Storing Set Families More Compactly with Top ZDDs

K. Matsuda (The University of Tokyo)
S. Denzumi (The University of Tokyo)
K. Sadakane (The University of Tokyo)

18th Symposium on Experimental Algorithms
June 16—18, 2020
Abstract

• Purpose
  – Compress zero-suppressed binary decision diagram (ZDD) ≒ labeled binary directed acyclic graph (DAG)

• Method
  – Expand a tree compression algorithm to DAGs

• Result
  – Theoretic: Exponentially smaller than input
  – Experimental: Smaller than a related research in almost all cases
Contents

• Preliminary
  - ZDD
  - Tree compression algorithms

• Proposed data structure
  - Construction algorithm
  - Complexity analysis

• Experiment

• Conclusion
Preliminary

ZDD
DAG compression
Top tree compression
ZDD

• Zero-suppressed binary decision diagram [Minato 93]
  - Labeled binary directed acyclic graph
  - Represents a family of sets
  - Share equivalent subgraphs

• Terminology
  - Branching nodes
    * Label
    * 0-edges and 1-edges
  - Sink nodes
    * Top or bottom

\[
\{ \{1, 2\}, \{1, 3\}, \{2, 3\} \}
\]

\[
\begin{array}{c}
1 \\
2 \\
3 \\
\perp \\
\top
\end{array}
\]
Tree compression methods

• Tree grammar
  – Based on grammar compression for strings [Charikar et al. 05]
  – Traversing on compressed representations require linear time to the size of grammar
    [Busatto et al. 04,], [Lohrey et al. 13]

• Succinct data structures
  – Labeled tree: LOUDS [Jacobson 89]
    BP [Munro, Raman 01]
  – Unlabeled tree: [Ferragina et al. 09]
Tree compression

• Transform-based compression
  - Shares equivalent sub structures
  - DAG compression [Downey et al. 80]
    * Shares all equivalent subtrees
  - Top DAG compression [Bille et al. 13]
    * Shares equivalent subcomponents
DAG compression

• Compress labeled DAGs
  - [Downey et al. 80]
  - Share all equivalent subtrees
Problem of DAG comp.

• Cannot compress substructures that repeats vertically

• Example:

\[
\text{Simple, but not compressed}
\]
Top DAG compression

- Compress labeled DAGs [Bille et al. 13]
  - Transform an input tree to top tree, and compress the top tree by DAG compression

Input tree $T$

Top tree $T$

Top DAG $TD$
Top DAG compression

• In comparison to DAG compression:
  – [Best case] $O(n / \log_\sigma n)$ times smaller
  – [Worst case] $O(\log_\sigma n)$ times larger

• Greedy construction [Bille et al. 13]
  – #node = $O(n \log \log_\sigma n / \log_\sigma n)$
  – Proof is in [Hübschle-Schneider and Raman 15]

• Optimal construction [Lohrey et al. 17], [Dudek, Gawrychowski 18]
  – #node = $O(n/\log_\sigma n)$
    (information theoretic lowerbound)

(n: #node of the input tree, $\sigma$: #label)
Top tree

- A binary tree \( \mathcal{T} \) that represents the way to decompose the input tree \( T \)
  - Each node of the top tree corresponds to a cluster of \( T \)
  - The root of the top tree corresponds to whole \( T \)
  - A cluster is an induced subgraph of a set of connected edges
  - Every cluster has at most 2 boundary nodes
  - A cluster is made by horizontal or vertical merge of 2 clusters that have the same node as a boundary node
Top tree

- A binary tree $\mathcal{T}$ that represents the way to decompose the input tree $T$
  - A cluster is an induced subgraph of a set of connected edges
  - Every cluster has at most 2 boundary nodes
Top tree

- Example:

Input tree $T$  

```
1
/   \
/     \     
2       7
|       |
|       |
3       6
|       |
|       |
4       5
```

Top tree $T$

```
V(b)
/   \
/     \     
H(c)       H(c)
/   \
/     \     
H(e)       H(e)
/   \
/     \     
V(b)       V(b)
/   \
/     \     
V(b)       V(b)
/   \
/     \     
V(b)       V(b)
/   \
/     \     
V(b)       V(b)
/   \
/     \     
V(b)       V(b)
/   \
/     \     
V(b)       V(b)
/   \
/     \     
V(b)       V(b)
/   \
/     \     
V(b)       V(b)
```

Example:

2020-06-16

18th Symposium on Experimental Algorithms
Top tree

• Example:

Input tree $T$

top DAG $\mathcal{T}_D$
Advantage of top DAG

• Top DAG compression allows sharing the same substructure that appear at different height.

\[ \log n \]
Two types of merging

- Vertical merge: (a), (b)
- Horizontal merge: (c), (d), (e)

[Bille et al. 13]
Horizontal merge

• Merge two clusters that have the same node as their top boundary nodes
Vertical merge

