How Graph Structure Makes Hard Problems Tractable

Tuukka Korhonen





13 November 2026

HIIT

HIIT Special Seminar

Role of Structure in Algorithms



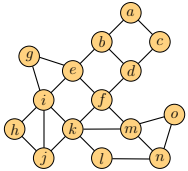
- There are really hard computational problems (NP-hard)
- Yet we solve these problems routinely in practice

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



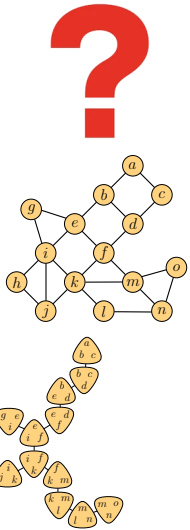


Role of Structure in Algorithms

- 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?



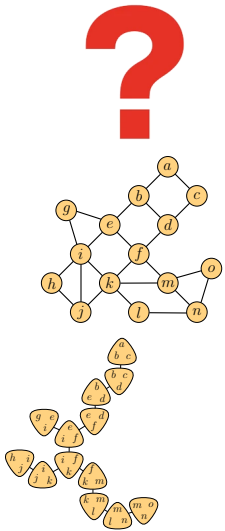
Role of Structure in Algorithms

- 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. Algorithms for exploiting graph-theoretic structure



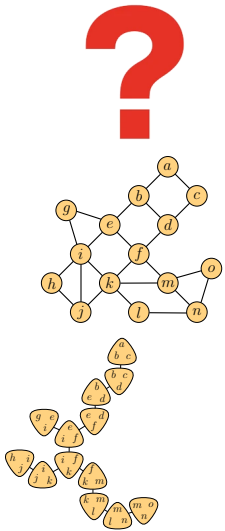
Role of Structure in Algorithms

- 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. Algorithms for exploiting graph-theoretic structure
2. Algorithms for discovering graph-theoretic structure



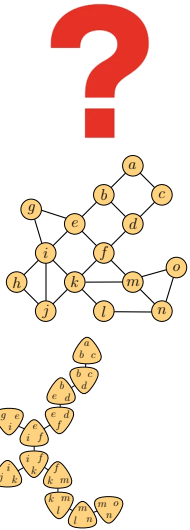
Role of Structure in Algorithms

- 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. Algorithms for exploiting graph-theoretic structure
2. Algorithms for discovering graph-theoretic structure
3. Characterizing the most general exploitable graph-theoretic structure



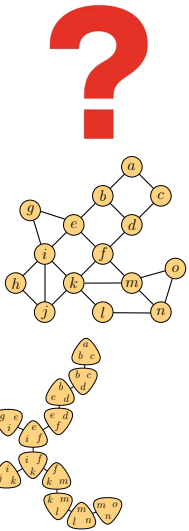
Role of Structure in Algorithms

- 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

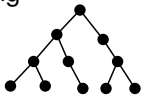
My research: What can theory say about this?

1. Algorithms for exploiting graph-theoretic structure
2. **Algorithms for discovering graph-theoretic structure**
3. Characterizing the most general exploitable graph-theoretic structure



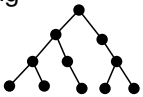
Treewidth

Many NP-hard graph problems become easy if the underlying graph is a tree

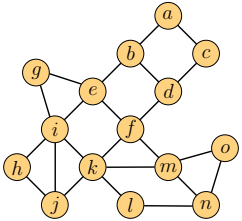


Treewidth

Many NP-hard graph problems become easy if the underlying graph is a tree

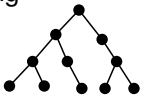


What if the graph is not a tree, but almost?

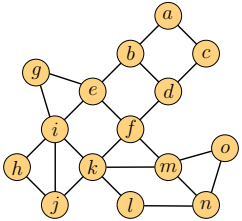


Treewidth

Many NP-hard graph problems become easy if the underlying graph is a tree



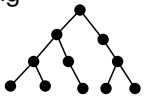
What if the graph is not a tree, but almost?



