Social and Graph Data Management: Width Measures Theory

Pablo Mollá Chárlez

Contents

1 Introduction

Width measures play a fundamental role in graph theory and its applications, providing insights into the structural complexity of graphs. Among these, cutwidth, pathwidth, and treewidth are key measures that quantify how closely a graph resembles simpler structures, such as trees or paths. These measures can **guide the development of efficient algorithms for complex problems**.

- **Treewidth** quantifies how "tree-like" a graph is, with lower treewidth indicating a closer resemblance to a tree. This measure, referenced in over 32,000 publications on Google Scholar, is a cornerstone of the Graph Minors Project by Robertson and Seymour. It plays a critical role in identifying graphs where computational problems, otherwise challenging, can be solved more efficiently.
- **Pathwidth** refines this idea, assessing how "path-like" a graph is. With around 17,000 references in scholarly work, it is particularly relevant in scenarios requiring linear representations, such as scheduling or routing.
- **Cutwidth**, with approximately 8,000 references, offers a related perspective by focusing on the edges crossing a separator rather than the nodes within it.

Understanding and utilizing these measures is essential because many real-world graphs, including social networks, exhibit relatively low treewidth. This structural property makes them easier to algorithms that leverage the simplicity of trees, where such algorithms are more computationally tractable and precise, providing exact solutions.

2 Pathwidth

2.1 What Makes a Graph Path-Like?

A graph is considered path-like if its structure closely resembles a single path. The measure pathwidth(G) quantifies how much a path needs to be "thickened" to represent the graph G. The essential idea lies in the concept of **separators**: in a path-like graph, no edge directly connects vertices from the "left" side to the "right" side of the separator. This ensures that the graph can be decomposed into a sequence of overlapping subsets (bags), with each subset capturing the local connectivity.

2.2 Path-Decomposition

A path-decomposition of a graph $G = (V, E)$ is a sequence of subsets (or bags) $X_1, X_2, \ldots, X_r$, where $X_i \subseteq V$, satisfying the following conditions:

1. **Edge Coverage:** For every edge $\{u, v\} \in E$, there exists at least one bag X_i that contains both endpoints u and v.
2. **Contiguous Appearance:** For each vertex $v \in V$, the bags containing v form a contiguous subsequence $X_i, X_{i+1}, \ldots, X_j$.
3. **Vertex Coverage:** Each vertex $v \in V$ must appear in at least one bag.

2.2.1 Width of a Path-Decomposition

The width of a path-decomposition is calculated as:

$$\text{width} = \max_{1 \le i \le r} \{|X_i| - 1\},$$

where $|X_i|$ represents the size of bag X_i. The pathwidth of G is the minimum width over all possible path-decompositions of G:

$$\text{pathwidth}(G) = \min\{\text{width of } X_1, \ldots, X_r\}.$$

In this last formula, the mininum width from multiple possible decompositions is considered.

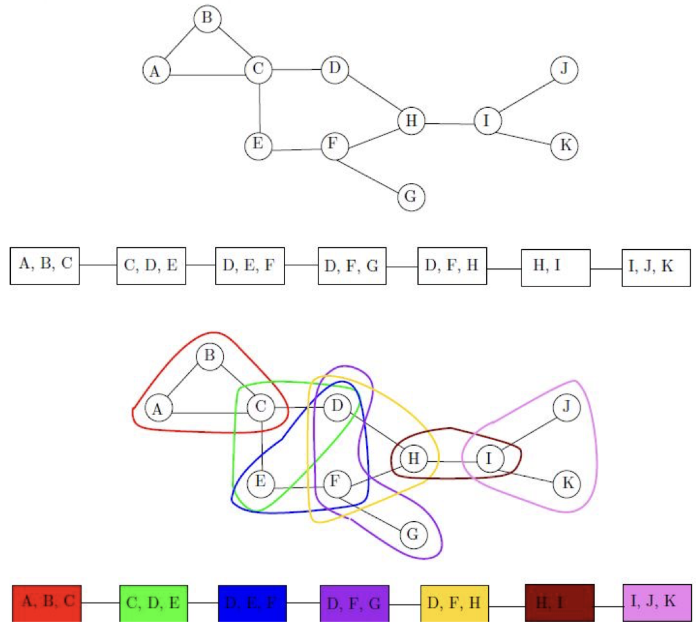

Figure 1: Path Decomposition of Width 2

In the previous image, we see a path-decomposition of a graph with width 2. The graph is divided into overlapping subsets $\{A, B, C\}, \{C, D, E\}, \ldots, \{I, J, K\}$. Each bag satisfies:

1. **Every edge is covered** by a bag, like the bag $\boxed{\text{A, B, C}}$ which covers $\{A, B\}, \{B, C\}$ and $\{A, C\}$. ✓
2. **Contiguous subsequences**: For example, vertex D appears in $\{C, D, E\}, \{D, E, F\}, \{D, F, G\}$, maintaining contiguity. Bear in mind, that between any 2 bags there must always be at least one "bridge" vertex. ✓
3. **Each vertex appears in at least one bag**. ✓

The path-decomposition illustrates how the graph is "thickened" to a simple path. **Another possible representation** of a path decomposition is using the **interval graphs**, where each vertex corresponds to an interval on a timeline. Path-decomposition aligns with these intervals:

- Each bag corresponds to overlapping intervals, ensuring edge coverage and contiguity.
- This visualization emphasizes how pathwidth relates to "linear-like" structures.

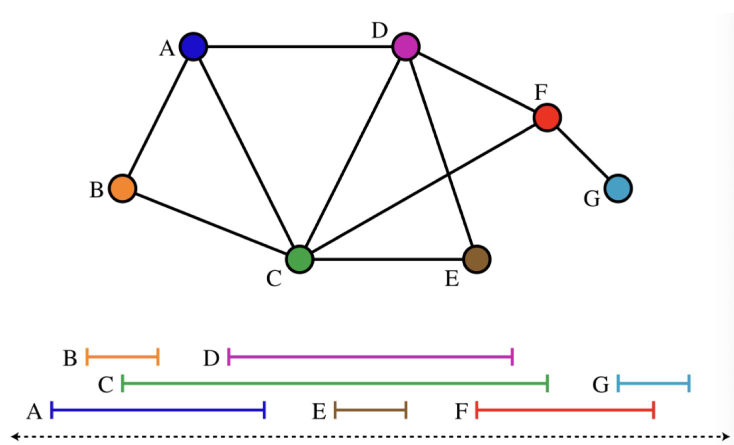

Figure 2: Interval Graph

2.2.2 Example of Path Decomposition

Let's consider a graph G and its path-decomposition with the following bags:

$$X_1 = \{A\}, \quad X_2 = \{A, B\}, \quad X_3 = \{B, C\}, \quad X_4 = \{B, C, D\}, \quad X_5 = \{C, D\}.$$

Let's check the contiguous appearance:

- **Vertex A:** Appears in X_1, X_2 (consecutive). ✓
- **Vertex B:** Appears in X_2, X_3, X_4 (consecutive). ✓
- **Vertex C:** Appears in X_3, X_4, X_5 (consecutive). ✓
- **Vertex D:** Appears in X_4, X_5 (consecutive). ✓

All vertices appear in contiguous bags, so this is a valid path-decomposition. For instance, an invalid path decomposition for the same graph would be:

$$X_1 = \{A\}, \quad X_2 = \{A, B\}, \quad X_3 = \{B, C\}, \quad X_4 = \{C\}, \quad X_5 = \{C, D\}, \quad X_6 = \{D, B\}.$$

Let's check the contiguous appearance:

- **Vertex A:** Appears in X_1, X_2 (consecutive). ✓
- **Vertex B:** Appears in X_2, X_3, X_6 but skips X_4 and X_5. ✗
- **Vertex C:** Appears in X_3, X_4, X_5 (consecutive). ✓
- **Vertex D:** Appears in X_4, X_5 (consecutive). ✓

The appearance of vertex B is not contiguous, so this decomposition is invalid.

2.3 Nice Path-Decomposition

A nice path-decomposition helps to simplify the structure of a path-decomposition by ensuring:

1. **Start and End Bags:** The first and last bags X_1 and X_r contain exactly one vertex ($|X_1| = |X_r| = 1$).
2. **Introduce and Forget Operations:**
   - Introduce: $X_{i+1} = X_i \cup \{v\}$, where a vertex v is added.
   - Forget: $X_{i+1} = X_i - \{v\}$, where a vertex v is removed.

This structure ensures consistency and makes algorithmic descriptions more straightforward, particularly in dynamic programming approaches.

2.3.1 Nice Path-Decomposition Example

A nice path-decomposition is a stricter version, with additional constraints. For the same previous example graph, a nice decomposition could be:

$$X_1 = \{A\}, \quad X_2 = \{A, B\}, \quad X_3 = \{B, C\}, \quad X_4 = \{C, D\}, \quad X_5 = \{D\}.$$

Here:

- X_1 (start) and X_5 (end) have one vertex each.
- $X_2 = X_1 \cup \{B\}$ (introduce B).
- $X_3 = X_2 \setminus \{A\} \cup \{C\}$ (forget A, introduce C).
- $X_4 = X_3 \setminus \{B\} \cup \{D\}$ (forget B, introduce D).
- $X_5 = X_4 \setminus \{C\}$ (forget C).

