How Graph Structure Makes Hard Problems Tractable

Tuukka Korhonen

1 April 2026

CS Special Seminar

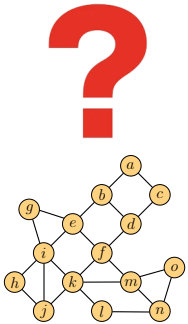
Role of Structure in Algorithms

- There are really hard computational problems (NP-hard)
- Yet we solve these problems routinely in practice

Role of Structure in Algorithms

- There are really hard computational problems (NP-hard)
- Yet we solve these problems routinely in practice

- Hardness is about the worst case
- Real-life instances contain structure that can be exploited

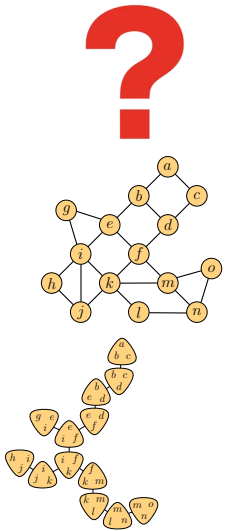


Role of Structure in Algorithms

- There are really hard computational problems (NP-hard)
- Yet we solve these problems routinely in practice

- Hardness is about the worst case
- Real-life instances contain structure that can be exploited

My research: What can theory say about this?



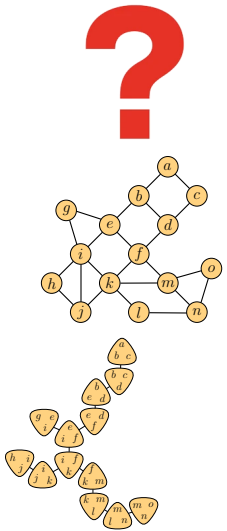
Role of Structure in Algorithms

- There are really hard computational problems (NP-hard)
- Yet we solve these problems routinely in practice

- Hardness is about the worst case
- Real-life instances contain structure that can be exploited

My research: What can theory say about this?

1. Drawing the line between easy and hard instances using graph-theoretic methods
2. Designing algorithms to discover the graph-theoretic structure that can be exploited



Outline of the Talk

⇒ Background and motivation

– My contributions

– Future research

Example: Constraint Satisfaction Problems (CSPs)

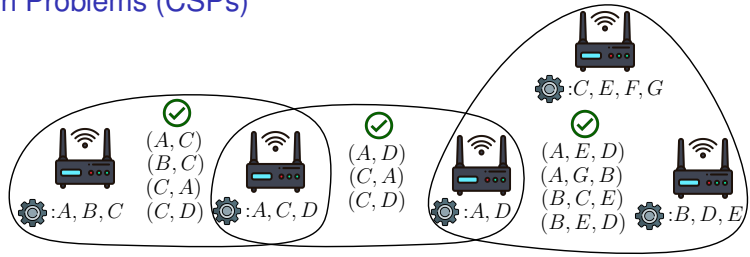
Example: Constraint Satisfaction Problems (CSPs)

Wifi setting assignment:

Variables: Transmitters

Domains: Settings of transmitters

Constraints: Compatible settings



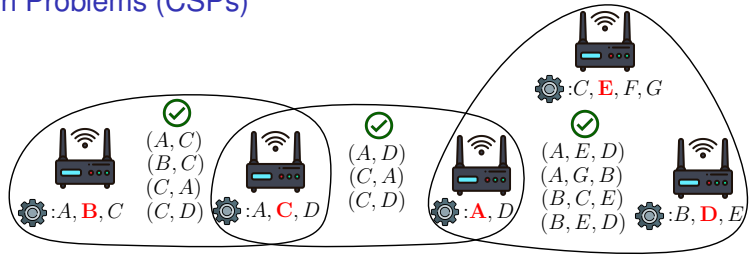
Example: Constraint Satisfaction Problems (CSPs)

Wifi setting assignment:

Variables: Transmitters

Domains: Settings of transmitters

Constraints: Compatible settings



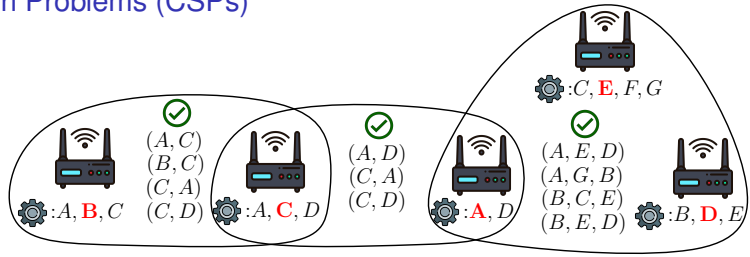
Example: Constraint Satisfaction Problems (CSPs)

Wifi setting assignment:

Variables: Transmitters

Domains: Settings of transmitters

Constraints: Compatible settings

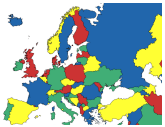


Graph coloring:

Variables: Vertices

Domains: Possible colors

Constraints: Endpoints of each edge get different colors



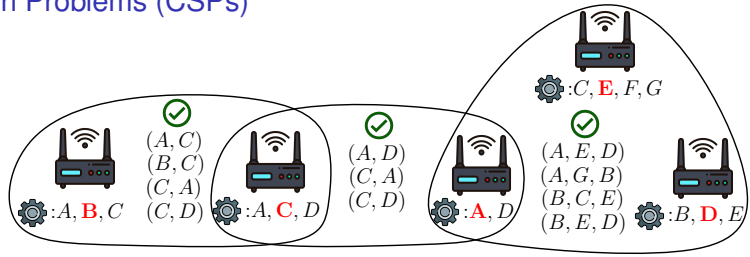
Example: Constraint Satisfaction Problems (CSPs)

Wifi setting assignment:

Variables: Transmitters

Domains: Settings of transmitters

Constraints: Compatible settings

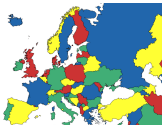


Graph coloring:

Variables: Vertices

Domains: Possible colors

Constraints: Endpoints of each edge get different colors

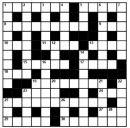


Crosswords:

Variables: Squares

Domains: Letters

Constraints: Letters must form words in the dictionary



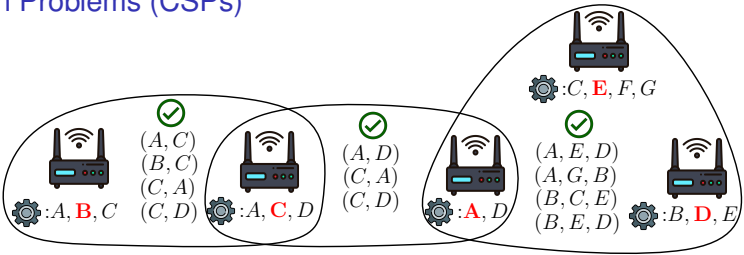
Example: Constraint Satisfaction Problems (CSPs)

Wifi setting assignment:

Variables: Transmitters

Domains: Settings of transmitters

Constraints: Compatible settings

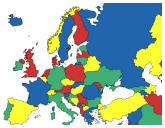


Graph coloring:

Variables: Vertices

Domains: Possible colors

Constraints: Endpoints of each edge get different colors

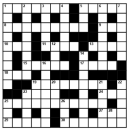


Crosswords:

Variables: Squares

Domains: Letters

Constraints: Letters must form words in the dictionary



Join evaluation in databases:

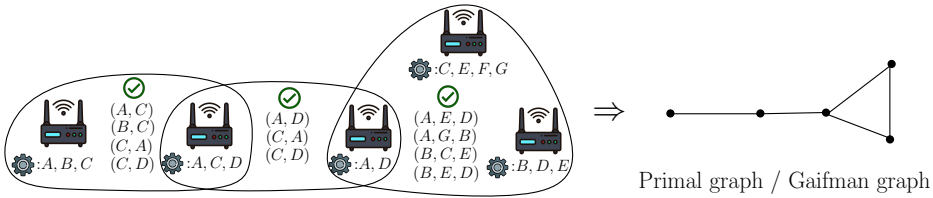
Variables: Attributes (columns)

