$ on $\mathcal{P}$ naturally. Assume the branch-width of $\lambda^{\mathcal{P}}$ is efficiently computable.

2. If $|\mathcal{P}| \leq 2$, then we have a decomposition. Done.

3. Ranging over all pairs $P_1, P_2 \in \mathcal{P}$, and $\mathcal{P}' = \mathcal{P} \setminus \{P_1, P_2\} \cup \{P_1 \cup P_2\}$; whenever $\mathrm{bw}(\lambda^{\mathcal{P}'}) \leq \mathrm{bw}(\lambda^{\mathcal{P}})$ happens, go to the next step.

4. Recursively obtain a branch-decomposition of $\mathcal{P}'$. Split the leaf of $P_1 \cup P_2$ into two new leaves labeled P_1 and P_2. Return.

An idea motivated by Geelen [private communication]...

1. Take an arbitrary partition $\mathcal{P}$ of our E, and extend connectivity λ to $\lambda^{\mathcal{P}}$ on $\mathcal{P}$ naturally. Assume the branch-width of $\lambda^{\mathcal{P}}$ is efficiently computable.
2. If $|\mathcal{P}| \leq 2$, then we have a decomposition. Done.
3. Ranging over all pairs $P_1, P_2 \in \mathcal{P}$, and $\mathcal{P}' = \mathcal{P} \setminus \{P_1, P_2\} \cup \{P_1 \cup P_2\}$; whenever $\mathrm{bw}(\lambda^{\mathcal{P}'}) \leq \mathrm{bw}(\lambda^{\mathcal{P}})$ happens, go to the next step.
4. Recursively obtain a branch-decomposition of $\mathcal{P}'$. Split the leaf of $P_1 \cup P_2$ into two new leaves labeled P_1 and P_2. Return.

This scheme leads to $O(n^3)$ calls to $\lambda^{\mathcal{P}}$-queries,

An idea motivated by Geelen [private communication]...

1. Take an arbitrary partition $\mathcal{P}$ of our E, and extend connectivity λ to $\lambda^{\mathcal{P}}$ on $\mathcal{P}$ naturally. Assume the branch-width of $\lambda^{\mathcal{P}}$ is efficiently computable.

2. If $|\mathcal{P}| \leq 2$, then we have a decomposition. Done.

3. Ranging over all pairs $P_1, P_2 \in \mathcal{P}$, and $\mathcal{P}' = \mathcal{P} \setminus \{P_1, P_2\} \cup \{P_1 \cup P_2\}$; whenever $\mathrm{bw}(\lambda^{\mathcal{P}'}) \leq \mathrm{bw}(\lambda^{\mathcal{P}})$ happens, go to the next step.

4. Recursively obtain a branch-decomposition of $\mathcal{P}'$. Split the leaf of $P_1 \cup P_2$ into two new leaves labeled P_1 and P_2. Return.

This scheme leads to $O(n^3)$ calls to $\lambda^{\mathcal{P}}$-queries, but we manage to speed it up to just $O(n^2)$ queries:

- At the top level of recursion, process not only the first admissible pair P_1, P_2 in step 3, but all such pairwise disjoint pairs.

An idea motivated by Geelen [private communication]. . .

1. Take an arbitrary partition $\mathcal{P}$ of our E, and extend connectivity λ to $\lambda^{\mathcal{P}}$ on $\mathcal{P}$ naturally. Assume the branch-width of $\lambda^{\mathcal{P}}$ is efficiently computable.

2. If $|\mathcal{P}| \leq 2$, then we have a decomposition. Done.

3. Ranging over all pairs $P_1, P_2 \in \mathcal{P}$, and $\mathcal{P}' = \mathcal{P} \setminus \{P_1, P_2\} \cup \{P_1 \cup P_2\}$; whenever $\mathrm{bw}(\lambda^{\mathcal{P}'}) \leq \mathrm{bw}(\lambda^{\mathcal{P}})$ happens, go to the next step.

4. Recursively obtain a branch-decomposition of $\mathcal{P}'$. Split the leaf of $P_1 \cup P_2$ into two new leaves labeled P_1 and P_2. Return.

