Polynomial algorithm to compute the toughness of graphs with bounded treewidth

Gyula Y. Katona

Department of Computer Science and Information Theory
Budapest University of Technology and Economics

September 21, 2022

Toughness

Definition

Let t be a positive real number. A graph G is called t-tough, if

$$c(G - S) \leq \frac{|S|}{t}$$

for any cutset S of G.
The toughness of G, denoted by $\tau(G)$, is the largest t for which G is t-tough, taking $\tau(K_n) = \infty$ for all $n \geq 1$.

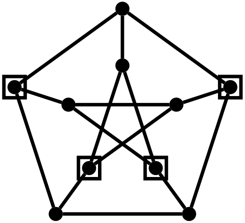
Toughness

Definition

Let t be a positive real number. A graph G is called t-tough, if

$$c(G - S) \leq \frac{|S|}{t}$$

for any cutset S of G.
The toughness of G, denoted by $\tau(G)$, is the largest t for which G is t-tough, taking $\tau(K_n) = \infty$ for all $n \geq 1$.



The Petersen graph is 4/3-tough.

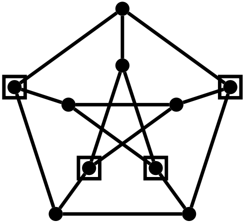
Toughness

Definition

Let t be a positive real number. A graph G is called t-tough, if

$$c(G-S) \leq \frac{|S|}{t}$$

for any cutset S of G.
The toughness of G, denoted by $\tau(G)$, is the largest t for which G is t-tough, taking $\tau(K_n) = \infty$ for all $n \geq 1$.



The Petersen graph is 4/3-tough.

A cycle is 1-tough.

Toughness

In other words: for a non-complete, connected graph G,

$$\tau(G) = \min_{S \text{ cutset}} \frac{|S|}{c(G-S)}.$$

S is called a **tough set** if it gives the ratio $\tau(G)$.

Observation

For a non-complete, connected graph G on n vertices the toughness $\tau(G)$ is a rational number $\frac{p}{q}$ with $1 \le p, q \le n$.

Proof.

Clearly $1 \le |S| \le n-2$ and $2 \le c(G-S) \le n-1$. □

MŰEGYETEM 1782

Complexity of toughness

Let t be an arbitrary positive rational number and consider the following problem.

***t*-TOUGH**

Instance: A graph G,

Question: Is it true that $\tau(G) \geq t$?

Theorem (Bauer, Hakimi, Schmeichel, 1990)

*For any positive rational number t, **t**-TOUGH is coNP-complete.*

Complexity of toughness

Let t be an arbitrary positive rational number and consider the following problem.

***t*-TOUGH**

Instance: A graph G,

Question: Is it true that $\tau(G) \geq t$?

Theorem (Bauer, Hakimi, Schmeichel, 1990)

For any positive rational number t, ***t*-TOUGH** *is coNP-complete.*

Theorem (Bauer, van den Heuvel, Morgana, Schmeichel, 1998)

1-TOUGH *is coNP-complete for r-regular graphs for all $r \geq 3$.*

Complexity of toughness in special graph classes

Theorem (Kratsch, Lehel, Müller, 1996)

The problem **1**-TOUGH *is coNP-complete for bipartite graphs.*

Theorem (GY. K., K. Varga, 2022)

For any positive rational number $t \leq 1$ the problem ***t***-TOUGH *remains coNP-complete for bipartite graphs.*

Complexity of toughness in special graph classes

Theorem (Kratsch, Lehel, Müller, 1996)

The problem **1**-TOUGH *is coNP-complete for bipartite graphs.*

Theorem (GY. K., K. Varga, 2022)

For any positive rational number $t \leq 1$ the problem $\boldsymbol{t}$-TOUGH *remains coNP-complete for bipartite graphs.*

Deciding the toughness remains coNP-hard in many other special graph classes.

