A Backtracking-Based Algorithm for Computing Hypertree-Decompositions

Georg Gottlob and Marko Samer

Draft

(Part 1)
Motivation: Solving Graphical Models

- 'Convert' reasoning problem to tree structure by decomposition:
  - Given a tree decomposition of width $w$, we can solve the reasoning problem in:
    - time $O((r + m) \cdot \deg \cdot k^{w+1})$
    - space $O(m \cdot k^{sep})$
  - Given a hypertree decomposition of width $hw$, we can solve the reasoning problem (absorbing rel. to 0) in:
    - time $O(m \cdot \deg \cdot hw \cdot \log(t) \cdot t^{hw})$
    - space $O(t^{hw})$
Motivation: Solving Graphical Models

- 'Convert' reasoning problem to tree structure by decomposition:
  - Given a tree decomposition of width $w$, we can solve the reasoning problem in
    - time $O((r + m) \cdot \text{deg} \cdot k^{w+1})$
    - space $O(m \cdot k^{\text{sep}})$
  - Given a hypertree decomposition of width $hw$, we can solve the reasoning problem (absorbing rel. to 0) in
    - time $O(m \cdot \text{deg} \cdot hw \cdot \log(t) \cdot t^{hw})$
    - space $O(t^{hw})$

- Question: How to compute hypertree decomposition?
Problem hardness

- Given a reasoning problem, finding a hypertree decomposition with minimal width is NP hard in general.

  - In their paper, the authors suggest an algorithm that, for a problem and given $k$, finds a hypertree decomposition of width at most $k$ (if one exists) in polynomial time.
    - First step: Nondeterministic algorithm.
    - Second step: Introduce heuristic to achieve determinism.
Definitions

- View reasoning problem as its hypergraph $H = (V,E)$
  - Vertices $V$ are the variables of the problem
  - Hyperedges $E$ are the scopes of the problem's functions / relations, each one a subset of $V$. 

![Hypergraph Diagram]
Definitions (our way)

A tree decomposition of a reasoning problem with hypergraph $H = (V, E)$ is a triple $(T, \chi, \psi)$ where $T = (V_T, E_T)$ is a tree and $\chi: V_T \rightarrow 2^V$ and $\psi: V_T \rightarrow 2^E$ are labeling functions, satisfying the following:

1. For each hyperedge $X \in E$, there is exactly one vertex $v \in V_T$ such that $X \in \psi(v)$.

2. If $X \in \psi(v)$, then $X \subseteq \chi(v)$.

3. For each variable $x \in V$, the set $\{v \in V_T \mid x \in \chi(v)\}$ induces a connected subtree of $T$. This is also called the running intersection or the connectedness property.

The treewidth of a tree decomposition is $w = \max_{v \in V_T} |\chi(v)| - 1$. $T$ is a hypertree decomposition if the following additional condition is satisfied:

4. For each $v \in V_T : \chi(v) \subseteq \bigcup \psi(v)$.

The hypertree width of a hypertree decomposition is then $hw = \max_{v \in V_T} |\psi(v)|$. 
Definitions (their way)

A tree decomposition of a reasoning problem with hypergraph $H = (V, E)$ is a triple $(T, \chi, \psi)$ where $T = (V_T, E_T)$ is a tree and $\chi: V_T \rightarrow 2^V$ and $\psi: V_T \rightarrow 2^E$ are labeling functions, satisfying the following:

1. For each hyperedge $X \in E$, there is one vertex $v \in V_T$ such that $X \subseteq \chi(v)$.
2. For each variable $x \in V$, the set $\{v \in V_T | x \in \chi(v)\}$ induces a connected subtree of $T$.
3. For each $v \in V_T$ : $\chi(v) \subseteq \bigcup \psi(v)$.
4. For each $v \in V_T$ : $\bigcup \psi(v) \cap \chi(T_v) \subseteq \chi(v)$.