Domains: Values of attributes

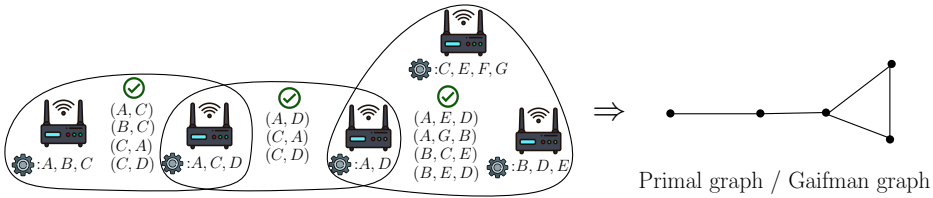
Constraints: Tables of the database

Solving CSP Using Graph Structure

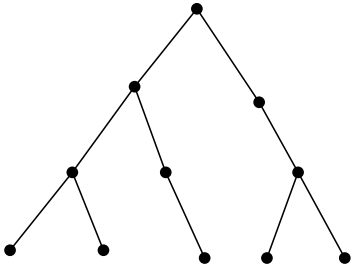
Solving CSP Using Graph Structure



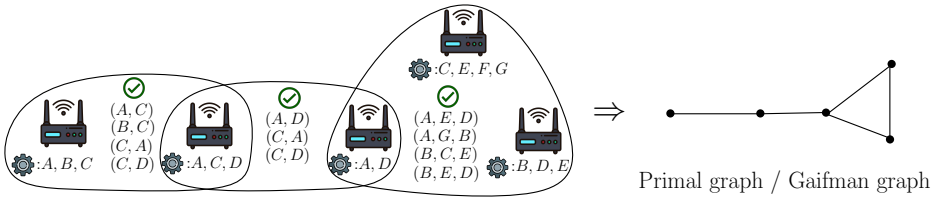
Solving CSP Using Graph Structure



Warmup: What if the primal graph is a tree?

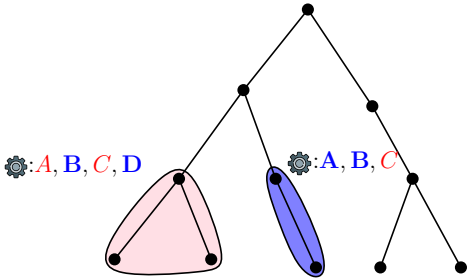


Solving CSP Using Graph Structure

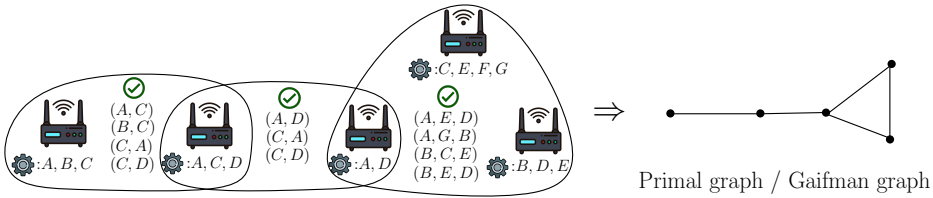


Warmup: What if the primal graph is a tree?

Bottom-up dynamic programming

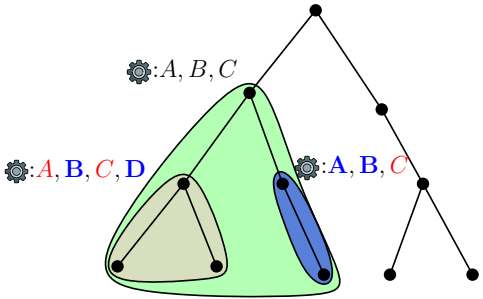


Solving CSP Using Graph Structure

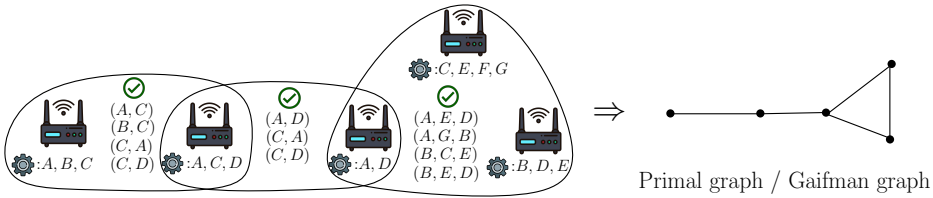


Warmup: What if the primal graph is a tree?

Bottom-up dynamic programming



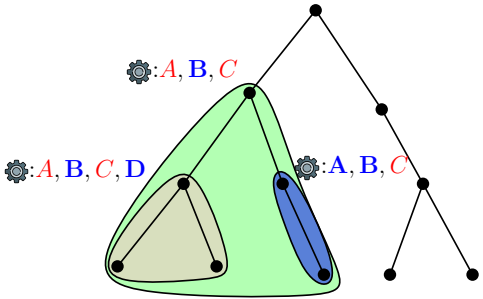
Solving CSP Using Graph Structure



Warmup: What if the primal graph is a tree?

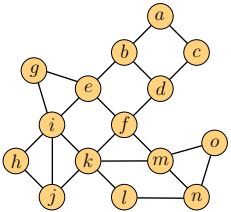
Bottom-up dynamic programming

⇒ Linear-time algorithm



Treewidth

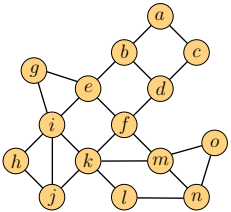
What if the primal graph is not a tree?



Treewidth

What if the primal graph is not a tree?

The graph-theoretic notion of **Treewidth**

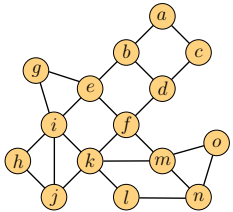


Treewidth

What if the primal graph is not a tree?

The graph-theoretic notion of **Treewidth**

- Trees have treewidth 1
- The example graphs have treewidth 2
- Cliques have maximum treewidth, $n - 1$



Treewidth

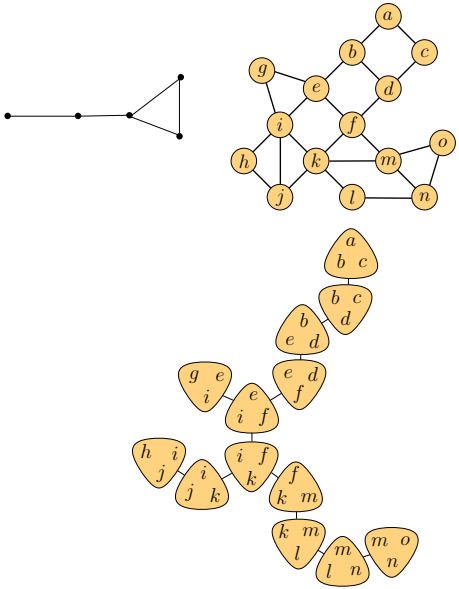
What if the primal graph is not a tree?

The graph-theoretic notion of **Treewidth**

- Trees have treewidth 1
- The example graphs have treewidth 2
- Cliques have maximum treewidth, $n - 1$

Treewidth of a graph: Minimum width of tree decomposition

Width of a tree decomposition: Maximum bag size -1



Treewidth

What if the primal graph is not a tree?

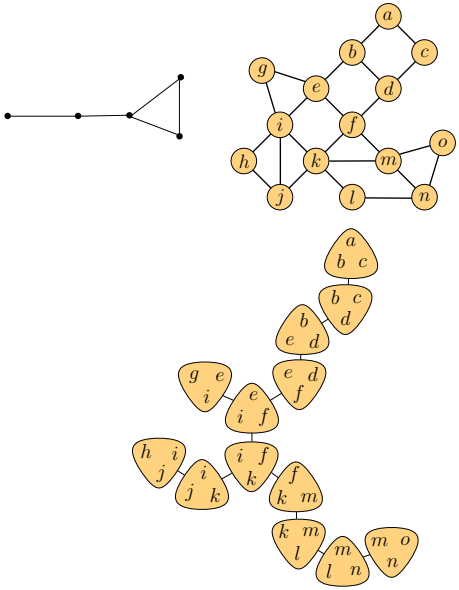
The graph-theoretic notion of **Treewidth**

- Trees have treewidth 1
- The example graphs have treewidth 2
- Cliques have maximum treewidth, $n - 1$

Treewidth of a graph: Minimum width of tree decomposition

Width of a tree decomposition: Maximum bag size -1

Dynamic programming on tree decomposition to solve CSP



Treewidth

What if the primal graph is not a tree?