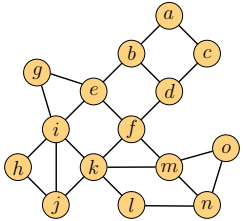
The notion of **Treewidth**

Treewidth

Many NP-hard graph problems become easy if the underlying graph is a tree



What if the graph is not a tree, but almost?

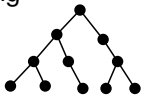


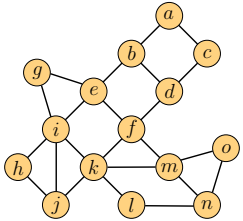
The notion of **Treewidth**

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

Treewidth

Many NP-hard graph problems become easy if the underlying graph is a tree





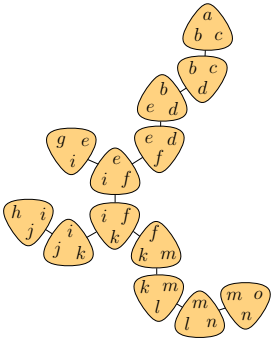
What if the graph is not a tree, but almost?

The notion of **Treewidth**

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

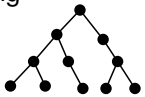
Treewidth of a graph: Minimum width of tree decomposition

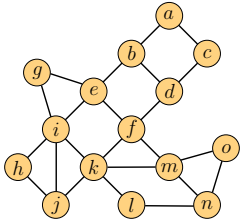
Width of a tree decomposition: Maximum bag size -1



Treewidth

Many NP-hard graph problems become easy if the underlying graph is a tree





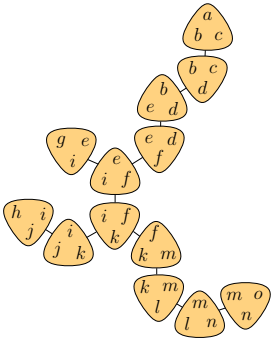
What if the graph is not a tree, but almost?

The notion of **Treewidth**

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

Treewidth of a graph: Minimum width of tree decomposition

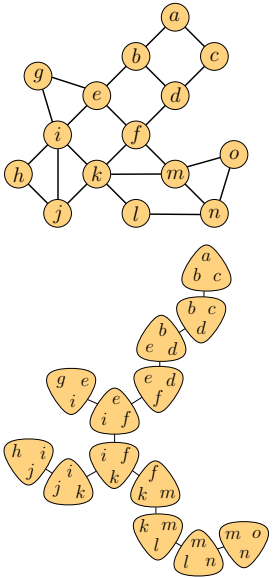
Width of a tree decomposition: Maximum bag size -1



Introduced by [Robertson & Seymour'84; Arnborg & Proskurowski'89, Bertelè & Brioschi'72, Halin'76]

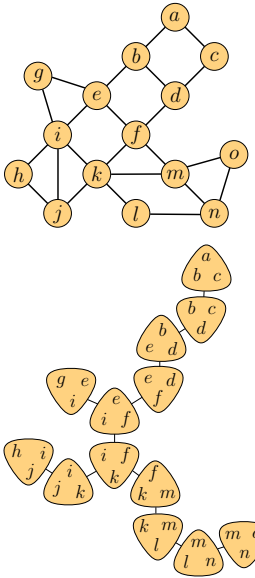
Why Treewidth

- Given a tree decomposition of width k, many computational problems can be solved in time $2^{\mathcal{O}(k)}n$



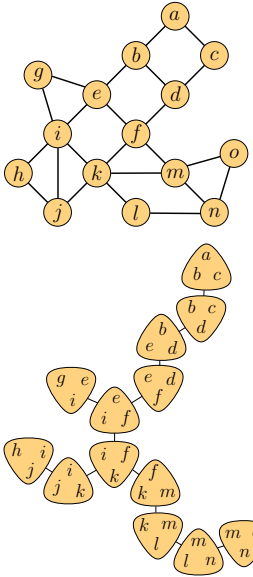
Why Treewidth

- Given a tree decomposition of width k, many computational problems can be solved in time $2^{\mathcal{O}(k)}n$
- Not limited to pure graph problems: The graph can capture the structure of any discrete problem