$\chi(T_v)$ here denotes all variables occurring in the nodes $V'_T$ of the subtree rooted at $v$, formally $\bigcup_{v' \in V'_T} \chi(v')$. 
Definitions

- Slight differences in the definitions
  - Dechter: “Each hyperedge assigned to exactly one cluster”.
  - Gottlob: “Hyperedges can be assigned to multiple clusters or none at all.”
Alternative valid decompositions

\( \psi = \{A, B\} \)
\( \chi = \{a, b, c, d, e, f\} \)

\( \psi = \{A, C, D\} \)
\( \chi = \{a, c, d, f, g, i\} \)

\( \psi = \{F, H\} \)
\( \chi = \{a, b, e, h, j\} \)

\( \psi = \{A, B, C\} \)
\( \chi = \{a, b, c, d, e, f, g\} \)

\( \psi = \{C, D\} \)
\( \chi = \{a, c, d, f, g, i\} \)

\( \psi = \{F, H\} \)
\( \chi = \{a, b, e, h, j\} \)

\( \psi = \{A, B, C\} \)
\( \chi = \{a, b, c, d, e, f, g\} \)

\( \psi = \{C, D\} \)
\( \chi = \{a, c, d, f, g, i\} \)

\( \psi = \{F, H\} \)
\( \chi = \{a, b, e, h, j\} \)

\( \psi = \{E\} \)
\( \chi = \{g, i\} \)

\( \psi = \{H\} \)
\( \chi = \{e, j\} \)
Algorithm $k$-decomp (1)

- Gottlob et. al. propose a nondeterministic algorithm for checking and finding a hypertree decomposition:

Algorithm 1 $k$-decomp($H$Graph)

1. $H$Tree := $k$-decomposable(edges($H$Graph), $\emptyset$);
2. return $H$Tree;
Algorithm \( k \)-decomp (2)

**Algorithm 2** \( k \)-decomposable(Edges, OldSep)

1. **guess** \( \text{Separator} \subseteq \text{edges(HGraph)} \) \textbf{such that} \(|\text{Separator}| \leq k\);
2. **check** that the following two conditions hold:
   - \( \bigcup \text{Edges} \cap \bigcup \text{OldSep} \subseteq \bigcup \text{Separator} \);
   - \( \text{Separator} \cap \text{Edges} \neq \emptyset \);
3. **if** one of these checks fails **then** return NULL;
4. \( \text{Components} := \text{separate(Edges, Separator)} \);
5. \( \text{Subtrees} := \emptyset \);
6. **for each** \( \text{Comp} \in \text{Components} \) **do**
   - \( \text{HTree} := \text{k-decomposable(Comp, Separator)} \);
   - **if** \( \text{HTree} = \text{NULL} \) **then**
     - return NULL;
   - **else**
     - \( \text{Subtrees} := \text{Subtrees} \cup \{ \text{HTree} \} \);
   - **endif**
7. **endfor**
8. \( \text{Chi} := (\bigcup \text{Edges} \cap \bigcup \text{OldSep}) \cup (\text{Separator} \cap \text{Edges}) \);
9. \( \text{HTree} := \text{getHTNode(Separator, Chi, Subtrees)} \);
10. **return** \( \text{HTree} \);
Example

\[ \psi = \{A, B\} \]
\[ \chi = \{a, b, c, d, e, f\} \]
Example

\[ \psi = \{A, B\} \]
\[ \chi = \{a, b, c, d, e, f\} \]
Example

\[ \psi = \{A, B\} \]
\[ \chi = \{a, b, c, d, e, f\} \]

\[ \psi = \{C, D\} \]
\[ \chi = \{a, c, d, f, i, g\} \]
Example

\[ \psi = \{A, B\} \]
\[ \chi = \{a, b, c, d, e, f\} \]

\[ \psi = \{C, D\} \]
\[ \chi = \{a, c, d, f, i, g\} \]
Example

\[ \psi = \{A, B\} \]
\[ \chi = \{a, b, c, d, e, f\} \]

\[ \psi = \{C, D\} \]
\[ \chi = \{a, c, d, f, i, g\} \]

\[ \psi = \{H, F\} \]
\[ \chi = \{a, b, e, h, j\} \]
To be continued

• Nondeterminism:
  – Cannot be implemented, only theoretical interest:
    • Gottlob et al. show that problem of deciding whether a problem's hypertree width is bounded by $k$ is in $P$.