The graph-theoretic notion of **Treewidth**

- Trees have treewidth 1
- The example graphs have treewidth 2
- Cliques have maximum treewidth, $n - 1$

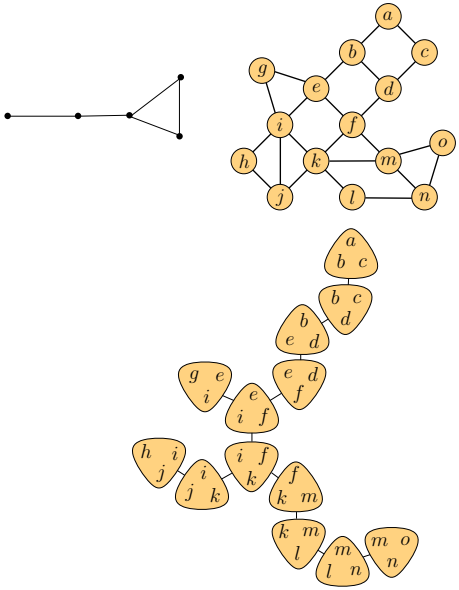
Treewidth of a graph: Minimum width of tree decomposition

Width of a tree decomposition: Maximum bag size -1

Dynamic programming on tree decomposition to solve CSP

$\Rightarrow$ Running time $\mathcal{O}(D^{k+1} \cdot n)$

- n: Number of variables
- D: Domain size
- k: Treewidth



Can You Beat Treewidth?

- We can solve CSP in time $\mathcal{O}(D^{k+1} \cdot n)$, where
  - n: Number of variables
  - D: Domain size
  - k: Treewidth
- Could there be another graph parameter that works better than treewidth?

Can You Beat Treewidth?

- We can solve CSP in time $\mathcal{O}(D^{k+1} \cdot n)$, where
  - n: Number of variables
  - D: Domain size
  - k: Treewidth
- Could there be another graph parameter that works better than treewidth?

[Grohe, Schwentick & Segoufin, STOC'01]

&

[Marx, FOCS'07]

: No (under certain assumptions)

Theorem (informal)

For **every** graph G of treewidth $\mathrm{tw}(G) = k$, the factor D^k in the running time is unavoidable for solving CSPs with primal graph G, assuming the Exponential Time Hypothesis.

Can You Beat Treewidth?

- We can solve CSP in time $\mathcal{O}(D^{k+1} \cdot n)$, where
  - n: Number of variables
  - D: Domain size
  - k: Treewidth
- Could there be another graph parameter that works better than treewidth?

[Grohe, Schwentick & Segoufin, STOC'01]

&

[Marx, FOCS'07]

: No (under certain assumptions)

Theorem (informal)

For **every** graph G of treewidth $\mathrm{tw}(G) = k$, the factor D^k in the running time is unavoidable for solving CSPs with primal graph G, assuming the Exponential Time Hypothesis.

Instance-optimal over the primal graphs

Treewidth: History and Practical Applications

Treewidth: History and Practical Applications

Treewidth was (re-)invented by:

- Bertele & Brioschi for dynamic programming, 1972
- Robertson & Seymour for theory of graph minors, 1983
- Baker for approximation algorithms, 1983
- Dechter & Pearl for constraint satisfaction, 1985
- Courcelle for logic, 1988

...

Treewidth: History and Practical Applications

Treewidth was (re-)invented by:

- Bertele & Brioschi for dynamic programming, 1972
- Robertson & Seymour for theory of graph minors, 1983
- Baker for approximation algorithms, 1983
- Dechter & Pearl for constraint satisfaction, 1985
- Courcelle for logic, 1988

...

Modern practical applications include:

Treewidth: History and Practical Applications

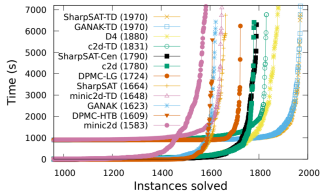
Treewidth was (re-)invented by:

- Bertele & Brioschi for dynamic programming, 1972
- Robertson & Seymour for theory of graph minors, 1983
- Baker for approximation algorithms, 1983
- Dechter & Pearl for constraint satisfaction, 1985
- Courcelle for logic, 1988

...

Modern practical applications include:

- #SAT (counting the number of solutions for SAT)



Treewidth: History and Practical Applications

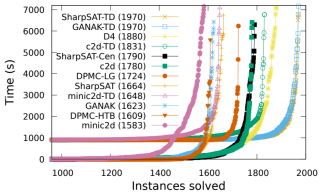
Treewidth was (re-)invented by:

- Bertele & Brioschi for dynamic programming, 1972
- Robertson & Seymour for theory of graph minors, 1983
- Baker for approximation algorithms, 1983
- Dechter & Pearl for constraint satisfaction, 1985
- Courcelle for logic, 1988

...

Modern practical applications include:

- #SAT (counting the number of solutions for SAT)
- Quantum computer simulation (tensor network contraction)



Treewidth: History and Practical Applications

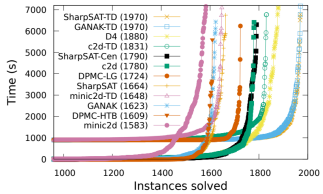
Treewidth was (re-)invented by:

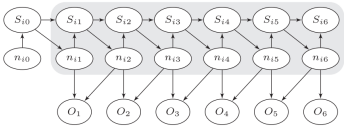
- Bertele & Brioschi for dynamic programming, 1972
- Robertson & Seymour for theory of graph minors, 1983
- Baker for approximation algorithms, 1983
- Dechter & Pearl for constraint satisfaction, 1985
- Courcelle for logic, 1988

...

Modern practical applications include:

- #SAT (counting the number of solutions for SAT)
- Quantum computer simulation (tensor network contraction)
- Probabilistic inference (junction tree algorithm)





Treewidth: History and Practical Applications

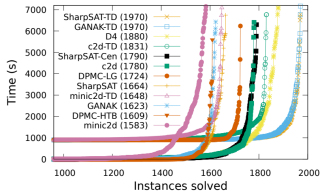
Treewidth was (re-)invented by:

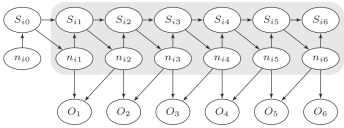
- Bertele & Brioschi for dynamic programming, 1972
- Robertson & Seymour for theory of graph minors, 1983
- Baker for approximation algorithms, 1983
- Dechter & Pearl for constraint satisfaction, 1985
- Courcelle for logic, 1988

...

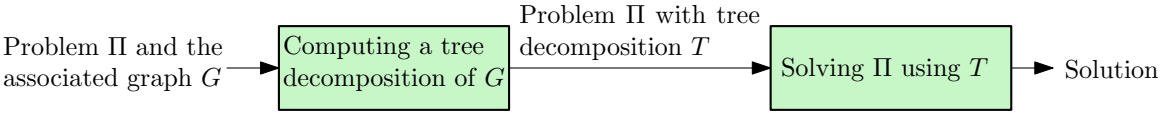
Modern practical applications include:

- #SAT (counting the number of solutions for SAT)
- Quantum computer simulation (tensor network contraction)
- Probabilistic inference (junction tree algorithm)
- Join evaluation in databases (fractional hypertreewidth, submodular width)