- Merge two clusters that have the same node as their top and bottom boundary.
Top tree construction

• Top tree is not uniquely determined from the input tree

• Greedy construction
  – Repeat 1—3 until the tree T become 1 edge
  – 1. Choose pairs of clusters that can be horizontally merged as much as possible
  – 2. Choose pairs of clusters that can be vertically merged from remaining nodes as much as possible
  – 3. Merge the all pairs chosen at 1 and 2
Greedy construction

- Example

Tree $T$  top tree $\mathcal{T}$
Greedy construction

• Example

Tree $T$

top tree $\mathcal{T}$
Greedy construction

• Example

Tree $T$ vs. top tree $\mathcal{T}$
Greedy construction

• Example

Tree $T$

Top tree $\mathcal{T}$
Greedy construction

• Example

Tree $T$

Top tree $\mathcal{T}$
Greedy construction

• Example

Tree $T$  top tree $\mathcal{T}$
Greedy construction

• Example

Tree $T$  top tree $\mathcal{T}$
Complexity: Greedy method

- $n$: #node of an input tree, $\sigma$: #label

**Theorem**
[Bille et al. 13]
The height of the top tree made by greedy construction is $O(\log n)$

**Theorem**
[Hübchle-Schneider, Raman 15]
The number of nodes of top DAG obtained after DAG compression to the top tree made by greedy construction is $O( (n \log \log_\sigma n) / \log_\sigma n)$
Operations on top DAG

• Following operations are in $O(\log n)$ time
  – $(x$: $x$-th node in DFS, $T(x)$: a subtree rooted by $x$)
  – access($x$): label of $x$
  – parent($x$): preorder of the parent of $x$
  – depth($x$): depth of $x$
  – height($x$): height of $x$
  – size($x$): number of nodes in $T(x)$
  – firstchild($x$): preorder of the first child of $x$
  – nextsibling($x$): preorder of the next sibling of $x$
  – la($x$, $i$): preorder of $i$-th ancestor of $x$
  – nca($x$, $y$): preorder of nearest common ancestor of $x$ and $y$
Proposed method

Top ZDD

Construction algorithm
Experiment
Construction of top ZDD

• 1. Find a spanning tree from input ZDD
  - The edges not included in the spanning tree is called non tree edges

• 2. Transform the spanning tree to a top tree by greedy construction

• 3. For each non tree edge, store the edge at the nearest common ancestor of the source node and the destination node (Edges point sink nodes are exception)

• 4. Share equivalent subtrees
Example of construction

• Step 0. Input

Original ZDD
Example of construction

• Step 1. Find a spanning tree
Example of construction

• Step 2. Construct a top tree

Original ZDD
Example of construction

• Step 3. Store non tree edges

Original ZDD
Example of construction

- Step 4. Share equivalent subtrees
Theoretical results

**Theorem**
Memory usage of top ZDD is $O(\log n)$ in the best case

- Examples

**Theorem**
Edge traversal is $O(\log^2 n)$ time
Experiment

• Compared data structures
  - Top ZDD: memory usage (byte)
  - DenseZDD: memory usage (byte)
    * Static ZDD using succinct data structure [Denzumi et al. 14]
  - ZDD: \((2n \log n + n \log \sigma)\) (\(n: \#\) node, \(\sigma: \#\) label) (byte)

• Data sets
  - \(\{S \subseteq U | |S| \leq B\}\), where \(|U| = A\)
  - \(\{S \subseteq U | \forall e \in S, \exists f \in S \text{ s.t. } |e - f| \leq B\} \cup U\), where \(U = \{1, ..., A\}\)
  - \(2^U\), where \(|U| = A\)
  - Solutions of knapsack problems
  - Sets of matching edges of graphs
  - Frequent item sets
Experimental results 1/6

• Data: For an underlying set $U$ of size $A$, 
  \[ \{S \subseteq U \mid |S| \leq B\} \]

<table>
<thead>
<tr>
<th></th>
<th>top ZDD</th>
<th>DenseZDD</th>
<th>$(2n \log n + n \log c)/8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A = 100, B = 50$</td>
<td>3,823</td>
<td>9,544</td>
<td>9,882</td>
</tr>
<tr>
<td>$A = 400, B = 200$</td>
<td>13,614</td>
<td>146,550</td>
<td>206,025</td>
</tr>
<tr>
<td>$A = 1000, B = 500$</td>
<td>43,151</td>
<td>966,519</td>
<td>1,440,375</td>
</tr>
</tbody>
</table>

• Top ZDDs are 2—20 times smaller than DenseZDDs

2020-06-16

18th Symposium on Experimental Algorithms
Experimental results 2/6

• Data: For an underlying set $U$ of size $A$,
  $$\{S \subseteq U \mid \forall e \in S, \exists f \in S \text{ s.t. } |e - f| \leq B\} \cup U$$