• Next time:
  – Transform $k$-decomp into a deterministic algorithm with polynomial runtime:
    • Replace “guess and check” (lines 1-4) by backtracking-based search.
  – Benchmark results
A Backtracking-Based Algorithm for Computing Hypertree-Decompositions

Georg Gottlob and Marko Samer
Draft

(Part 2)
Algorithm det-\(k\)-decomp (1)

- Replace “guess and check”:
  - Heuristic backtrack search, keeping track of failed and succeeded decompositions
  - Key: Find \textit{Separator} that satisfies two conditions:
    
    \begin{align*}
    1. \quad & \bigcup \text{Edges} \cap \bigcup \text{OldSep} \subseteq \bigcup \text{Separator} \quad \Rightarrow \text{Connectivity} \\
    2. \quad & \text{Separator} \cap \text{Edges} \neq \emptyset \quad \Rightarrow \text{Monotonicity}
    \end{align*}

Algorithm 3 \textit{det-\(k\)-decomp}(\textit{HGraph})

\begin{algorithmic}
\State \textit{FailSeps} := \emptyset;
\State \textit{SuccSeps} := \emptyset;
\State \textit{HTree} := \textit{decompCov}(\textit{edges}(\textit{HGraph}), \emptyset);
\If {\textit{HTree} \neq \text{NULL}}
\State \textit{HTree} := \textit{expand}(\textit{HTree});
\EndIf
\State \text{return} \textit{HTree};
\end{algorithmic}
Algorithm det-\(k\)-decomp (2)

- Algorithm outline:

```
  Enforce Condition 1
  Enforce Condition 2 & compute components
  Check & decompose components recursively
  
  \( k\)-decomp
    \rightarrow decompCov
    \rightarrow decompAdd
    \rightarrow decompSub
    \rightarrow expand
    \rightarrow cover
    \rightarrow separate
```
Algorithm det-\(k\)-decomp (3)

- **decompCov** enforces condition 1:
  \[ \bigcup Edges \cap \bigcup OldSep \subseteq \bigcup Separator \]

---

**Algorithm 4** \text{decompCov}(Edges, Conn)

```plaintext
if \(|Edges| \leq k\) then
    \text{HTree} := \text{getHTNode}(Edges, \bigcup Edges, \emptyset);
    return \text{HTree};
endif
BoundEdges := \{e \in \text{edges}(HGraph) \mid e \cap Conn \neq \emptyset\};
for each CovSep \in \text{cover}(Conn, BoundEdges) do
    \text{HTree} := \text{decompAdd}(Edges, Conn, CovSep);
    if \text{HTree} \neq \text{NULL} then
        return \text{HTree};
    endif
endfor
return \text{NULL};
```
Algorithm det-$k$-decomp (4)

- \textit{decompAdd} enforces condition 2 and decomposes

  \textbf{condition 2:} \( \textit{Separator} \cap \textit{Edges} \neq \emptyset \)

\begin{algorithm}
\caption{\textit{decompAdd}(\textit{Edges}, \textit{Conn}, \textit{CovSep})}
\begin{algorithmic}[1]
\STATE \textit{InCovSep} := \textit{CovSep} \cap \textit{Edges};
\IF {\textit{InCovSep} \neq \emptyset \text{ or } k - |\textit{CovSep}| > 0}
\STATE \textbf{if} \textit{InCovSep} = \emptyset \textbf{ then AddSize} := 1 \textbf{ else AddSize} := 0 \textbf{ endif};
\STATE \textbf{for each} \textit{AddSep} \subseteq \textit{Edges} \text{ s.t. } |\textit{AddSep}| = \textit{AddSize} \textbf{ do}
\STATE \hspace{0.5em} \textit{Separator} := \textit{CovSep} \cup \textit{AddSep};
\STATE \hspace{0.5em} \textit{Components} := \textit{separate}(\textit{Edges}, \textit{Separator});
\STATE \hspace{0.5em} \textbf{if} \ \forall \textit{Comp} \in \textit{Components}. (\textit{Separator}, \textit{Comp}) \notin \textit{FailSeps} \textbf{ then}
\STATE \hspace{1em} \textit{Subtrees} := \textit{decompSub}(\textit{Components}, \textit{Separator});
\STATE \hspace{1em} \textbf{if} \ \textit{Subtrees} \neq \emptyset \textbf{ then}
\STATE \hspace{2em} \textit{Chi} := \textit{Conn} \cup \bigcup (\textit{InCovSep} \cup \textit{AddSep});
\STATE \hspace{2em} \textit{HTree} := \textit{getHTNode}(\textit{Separator}, \textit{Chi}, \textit{Subtrees});
\STATE \hspace{2em} \textbf{return} \textit{HTree};
\STATE \hspace{1em} \textbf{endif}
\STATE \textbf{endif}
\STATE \textbf{endfor}
\STATE \textbf{endif}
\STATE \textbf{return} \textit{NULL};
\end{algorithmic}
\end{algorithm}
Algorithm det-\textit{k-decomp} (5)

