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

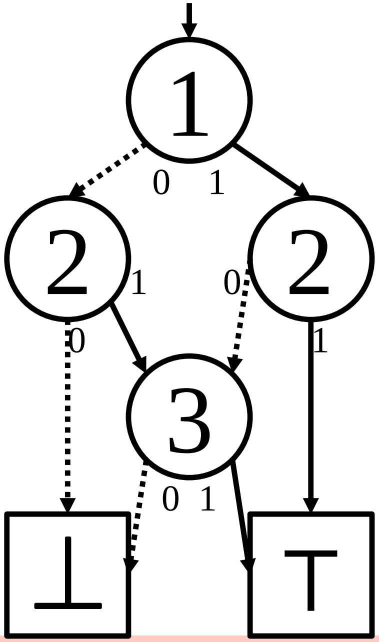
ZDD
DAG compression
Top tree compression

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} }



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]

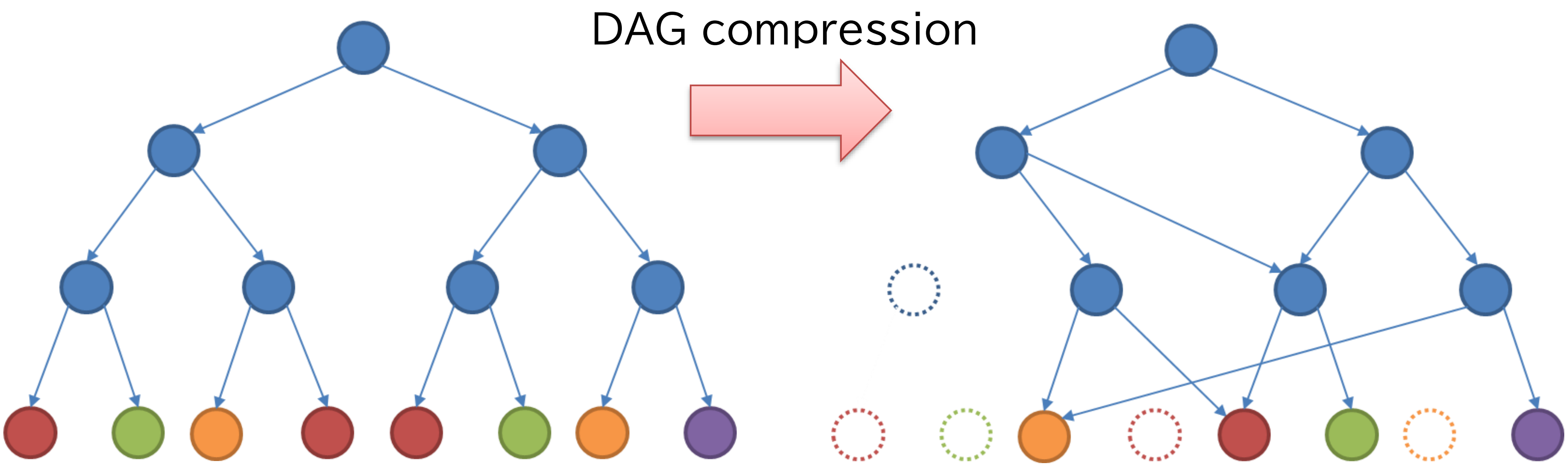
- Succinct data structures
  - Labeled tree: LOUDS [Jacobson 89]
    BP [Munro, Raman 01]
  - Unlabeled tree: [Ferragina et al. 09]