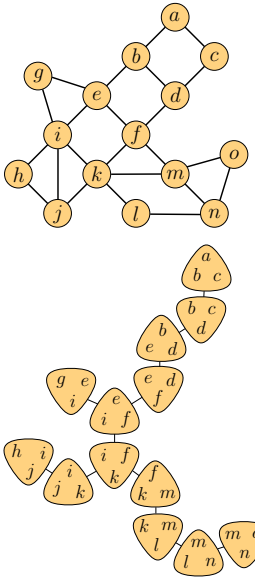
Why Treewidth

- Given a tree decomposition of width k, many computational problems can be solved in time $2^{\mathcal{O}(k)}n$
- Not limited to pure graph problems: The graph can capture the structure of any discrete problem
  - Constraint satisfaction



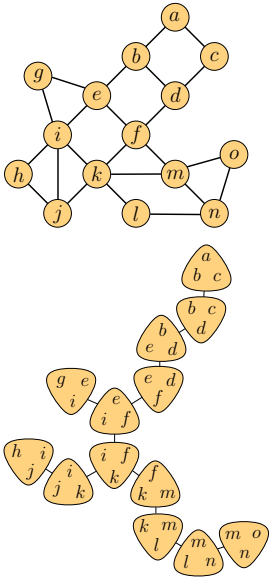
Why Treewidth

- Given a tree decomposition of width k, many computational problems can be solved in time $2^{\mathcal{O}(k)}n$
- Not limited to pure graph problems: The graph can capture the structure of any discrete problem
  - Constraint satisfaction
  - Quantum computing simulation



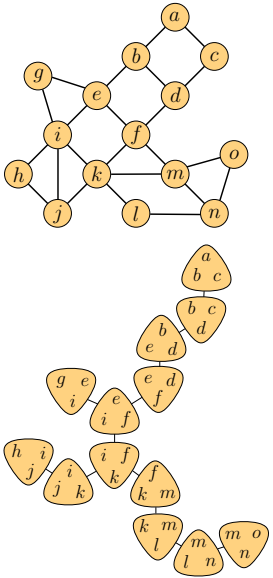
Why Treewidth

- Given a tree decomposition of width k, many computational problems can be solved in time $2^{\mathcal{O}(k)}n$
- Not limited to pure graph problems: The graph can capture the structure of any discrete problem
  - Constraint satisfaction
  - Quantum computing simulation
  - Probabilistic inference



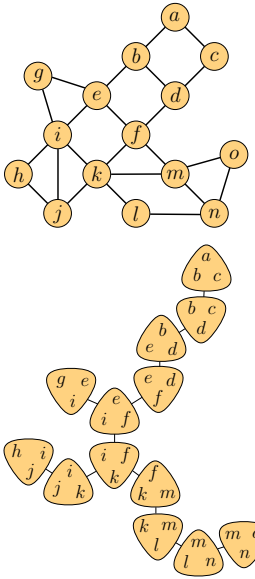
Why Treewidth

- Given a tree decomposition of width k, many computational problems can be solved in time $2^{\mathcal{O}(k)}n$
- Not limited to pure graph problems: The graph can capture the structure of any discrete problem
  - Constraint satisfaction
  - Quantum computing simulation
  - Probabilistic inference
  - Join evaluation in databases