- \textit{decompSub} recursively decomposes the components
  - checks for previous processing of components

\begin{algorithm}
\caption{decompSub(Components, Separator)}
\begin{footnotesize}
\begin{algorithmic}
\State \textbf{Subtrees} := \emptyset;
\For{\textit{each} \textit{Comp} \in \textit{Components}}
  \State \textit{ChildConn} := $\bigcup \textit{Comp} \cap \bigcup \textit{Separator}$;
  \If{\langle \textit{Separator}, \textit{Comp} \rangle \in \textit{SuccSeps}}
    \State \textit{HTree} := \textit{getHTNode}($\textit{Comp}, \textit{ChildConn}, \emptyset$);
  \Else
    \State \textit{HTree} := \textit{decompCov}($\textit{Comp}, \textit{ChildConn}$);
  \EndIf
  \If{\textit{HTree} = \textit{NULL}}
    \State \textit{FailSeps} := \textit{FailSeps} \cup \{\langle \textit{Separator}, \textit{Comp} \rangle\};
    \State \Return \emptyset;
  \Else
    \State \textit{SuccSeps} := \textit{SuccSeps} \cup \{\langle \textit{Separator}, \textit{Comp} \rangle\};
  \EndIf
\EndFor
\Return \textit{Subtrees};
\end{algorithmic}
\end{footnotesize}
\end{algorithm}
Complexity analysis

• Bounds:
  
  – Number of recursive calls:
    • Number of separators bounded by
      \[ \Psi = \sum_{i=1}^{k} \binom{n}{i} = \sum_{i=1}^{k} \frac{n!}{i!(n-i)!} \]
    • At most \( m \) subcomponents each time.
    • Number of recursive calls thus bounded by \( \mathcal{O}(\Psi m) \).
  
  – Each recursive call:
    • Loops in \( \text{decompCov} \) bounded by
      \[ \Phi = \sum_{i=1}^{k} \binom{\min(n,ck)}{i} = \sum_{i=1}^{k} \frac{\min(n,ck)!}{i!(\min(n,ck) - i)!} \]
    • Loops in \( \text{decompAdd} \) bounded by \( n \).
    • Loops in \( \text{decompSub} \) bounded by \( m \).
    • Single recursive call therefore bounded bounded by \( \mathcal{O}(\Phi nm) \).

• Total complexity bound:

\[ \mathcal{O}(\Psi \Phi n m^2) = \mathcal{O}(n^{k+1} \min(n,ck)^k m^2) \]
Comparison

- **Compare to algorithm opt-\(k\)-decomp:**
  - **Complexity** \(O(n^{2k} m^2)\).
  - Often \(ck \ll n\), hence det-\(k\)-decomp is \(O(n^{k+1} (ck)^k m^2)\).
  - Lower memory usage
Heuristic for procedure *cover*

- **Choosing *CovSep* candidates:**
  - Assign weights to *BoundEdges*:
    - Number of vertices in Conn each edge contains
  - Order by decreasing weight
  - Greedily cover from first to last.