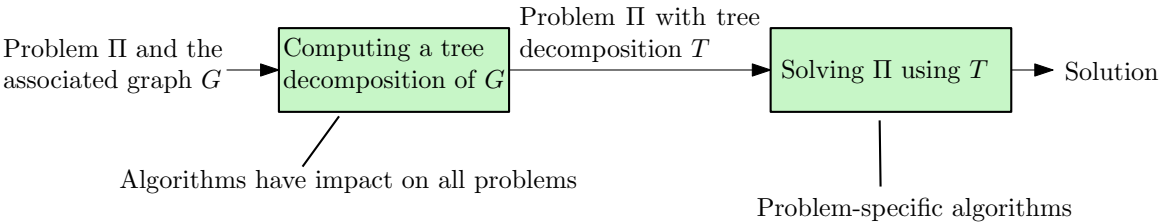




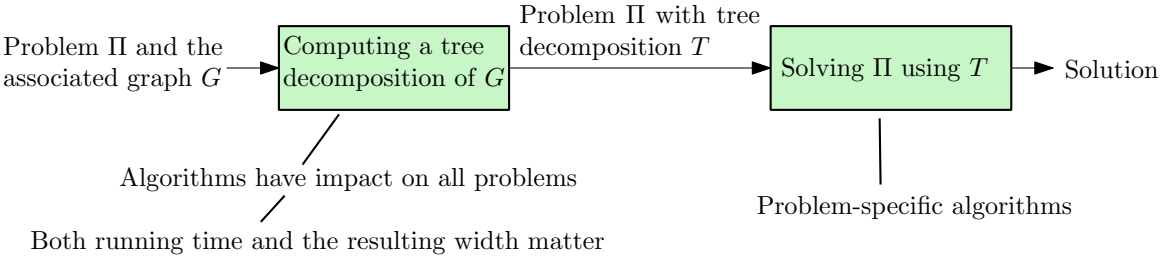
Computing Treewidth



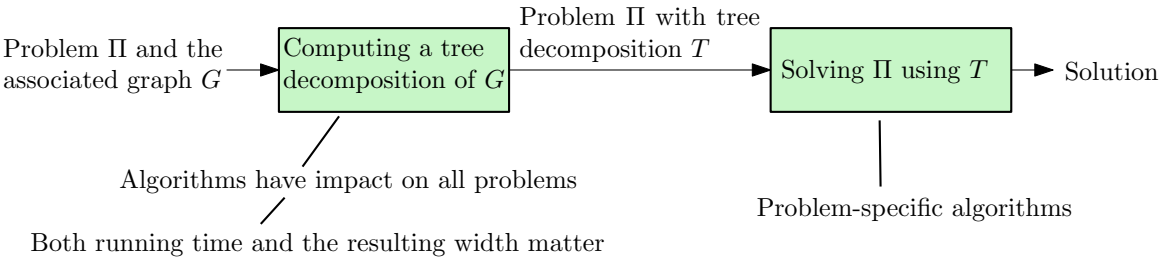
Computing Treewidth



Computing Treewidth

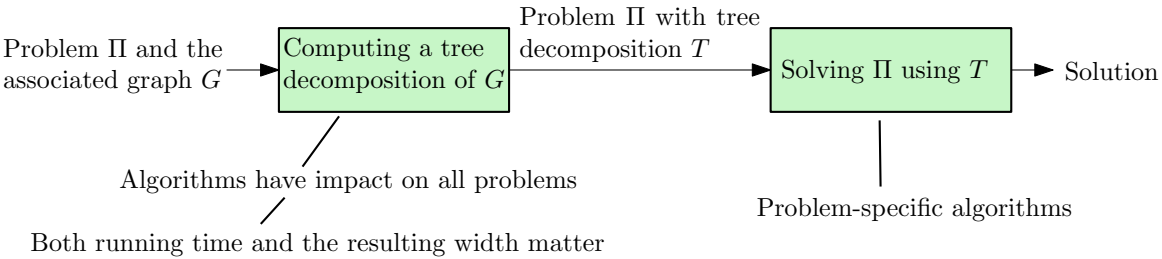


Computing Treewidth



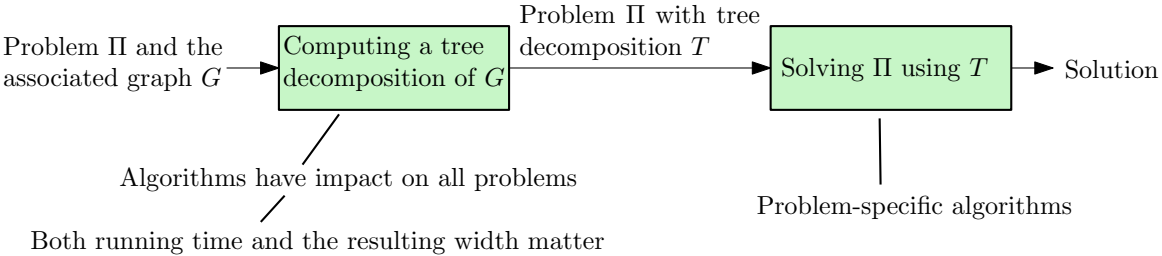
- Computing minimum-width tree decomposition is NP-hard

Computing Treewidth



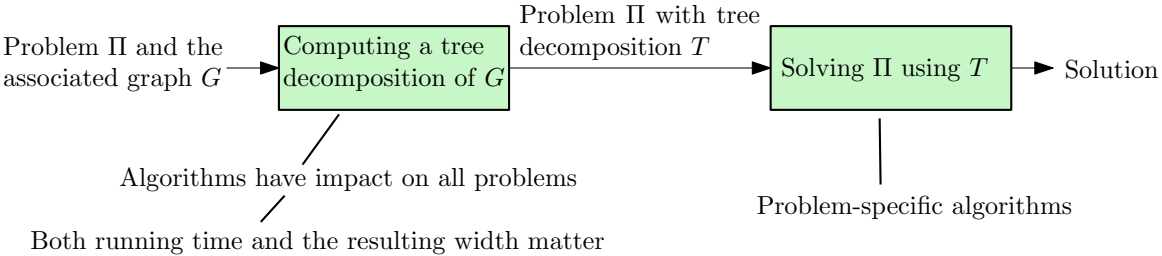
- Computing minimum-width tree decomposition is NP-hard
- Constant-approximation conjectured to be hard

Computing Treewidth



- Computing minimum-width tree decomposition is NP-hard
- Constant-approximation conjectured to be hard
- Solution: Efficient algorithms when treewidth k is small

Computing Treewidth



- Computing minimum-width tree decomposition is NP-hard
- Constant-approximation conjectured to be hard
- Solution: Efficient algorithms when treewidth k is small
- [Robertson & Seymour, 1987]: Treewidth can be 4-approximated in time $\mathcal{O}(3^{3k}n^2)$

Outline of the Talk

– Background and motivation

⇒ My contributions

– Future research

Algorithms for Computing Treewidth

Ref.	Appx. ratio	Run. time
[RS'87]	4	$3^{3k} n^2$
[ACP'87]	exact	n^{k+1}
[RS'87]	exact	$f(k) \cdot n^2$
[Lag'90]	8	$k^{\mathcal{O}(k)} n \log^2 n$
[MT'91]	6	$k^{\mathcal{O}(k)} n \log^2 n$
[Ree'92]	8	$k^{\mathcal{O}(k)} n \log n$
[Bod'93]	exact	$2^{\mathcal{O}(k^3)} n$
[BGHK'95]	$\mathcal{O}(\log n)$	$\text{poly}(n)$
[Ami'01]	$\mathcal{O}(\log k)$	$\text{poly}(n)$

Ref.	Appx. ratio	Run. time
[FHL'05]	$\mathcal{O}(\sqrt{\log k})$	$\text{poly}(n)$
[BDDFLP'13]	3	$2^{\mathcal{O}(k)} n \log n$
[BDDFLP'13]	5	$2^{\mathcal{O}(k)} n$
[FTV'14]	exact	1.7347^n
[FLSPW'17]	$\mathcal{O}(k)$	$k^{\mathcal{O}(1)} n \log n$
[BF'21]	5	$2^{7k} n \log n$
[DY'24]	$\mathcal{O}(\log n)$	$k^{\mathcal{O}(1)} n \log^{\mathcal{O}(1)} n$

Algorithms for Computing Treewidth

Ref.	Appx. ratio	Run. time
[RS'87]	4	$3^{3k}n^2$
[ACP'87]	exact	n^{k+1}
[RS'87]	exact	$f(k)\cdot n^2$
[Lag'90]	8	$k^{\mathcal{O}(k)}n\log^2 n$
[MT'91]	6	$k^{\mathcal{O}(k)}n\log^2 n$
[Ree'92]	8	$k^{\mathcal{O}(k)}n\log n$
[Bod'93]	exact	$2^{\mathcal{O}(k^3)}n$
[BGHK'95]	$\mathcal{O}(\log n)$	$\mathrm{poly}(n)$
[Ami'01]	$\mathcal{O}(\log k)$	$\mathrm{poly}(n)$