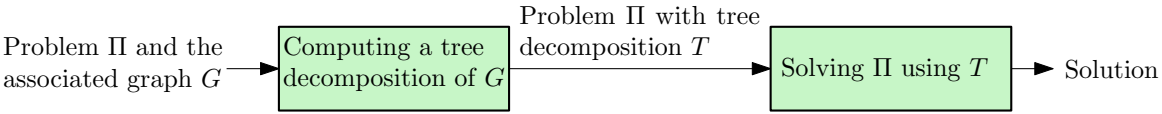


Why Treewidth

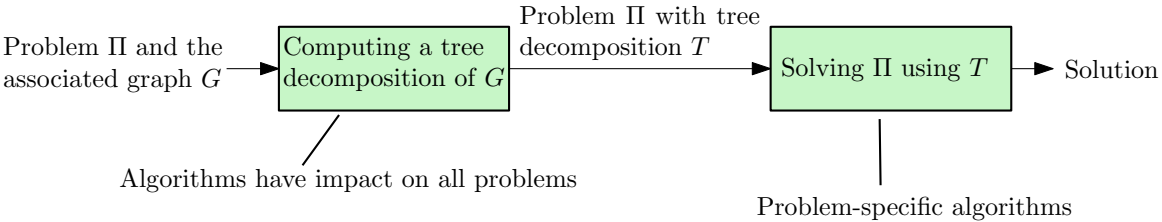
- Given a tree decomposition of width k, many computational problems can be solved in time $2^{\mathcal{O}(k)}n$
- Not limited to pure graph problems: The graph can capture the structure of any discrete problem
  - Constraint satisfaction
  - Quantum computing simulation
  - Probabilistic inference
  - Join evaluation in databases
- Often treewidth is the **optimal** parameter: For **every** graph, treewidth characterizes how hard problems can be embedded into it
  [Robertson&Seymour'85; Seese'91; Grohe,Schwentick&Segoufin'01; Marx'07; Courcelle&Oum'07; Kwisthout,Bodlaender&van der Gaag'10]