<table>
<thead>
<tr>
<th>$A = 500, B = 250$</th>
<th>top ZDD</th>
<th>DenseZDD</th>
<th>$(2n \log n + n \log c)/8$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2,431</td>
<td>227,798</td>
<td>321,594</td>
</tr>
<tr>
<td>$A = 1000, B = 500$</td>
<td>2,511</td>
<td>321,594</td>
<td>1,440,375</td>
</tr>
</tbody>
</table>

• Top ZDDs are 100—125 times smaller than DenseZDDs
Experimental results 3/6

• Data: For an underlying set $U$ of size $A$, $2^U$

<table>
<thead>
<tr>
<th></th>
<th>top ZDD</th>
<th>DenseZDD</th>
<th>$(2n \log n + n \log c)/8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A = 1000$</td>
<td>2,254</td>
<td>4,185</td>
<td>3,750</td>
</tr>
<tr>
<td>$A = 50000$</td>
<td>2,464</td>
<td>178,764</td>
<td>300,000</td>
</tr>
</tbody>
</table>

• Top ZDDs are highly effective because the inputs have simple structure
• Data: Solutions of knapsack problems
  \[- w_i \in [1, W]: \text{random weight } (w_i \geq w_{i+1}),\ C: \text{capacity}\]

<table>
<thead>
<tr>
<th></th>
<th>top ZDD</th>
<th>DenseZDD</th>
<th>((2n \log n + n \log c)/8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A = 100, W = 1000, C = 10000)</td>
<td>1,658,494</td>
<td>1,730,401</td>
<td>2,444,405</td>
</tr>
<tr>
<td>(A = 200, W = 100, C = 5000)</td>
<td>1,032,596</td>
<td>1,516,840</td>
<td>2,181,688</td>
</tr>
<tr>
<td>(A = 1000, W = 100, C = 1000)</td>
<td>2,080,925</td>
<td>2,929,191</td>
<td>4,491,025</td>
</tr>
<tr>
<td>(A = 5000, W = 100, C = 200)</td>
<td>1,135,613</td>
<td>1,740,841</td>
<td>2,884,279</td>
</tr>
<tr>
<td>(A = 1000, W = 10, C = 1000)</td>
<td>1,382,933</td>
<td>2,618,970</td>
<td>3,990,350</td>
</tr>
<tr>
<td>(A = 1000, W = 100, C = 1000)</td>
<td>565,500</td>
<td>656,728</td>
<td>1,056,907</td>
</tr>
</tbody>
</table>

• Top ZDDs are better than DenseZDDs
Experimental results 5/6

- Data: Sets of matching edges of graphs

<table>
<thead>
<tr>
<th></th>
<th>top ZDD</th>
<th>DenseZDD</th>
<th>((2n \log n + n \log c)/8)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>8 × 8 grid graph</strong></td>
<td>12,196</td>
<td>16,150</td>
<td>18,014</td>
</tr>
<tr>
<td><strong>Perfect graph</strong></td>
<td>23,038</td>
<td>16,304</td>
<td>25,340</td>
</tr>
<tr>
<td>(K_{12})</td>
<td>30,780</td>
<td>39,831</td>
<td>50,144</td>
</tr>
</tbody>
</table>

(“Interroute”: a graph of real network
http://www.topology-zoo.org/dataset.html)

- Top ZDDs lose in one case, but not 1.5 times bigger than DenseZDD
Experimental results 6/6

• Data: Frequent item sets ($p$: threshold)

<table>
<thead>
<tr>
<th>Data</th>
<th>top ZDD</th>
<th>DenseZDD</th>
<th>$(2n \log n + n \log c)/8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>“mushroom” ($p = 0.001$)</td>
<td>104,580</td>
<td>91,757</td>
<td>123,576</td>
</tr>
<tr>
<td>“retail” ($p = 0.00025$)</td>
<td>59,854</td>
<td>65,219</td>
<td>62,766</td>
</tr>
<tr>
<td>T40I10D100K” ($p = 0.005$)</td>
<td>224,378</td>
<td>188,400</td>
<td>248,656</td>
</tr>
</tbody>
</table>

(Data are from [http://fimi.uantwerpen.be/data/](http://fimi.uantwerpen.be/data/))

• Not so big differences between top ZDDs and DenseZDDs
Conclusion

• Proposed compression method for ZDD
  - Expand top tree compression
  - Choose a spanning tree from an input ZDD
  - Unlike DenseZDD, top ZDD does not separate the spanning tree and non-tree edges
  - Experiments showed efficiency of top ZDDs

• Future work
  - Dynamic programming on top ZDDs
  - Faster operations on top ZDDs
  - Finding better spanning trees for compression
  - Complexity of finding optimal spanning tree
  - Applying proposed method for general DAGs
Thank you for listening!