Treewidth

Simon Rey and Ronald de Haan

June Project

Institute for Logic, Language and Computation
University of Amsterdam

Further Parameterized Results

During the first lecture we presented several techniques to develop `FPT` algorithms. Today we will focus on a specific parameter that has been extensively studied in the world of parameterized complexity: the *treewidth* of a graph.

After giving some basic definitions, we will present some algorithmic results about computing the treewidth of a graph. Some examples on how to use such a parameter will then be given. The lecture will be concluded by two further topics about treewidth.

Shall we?

Motivation

Graphs are among the most extensively studied mathematical structures in computer science. Among the famous *21 NP-complete problems* that Richard Karp introduced in his seminal paper in 1972, 11 of them are about graphs.

Although most problems on graphs turn out to be hard to solve (e.g., NP-complete), many are solvable in polynomial time when the input graph is a *tree*.

↪ As newly appointed parametrized complexity expert which bell should that ring?

If some problems that are hard in general are easy on trees, let's try to parameterize them with a measure of how close to a tree the input graph is. That is the *treewidth*.

1. Tree Decomposition of a Graph and Treewidth

Tree Decomposition of a Graph and Treewidth

Definition

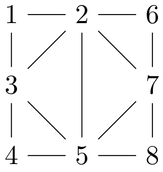
Tree Decomposition

Definition: Tree Decomposition

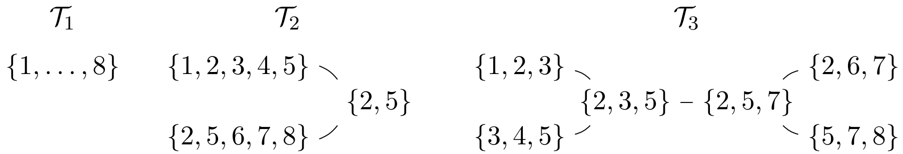
A tree decomposition of a graph $G = \langle V, E \rangle$ is a *tree* $\mathcal{T} = \langle V_\mathcal{T}, E_\mathcal{T} \rangle$ whose vertices, called *bags*, are subsets of vertices of G ($V_\mathcal{T} \subseteq 2^V$) such that:

- Every vertex of G appear in at least on bag: $\bigcup_{X \in V_\mathcal{T}} X = V$;
- For every edge $(u, v) \in E$, there exists a bag $X \in V_\mathcal{T}$ for which $\{u, v\} \subseteq X$;
- For every vertex $v \in V$, the set $\{X \in V_\mathcal{T} \mid v \in X\}$ is a connected subtree of $\mathcal{T}$.

A graph G:



Three possible tree decompositions of G:



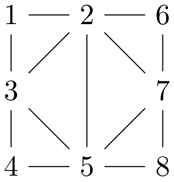
Treewidth

As illustrated on the previous slide, to every graph correspond (potentially) several tree decompositions. Some decompositions are *better* than others, in particular when they involve *small bags*. That is the idea behind the *treewidth* of a graph.

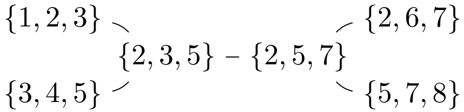
<u>Definition</u>: Treewidth

The *treewidth* of a graph G, denoted $tw(G)$ is the minimum over all tree decompositions of G of the size of the largest bag minus 1 in the tree decomposition. Formally:

$$tw(G) = \min \left\{ \max_{X \in V_\mathcal{T}} |X| - 1 \mid \mathcal{T} = \langle V_\mathcal{T}, E_\mathcal{T} \rangle \text{ is a tree decomposition of } G \right\}.$$



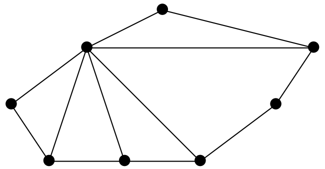
The graph G has treewidth $tw(G) = 3$ with the following optimal tree decomposition.



Classes with Bounded Treewidth

Let's play: I give you a class of graphs, you me give a nice bound on its treewidth!

- Trees? 1
- Forests? 1
- Cycle? 2
- Complete graphs of size n? $n - 1$
- Outerplanar graphs? 2



Tree Decomposition of a Graph and Treewidth

Computing a Tree Decomposition

An FPT Algorithm for the Treewidth

Let's focus a bit on the problem of computing a tree decomposition.

Unsurprisingly, computing a tree decomposition amount to solving NP-hard problems.