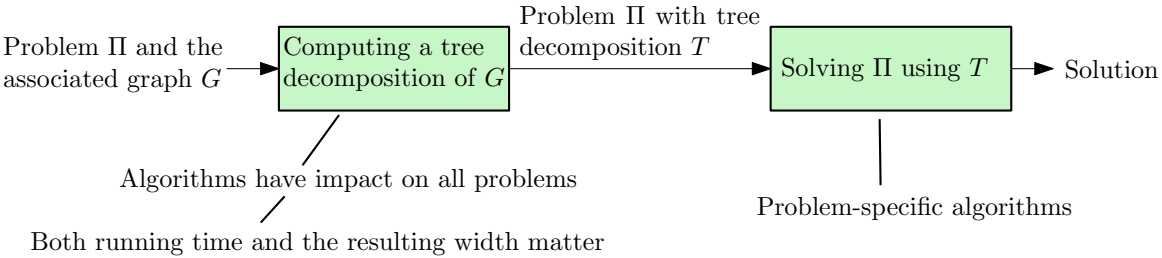
Computing Treewidth



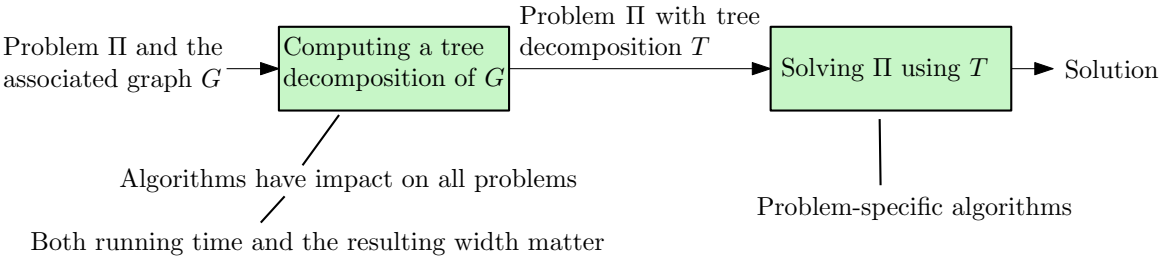
Computing Treewidth



Computing Treewidth

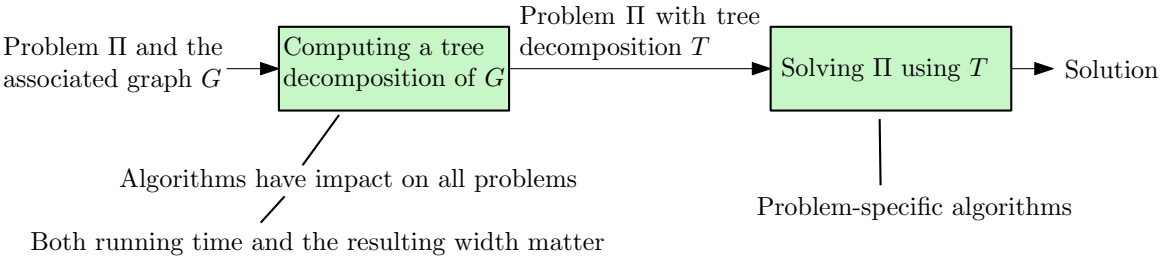


Computing Treewidth



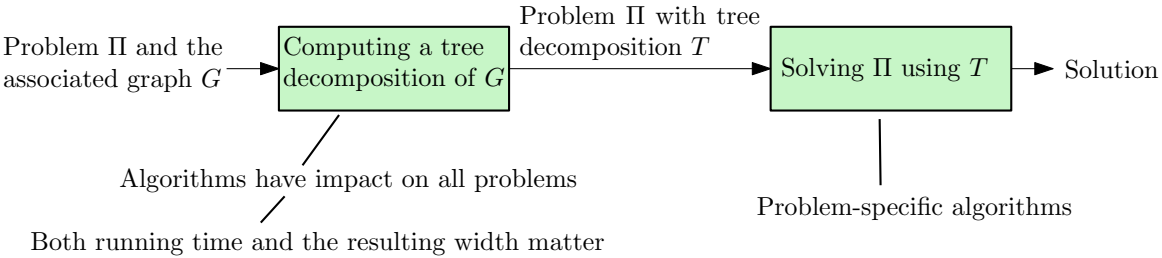
- Computing minimum-width tree decomposition is NP-hard

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

Algorithms for Computing Treewidth

Ref.	Appx. ratio	Run. time
[RS'87]	4	$3^{3k} n^2$
[ACP'87]	exact	n^{k+2}
[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
[FLPSW'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+2}
[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

Algorithms for Computing Treewidth

Ref.	Appx. ratio	Run. time
[RS'87]	4	$3^{3k} n^2$
[ACP'87]	exact	n^{k+2}
[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
  - 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+2}
[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	$2^{\mathcal{O}(k)} n$
[**K**L'23]	exact	$2^{\mathcal{O}(k^2)} n^4$
[**K**L'23]	$1+\varepsilon$	$k^{\mathcal{O}(k/\varepsilon)} n^4$

- [**Korhonen**, FOCS'21]: New "local improvement" technique for approximating treewidth
  - 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+2}
[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	$2^{\mathcal{O}(k)} n$
[**K**L'23]	exact	$2^{\mathcal{O}(k^2)} n^4$
[**K**L'23]	$1+\varepsilon$	$k^{\mathcal{O}(k/\varepsilon)} n^4$

- [**Korhonen**, FOCS'21]: New "local improvement" technique for approximating treewidth
  - 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+2}
[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	$2^{\mathcal{O}(k)} n$
[**K**L'23]	exact	$2^{\mathcal{O}(k^2)} n^4$
[**K**L'23]	$1+\varepsilon$	$k^{\mathcal{O}(k/\varepsilon)} n^4$

- [**Korhonen**, FOCS'21]: New "local improvement" technique for approximating treewidth
  - 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+2}
[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	$2^{\mathcal{O}(k)} n$
[**K**L'23]	exact	$2^{\mathcal{O}(k^2)} n^4$
[**K**L'23]	$1+\varepsilon$	$k^{\mathcal{O}(k/\varepsilon)} n^4$

- [**Korhonen**, FOCS'21]: New "local improvement" technique for approximating treewidth
  - 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 about dynamic treewidth asked by [Bodlaender'93; Alman,Mnich&Vassilevska Williams'20]