Complexity of toughness in special graph classes

Theorem (Kratsch, Lehel, Müller, 1996)

The problem **1**-TOUGH *is coNP-complete for bipartite graphs.*

Theorem (GY. K., K. Varga, 2022)

For any positive rational number $t \leq 1$ the problem ***t***-TOUGH *remains coNP-complete for bipartite graphs.*

Deciding the toughness remains coNP-hard in many other special graph classes.

Famous open cases: planar graphs, chordal graphs

Complexity of toughness in special graph classes

For some other special graph classes there are polynomial algorithms to compute the toughness:

- Claw-free graphs (Matthews, Sumner, 1984)
- Split graphs (Kratsch, Lehel, Müller, 1996; Woeginger, 1998)
- Interval graphs (Kratsch, Kloks, Müller, 1994)
- $2K_2$-free graphs (Broersma, Patel, Pyatkin, 2014)
- some other more special classes ...

MŰEGYETEM 1782

Main result

Theorem

There exists an algorithm to compute the toughness of a graph G width running time $\mathcal{O}\left(n^3 \cdot \mathrm{tw}(G)^{2\mathrm{tw}(G)}\right)$, where n is the number of vertices in G and $\mathrm{tw}(G)$ is the treewidth of G.

Main result

Theorem

There exists an algorithm to compute the toughness of a graph G width running time $\mathcal{O}\left(n^3 \cdot \mathrm{tw}(G)^{2\mathrm{tw}(G)}\right)$*, where* n *is the number of vertices in G and* $\mathrm{tw}(G)$ *is the treewidth of G.*

Corollary

The toughness can be computed in polynomial time for graphs width bounded treewidth.

Main result

Theorem

There exists an algorithm to compute the toughness of a graph G width running time $\mathcal{O}\left(n^3 \cdot \mathrm{tw}(G)^{2\mathrm{tw}(G)}\right)$*, where* n *is the number of vertices in G and* $\mathrm{tw}(G)$ *is the treewidth of G.*

Corollary

The toughness can be computed in polynomial time for graphs width bounded treewidth.

Corollary

Toughness is FPT parameterized with treewidth.

Tree decomposition

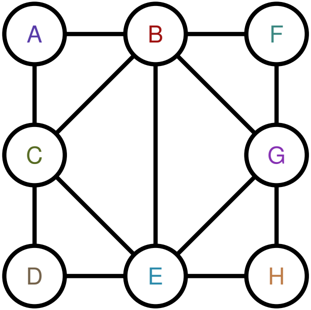
Let $G = (V, E)$ be a graph. Let $(X_t)_{t \in V(T)}$ be a family of vertex sets $X_t \subseteq V$ (bags) indexed by the nodes of a tree T. The pair $(T, \{X_t \mid t \in V(T)\})$ is a tree decomposition of G if it satisfies the following conditions:

- $\bigcup_{t \in V(T)} X_i = V$;
- for every edge $e = vw \in E$ there is a $t \in V(T)$ with $v, w \in X_t$;
- if $i, j, k \in V(T)$ and node j is on the path in T between nodes i and k, then $X_i \cap X_k \subseteq X_j$.

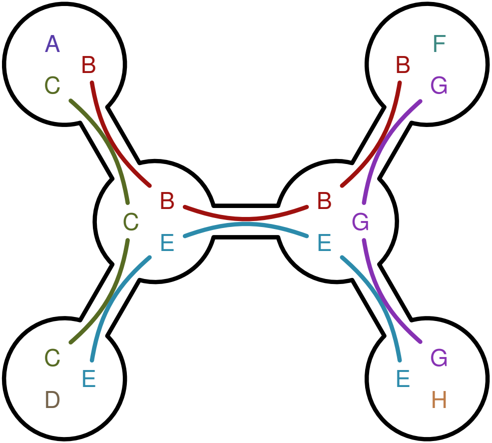
The width of the tree decomposition is $\max_{t \in V(T)} |X_t| - 1$.
The treewidth of a G is the minimum width of a tree decomposition of G.

