Fast Algorithms for Rank-Width

Martin Beyß

RWTH Aachen University

2012-10-28
Structure

1. Overview
2. Width-Measures
3. An Upper Bound Algorithm
4. A Lower Bound Algorithm
5. Conclusion
1. Overview

2. Width-Measures

3. An Upper Bound Algorithm

4. A Lower Bound Algorithm

5. Conclusion
Tree-Width
- measures similarity to a tree
- low only on sparse graphs
- Courcelle’s Theorem: \( FPT \)-Algorithm for \( MSO_2 \) Model Checking

Clique-Width
- can be low on dense graphs
- equivalent to Rank-Width
- Courcelle’s Theorem for Clique-Width and \( MSO_1 \)
Decomposition of a Graph

\[ G = (V, E) \]
\[ V = \{a, b, c, d, e, f, g\} \]
\[ \overrightarrow{T} = (V_{\overrightarrow{T}}, E_{\overrightarrow{T}}) \]
\[ \mathcal{L} : V_{\overrightarrow{T}} \rightarrow 2^V \]

many possibilities
Decomposition of a Graph

- $G = (V, E)$
  - $V = \{a, b, c, d, e, f, g\}$
- $\vec{T} = (V_{\vec{T}}, E_{\vec{T}})$
- $\mathcal{L} : V_{\vec{T}} \rightarrow 2^V$

many possibilities
Decomposition of a Graph

- \( G = (V, E) \)
  - \( V = \{a, b, c, d, e, f, g\} \)
- \( \vec{T} = (V_{\vec{T}}, E_{\vec{T}}) \)
- \( \mathcal{L} : V_{\vec{T}} \rightarrow 2^V \)

\[
\begin{align*}
V & \rightarrow \{a, b, c\} & \{d, e, f, g\} \\
\{a, b\} & \rightarrow \{a\} & \{b\} & \{d\} & \{e\} & \{f\} & \{g\} \\
\{c\} & \rightarrow \\
\{d, e\} & \rightarrow \\
\{f, g\} & \rightarrow \\
\end{align*}
\]

many possibilities
Decomposition of a Graph

- \( G = (V, E) \)
  - \( V = \{a, b, c, d, e, f, g\} \)
- \( \vec{T} = (V_\vec{T}, E_\vec{T}) \)
- \( \mathcal{L} : V_\vec{T} \rightarrow 2^V \)

Many possibilities
Width of a Decomposition

- **Width Function** $f : 2^V \rightarrow \mathbb{R}$
- **Width of Decomposition**: $\max\{f(\text{Node label})\}$

Width of $f$: Minimum Width of all possible Decompositions
Width of a Decomposition

- **Width Function** $f : 2^V \rightarrow \mathbb{R}$
- **Width of Decomposition**: $\max \{ f(\text{Node label}) \}$

Width of $f$: Minimum Width of all possible Decompositions
Width of a Decomposition

- **Width Function** $f : 2^V \rightarrow \mathbb{R}$
- **Width of Decomposition**:
  \[ \max \{ f(\text{Node label}) \} \]

**Width of $f$**: Minimum Width of all possible Decompositions
**Rank-Width**

**Width of the cut-rank function** $cutrk(X)$

$$\begin{array}{cccccccc}
a & b & c & d & e & f & g \\
\hline
a & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
b & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
c & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
d & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
e & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
f & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
g & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\end{array}$$
Rank-Width

Width of the cut-rank function $cutrk(X)$

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>f</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>g</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Rank-Width

Width of the cut-rank function $cutrk(X)$

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>f</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>g</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Diagram:
```
  e  
 /   
 d---
     
  b  
 /   
 c---
     
  a  
    '
  f  
```

Table:
```
a 0 0 1 1 1 1 0
b 0 0 1 1 0 0 1
c 1 1 0 0 0 1 0
d 1 1 0 0 0 0 0
e 1 0 0 0 0 0 0
f 1 0 1 0 0 0 0
g 0 1 0 0 0 0 0
```
Rank-Width

Width of the cut-rank function $\text{cutrk}(X)$

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Diagram:

- $V$
- $\{a,b,c\}$
- $\{d,e,f,g\}$
- $\{a,b\}$
- $\{c\}$
- $\{d,e\}$
- $\{f,g\}$
- $\{a\}$
- $\{b\}$
- $\{d\}$
- $\{e\}$
- $\{f\}$
- $\{g\}$
Boolean-Width

**Neighbourhood Sets**

e
\[ a \]

d
\[ c \]

b

g

\[ \text{cut-bool} \]

\[ \text{cutbool}(X) = \log_2 | UN(X)| \]

expensive!
Boolean-Width

Neighbourhood Sets

\[ UN(\{a, b, c\}) \]

\[ cutbool(X) = \log_2 |UN(X)| \]

expensive!
Boolean-Width

Neighbourhood Sets

\[ UN(\{a, b, c\}) = \{\emptyset\} \]

\[ cutbool(X) = \log_2 |UN(X)| \]

expensive!
Boolean-Width

Neighbourhood Sets

\[ UN(\{a, b, c\}) = \{\emptyset, \{d, e, f\}\} \]

\[ cutbool(X) = \log_2 |UN(X)| \] expensive!
Boolean-Width

Neighbourhood Sets

\[ UN(\{a, b, c\}) = \{\emptyset, \{d, e, f\}, \{d, g\}\} \]

cut-bool

\[ cutbool(X) = \log_2 |UN(X)| \]