Algorithms for Computing Treewidth

Ref.	Appx. ratio	Run. time
[RS'87]	4	$3^{3k}n^2$
[ACP'87]	exact	n^{k+2}
[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	$2^{\mathcal{O}(k)} n$
[**K**L'23]	exact	$2^{\mathcal{O}(k^2)} n^4$
[**K**L'23]	$1+\varepsilon$	$k^{\mathcal{O}(k/\varepsilon)} n^4$

- [**Korhonen**, FOCS'21]: New "local improvement" technique for approximating treewidth
  - 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 about dynamic treewidth asked by [Bodlaender'93; Alman,Mnich&Vassilevska Williams'20]
- [**Korhonen**, FOCS'25]: Dynamic treewidth in $2^{\mathcal{O}(k)} \log n$ time per operation

Algorithms for Computing Treewidth

Ref.	Appx. ratio	Run. time
[RS'87]	4	$3^{3k}n^2$
[ACP'87]	exact	n^{k+2}
[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	$2^{\mathcal{O}(k)} n$
[**K**L'23]	exact	$2^{\mathcal{O}(k^2)} n^4$
[**K**L'23]	$1+\varepsilon$	$k^{\mathcal{O}(k/\varepsilon)} n^4$

- [**Korhonen**, FOCS'21]: New "local improvement" technique for approximating treewidth
  - 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 about dynamic treewidth asked by [Bodlaender'93; Alman,Mnich&Vassilevska Williams'20]
- [**Korhonen**, FOCS'25]: Dynamic treewidth in $2^{\mathcal{O}(k)} \log n$ time per operation
  - Fully solves the dynamic treewidth problem and is arguably optimal

Branchwidth of Connectivity Functions

- Branchwidth of connectivity functions: Abstract width parameter that captures many graph width parameters

Branchwidth of Connectivity Functions

- Branchwidth of connectivity functions: Abstract width parameter that captures many graph width parameters
  - Branchwidth of graphs/hypergraphs/matroids
  - Rankwidth
  - Carving-width

Branchwidth of Connectivity Functions

- Branchwidth of connectivity functions: Abstract width parameter that captures many graph width parameters
  - Branchwidth of graphs/hypergraphs/matroids
  - Rankwidth
  - Carving-width

- 3-approximation in $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$-time [Oum&Seymour'06]

Branchwidth of Connectivity Functions

- Branchwidth of connectivity functions: Abstract width parameter that captures many graph width parameters
  - Branchwidth of graphs/hypergraphs/matroids
  - Rankwidth
  - Carving-width

- 3-approximation in $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$-time [Oum&Seymour'06]

- Exact in $n^{\mathcal{O}(k)}$-time [Oum&Seymour'07]

Branchwidth of Connectivity Functions

- Branchwidth of connectivity functions: Abstract width parameter that captures many graph width parameters
  - Branchwidth of graphs/hypergraphs/matroids
  - Rankwidth
  - Carving-width

- 3-approximation in $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$-time [Oum&Seymour'06]
- Exact in $n^{\mathcal{O}(k)}$-time [Oum&Seymour'07]
- Exact in $2^{\mathcal{O}(k^2)} n^{\mathcal{O}(1)}$-time [**Korhonen**&Oum, FOCS'26]

Branchwidth of Connectivity Functions

- Branchwidth of connectivity functions: Abstract width parameter that captures many graph width parameters
  - Branchwidth of graphs/hypergraphs/matroids
  - Rankwidth
  - Carving-width

- 3-approximation in $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$-time [Oum&Seymour'06]
- Exact in $n^{\mathcal{O}(k)}$-time [Oum&Seymour'07]
- Exact in $2^{\mathcal{O}(k^2)} n^{\mathcal{O}(1)}$-time [**Korhonen**&Oum, FOCS'26]
  - Solves open problem on whether matroid branchwidth is FPT asked by [Hliněný'05]