PROPOSITION:

Deciding whether G has treewidth $tw(G) \leq k$, for a given $k \in \mathbb{N}$, is NP-complete.

What's the reflex now when you see such a result? Let's look for FPT algorithms!

THEOREM: BODLAENDER ALGORITHM'S

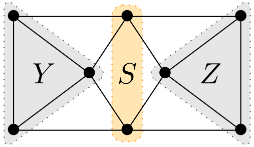
There exists an algorithm taking as input a graph $G = \langle V, E \rangle$ and an integer $k \in \mathbb{N}$ that runs in time $k^{\mathcal{O}(k^3)} \times |V|$ and that either construct a tree decomposition of G of *width at most k* or concludes that $tw(G) > k$. $(3^{27} = 7\ 625\ 597\ 484\ 987)$

Approximation algorithms have been proposed with different trade-off in the exponents on k and $|V|$ such as: $3 + 2/3$-approx. in $\mathcal{O}(2^{3.6982k} k^3 \times |V|^2)$ or 5-approx. in $2^{\mathcal{O}(k)} \times |V|$.

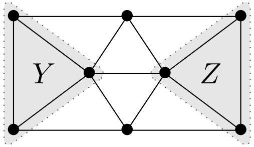
An Attempt at Explaining How - An Obstacle to Small Treewidth

All algorithms computing tree decompositions are quite involved and long to explain. Let's sketch the structure of one of them for a graph $G = \langle V, E \rangle$.

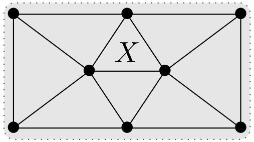
First, we need to find what are interesting *obstacle* to having a small treewidth. Two sets of vertices $Y, Z \subseteq V$ of the same size are *separable* if there is a $S \subseteq V$ of size $|S| < |Y| = |Z|$ such that there are no path from $Y \setminus S$ to $Z \setminus S$ when removing S from the graph. A set $X \subseteq V$ is *k-linked* if $|X| \geq k$ and X contains no separable subsets of size at most k.



S separates Y and Z



Y and Z are not separable



X is 3-linked

PROPOSITION:

If G contains a $(k+1)$-linked set of size at least $3k$, then G has treewidth at least k.

An Attempt at Explaining How - Idea of the Algorithm

The algorithm we sketch will then ensure the following two points:

- Either it finds a tree decomposition of width less than $4k$
- Or, it discovers a $(k+1)$-linked set of size at least $3k$ witnessing that G has $tw(G) \geq k$.

Good news: there exists an algorithm running in `FPT` time that decides whether a set of vertices $X \subseteq V$ is $(k+1)$-linked and which produces a witness showing why not otherwise.

We will then construct a recursive algorithm that will break down the graph into several bags that will form the tree decomposition. If at any steps it cannot find a suitable bag, it will discover a $(k+1)$-linked set of size at least $3k$.

An Attempt at Explaining How - A Recursive Procedure

At a given step, the recursive process separates the graph G into several connected components by removing a set of vertices (that are placed in a bag). The two following facts are maintained at any intermediate step of the algorithm:

- The current solution is a tree decomposition for the subgraph of G induced by U defined as the union of all the bags that have been formed so far;
- Each connected component C of the graph induced by $V \setminus U$ has at most $3k$ neighbors in U and there is a single bag V_t containing all the neighbors.

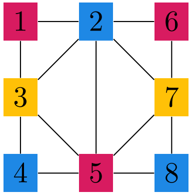
What is happening inside an intermediate step is roughly the following:

- Consider a given connected component C of the graph induced by $V \setminus U$;
- Call X the set of neighbors of C (vertices with an edge linking to X);
- When $|X| < 3k$ there is an easy way to extend the tree decomposition;
- Otherwise, if $|X| = 3k$ we need to test whether X is $(k+1)$-linked;
- If so, we can terminate the algorithm as we know $tw(G) \geq k$;
- Otherwise, we extend the tree decomposition with the witness that it is not the case.

2. Treewidth and Dynamic Programming

3-Colorability

3-Colorability	
Instance:	A graph $G = \langle V, E \rangle$
Parameter:	$k = tw(G)$ the treewidth of G
Question:	Is there a mapping $c : V \to \{1, 2, 3\}$ such that for all $v_1, v_2 \in V$, $v_1 \neq v_2$,we have $c(v_1) \neq c(v_2)$?



Let's show that the 3-Colorability problem is fixed-parameter tractable when parameterized with the treewidth of the input graph.