This satisfies all the conditions of a nice path-decomposition.

2.4 Pathwidth Challenges

Computing pathwidth is inherently difficult:

- Checking whether pathwidth$(G) \geq k$ is an NP-hard problem.
- However, for graphs with small k, approximations and exact computations become more feasible, often leveraging dynamic programming or heuristics.

3 Treewidth

3.1 Tree-Decomposition

A tree-decomposition of a graph $G = (V, E)$ consists of:

1. A tree T whose nodes are bags $(X_1, X_2, \ldots, X_r)$, where each bag X_i is a subset of V (the vertices of G).
2. The following conditions must hold:
   - **Edge Coverage:** For each edge $\{u, v\} \in E$, there must exist a bag X_i that contains both u and v.
   - **Contiguous Subtree:** For each vertex $v \in V$, the set of bags containing v forms a contiguous subtree in T. This ensures that the decomposition is consistent.
   - **Vertex Inclusion:** Every vertex in G must appear in at least one bag, although isolated vertices have no impact on treewidth.

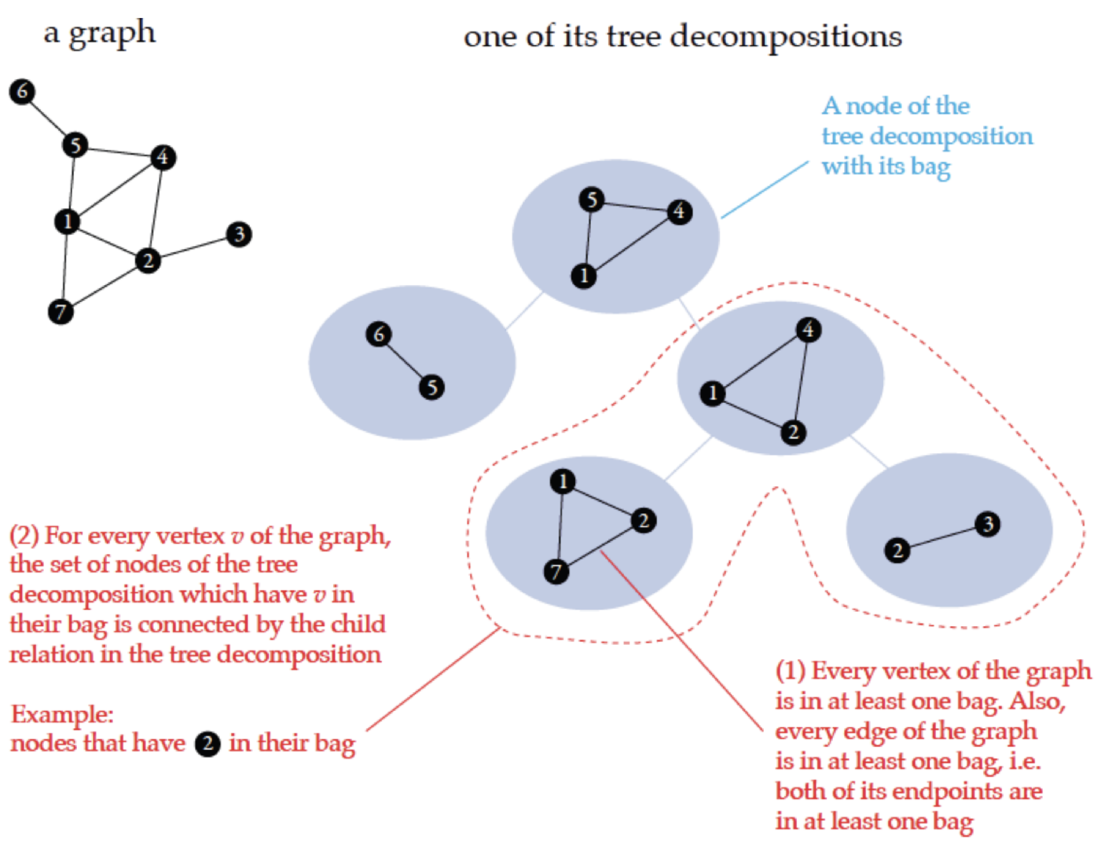

Figure 3: Tree Decomposition of Width 2

3.2 Width of a Tree-Decomposition

The width of a tree-decomposition is defined as:

$$\mathrm{width}(T) = \max_{1 \le i \le r} \{|X_i| - 1\},$$

i.e., the size of the largest bag minus one. The treewidth of G is the minimum width over all possible tree-decompositions of G.