Tree decomposition

Graph G



Tree decomposition (T, X)



Bounded treewidth

Theorem (Bodlaender, 1996)

A tree decomposition with width $\mathrm{tw}(G)$ *can be constructed in* $\mathrm{tw}(G)^{\mathcal{O}(\mathrm{tw}(G)^3)} \cdot n$ *time.*

MŰEGYETEM 1782

Bounded treewidth

Theorem (Bodlaender, 1996)

A tree decomposition with width $\mathrm{tw}(G)$ *can be constructed in* $\mathrm{tw}(G)^{\mathcal{O}(\mathrm{tw}(G)^3)} \cdot n$ *time.*

Many NP-hard problems are FPT parameterized with treewidth, so they are solvable in polynomial time if the treewidth is bounded.

Trees

Graphs with $\mathrm{tw}(G) = 1$ are trees.

Trees

Graphs with $\mathrm{tw}(G) = 1$ are trees.

Lemma

The toughness of a tree is $1/\Delta(G)$.

Proof.

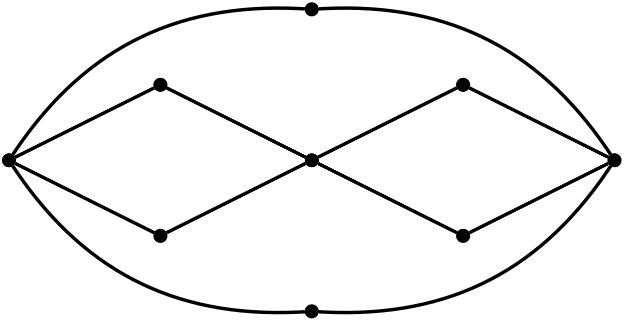
Every tough set is a single vertex with maximum degree. $\square$

Series parallel graphs

Graphs with $\mathrm{tw}(G) = 2$ are series parallel graphs.

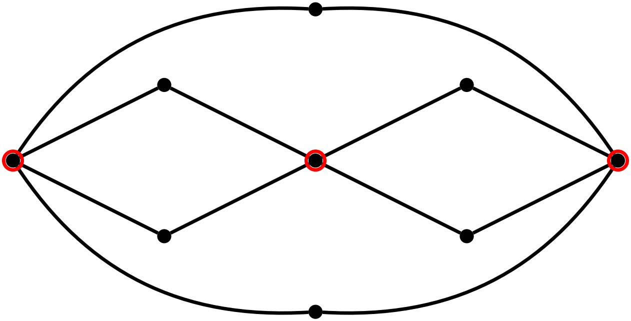
Series parallel graphs

Graphs with $\mathrm{tw}(G) = 2$ are series parallel graphs.



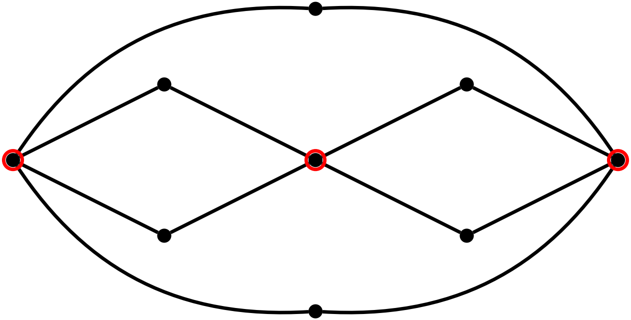
Series parallel graphs

Graphs with $\mathrm{tw}(G) = 2$ are series parallel graphs.



Series parallel graphs

Graphs with $\mathrm{tw}(G) = 2$ are series parallel graphs.



A polynomial algorithm can be designed using dynamic programming on the series-parallel decomposition tree.