Proposition:

For every graph $G = \langle V, E \rangle$, the 3-Colorability problem can be soled in time $2^{k^{\mathcal{O}(1)}} \times |V|$ where $k = tw(G)$ if the treewidth of G.

3-Colorability - The Algorithm

We devise a *dynamic programming* algorithm for 3-Colorability. For an input graph $G = \langle V, E\rangle$ of treewidth $k = tw(G)$, the first step consists in computing a tree decomposition $\mathcal{T} = \langle V_\mathcal{T}, E_\mathcal{T}\rangle$ of width at most k using Bodleander's algorithm.

We then recursively compute partial solutions of the 3-Colorability going from the leaves of $\mathcal{T}$ to its root. For any vertex $X \in V_\mathcal{T}$ we compute two sets:

- $Col(X)$: the set of all 3-coloring of the subgraph of G induced by the vertices in X;

↪ Can be computed in time $\mathcal{O}(3^{k+1} \times k^2)$ by going through all possible 3-colorings.

- $ExtCol(X)$: the set of all 3-coloring in $Col(X)$ that can be extended into a 3-coloring of the subgraph of G induced by the vertices in X and in all of its children in $\mathcal{T}$.

↪ If X is a leaf of $\mathcal{T}$, then $ExtCol(X) = Col(X)$. Otherwise, $ExtCol(X)$ is the set of all the $c \in Col(X)$ such that for all children X_i of X in $\mathcal{T}$, there exists $c_i \in ExtCol(X_i)$ that coincides with c on vertices $X_i \cap X$. This can be computed in time $\mathcal{O}(3^{2(k+1)} \times k)$.

Since $|V_\mathcal{T}| \leq |V|$, the overall running is $2^{k^{\mathcal{O}(1)}} \times |V|$. ■

Treewidth and Dynamic Programming

The algorithm we have presented in the previous example can be converted into a general strategy to proof `FPT` results when parameterized by the treewidth. The steps are:

- Computing a tree decomposition using Bodleander's algorithm;
- Developing a dynamic programming technique to obtain partial solutions for each bag in the tree;
- Finding an efficient way to combine the partial solutions into a global one.

This can be turned into the so called *win/win* approach. Can't wait to know what this is it right?

3. The Win/Win Approach: Treewidth Reductions

In Both Cases, You Win

Through the use of dynamic programming, we presented a nice technique to develop `FPT` algorithms for instance of bounded treewidth. Let's explore this idea further with the *win/win* approach.

This approach is based on the idea that for many problems, *large treewidth implies trivial instances*. In both cases—small and large treewidth—we thus have an efficient way of solving the problem. This approach can be summarized this way:

- Compute the treewidth of the input;
- If the treewidth is small, use the dynamic programming approach that we discussed;
- If the treewidth is large, the graph probably has an underlying structure (such as a grid), exploit it to trivially solve the problem.

↳ Let's see a simple example of that!

The Famous BAR FIGHT PREVENTION problem

VERTEX COVER	
Instance:	A graph $G = \langle V, E \rangle$ and an integer $k \in \mathbb{N}$
Parameter:	$tw(G)$ the treewidth of G
Question:	Is there a set of vertices $S \subseteq V$ of size $\|S\| \leq k$ such that for all edges $(u, v) \in E$, either $u \in S$ or $V \in S$?

What can we say about the treewidth of a graph G that has a vertex cover S of size k? *It is at most k*: a path in which every bag consists in S plus a vertex outside of S is a tree decomposition of G.

Let's then do the following: First, check whether G has *treewidth at most k* using Bodleander's algorithm. If not, *reject* the instance. If G has treewidth at most k, use a *dynamic programming algorithm* that can solve VERTEX COVER for G in time $2^k \times k^{\mathcal{O}(1)} \times |V|$. I won't present the algorithm here, but it exists, I promise you.

The overall procedure runs in FPT *time*. ■

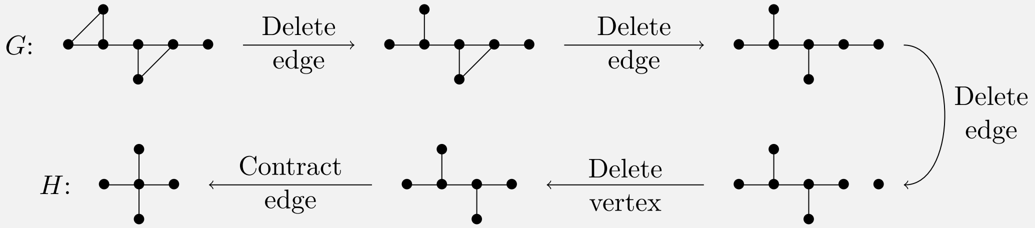
Graph Minors

Efficiently applying this technique relies on understanding the implications that a large treewidth has on the structure of a graph. This can be studied via *graph minors*.