Branchwidth of Connectivity Functions

- Branchwidth of connectivity functions: Abstract width parameter that captures many graph width parameters
  - Branchwidth of graphs/hypergraphs/matroids
  - Rankwidth
  - Carving-width

- 3-approximation in $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$-time [Oum&Seymour'06]
- Exact in $n^{\mathcal{O}(k)}$-time [Oum&Seymour'07]
- Exact in $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(1)}$-time [**Korhonen**&Oum, FOCS'26]
  - Solves open problem on whether matroid branchwidth is FPT asked by [Hliněný'05]
  - Improves the best dependence on k in FPT algorithms for **every** special case with previous FPT algorithms

Branchwidth of Connectivity Functions

- Branchwidth of connectivity functions: Abstract width parameter that captures many graph width parameters
  - Branchwidth of graphs/hypergraphs/matroids
  - Rankwidth
  - Carving-width

- 3-approximation in $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$-time [Oum&Seymour'06]
- Exact in $n^{\mathcal{O}(k)}$-time [Oum&Seymour'07]
- Exact in $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(1)}$-time [**Korhonen**&Oum, FOCS'26]
  - Solves open problem on whether matroid branchwidth is FPT asked by [Hliněný'05]
  - Improves the best dependence on k in FPT algorithms for **every** special case with previous FPT algorithms
- Exact in $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$-time [**Korhonen**&Oum, unpublished, 2026] (using GPT-6)

Branchwidth of Connectivity Functions

- Branchwidth of connectivity functions: Abstract width parameter that captures many graph width parameters
  - Branchwidth of graphs/hypergraphs/matroids
  - Rankwidth
  - Carving-width

- 3-approximation in $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$-time [Oum&Seymour'06]

- Exact in $n^{\mathcal{O}(k)}$-time [Oum&Seymour'07]

- Exact in $2^{\mathcal{O}(k^2)} n^{\mathcal{O}(1)}$-time [**Korhonen**&Oum, FOCS'26]
  - Solves open problem on whether matroid branchwidth is FPT asked by [Hliněný'05]
  - Improves the best dependence on k in FPT algorithms for **every** special case with previous FPT algorithms

- Exact in $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$-time [**Korhonen**&Oum, unpublished, 2026] (using GPT-6)
  - Optimal dependence on k, unconditionally

My Research in the AI Age

Me and everyone else in the TCS community still figuring it out

My Research in the AI Age

Me and everyone else in the TCS community still figuring it out

Possible direction: Theory soon “finished”, pivot into theory-inspired experimental algorithmics

My Research in the AI Age

Me and everyone else in the TCS community still figuring it out

Possible direction: Theory soon “finished”, pivot into theory-inspired experimental algorithmics

- Would benefit from my previous experience in theory-inspired practical algorithms research

My Research in the AI Age

Me and everyone else in the TCS community still figuring it out

Possible direction: Theory soon “finished”, pivot into theory-inspired experimental algorithmics

- Would benefit from my previous experience in theory-inspired practical algorithms research
- Super fast iteration thanks to AI coding + AI TCS

My Research in the AI Age

Me and everyone else in the TCS community still figuring it out

Possible direction: Theory soon “finished”, pivot into theory-inspired experimental algorithmics

- Would benefit from my previous experience in theory-inspired practical algorithms research
- Super fast iteration thanks to AI coding + AI TCS

My direction: Theory not yet finished, participate and attempt to understand the progress

My Research in the AI Age

Me and everyone else in the TCS community still figuring it out

Possible direction: Theory soon “finished”, pivot into theory-inspired experimental algorithmics

- Would benefit from my previous experience in theory-inspired practical algorithms research
- Super fast iteration thanks to AI coding + AI TCS

My direction: Theory not yet finished, participate and attempt to understand the progress

- If human understanding is still valued, this is the best time to work on theory

My Research in the AI Age

Me and everyone else in the TCS community still figuring it out

Possible direction: Theory soon “finished”, pivot into theory-inspired experimental algorithmics