Nice tree decomposition

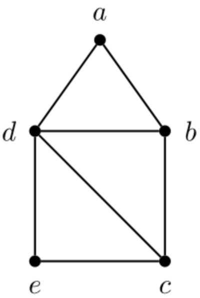
A rooted tree decomposition $(T, \{X_t : t \in T\})$ of a graph G is nice if every node $t \in V(T) \setminus$ root is of one of the following four types:

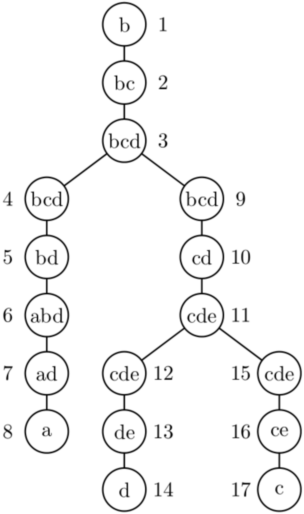
- Leaf: no children and $|X_t| = 1$.
- Introduce: a unique child t' and $X_t = X_{t'} \cup \{v\}$ with $v \notin X_{t'}$.
- Forget: a unique child t' and $X_t = X_{t'} \setminus \{v\}$ with $v \in X_{t'}$.
- Join: two children t_1 and t_2 with $X_t = X_{t_1} = X_{t_2}$.

Theorem (Bodlaender, Kloks, 1996)

A tree decomposition $(T, \{X_t : t \in T\})$ of width $\mathrm{tw}(G)$ of an n-vertex graph G can be transformed in time $\mathcal{O}\left(\mathrm{tw}(G)^2 \cdot n\right)$ into a nice tree decomposition of G of width $\mathrm{tw}(G)$ and $4n$ nodes.

Nice tree decomposition





The algorithm

- Take a nice, rooted tree decomposition and compute the following information for each vertex $t \in V(T)$ in a bottom up order.

The algorithm

- Take a nice, rooted tree decomposition and compute the following information for each vertex $t \in V(T)$ in a bottom up order.
- V_t: all vertices of G appearing in bags that are descendants of t
- $G_t := G[V_t]$

The algorithm

- Take a nice, rooted tree decomposition and compute the following information for each vertex $t \in V(T)$ in a bottom up order.
- V_t: all vertices of G appearing in bags that are descendants of t
- $G_t := G[V_t]$
- $\textsc{Mnc}(t, s, Q, \mathcal{P})$: the maximum number of components of $G_t - S$ where the maximum is taken for all sets $S \subseteq V_t$ having

The algorithm

- Take a nice, rooted tree decomposition and compute the following information for each vertex $t \in V(T)$ in a bottom up order.
- V_t: all vertices of G appearing in bags that are descendants of t
- $G_t := G[V_t]$
- $\textsc{Mnc}(t, s, Q, \mathcal{P})$: the maximum number of components of $G_t - S$ where the maximum is taken for all sets $S \subseteq V_t$ having
  - $|S| = s$,

The algorithm

- Take a nice, rooted tree decomposition and compute the following information for each vertex $t \in V(T)$ in a bottom up order.
- V_t: all vertices of G appearing in bags that are descendants of t
- $G_t := G[V_t]$
- $\textsc{Mnc}(t, s, Q, \mathcal{P})$: the maximum number of components of $G_t - S$ where the maximum is taken for all sets $S \subseteq V_t$ having
  - $|S| = s$,
  - $S \cap X_t = Q$, and

The algorithm

- Take a nice, rooted tree decomposition and compute the following information for each vertex $t \in V(T)$ in a bottom up order.
- V_t: all vertices of G appearing in bags that are descendants of t
- $G_t := G[V_t]$
- $\textsc{Mnc}(t, s, Q, \mathcal{P})$: the maximum number of components of $G_t - S$ where the maximum is taken for all sets $S \subseteq V_t$ having
  - $|S| = s$,
  - $S \cap X_t = Q$, and
  - $\mathcal{P}$ is the partition of $X_t - Q$ ($= X_t - S$) that is the partition of $G_t - S$ to components restricted to $X_t - Q$.