This scheme leads to $O(n^3)$ calls to $\lambda^{\mathcal{P}}$-queries, but we manage to speed it up to just $O(n^2)$ queries:

- At the top level of recursion, process not only the first admissible pair P_1, P_2 in step 3, but all such pairwise disjoint pairs.

- At deeper levels, process only such pairs that P_1 of it has been processed one level up.

Gadgets for partitioned matroids

True standing: We do not know how to devise forbidden minor characterizations for the branch-width of partitioned matroids,

Gadgets for partitioned matroids

True standing: We do not know how to devise forbidden minor characterizations for the branch-width of partitioned matroids, but we can replace the parts of $\mathcal{P}$ by appropriate "unbreakable" (*titanic*) gadgets.

Titanic gadget – for $P \in \mathcal{P}$ and $\ell = \lambda(P)$,

we replace $P \subseteq E$ in M with a copy of the *uniform matroid* $U_{\ell-1,3\ell-5}$.

(To "replace" means to use the proper *matroid amalgam*.)

Gadgets for partitioned matroids

True standing: We do not know how to devise forbidden minor characterizations for the branch-width of partitioned matroids, but we can replace the parts of $\mathcal{P}$ by appropriate "unbreakable" (*titanic*) gadgets.

Titanic gadget – for $P \in \mathcal{P}$ and $\ell = \lambda(P)$,

we replace $P \subseteq E$ in M with a copy of the *uniform matroid* $U_{\ell-1,3\ell-5}$.

(To "replace" means to use the proper *matroid amalgam*.)

Lemma 5. *If $\mathrm{bw}(\lambda) \leq k$, and $\mathcal{P}$ is a titanic partition of width $\leq k$ (i.e. each part $P \in \mathcal{P}$ is titanic of $\lambda(P) \leq k$);*

then the branch-width of $\lambda^{\mathcal{P}}$ is $\leq k$.

Gadgets for partitioned matroids

True standing: We do not know how to devise forbidden minor characterizations for the branch-width of partitioned matroids, but we can replace the parts of $\mathcal{P}$ by appropriate "unbreakable" (*titanic*) gadgets.

Titanic gadget – for $P \in \mathcal{P}$ and $\ell = \lambda(P)$,

we replace $P \subseteq E$ in M with a copy of the *uniform matroid* $U_{\ell-1,3\ell-5}$.

(To "replace" means to use the proper *matroid amalgam*.)

Lemma 5. *If* $\mathrm{bw}(\lambda) \leq k$*, and* $\mathcal{P}$ *is a titanic partition of width* $\leq k$ *(i.e. each part* $P \in \mathcal{P}$ *is titanic of* $\lambda(P) \leq k$*);*

then the branch-width of $\lambda^{\mathcal{P}}$ *is* $\leq k$*.*

Starting with a partitioned matroid $M, \mathcal{P}$, we arrive at *normalized* matroid $M^{\#}$.

($M^{\#}$ may require a slightly larger field to be represented over.)

Theorem 6. *The branch-width of* $\lambda^{\mathcal{P}}$ *on* M *is equal to* $\mathrm{bw}(M^{\#})$*.*

4 Our new Algorithm, a Sketch

We want to "squeeze" the alg. of Section 3 inside the old matroid bw. algorithm…

- Start with a $3k$-decomposition of $M^{\#}$ (Theorem 2), otherwise NO.

4 Our new Algorithm, a Sketch

We want to “squeeze” the alg. of Section 3 inside the old matroid bw. algorithm…

- Start with a $3k$-decomposition of $M^{\#}$ (Theorem 2), otherwise NO.
- Now, “merging” parts $P_1, P_2 \in \mathcal{P}$ means adding a new titanic gadget, which can be done linearly while increasing the decomp. width by $< \ell$.

4 Our new Algorithm, a Sketch

We want to "squeeze" the alg. of Section 3 inside the old matroid bw. algorithm...