A minor H of a graph G is a graph that can be obtained using the following operations:

- *Deleting* some edges of G;
- *Deleting* some vertices of G;
- *Contracting* some edges of G: merging two connected vertices.

<u>Example</u>: The following sequence of operations shows that H is a minor of G.



Excluded Grid Theorem

One of the most important result (coming soon!) in that area states that graphs with large treewidth contain a grid as a minor. But what is a grid?

For any positive integer $t \in \mathbb{N}_{>0}$, a *$t \times t$ grid*, denoted by $\boxplus_t$, is a graph with vertices $\{(x, y) \mid x, y \in \{1, \ldots, t\}\}$. It can be shown that the grid $\boxplus_t$ has treewidth $tw(\boxplus_t) = t$.

Since $tw(\boxplus_t) = t$, any graph containing $\boxplus_t$ as a minor must have treewidth at least t. More surprising, the conserve also holds.

Theorem: Excluded Grid Theorem

There exists a function $g(t)$ that is $\mathcal{O}(t^{98+o(1)})$ such that every graph of treewidth larger than $g(t)$ contains $\boxplus_t$ as a minor.

The bound has been successively improved to $\mathcal{O}(t^{36+o(1)})$, $\mathcal{O}(t^{19+o(1)})$ and lately $\mathcal{O}(t^{9+o(1)})$.

This theorem yielded many deep algorithmic results that are way beyond the scope of this project. Just keep in mind that it is an active line of work!

4. Game Theoretic Characterization of Treewidth

The Robber Game

The k-robber game, played on a graph $G = \langle V, E \rangle$, has the following rules:

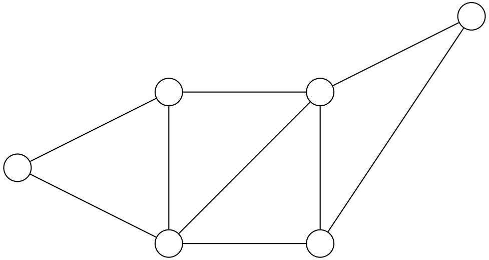
- There are k cops and a robber ;
- At the beginning, the k cops are placed G, the robber then selects a starting position.
- During a turn, a subset of cops take a helicopter and announce on which vertices they will land (one for each of them;
- After the lift off, the robber can move along the edges of G but cannot go through a vertex occupied by a cop;
- Once the robber has selected his next position, the cops land and a new turn starts;
- The positions of the robber is always known by the cops;
- The cops win if one of them land on the position of the robber;
- The robber wins if she can avoid the cops indefinitely.

An Example of the Game - Initial State

Beginning of the game

Three cops

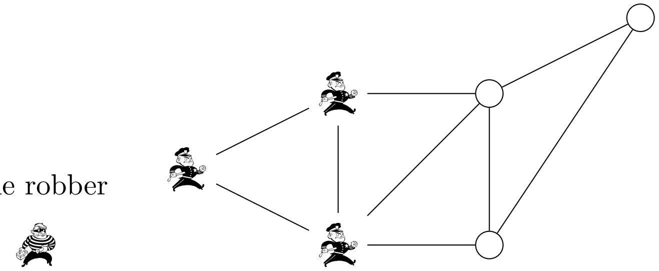
One robber



An Example of the Game - Starting Positions for the Cops

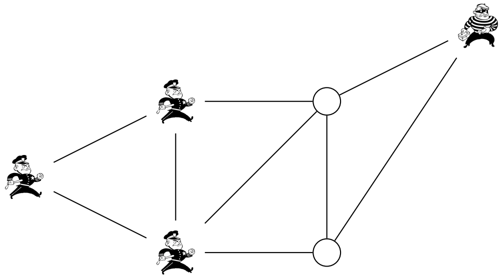
The cops select their initial position

One robber



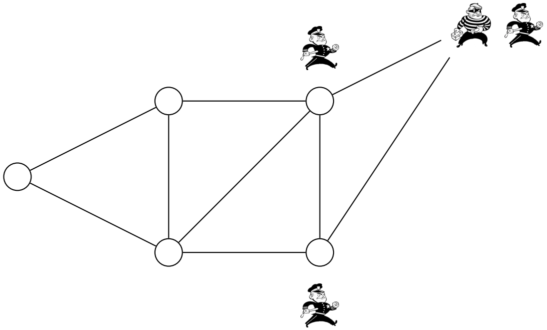
An Example of the Game - Starting Positions for the Robber