The algorithm

- Take a nice, rooted tree decomposition and compute the following information for each vertex $t \in V(T)$ in a bottom up order.
- V_t: all vertices of G appearing in bags that are descendants of t
- $G_t := G[V_t]$
- $\textsc{Mnc}(t, s, Q, \mathcal{P})$: the maximum number of components of $G_t - S$ where the maximum is taken for all sets $S \subseteq V_t$ having
  - $|S| = s$,
  - $S \cap X_t = Q$, and
  - $\mathcal{P}$ is the partition of $X_t - Q$ ($= X_t - S$) that is the partition of $G_t - S$ to components restricted to $X_t - Q$.
- For every t compute $\textsc{Mnc}(t, s, Q, \mathcal{P})$ for each possible value of $0 \leq s < n$, $Q \subseteq X_t$ and $\mathcal{P}$ using the previously computed info for the child/children of t.

The algorithm

- Take a nice, rooted tree decomposition and compute the following information for each vertex $t \in V(T)$ in a bottom up order.
- V_t: all vertices of G appearing in bags that are descendants of t
- $G_t := G[V_t]$
- $\textsc{Mnc}(t, s, Q, \mathcal{P})$: the maximum number of components of $G_t - S$ where the maximum is taken for all sets $S \subseteq V_t$ having
  - $|S| = s$,
  - $S \cap X_t = Q$, and
  - $\mathcal{P}$ is the partition of $X_t - Q$ $(= X_t - S)$ that is the partition of $G_t - S$ to components restricted to $X_t - Q$.
- For every t compute $\textsc{Mnc}(t, s, Q, \mathcal{P})$ for each possible value of $0 \leq s < n$, $Q \subseteq X_t$ and $\mathcal{P}$ using the previously computed info for the child/children of t.
- The total size of information for one vertex of t is $\mathcal{O}\left(n \cdot \mathrm{tw}(G)^{\mathrm{tw}(G)}\right)$.

The algorithm

- For the root r of the tree compute:

$$\tau(G) = \min\left\{ \frac{s}{\textsc{Mnc}(r, s, Q, \mathcal{P})} \,\middle|\, 0 \le s < n; \textsc{Mnc}(r, s, Q, \mathcal{P}) \ge 2 \right\}$$

MŰEGYETEM 1782

How to compute $\textsc{Mnc}(t, s, Q, \mathcal{P})$?

- Leaf: trivial
- Forget: easy

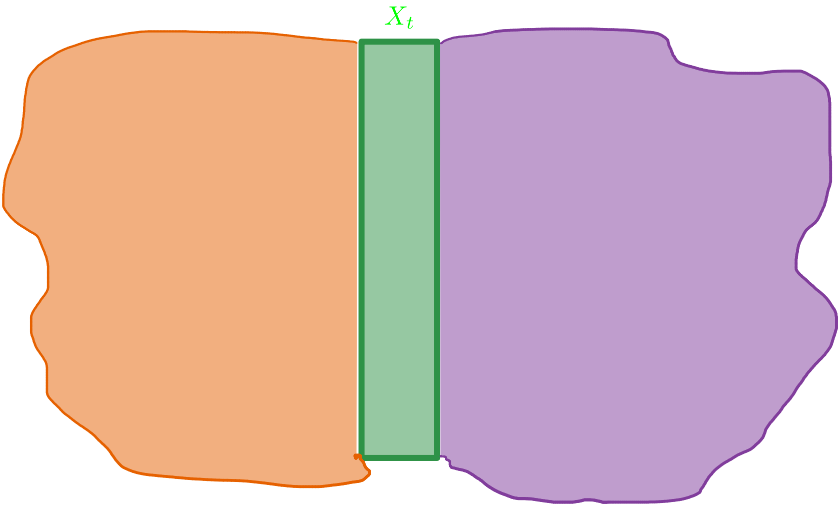
How to compute $\textsc{Mnc}(t, s, Q, \mathcal{P})$?