Ref.	Appx. ratio	Run. time
[FHL'05]	$\mathcal{O}(\sqrt{\log k})$	$\mathrm{poly}(n)$
[BDDFLP'13]	3	$2^{\mathcal{O}(k)}n\log n$
[BDDFLP'13]	5	$2^{\mathcal{O}(k)}n$
[FTV'14]	exact	1.7347^n
[FLSPW'17]	$\mathcal{O}(k)$	$k^{\mathcal{O}(1)}n\log n$
[BF'21]	5	$2^{7k}n\log n$
[DY'24]	$\mathcal{O}(\log n)$	$k^{\mathcal{O}(1)}n\log^{\mathcal{O}(1)} n$
[Kor'21]	2	$2^{\mathcal{O}(k)}n$

- [Korhonen, FOCS'21]: New "local improvement" technique for approximating treewidth

Algorithms for Computing Treewidth

Ref.	Appx. ratio	Run. time
[RS'87]	4	$3^{3k} n^2$
[ACP'87]	exact	n^{k+1}
[RS'87]	exact	$f(k) \cdot n^2$
[Lag'90]	8	$k^{\mathcal{O}(k)} n \log^2 n$
[MT'91]	6	$k^{\mathcal{O}(k)} n \log^2 n$
[Ree'92]	8	$k^{\mathcal{O}(k)} n \log n$
[Bod'93]	exact	$2^{\mathcal{O}(k^3)} n$
[BGHK'95]	$\mathcal{O}(\log n)$	$\text{poly}(n)$
[Ami'01]	$\mathcal{O}(\log k)$	$\text{poly}(n)$

Ref.	Appx. ratio	Run. time
[FHL'05]	$\mathcal{O}(\sqrt{\log k})$	$\text{poly}(n)$
[BDDFLP'13]	3	$2^{\mathcal{O}(k)} n \log n$
[BDDFLP'13]	5	$2^{\mathcal{O}(k)} n$
[FTV'14]	exact	1.7347^n
[FLSPW'17]	$\mathcal{O}(k)$	$k^{\mathcal{O}(1)} n \log n$
[BF'21]	5	$2^{7k} n \log n$
[DY'24]	$\mathcal{O}(\log n)$	$k^{\mathcal{O}(1)} n \log^{\mathcal{O}(1)} n$
[Kor'21]	2	$2^{\mathcal{O}(k)} n$

- [Korhonen, FOCS'21]: New "local improvement" technique for approximating treewidth
  - Improves approximation ratio below 3, breaking the RS-barrier (open problem asked by [BDDFLP'13])
  - Achieves running time $2^{\mathcal{O}(k)} n$, while simpler than previous such algorithms

Algorithms for Computing Treewidth

Ref.	Appx. ratio	Run. time
[RS'87]	4	$3^{3k}n^2$
[ACP'87]	exact	n^{k+1}
[RS'87]	exact	$f(k)\cdot n^2$
[Lag'90]	8	$k^{\mathcal{O}(k)}n\log^2 n$
[MT'91]	6	$k^{\mathcal{O}(k)}n\log^2 n$
[Ree'92]	8	$k^{\mathcal{O}(k)}n\log n$
[Bod'93]	exact	$2^{\mathcal{O}(k^3)}n$
[BGHK'95]	$\mathcal{O}(\log n)$	$\mathrm{poly}(n)$
[Ami'01]	$\mathcal{O}(\log k)$	$\mathrm{poly}(n)$

Ref.	Appx. ratio	Run. time
[FHL'05]	$\mathcal{O}(\sqrt{\log k})$	$\mathrm{poly}(n)$
[BDDFLP'13]	3	$2^{\mathcal{O}(k)}n\log n$
[BDDFLP'13]	5	$2^{\mathcal{O}(k)}n$
[FTV'14]	exact	1.7347^n
[FLSPW'17]	$\mathcal{O}(k)$	$k^{\mathcal{O}(1)}n\log n$
[BF'21]	5	$2^{7k}n\log n$
[DY'24]	$\mathcal{O}(\log n)$	$k^{\mathcal{O}(1)}n\log^{\mathcal{O}(1)} n$
[Kor'21]	2	$2^{\mathcal{O}(k)}n$
[KL'23]	exact	$2^{\mathcal{O}(k^2)}n^4$
[KL'23]	$1+\varepsilon$	$k^{\mathcal{O}(k/\varepsilon)}n^4$

- [Korhonen, FOCS'21]: New "local improvement" technique for approximating treewidth
  - Improves approximation ratio below 3, breaking the RS-barrier (open problem asked by [BDDFLP'13])
  - Achieves running time $2^{\mathcal{O}(k)}n$, while simpler than previous such algorithms
- [Korhonen & Lokshtanov, STOC'23]: Lifting the technique to exact and $(1+\varepsilon)$-approximation

Algorithms for Computing Treewidth

Ref.	Appx. ratio	Run. time
[RS'87]	4	$3^{3k}n^2$
[ACP'87]	exact	n^{k+1}
[RS'87]	exact	$f(k)\cdot n^2$
[Lag'90]	8	$k^{\mathcal{O}(k)}n\log^2 n$
[MT'91]	6	$k^{\mathcal{O}(k)}n\log^2 n$
[Ree'92]	8	$k^{\mathcal{O}(k)}n\log n$
[Bod'93]	exact	$2^{\mathcal{O}(k^3)}n$
[BGHK'95]	$\mathcal{O}(\log n)$	$\mathrm{poly}(n)$
[Ami'01]	$\mathcal{O}(\log k)$	$\mathrm{poly}(n)$

Ref.	Appx. ratio	Run. time
[FHL'05]	$\mathcal{O}(\sqrt{\log k})$	$\mathrm{poly}(n)$
[BDDFLP'13]	3	$2^{\mathcal{O}(k)}n\log n$
[BDDFLP'13]	5	$2^{\mathcal{O}(k)}n$
[FTV'14]	exact	1.7347^n
[FLSPW'17]	$\mathcal{O}(k)$	$k^{\mathcal{O}(1)}n\log n$
[BF'21]	5	$2^{7k}n\log n$
[DY'24]	$\mathcal{O}(\log n)$	$k^{\mathcal{O}(1)}n\log^{\mathcal{O}(1)} n$
[Kor'21]	2	$2^{\mathcal{O}(k)}n$
[KL'23]	exact	$2^{\mathcal{O}(k^2)}n^4$
[KL'23]	$1+\varepsilon$	$k^{\mathcal{O}(k/\varepsilon)}n^4$

- [Korhonen, FOCS'21]: New "local improvement" technique for approximating treewidth
  - Improves approximation ratio below 3, breaking the RS-barrier (open problem asked by [BDDFLP'13])
  - Achieves running time $2^{\mathcal{O}(k)}n$, while simpler than previous such algorithms
- [Korhonen & Lokshtanov, STOC'23]: Lifting the technique to exact and $(1+\varepsilon)$-approximation
  - Solves open problem about breaking the $2^{\mathcal{O}(k^3)}$-barrier asked by [DF'99], [Telle'06], [BDDFLP'13], [BJT'21]

Algorithms for Computing Treewidth

Ref.	Appx. ratio	Run. time
[RS'87]	4	$3^{3k} n^2$
[ACP'87]	exact	n^{k+1}
[RS'87]	exact	$f(k) \cdot n^2$
[Lag'90]	8	$k^{\mathcal{O}(k)} n \log^2 n$
[MT'91]	6	$k^{\mathcal{O}(k)} n \log^2 n$
[Ree'92]	8	$k^{\mathcal{O}(k)} n \log n$
[Bod'93]	exact	$2^{\mathcal{O}(k^3)} n$
[BGHK'95]	$\mathcal{O}(\log n)$	$\text{poly}(n)$
[Ami'01]	$\mathcal{O}(\log k)$	$\text{poly}(n)$

Ref.	Appx. ratio	Run. time
[FHL'05]	$\mathcal{O}(\sqrt{\log k})$	$\text{poly}(n)$
[BDDFLP'13]	3	$2^{\mathcal{O}(k)} n \log n$
[BDDFLP'13]	5	$2^{\mathcal{O}(k)} n$
[FTV'14]	exact	1.7347^n
[FLSPW'17]	$\mathcal{O}(k)$	$k^{\mathcal{O}(1)} n \log n$
[BF'21]	5	$2^{7k} n \log n$
[DY'24]	$\mathcal{O}(\log n)$	$k^{\mathcal{O}(1)} n \log^{\mathcal{O}(1)} n$
[Kor'21]	2	$2^{\mathcal{O}(k)} n$
[KL'23]	exact	$2^{\mathcal{O}(k^2)} n^4$
[KL'23]	$1+\varepsilon$	$k^{\mathcal{O}(k/\varepsilon)} n^4$