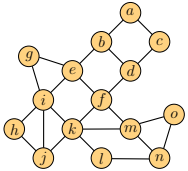
- Would benefit from my previous experience in theory-inspired practical algorithms research
- Super fast iteration thanks to AI coding + AI TCS

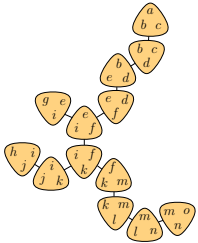
My direction: Theory not yet finished, participate and attempt to understand the progress

- If human understanding is still valued, this is the best time to work on theory
- Work on both using AI to progress my research and understanding/exploring/disseminating results published by OpenAI/Anthropic/other frontier labs

Conclusion

Structure matters for solving algorithmic problems



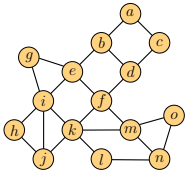


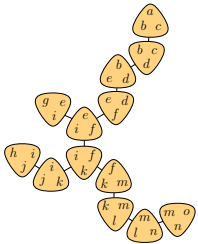
Conclusion

Structure matters for solving algorithmic problems

Main themes of my research:

1. Algorithms for exploiting graph-theoretic structure
2. **Algorithms for discovering graph-theoretic structure**
3. Characterizing the most general exploitable graph-theoretic structure





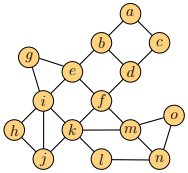
Conclusion

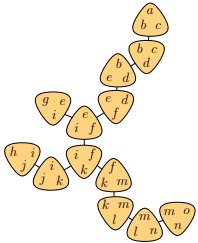
Structure matters for solving algorithmic problems

Main themes of my research:

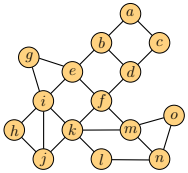
1. Algorithms for exploiting graph-theoretic structure
2. **Algorithms for discovering graph-theoretic structure**
3. Characterizing the most general exploitable graph-theoretic structure

Future: AI is changing how research is done in TCS





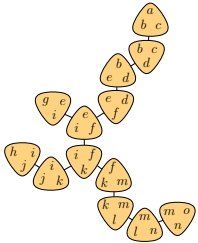
Conclusion



Structure matters for solving algorithmic problems

Main themes of my research:

1. Algorithms for exploiting graph-theoretic structure
2. **Algorithms for discovering graph-theoretic structure**
3. Characterizing the most general exploitable graph-theoretic structure



Future: AI is changing how research is done in TCS

- In the short term, continue on my research accelerated by AI

Conclusion

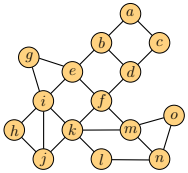
Structure matters for solving algorithmic problems

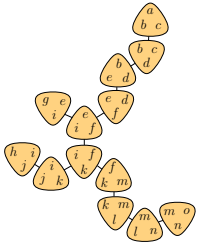
Main themes of my research:

1. Algorithms for exploiting graph-theoretic structure
2. **Algorithms for discovering graph-theoretic structure**
3. Characterizing the most general exploitable graph-theoretic structure

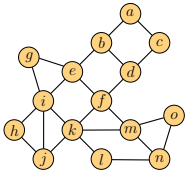
Future: AI is changing how research is done in TCS

- In the short term, continue on my research accelerated by AI
- In the long term, rethink how to best contribute to algorithms research





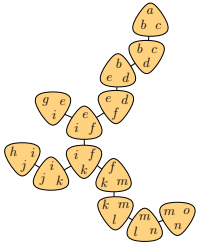
Conclusion



Structure matters for solving algorithmic problems

Main themes of my research:

1. Algorithms for exploiting graph-theoretic structure
2. **Algorithms for discovering graph-theoretic structure**
3. Characterizing the most general exploitable graph-theoretic structure



Future: AI is changing how research is done in TCS

- In the short term, continue on my research accelerated by AI
- In the long term, rethink how to best contribute to algorithms research

Thank you!