- Leaf: trivial
- Forget: easy
- Introduce: harder

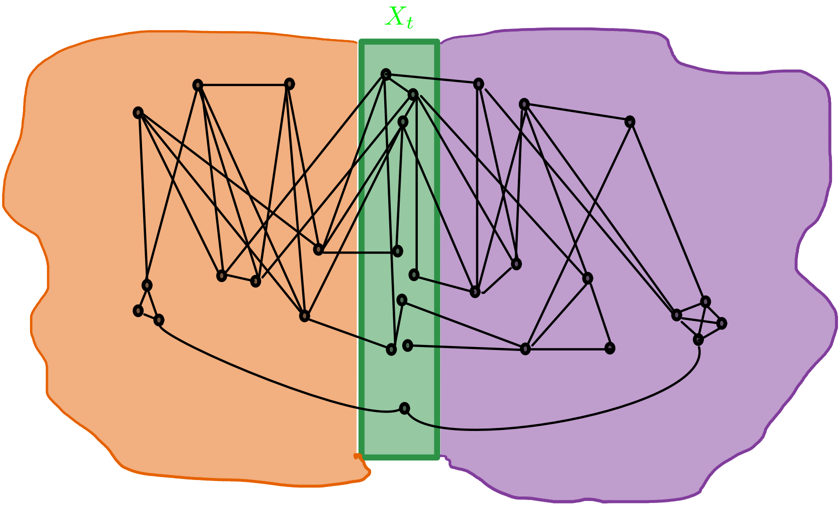
How to compute $\textsc{Mnc}(t, s, Q, \mathcal{P})$?

- Leaf: trivial
- Forget: easy
- Introduce: harder
- Join: hardest case

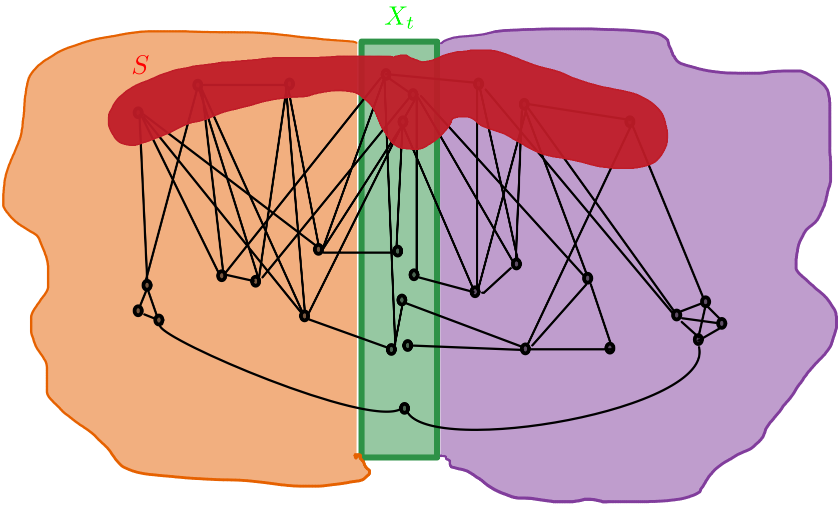
How to compute for join?



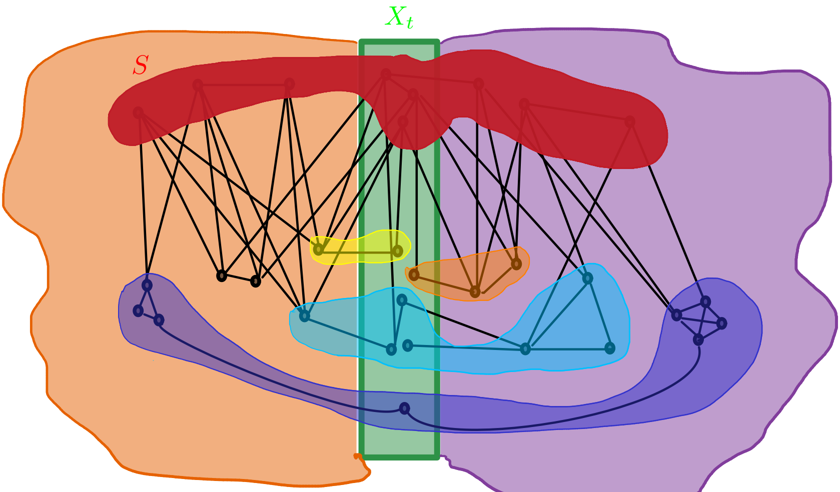
How to compute for join?