- [Korhonen, FOCS'21]: New "local improvement" technique for approximating treewidth
  - Improves approximation ratio below 3, breaking the RS-barrier (open problem asked by [BDDFLP'13])
  - Achieves running time $2^{\mathcal{O}(k)} n$, while simpler than previous such algorithms
- [Korhonen & Lokshtanov, STOC'23]: Lifting the technique to exact and $(1+\varepsilon)$-approximation
  - Solves open problem about breaking the $2^{\mathcal{O}(k^3)}$-barrier asked by [DF'99], [Telle'06], [BDDFLP'13], [BJT'21]
- [Korhonen, Majewski, Nadara, Pilipczuk & Sokołowski, FOCS'23]: First dynamic treewidth algorithm

Algorithms for Computing Treewidth

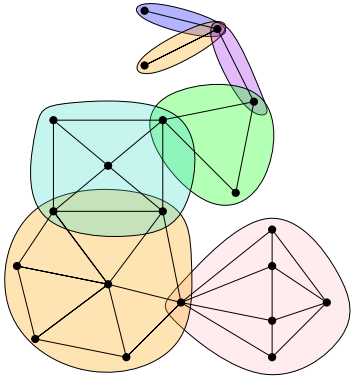
Ref.	Appx. ratio	Run. time
[RS'87]	4	$3^{3k}n^2$
[ACP'87]	exact	n^{k+1}
[RS'87]	exact	$f(k)\cdot n^2$
[Lag'90]	8	$k^{\mathcal{O}(k)}n\log^2 n$
[MT'91]	6	$k^{\mathcal{O}(k)}n\log^2 n$
[Ree'92]	8	$k^{\mathcal{O}(k)}n\log n$
[Bod'93]	exact	$2^{\mathcal{O}(k^3)}n$
[BGHK'95]	$\mathcal{O}(\log n)$	$\mathrm{poly}(n)$
[Ami'01]	$\mathcal{O}(\log k)$	$\mathrm{poly}(n)$

Ref.	Appx. ratio	Run. time
[FHL'05]	$\mathcal{O}(\sqrt{\log k})$	$\mathrm{poly}(n)$
[BDDFLP'13]	3	$2^{\mathcal{O}(k)}n\log n$
[BDDFLP'13]	5	$2^{\mathcal{O}(k)}n$
[FTV'14]	exact	1.7347^n
[FLSPW'17]	$\mathcal{O}(k)$	$k^{\mathcal{O}(1)}n\log n$
[BF'21]	5	$2^{7k}n\log n$
[DY'24]	$\mathcal{O}(\log n)$	$k^{\mathcal{O}(1)}n\log^{\mathcal{O}(1)} n$
[Kor'21]	2	$2^{\mathcal{O}(k)}n$
[KL'23]	exact	$2^{\mathcal{O}(k^2)}n^4$
[KL'23]	$1+\varepsilon$	$k^{\mathcal{O}(k/\varepsilon)}n^4$

- [Korhonen, FOCS'21]: New "local improvement" technique for approximating treewidth
  - Improves approximation ratio below 3, breaking the RS-barrier (open problem asked by [BDDFLP'13])
  - Achieves running time $2^{\mathcal{O}(k)}n$, while simpler than previous such algorithms
- [Korhonen & Lokshtanov, STOC'23]: Lifting the technique to exact and $(1+\varepsilon)$-approximation
  - Solves open problem about breaking the $2^{\mathcal{O}(k^3)}$-barrier asked by [DF'99], [Telle'06], [BDDFLP'13], [BJT'21]
- [Korhonen, Majewski, Nadara, Pilipczuk & Sokołowski, FOCS'23]: First dynamic treewidth algorithm
  - Partially solves problem of [Bodlaender'93] about dynamic treewidth

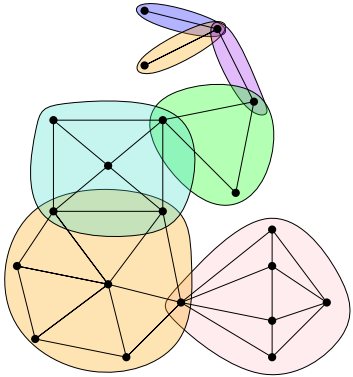
Beyond Treewidth

- In many settings, we wish to decompose a graph until remaining parts are “well-connected”



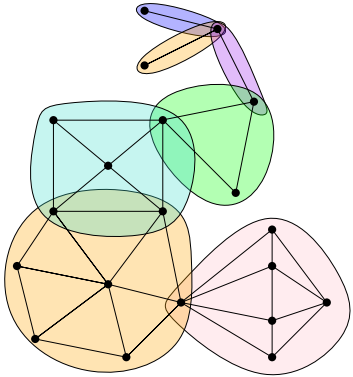
Beyond Treewidth

- In many settings, we wish to decompose a graph until remaining parts are “well-connected”
  - “k-connected components” (multiple different definitions)



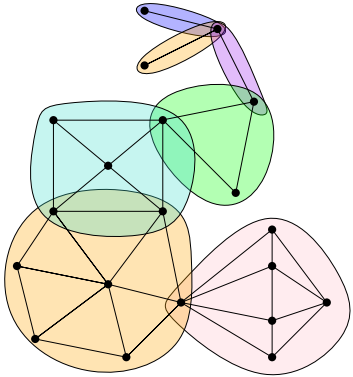
Beyond Treewidth

- In many settings, we wish to decompose a graph until remaining parts are “well-connected”
  - “k-connected components” (multiple different definitions)
- [Korhonen, STOC’25]: For every constant k, linear-time algorithms for
  - k-edge-connected components
  - k-Gomory-Hu tree
  - k-vertex separator
  - k-unbreakable tree decomposition
  - k-lean tree decomposition



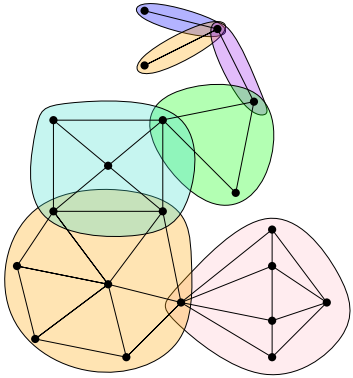
Beyond Treewidth

- In many settings, we wish to decompose a graph until remaining parts are “well-connected”
  - “k-connected components” (multiple different definitions)
- [Korhonen, STOC’25]: For every constant k, linear-time algorithms for
  - k-edge-connected components
  - k-Gomory-Hu tree
  - k-vertex separator
  - k-unbreakable tree decomposition
  - k-lean tree decomposition
- Answers a question dating to [Aho, Hopcroft & Ullman, ’74] $(*)$



Beyond Treewidth

- In many settings, we wish to decompose a graph until remaining parts are “well-connected”
  - “k-connected components” (multiple different definitions)
- [Korhonen, STOC’25]: For every constant k, linear-time algorithms for
  - k-edge-connected components
  - k-Gomory-Hu tree
  - k-vertex separator
  - k-unbreakable tree decomposition
  - k-lean tree decomposition
- Answers a question dating to [Aho, Hopcroft & Ullman, ’74] $(*)$



- [Korhonen, FOCS’25]: Adapted the techniques to give an optimal running time for dynamic treewidth

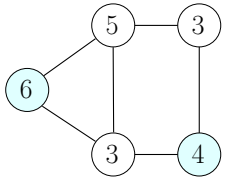
Outline of the Talk

– Background and motivation

– My contributions

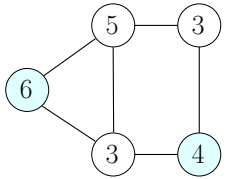
⇒ Future research

Understanding Tractability of Maximum Weight Independent Set Problem (MWIS)