The robber tries to go as far as possible from them



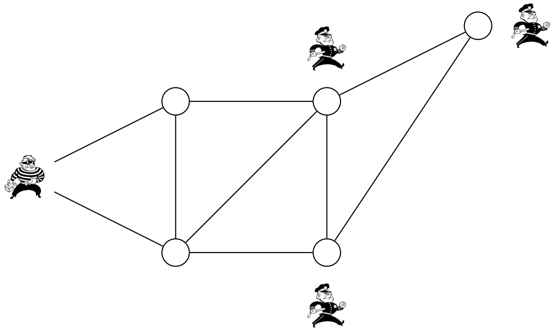
An Example of the Game - First Round Helicopter Lift Off

The cops rush to the robber



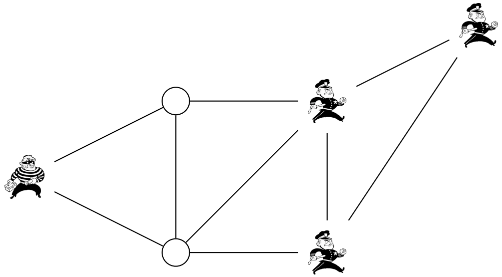
An Example of the Game - First Round Robber Move

The robber runs away as far as she can



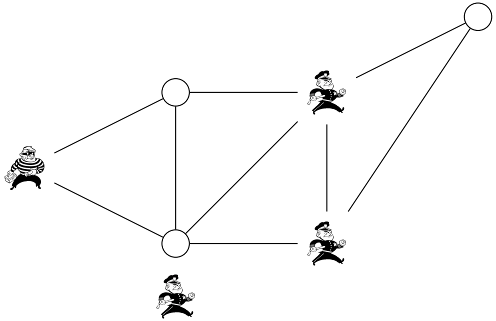
An Example of the Game - End of the First Round

Cops land and discover the robber escaped



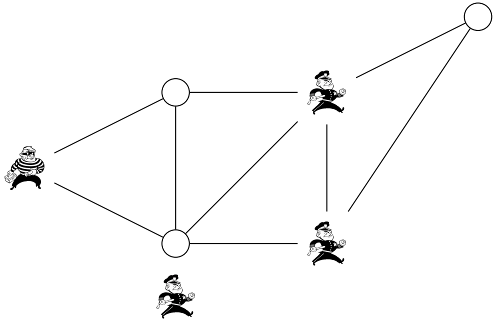
An Example of the Game - Second Round Helicopter Lift Off

The cops hired a graph theorist, he is now helping them



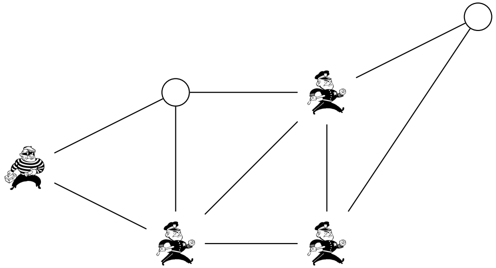
An Example of the Game - Second Round Robber Move

The Robber is afraid by such a tactical move, he freezes



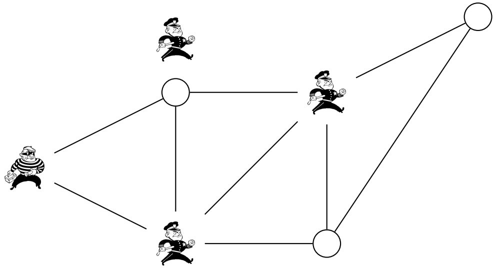
An Example of the Game - End of the Second Round

The Robber is afraid by such a tactical move, he freezes



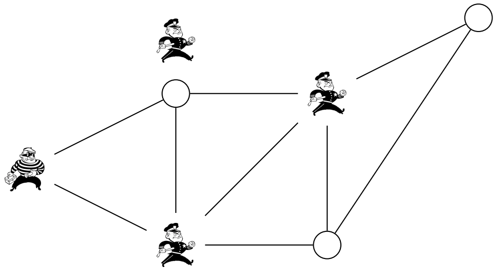
An Example of the Game - Third Round Helicopter Lift Off

Still moving cleverly



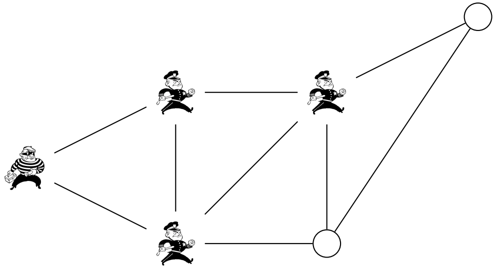
An Example of the Game - Third Round Robber Move

He is trapped, what can he do???