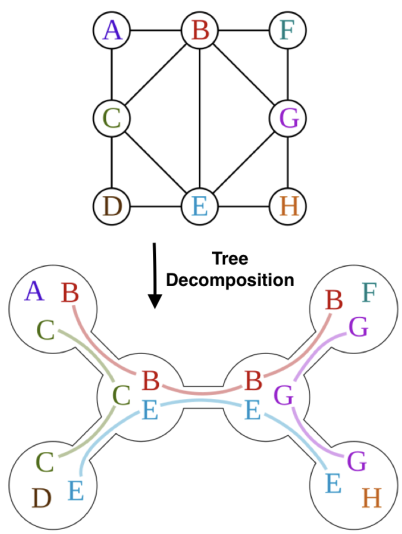

Figure 4: Tree Decomposition of Width 2

3.2.1 Example of Tree Decomposition

Consider the following graph:

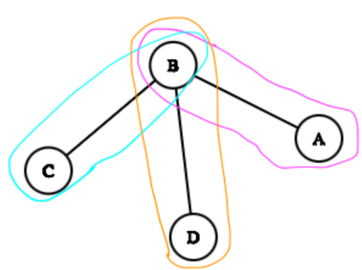

Figure 5: Simple Graph

One possible tree-decomposition consists of the following bags:

$$X_1 = \{A, B\}, \quad X_2 = \{B, C\}, \quad X_3 = \{B, D\}$$

These bags form a tree: X_1, X_2, X_3. Let's verify the conditions:

1. **Edge Coverage:** $\{A, B\} \in X_1$, $\{B, C\} \in X_2$, $\{B, D\} \in X_3$. ✓
2. **Contiguous Subtree:** **Vertex A** appears in X_1 only. **Vertex B** appears in X_1, X_2, X_3 (contiguous subtree). **Vertex C** appears in X_2 only. **Vertex D** appears in X_3 only. ✓

The largest bag size is 2, so the tree width is $2 - 1 = 1$.

3.3 Nice Tree-Decomposition

A nice tree-decomposition simplifies algorithms by introducing structured types of bags. There are four types of bags:

1. **Leaf Node:** A bag that contains exactly one vertex, e.g., $X = \{v\}$.
2. **Introduce Node:** Adds a vertex v to its child's bag: $X = Y \cup \{v\}$.
3. **Forget Node:** Removes a vertex v from its child's bag: $X = Y \setminus \{v\}$.
4. **Join Node:** Has two children, both with the same bag as X.

3.3.1 Nice Tree-Decomposition Example

For the previous simple graph 5, a nice tree-decomposition might look like this:

1. **Leaf:** $\{A\}$
2. **Introduce:** $\{A, B\}$
3. **Forget:** $\{B\}$
4. **Introduce:** $\{B, C\}$
5. **Forget:** $\{C\}$
6. **Introduce:** $\{B, D\}$
7. **Forget:** $\{D\}$

3.4 Properties of Treewidth

1. **Closed Under Graph Minors:** Graphs with treewidth $\leq k$ are closed under minors:
   - **Deleting Edges:** Reduces the graph's complexity.
   - **Deleting Vertices:** Also simplifies the graph.
   - **Contracting Edges:** Combines two vertices into one, preserving treewidth.

   If a graph G has treewidth $\leq k$, then any graph obtained by deleting edges, deleting vertices, or contracting edges will also have treewidth $\leq k$. These operations can reduce the treewidth but will never increase it. We call **minors** to the "smaller" version of the original graph G after having suffered a deletion (edges or vertices) and/or a contraction.

2. **Grid Treewidth:** Treewidth of $n \times n$ grid is n.
3. **Grid Minor Theorem:** Any graph with treewidth r has a grid minor of size $f(r)$, where $f(r)$ is a function dependent on r.
4. **Forbidden Minors:** Graphs of treewidth k can be characterized by a finite set of forbidden minors:
   - $k = 1$: Forbidden minor is a triangle.
   - $k = 2$: Forbidden minor is K_4 (complete graph with 4 vertices).

3.5 Treewidth Challenges

- Computing the treewidth of a graph is NP-hard, but approximation techniques and algorithms for small k are available.
- Practical algorithms often rely on heuristics to approximate treewidth efficiently.