expensive!
Neighbourhood Sets

UN(\{a, b, c\}) = 
\{\emptyset, \{d, e, f\}, \{d, g\}, \{d, e, f, g\}\}

cutbool(X) = \log_2 |UN(X)|
expensive!
Boolean-Width

Neighbourhood Sets

\[ UN(\{a, b, c\}) = \{\emptyset, \{d, e, f\}, \{d, g\}, \{d, e, f, g\}, \{f\}\} \]

\[ cutbool(X) = \log_2 |UN(X)| \]

expensive!
Boolean-Width

Neighbourhood Sets

UN(\{a, b, c\}) =
\{\emptyset, \{d, e, f\}, \{d, g\},
\{d, e, f, g\}, \{f\}\}

\text{cutbool}(X) = \log_2 |UN(X)|

expensive!
Boolean-Width

Neighbourhood Sets

\[ UN(\{a, b, c\}) = \{\emptyset, \{d, e, f\}, \{d, g\}, \{d, e, f, g\}, \{f\}, \{d, f, g\}\} \]

\[ \text{cut-bool} \]

\[ \text{cutbool}(X) = \log_2 |UN(X)| \]

expensive!
**Boolean-Width**

**Neighbourhood Sets**

\[ UN(\{a, b, c\}) = \{\emptyset, \{d, e, f\}, \{d, g\}, \{d, e, f, g\}, \{f\}, \{d, f, g\}\} \]

\[ cutbool(X) = \log_2 |UN(X)| \]

expensive!
Boolean-Width

Neighbourhood Sets

\[ UN(\{a, b, c\}) = \{ \emptyset, \{d, e, f\}, \{d, g\}, \{d, e, f, g\}, \{f\}, \{d, f, g\} \} \]

\[ cutbool(X) = \log_2 |UN(X)| \]

expensive!
Comparing Graph Parameters

- Tree-Width and Clique-Width
- Decompositions
- Rank-Width
- Boolean-Width
- Theoretical Connection
1. Overview

2. Width-Measures

3. **An Upper Bound Algorithm**

4. A Lower Bound Algorithm

5. Conclusion
Main Idea

Initialization: Greedy
Main Idea

Initialization: Greedy

\[ V \]

\{a,b,c\} \quad \{d,e,f,g\}
Main Idea

Initialization: Greedy

V

{a,b,c}  {d,e,f,g}

{a,b}  {c}
Main Idea

Initialization: Greedy
Main Idea

Improvement: Random and Greedy

\[
\begin{align*}
V \setminus \{a,b,c\} & \setminus \{d,e,f,g\} \\
\{a,b,c\} & \setminus \{d,e,f,g\} \\
\{a\} & \setminus \{b\} & \setminus \{c\} & \setminus \{d\} & \setminus \{e\} & \setminus \{f\} & \setminus \{g\}.
\end{align*}
\]
Main Idea

Improvement: Random and Greedy

\[ \{a,b,c\} \quad \{d,e,f,g\} \]
\[ \{a,b\} \quad \{c\} \quad \{d,e\} \quad \{f,g\} \]
\[ \{a\} \quad \{b\} \quad \{d\} \quad \{e\} \quad \{f\} \quad \{g\} \]
Main Idea

Improvement: Random and Greedy

\[ \{a, b, c\} \quad \{d, e, f, g\} \]

\[ \{a, b\} \quad \{c\} \quad \{d, e\} \quad \{f, g\} \]

\[ \{a\} \quad \{b\} \quad \{d\} \quad \{e\} \quad \{f\} \quad \{g\} \]
Main Idea

Improve: Random and Greedy

\[ V \{a,b,c\} \{d,e,f,g\} \{a,b\} \{c\} \{d,f\} \{e,g\} \{a\} \{b\} \{d\} \{e\} \{f\} \{g\} \]
Main Idea

Improvement: Random and Greedy

A diagram shows a tree structure with sets {a, b, c}, {d, e, f, g}, {a, b}, {c}, {d, f}, and {e, g} at the leaves. The tree is labeled with an arrow pointing to the root labeled V.
Main Idea

Improvement: Random and Greedy
Comparison to Tree-width

Dataset: 193 real life graphs from TreewidthLIB
Comparison to Boolean-width
1. Overview
2. Width-Measures
3. An Upper Bound Algorithm
4. A Lower Bound Algorithm
5. Conclusion
Main Idea

If an induced subgraph has rank-width $\geq k$ so does the graph

Algorithm outline

1. create a sequence of growing (by one vertex) induced subgraphs
2. enumerate all decompositions for these subgraphs
3. adapt decompositions for next induced subgraph
4. repeat 3

Search space reduction

use incomplete decompositions with exactly two leaves
Main Idea

If an induced subgraph has rank-width $\geq k$ so does the graph.

**Algorithm outline**

1. create a sequence of growing (by one vertex) induced subgraphs
2. enumerate all decompositions for these subgraphs
3. adapt decompositions for next induced subgraph
4. repeat 3

**Search space reduction**

use incomplete decompositions with exactly two leaves
Main Idea

If an induced subgraph has rank-width $\geq k$ so does the graph.

Algorithm outline

1. create a sequence of growing (by one vertex) induced subgraphs
2. enumerate all decompositions for these subgraphs
3. adapt decompositions for next induced subgraph
4. repeat 3

Search space reduction

use incomplete decompositions with exactly two leaves
LB in comparison to UB and boolean-width
LB in comparison to UB and boolean-width

![Graph 1](truewUB/rwLB)

![Graph 2](rwUB/rwLB)
1 Overview

2 Width-Measures

3 An Upper Bound Algorithm

4 A Lower Bound Algorithm

5 Conclusion
Boolean-Width is often smaller than Rank-Width

But Rank-Width is much easier to calculate