- Start with a $3k$-decomposition of $M^{\#}$ (Theorem 2), otherwise NO.
- Now, "merging" parts $P_1, P_2 \in \mathcal{P}$ means adding a new titanic gadget, which can be done linearly while increasing the decomp. width by $< \ell$.
- Testing bw. of the merged partition then means checking the appr. forbidden minors, which is again linear using the current decomposition.

4 Our new Algorithm, a Sketch

We want to “squeeze” the alg. of Section 3 inside the old matroid bw. algorithm…

- Start with a $3k$-decomposition of $M^{\#}$ (Theorem 2), otherwise NO.
- Now, “merging” parts $P_1, P_2 \in \mathcal{P}$ means adding a new titanic gadget, which can be done linearly while increasing the decomp. width by $< \ell$.
- Testing bw. of the merged partition then means checking the appr. forbidden minors, which is again linear using the current decomposition.
- Whenever a “mergeable” pair $P_1, P_2 \in \mathcal{P}$ is found, we must update our decomposition, down to width $\leq 3k$ again.

  This can be done in quadratic time, cf. the proof of Theorem 2.

4 Our new Algorithm, a Sketch

We want to "squeeze" the alg. of Section 3 inside the old matroid bw. algorithm...

- Start with a $3k$-decomposition of $M^{\#}$ (Theorem 2), otherwise NO.
- Now, "merging" parts $P_1, P_2 \in \mathcal{P}$ means adding a new titanic gadget, which can be done linearly while increasing the decomp. width by $< \ell$.
- Testing bw. of the merged partition then means checking the appr. forbidden minors, which is again linear using the current decomposition.
- Whenever a "mergeable" pair $P_1, P_2 \in \mathcal{P}$ is found, we must update our decomposition, down to width $\leq 3k$ again.

  This can be done in quadratic time, cf. the proof of Theorem 2.

Altogether, we really get $n^2 \times O(n) + n \times O(n^2) = O(n^3)$ time!

5 Application to Rank-width

How do we translate our matroid-based results to graph rank-width?

5 Application to Rank-width

How do we translate our matroid-based results to graph rank-width?

- bipartite G, $V(G) = U \cup W$:

$$\begin{array}{c|ccc|c} & & W & & \\ \hline & 0 & 1 & 1 & \\ U & 1 & 0 & 1 & \boldsymbol{I}_U \\ & 0 & 1 & 0 & \\ \hline \end{array}$$

Fact. The rank-width of a bipartite graph equals the branch-width of the binary matroid represented by its *bipartite adjacency matrix*.

5 Application to Rank-width

How do we translate our matroid-based results to graph rank-width?

- bipartite G, $V(G) = U \cup W$:

$$\begin{array}{c|ccc|c} & & W & & \\ \hline & 0 & 1 & 1 & \\ U & 1 & 0 & 1 & \boldsymbol{I}_U \\ & 0 & 1 & 0 & \\ \hline \end{array}$$

Fact. The rank-width of a bipartite graph equals the branch-width of the binary matroid represented by its *bipartite adjacency matrix*.

- What to do for non-bipartite graphs?

5 Application to Rank-width

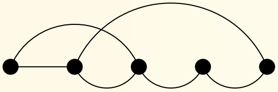
How do we translate our matroid-based results to graph rank-width?

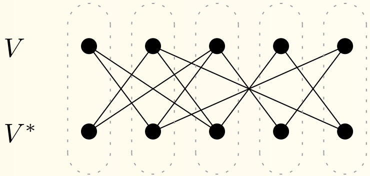
- bipartite G, $V(G) = U \cup W$:

$$
\begin{array}{c|ccc|c}
 & & W & & \\
\hline
 & 0 & 1 & 1 & \\
U & 1 & 0 & 1 & \boldsymbol{I}_U \\
 & 0 & 1 & 0 & \\
\hline
\end{array}
$$

Fact. The rank-width of a bipartite graph equals the branch-width of the binary matroid represented by its *bipartite adjacency matrix*.

- What to do for non-bipartite graphs?





Fact. We change every graph G to the associated bipartite graph with its canonical vertex partition. The value of rank-width exactly doubles.