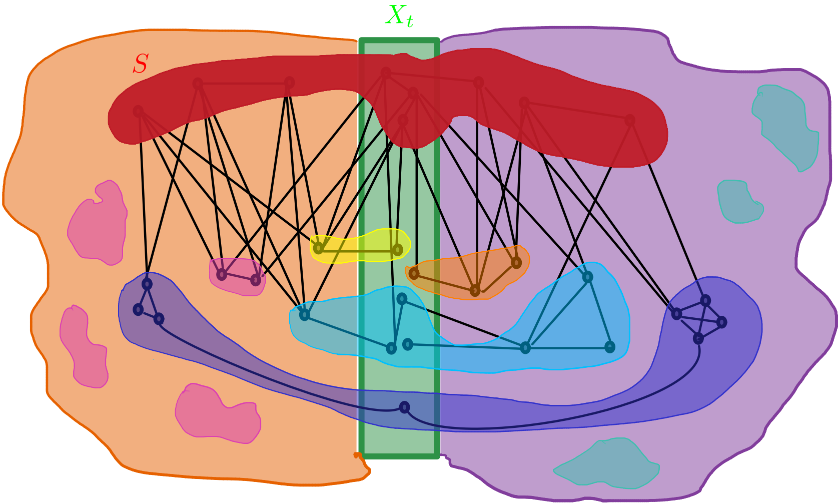
How to compute for join?



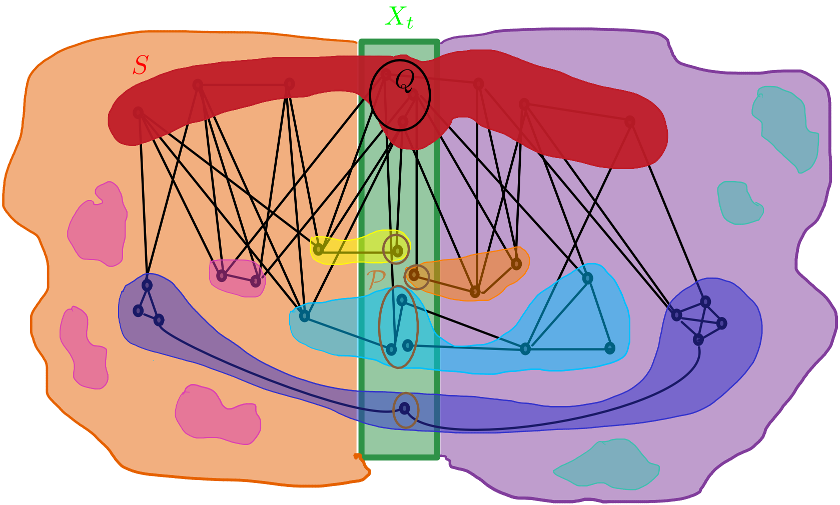
How to compute for join?



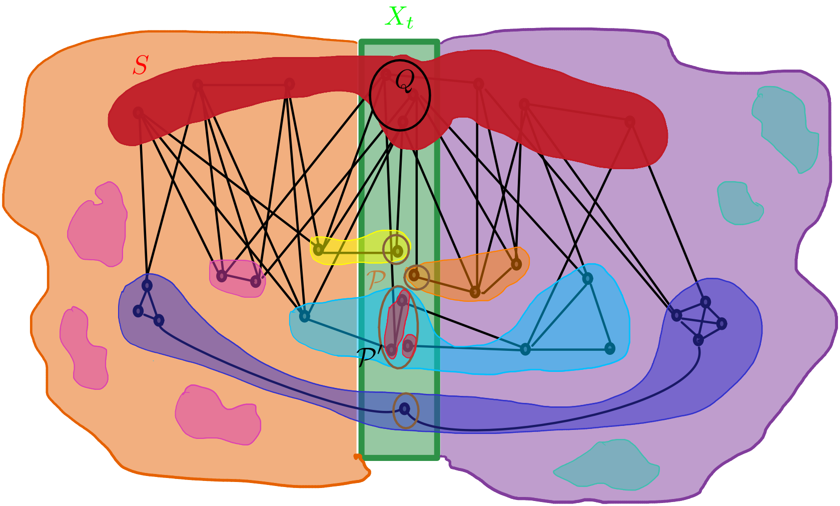
How to compute for join?