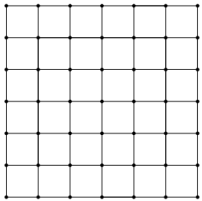


$\Rightarrow \quad 10$

Understanding Tractability of Maximum Weight Independent Set Problem (MWIS)

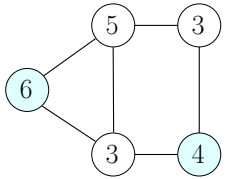


$\Rightarrow \quad 10$

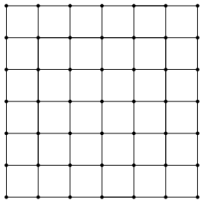


- [Korhonen, ICALP'21]: MWIS is hard on all graphs containing large *grid induced minors* (for a certain class of monotone algorithms)

Understanding Tractability of Maximum Weight Independent Set Problem (MWIS)

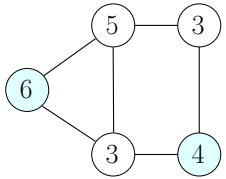


$\Rightarrow \quad 10$

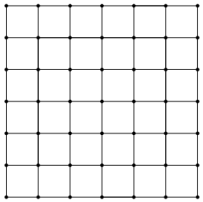


- [Korhonen, ICALP'21]: MWIS is hard on all graphs containing large *grid induced minors* (for a certain class of monotone algorithms)
- Conjecture [Dallard-Milanič-Štorgel, Gartland-Lokshtanov '21]: MWIS is easy for graphs **not** containing large grid induced minors

Understanding Tractability of Maximum Weight Independent Set Problem (MWIS)

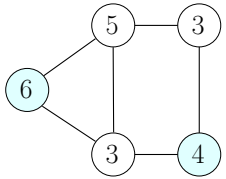


$\Rightarrow \quad 10$

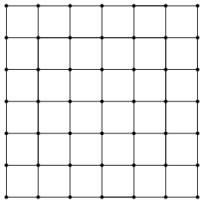


- [Korhonen, ICALP'21]: MWIS is hard on all graphs containing large *grid induced minors* (for a certain class of monotone algorithms)
- Conjecture [Dallard-Milanič-Štorgel, Gartland-Lokshtanov '21]: MWIS is easy for graphs **not** containing large grid induced minors

Understanding Tractability of Maximum Weight Independent Set Problem (MWIS)

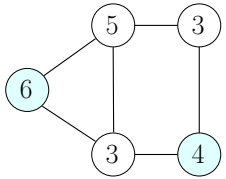


$\Rightarrow \quad 10$

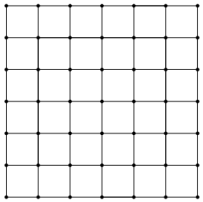


- [Korhonen, ICALP'21]: MWIS is hard on all graphs containing large *grid induced minors* (for a certain class of monotone algorithms)
- Conjecture [Dallard-Milanič-Štorgel, Gartland-Lokshtanov '21]: MWIS is easy for graphs **not** containing large grid induced minors
- [Korhonen '22]: Conjecture true for bounded-degree graphs

Understanding Tractability of Maximum Weight Independent Set Problem (MWIS)

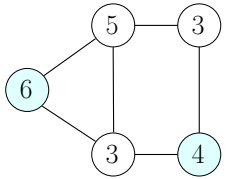


$\Rightarrow \quad 10$

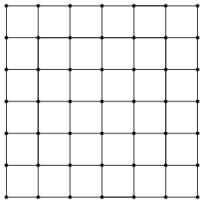


- [Korhonen, ICALP'21]: MWIS is hard on all graphs containing large *grid induced minors* (for a certain class of monotone algorithms)
- Conjecture [Dallard-Milanič-Štorgel, Gartland-Lokshtanov '21]: MWIS is easy for graphs **not** containing large grid induced minors
- [Korhonen '22]: Conjecture true for bounded-degree graphs
  - Solved an open problem in graph theory asked by e.g. Maria Chudnovsky

Understanding Tractability of Maximum Weight Independent Set Problem (MWIS)



$\Rightarrow \quad 10$



- [Korhonen, ICALP'21]: MWIS is hard on all graphs containing large *grid induced minors* (for a certain class of monotone algorithms)
- Conjecture [Dallard-Milanič-Štorgel, Gartland-Lokshtanov '21]: MWIS is easy for graphs **not** containing large grid induced minors
- [Korhonen '22]: Conjecture true for bounded-degree graphs
  - Solved an open problem in graph theory asked by e.g. Maria Chudnovsky
- Conjecture wide open, but progress on special cases, e.g. [Bonamy, Bonnet, Déprés, Esperet, Geniet, Hilaire, Thomassé, Wesolek, SODA'23], [Korhonen & Lokshtanov, SODA'24], [Chudnovsky, Gartland, Hajebi, Lokshtanov, Spirkl, SODA'25]

Fully Polynomial-Time Parameterized Graph Decomposition

Constant-factor approximation

$2^{\mathcal{O}(k)}n$ running time

Treewidth, k-unbreakable tree decomposition...

Fully Polynomial-Time Parameterized Graph Decomposition

Constant-factor approximation
$2^{\mathcal{O}(k)}n$ running time

Treewidth, k-unbreakable tree decomposition...

Fully polynomial-time?

Fully Polynomial-Time Parameterized Graph Decomposition

Constant-factor approximation
$2^{\mathcal{O}(k)}n$ running time

$\mathcal{O}(\sqrt{\log k})$-approximation
$k^{\mathcal{O}(1)}n$ running time

Treewidth, k-unbreakable tree decomposition...

?

Fully polynomial-time?

Fully Polynomial-Time Parameterized Graph Decomposition

Constant-factor approximation
$2^{\mathcal{O}(k)}n$ running time

$\mathcal{O}(\sqrt{\log k})$-approximation
$k^{\mathcal{O}(1)}n$ running time

Treewidth, k-unbreakable tree decomposition...

?

Fully polynomial-time?

State-of-the-art:

- Treewidth: $\mathcal{O}(k)$-approximation and $\mathcal{O}(\log n)$-approximation in $k^{\mathcal{O}(1)}n \operatorname{polylog} n$ time
  [Fomin, Lokshtanov, Pilipczuk, Saurabh & Wrochna '17], [Dong & Ye '24]

Fully Polynomial-Time Parameterized Graph Decomposition

Constant-factor approximation
$2^{\mathcal{O}(k)}n$ running time

$\mathcal{O}(\sqrt{\log k})$-approximation
$k^{\mathcal{O}(1)}n$ running time

Treewidth, k-unbreakable tree decomposition...

?

Fully polynomial-time?

State-of-the-art:

- Treewidth: $\mathcal{O}(k)$-approximation and $\mathcal{O}(\log n)$-approximation in $k^{\mathcal{O}(1)}n \operatorname{polylog} n$ time
  [Fomin, Lokshtanov, Pilipczuk, Saurabh & Wrochna '17], [Dong & Ye '24]
- Expander hierarchy: n^{ε}-approximate decomposition in $n^{1+\varepsilon}$ time for any $\varepsilon > 0$
  [Goranci, Räcke, Saranurak & Tan '21]

Fully Polynomial-Time Parameterized Graph Decomposition

Constant-factor approximation
$2^{\mathcal{O}(k)}n$ running time

$\mathcal{O}(\sqrt{\log k})$-approximation
$k^{\mathcal{O}(1)}n$ running time

Treewidth, k-unbreakable tree decomposition...

?

Fully polynomial-time?

State-of-the-art:

- Treewidth: $\mathcal{O}(k)$-approximation and $\mathcal{O}(\log n)$-approximation in $k^{\mathcal{O}(1)}n \operatorname{polylog} n$ time
  [Fomin, Lokshtanov, Pilipczuk, Saurabh & Wrochna '17], [Dong & Ye '24]
- Expander hierarchy: n^{ε}-approximate decomposition in $n^{1+\varepsilon}$ time for any $\varepsilon > 0$
  [Goranci, Räcke, Saranurak & Tan '21]

Potential techniques:

Fully Polynomial-Time Parameterized Graph Decomposition

Constant-factor approximation
$2^{\mathcal{O}(k)}n$ running time

$\mathcal{O}(\sqrt{\log k})$-approximation
$k^{\mathcal{O}(1)}n$ running time

Treewidth, k-unbreakable tree decomposition...

?

Fully polynomial-time?