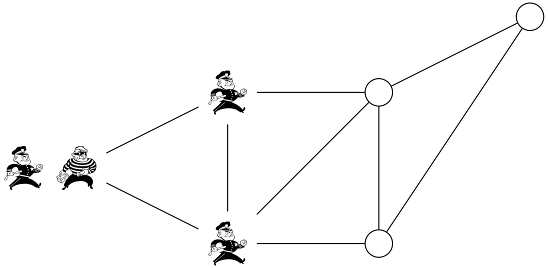
An Example of the Game - End of the Third Round

The cops are getting closer!



An Example of the Game - End of the Forth Round

The cops catch the robber and win the game.



A New Characterization of Treewidth

Apart from a nice animation, what is the added value of this game?

<u>Theorem</u>:

The two following facts are equivalent for a given graph G and $k \in \mathbb{N}$:

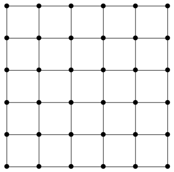
- The treewidth of G is $tw(G) \leq k$;
- There exists a winning strategy for $k + 1$ cops on the robber game on G.

Because the graph in the previous example is outerplanar, I was sure the cops could win. Indeed, outerplanar have treewidth 2, hence 3 cops are enough to catch the robber.

Let's see how we can use this characterization to find bounds on the treewidth of a grid.

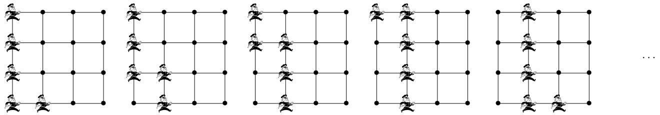
Upper Bound of the Treewidth of Grids

Recall that a $t \times t$ grid looks like that (for $t = 6$):



➥ What would be an efficient strategy for $t + 1$ cops?

With $t + 1$ cops, the grid can be searched column per column with the following strategy:

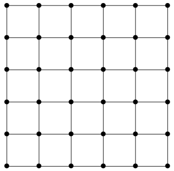


Proposition:

The treewidth of a $t \times t$ grid $\boxplus_t$ is at most t.

Lower Bound of the Treewidth of Grids

Let's look for lower bounds now!



➥ What would be an efficient strategy for $t-1$ cops?

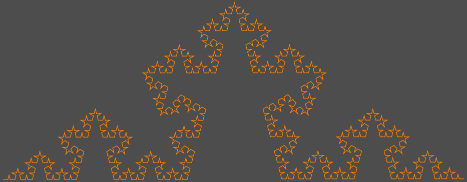
Always moving on a cell at the intersection of a column and row both without any cops. Such a cell always exists since there are not enough cops to fully cover a column or a row.

PROPOSITION:

The treewidth of a $t \times t$ grid $\boxplus_t$ is at least $t-1$.

Our two results show that $t-1 \leq tw(\boxplus_t) \leq t$. We actually know that $tw(\boxplus_t) = t$ but the proof is much harder.

5. Conclusion

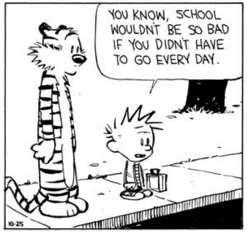


Summary

Today, we have:

- Introduced the idea of tree decomposition which yielded the definition of the treewidth;
- Studied how one can efficiently compute the treewidth of a given graph;
- Presented some algorithms working with the treewidth and the general underlying scheme for these algorithms;
- Studied the treewidth further by presenting the excluded grid theorem and another characterization through the robber game.

Next lecture will be devoted to the hardness theory of parameterized complexity: showing that some problems are not fixed-parameter tractable.


YOU KNOW, SCHOOL WOULDN'T BE SO BAD IF YOU DIDN'T HAVE TO GO EVERY DAY.
10-25