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>D</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>A</th>
<th>F</th>
<th>B</th>
<th>D</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>F</td>
<td>B</td>
<td>D</td>
<td>G</td>
<td>H</td>
</tr>
<tr>
<td>A</td>
<td>F</td>
<td>B</td>
<td>D</td>
<td>G</td>
<td>H</td>
</tr>
<tr>
<td>A</td>
<td>F</td>
<td>B</td>
<td>D</td>
<td>G</td>
<td>H</td>
</tr>
<tr>
<td>A</td>
<td>F</td>
<td>B</td>
<td>D</td>
<td>G</td>
<td>H</td>
</tr>
</tbody>
</table>
```
Example

- Run \textit{det}-\textit{k-decomp} on familiar example:

\[
\begin{align*}
\text{decompCov} & (\{A, B, C, D, E, F, G, H\}, \emptyset) \\
\text{decompAdd} & (\{A, B, C, D, E, F, G, H\}, \emptyset, \emptyset) \\
\text{decompSub} & (\{\{B, C, D, E, F, G, H\}\}, \{A\}) \\
\text{decompCov} & (\{B, C, D, E, F, G, H\}, \{a, b, c\}) \\
\text{decompAdd} & (\{B, C, D, E, F, G, H\}, \{a, b, c\}, \{A\}) \\
\text{decompSub} & (\{\{C, D, E\}, \{F, G, H\}\}, \{A, B\}) \\
\text{decompCov} & (\{C, D, E\}, \{a, c, d, f\}) \\
\text{decompAdd} & (\{C, D, E\}, \{a, c, d, f\}, \{C, D\}) \\
\text{decompSub} & (\{\{E\}\}, \{C, D\}) \\
\text{decompCov} & (\{E\}, \{g, i\}) \\
\text{decompCov} & (\{F, G, H\}, \{a, b, e\}) \\
\text{decompAdd} & (\{F, G, H\}, \{a, b, e\}, \{A, F\}) \\
\text{decompSub} & (\{\{G, H\}\}, \{A, F\}) \\
\text{decompCov} & (\{G, H\}, \{a, e, h\}) \\
\end{align*}
\]
Experimental results

• Compare performance:
  – det-\(k\)-decomp
  – Bucket Elimination heuristics
  – opt-\(k\)-decomp

• Report smallest hypertree width obtained within 1 hour.
Results (1)

- **Benchmarks from Daimler Chrysler (adder circuits etc.)**

<table>
<thead>
<tr>
<th>Instance (Atoms / Variables)</th>
<th>Min</th>
<th>opt-k-decomp</th>
<th>BE</th>
<th>det-k-decomp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Width</td>
<td>Time</td>
<td>Width</td>
</tr>
<tr>
<td>adder_15 (76 / 106)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>adder_25 (126 / 176)</td>
<td>2</td>
<td>2</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>adder_50 (251 / 351)</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>2</td>
</tr>
<tr>
<td>adder_75 (376 / 526)</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>2</td>
</tr>
<tr>
<td>adder_99 (496 / 694)</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>2</td>
</tr>
<tr>
<td>bridge_15 (137 / 137)</td>
<td>2</td>
<td>2</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>bridge_25 (227 / 227)</td>
<td>2</td>
<td>2</td>
<td>69</td>
<td>3</td>
</tr>
<tr>
<td>bridge_50 (452 / 452)</td>
<td>2</td>
<td>2</td>
<td>1105</td>
<td>3</td>
</tr>
<tr>
<td>bridge_75 (677 / 677)</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>3</td>
</tr>
<tr>
<td>bridge_99 (893 / 893)</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>3</td>
</tr>
<tr>
<td>NewSystem1 (84 / 142)</td>
<td>3</td>
<td>—</td>
<td>—</td>
<td>3</td>
</tr>
<tr>
<td>NewSystem2 (200 / 345)</td>
<td>3</td>
<td>—</td>
<td>—</td>
<td>4</td>
</tr>
<tr>
<td>NewSystem3 (278 / 474)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>5</td>
</tr>
<tr>
<td>NewSystem4 (418 / 718)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>5</td>
</tr>
<tr>
<td>atv_partial_system (88 / 125)</td>
<td>3</td>
<td>—</td>
<td>—</td>
<td>3</td>
</tr>
</tbody>
</table>
Results (2)

- **Hypergraphs extracted from 2D grids**
  - hypertree width known from construction

<table>
<thead>
<tr>
<th>Instance (Atoms / Variables)</th>
<th>Min</th>
<th>opt-(k)-decomp</th>
<th>BE</th>
<th>det-(k)-decomp</th>
</tr>
</thead>
<tbody>
<tr>
<td>grid2d.10 (50 / 50)</td>
<td>4</td>
<td>—</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>grid2d.15 (112 / 113)</td>
<td>6</td>
<td>—</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>grid2d.20 (200 / 200)</td>
<td>7</td>
<td>—</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>0</td>
<td>3140</td>
</tr>
<tr>
<td>grid2d.25 (312 / 313)</td>
<td>9</td>
<td>—</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>3</td>
<td>2000</td>
</tr>
<tr>
<td>grid2d.30 (450 / 450)</td>
<td>11</td>