State-of-the-art:

- Treewidth: $\mathcal{O}(k)$-approximation and $\mathcal{O}(\log n)$-approximation in $k^{\mathcal{O}(1)}n \operatorname{polylog} n$ time
  [Fomin, Lokshtanov, Pilipczuk, Saurabh & Wrochna '17], [Dong & Ye '24]
- Expander hierarchy: n^{ε}-approximate decomposition in $n^{1+\varepsilon}$ time for any $\varepsilon > 0$
  [Goranci, Räcke, Saranurak & Tan '21]

Potential techniques:

- BFS-tree packing for finding separators [Bonnet, Korhonen, Le, Li & Masařík '26]

Fully Polynomial-Time Parameterized Graph Decomposition

Constant-factor approximation
$2^{\mathcal{O}(k)}n$ running time

$\mathcal{O}(\sqrt{\log k})$-approximation
$k^{\mathcal{O}(1)}n$ running time

Treewidth, k-unbreakable tree decomposition...

?

Fully polynomial-time?

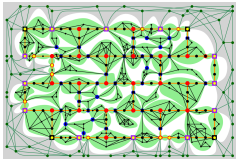
State-of-the-art:

- Treewidth: $\mathcal{O}(k)$-approximation and $\mathcal{O}(\log n)$-approximation in $k^{\mathcal{O}(1)}n \operatorname{polylog} n$ time
  [Fomin, Lokshtanov, Pilipczuk, Saurabh & Wrochna '17], [Dong & Ye '24]
- Expander hierarchy: n^{ε}-approximate decomposition in $n^{1+\varepsilon}$ time for any $\varepsilon > 0$
  [Goranci, Räcke, Saranurak & Tan '21]

Potential techniques:

- BFS-tree packing for finding separators [Bonnet, Korhonen, Le, Li & Masařík '26]
- Bodlaender's matching contraction [Bodlaender'93], [Korhonen'25]

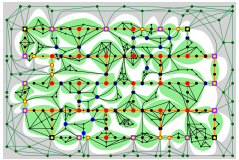
Understanding High Treewidth via Constant-Congestion Minors



High treewidth $\approx$ can embed planar graphs as minors

[Robertson & Seymour '86]

Understanding High Treewidth via Constant-Congestion Minors

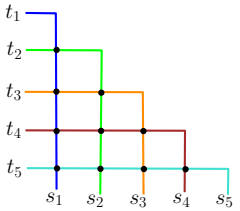


High treewidth $\approx$ can embed planar graphs as minors

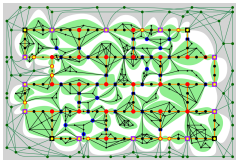
[Robertson & Seymour '86]

High treewidth $\approx$ can embed expander graphs as constant-congestion minors

[Chekuri & Chuzhoy '16]

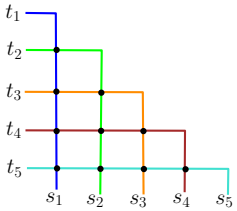


Understanding High Treewidth via Constant-Congestion Minors



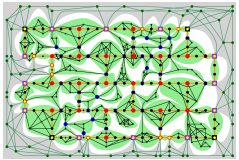
High treewidth $\approx$ can embed planar graphs as minors
[Robertson & Seymour '86]

High treewidth $\approx$ can embed expander graphs as constant-congestion minors
[Chekuri & Chuzhoy '16]



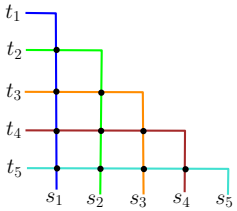
Goal: Framework for instance-optimality results by techniques of Chekuri & Chuzhoy

Understanding High Treewidth via Constant-Congestion Minors



High treewidth $\approx$ can embed planar graphs as minors
[Robertson & Seymour '86]

High treewidth $\approx$ can embed expander graphs as constant-congestion minors
[Chekuri & Chuzhoy '16]



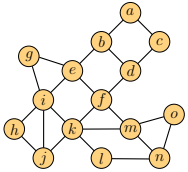
Goal: Framework for instance-optimality results by techniques of Chekuri & Chuzhoy

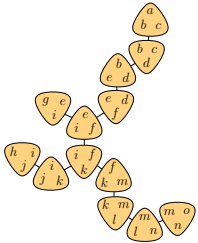
Concrete settings:

- Grohe-Schwentick-Segoufin-Marx CSP result for bounded-domain CSPs?
- SAT?
- Monadic Second-Order logic?

Conclusion

Structure matters for solving algorithmic problems

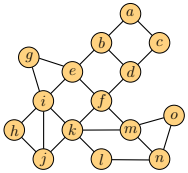


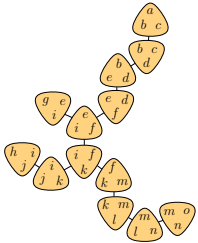


Conclusion

Structure matters for solving algorithmic problems

Two connected themes of research:



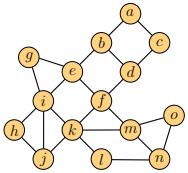


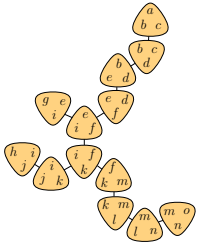
Conclusion

Structure matters for solving algorithmic problems

Two connected themes of research:

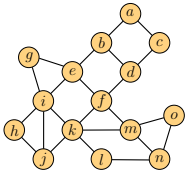
1. Drawing the line between easy and hard instances using graph-theoretic methods





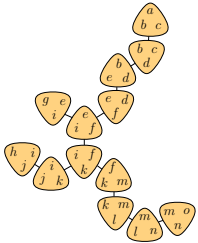
Conclusion

Structure matters for solving algorithmic problems



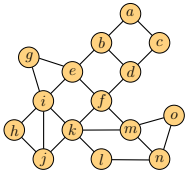
Two connected themes of research:

1. Drawing the line between easy and hard instances using graph-theoretic methods
   - Contribution/future: Understanding dense structure via MWIS
   - Future: Instance-optimality results via constant-congestion minors



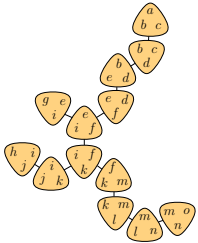
Conclusion

Structure matters for solving algorithmic problems



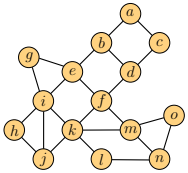
Two connected themes of research:

1. Drawing the line between easy and hard instances using graph-theoretic methods
   - Contribution/future: Understanding dense structure via MWIS
   - Future: Instance-optimality results via constant-congestion minors
2. Designing algorithms to discover the graph-theoretic structure that can be exploited



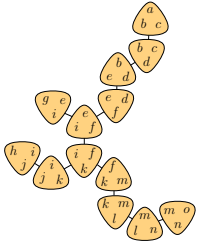
Conclusion

Structure matters for solving algorithmic problems



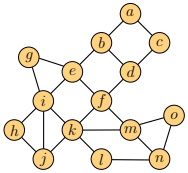
Two connected themes of research:

1. Drawing the line between easy and hard instances using graph-theoretic methods
   - Contribution/future: Understanding dense structure via MWIS
   - Future: Instance-optimality results via constant-congestion minors
2. Designing algorithms to discover the graph-theoretic structure that can be exploited
   - Contribution: Faster algorithms for computing treewidth
   - Contribution: Linear-time algorithms for k-connected components
   - Future: $k^{\mathcal{O}(1)}n$ time algorithms for these problems



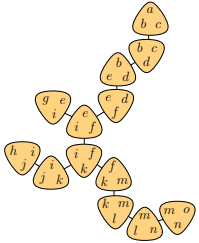
Conclusion

Structure matters for solving algorithmic problems



Two connected themes of research:

1. Drawing the line between easy and hard instances using graph-theoretic methods
   - Contribution/future: Understanding dense structure via MWIS
   - Future: Instance-optimality results via constant-congestion minors
2. Designing algorithms to discover the graph-theoretic structure that can be exploited
   - Contribution: Faster algorithms for computing treewidth
   - Contribution: Linear-time algorithms for k-connected components
   - Future: $k^{\mathcal{O}(1)}n$ time algorithms for these problems



Thank you!