How to compute for join?



How to compute for join?



Running time

- Number of vertices in the tree: $\mathcal{O}(n)$
- Computing for Leafs: $\mathcal{O}(1)$
- Computing for Introduce, Forget: $\mathcal{O}\left(n \cdot \mathrm{tw}(G)^{\mathrm{tw}(G)}\right)$
- Computing for Join: $\mathcal{O}\left(n^2 \cdot \mathrm{tw}(G)^{2\mathrm{tw}(G)}\right)$
- Computing at the end: $\mathcal{O}\left(n \cdot \mathrm{tw}(G)^{\mathrm{tw}(G)}\right)$

Running time

- Number of vertices in the tree: $\mathcal{O}(n)$
- Computing for Leafs: $\mathcal{O}(1)$
- Computing for Introduce, Forget: $\mathcal{O}\left(n \cdot \mathrm{tw}(G)^{\mathrm{tw}(G)}\right)$
- Computing for Join: $\mathcal{O}\left(n^2 \cdot \mathrm{tw}(G)^{2\mathrm{tw}(G)}\right)$
- Computing at the end: $\mathcal{O}\left(n \cdot \mathrm{tw}(G)^{\mathrm{tw}(G)}\right)$

Running time: $\mathcal{O}\left(n^3 \cdot \mathrm{tw}(G)^{2\mathrm{tw}(G)}\right)$

Running time

- Number of vertices in the tree: $\mathcal{O}(n)$
- Computing for Leafs: $\mathcal{O}(1)$
- Computing for Introduce, Forget: $\mathcal{O}\left(n \cdot \mathrm{tw}(G)^{\mathrm{tw}(G)}\right)$
- Computing for Join: $\mathcal{O}\left(n^2 \cdot \mathrm{tw}(G)^{2\mathrm{tw}(G)}\right)$
- Computing at the end: $\mathcal{O}\left(n \cdot \mathrm{tw}(G)^{\mathrm{tw}(G)}\right)$

Running time: $\mathcal{O}\left(n^3 \cdot \mathrm{tw}(G)^{2\mathrm{tw}(G)}\right)$

Conjecture

There exists an algorithm to compute the toughness of a graph G width running time $\mathcal{O}\left(n^2 \cdot 2^{\mathcal{O}(\mathrm{tw}(G))}\right)$.

Running time

- Number of vertices in the tree: $\mathcal{O}(n)$
- Computing for Leafs: $\mathcal{O}(1)$
- Computing for Introduce, Forget: $\mathcal{O}\left(n \cdot \mathrm{tw}(G)^{\mathrm{tw}(G)}\right)$
- Computing for Join: $\mathcal{O}\left(n^2 \cdot \mathrm{tw}(G)^{2\mathrm{tw}(G)}\right)$
- Computing at the end: $\mathcal{O}\left(n \cdot \mathrm{tw}(G)^{\mathrm{tw}(G)}\right)$

Running time: $\mathcal{O}\left(n^3 \cdot \mathrm{tw}(G)^{2\mathrm{tw}(G)}\right)$

Conjecture

There exists an algorithm to compute the toughness of a graph G width running time $\mathcal{O}\left(n^2 \cdot 2^{\mathcal{O}(\mathrm{tw}(G))}\right)$.

I believe that the methods invented by Bodlaender, Cygan, Kratsch and Nederlof (2013) will work here, too.

Open questions

Question

What is the complexity of t-TOUGH for

- *chordal graphs?*
- *planar graphs?*

The End