<td>—</td>
<td>19</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>7</td>
<td>1566</td>
</tr>
<tr>
<td>grid2d.35 (612 / 613)</td>
<td>12</td>
<td>—</td>
<td>23</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>15</td>
<td>1905</td>
</tr>
<tr>
<td>grid2d.40 (800 / 800)</td>
<td>14</td>
<td>—</td>
<td>26</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>28</td>
<td>2530</td>
</tr>
<tr>
<td>grid2d.45 (1012 / 1013)</td>
<td>16</td>
<td>—</td>
<td>31</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>51</td>
<td>2606</td>
</tr>
<tr>
<td>grid2d.50 (1250 / 1250)</td>
<td>17</td>
<td>—</td>
<td>33</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>86</td>
<td>2786</td>
</tr>
<tr>
<td>grid2d.60 (1800 / 1800)</td>
<td>21</td>
<td>—</td>
<td>41</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>204</td>
<td>2984</td>
</tr>
<tr>
<td>grid2d.70 (2450 / 2450)</td>
<td>24</td>
<td>—</td>
<td>48</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>474</td>
<td>2161</td>
</tr>
<tr>
<td>grid2d.75 (2812 / 2813)</td>
<td>26</td>
<td>—</td>
<td>48</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>631</td>
<td>2881</td>
</tr>
</tbody>
</table>
Results (3)

- **ISCAS89**
  - extracted from circuits
  - examples from practice

<table>
<thead>
<tr>
<th>Instance (Atoms / Variables)</th>
<th>Min</th>
<th>opt-$k$-decomp</th>
<th>BE</th>
<th>det-$k$-decomp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Width</td>
<td>Time</td>
<td>Width</td>
</tr>
<tr>
<td>s27 (13 / 17)</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>s208 (104 / 115)</td>
<td>$\geq 3$</td>
<td>7</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>s298 (133 / 139)</td>
<td>$\geq 3$</td>
<td>5</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>s344 (175 / 184)</td>
<td>$\geq 3$</td>
<td>7</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>s349 (176 / 185)</td>
<td>$\geq 3$</td>
<td>7</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>s382 (179 / 182)</td>
<td>$\geq 3$</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>s386 (165 / 172)</td>
<td></td>
<td>8</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>s400 (183 / 186)</td>
<td>$\geq 3$</td>
<td>6</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>s420 (212 / 231)</td>
<td>$\geq 3$</td>
<td>9</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>s444 (202 / 205)</td>
<td>$\geq 3$</td>
<td>6</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>s510 (217 / 236)</td>
<td>$\geq 3$</td>
<td>23</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>s526 (214 / 217)</td>
<td>$\geq 3$</td>
<td>8</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>s641 (398 / 433)</td>
<td></td>
<td>7</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>s713 (412 / 447)</td>
<td></td>
<td>7</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>s820 (294 / 312)</td>
<td>$\geq 3$</td>
<td>13</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>s832 (292 / 310)</td>
<td>$\geq 3$</td>
<td>12</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>s838 (422 / 457)</td>
<td>$\geq 3$</td>
<td>16</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>s953 (424 / 440)</td>
<td>$\geq 3$</td>
<td>40</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>s1196 (547 / 561)</td>
<td></td>
<td>35</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>s1238 (526 / 540)</td>
<td></td>
<td>34</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>s1423 (731 / 748)</td>
<td></td>
<td>18</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>s1488 (659 / 667)</td>
<td></td>
<td>23</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>s1494 (653 / 661)</td>
<td></td>
<td>24</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>s5378 (2958 / 2993)</td>
<td></td>
<td>85</td>
<td>141</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

- **Performance:**
  - Significantly outperforms opt-\(k\)-decomp.
    - Time- and memory-wise
  - Results better than or comparable to BE heuristic.
    - Only when time is not the issue and graphs are “not too large and complicated”