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

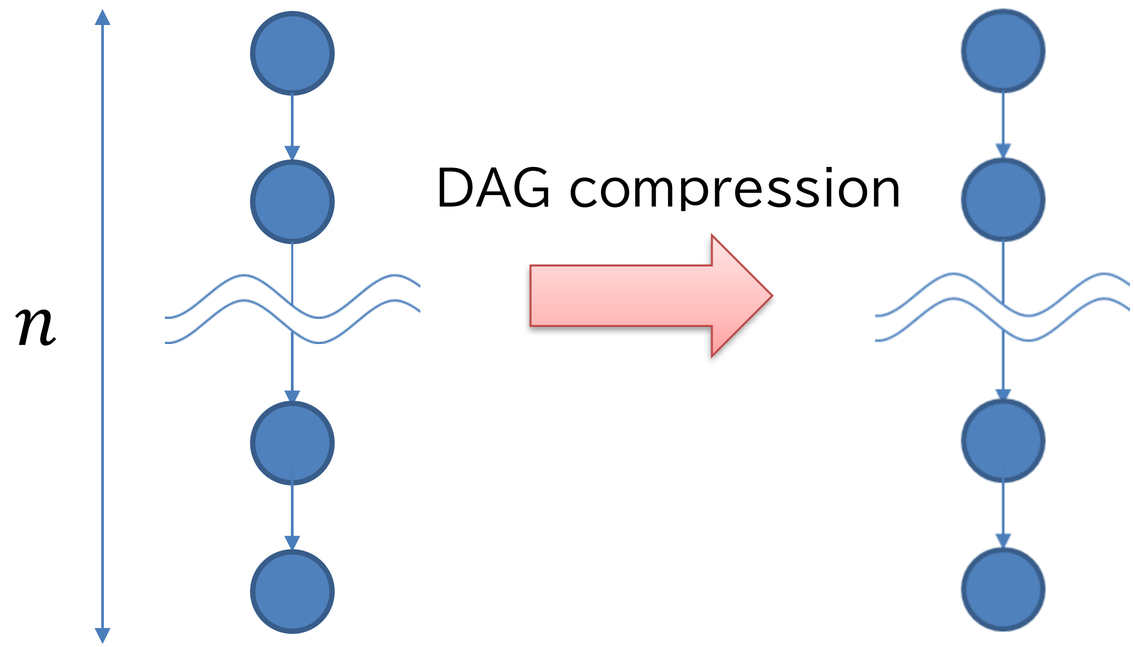
DAG compression

- Compress labeled DAGs
  - [Downey et al. 80]
  - Share all equivalent subtrees



Problem of DAG comp.

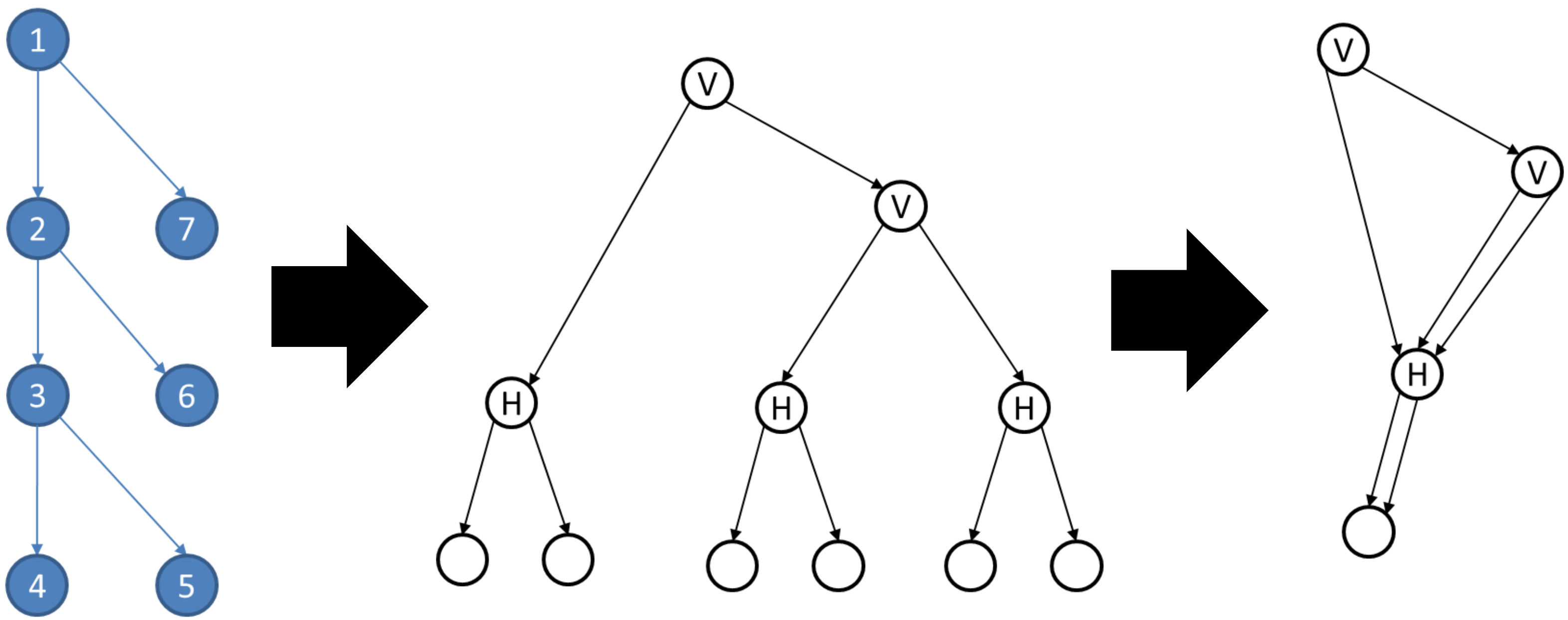
- Cannot compress substructures that repeats vertically
- Example:



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 $\mathcal{T}$

top DAG $\mathcal{TD}$

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 [Hübchle-Schneider and Raman 15]
- Optimal construction [Lohrey et al. 17], [Dudek, Gawrychowski 18]
  - #node = $O(n/\log_\sigma n)$ (information theoretic lowerbound)

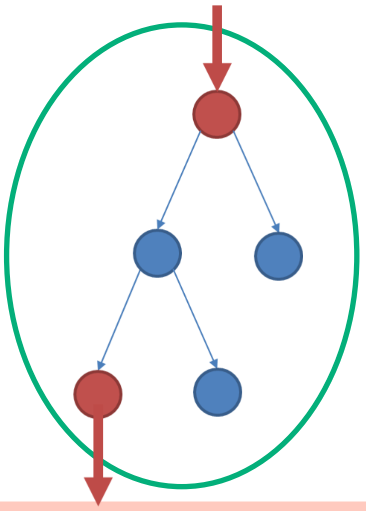
(n: #node of the input tree, σ: #label)

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

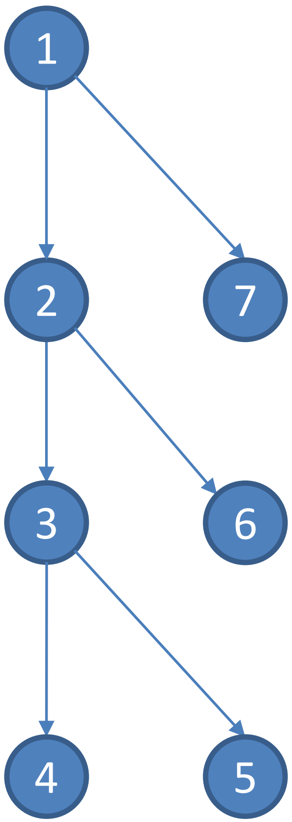
Top tree

- 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

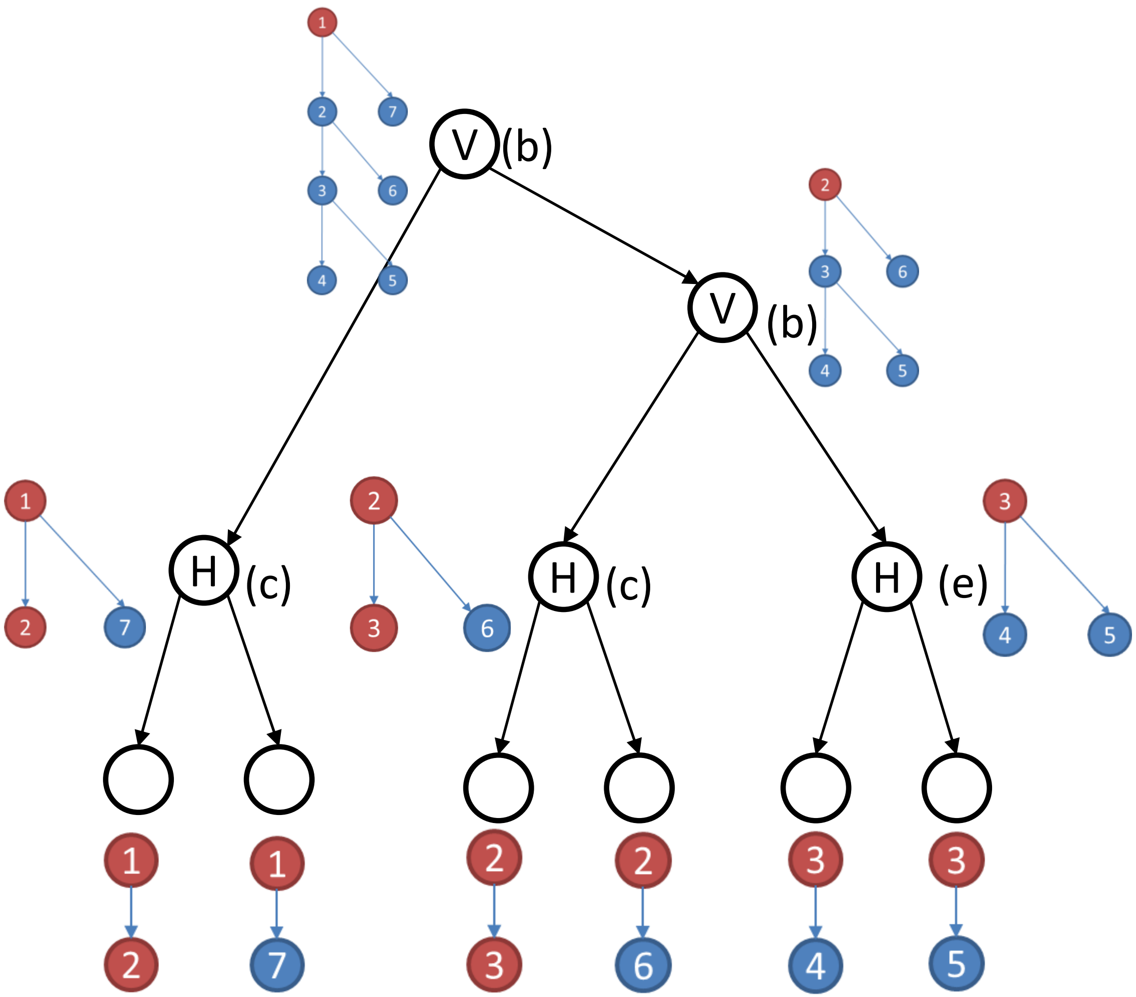


Top tree

- Example:



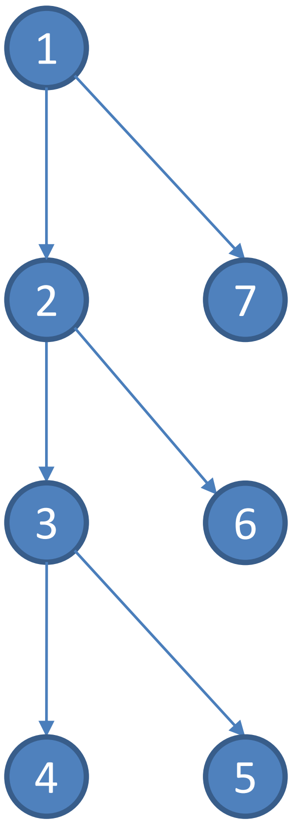
Input tree T



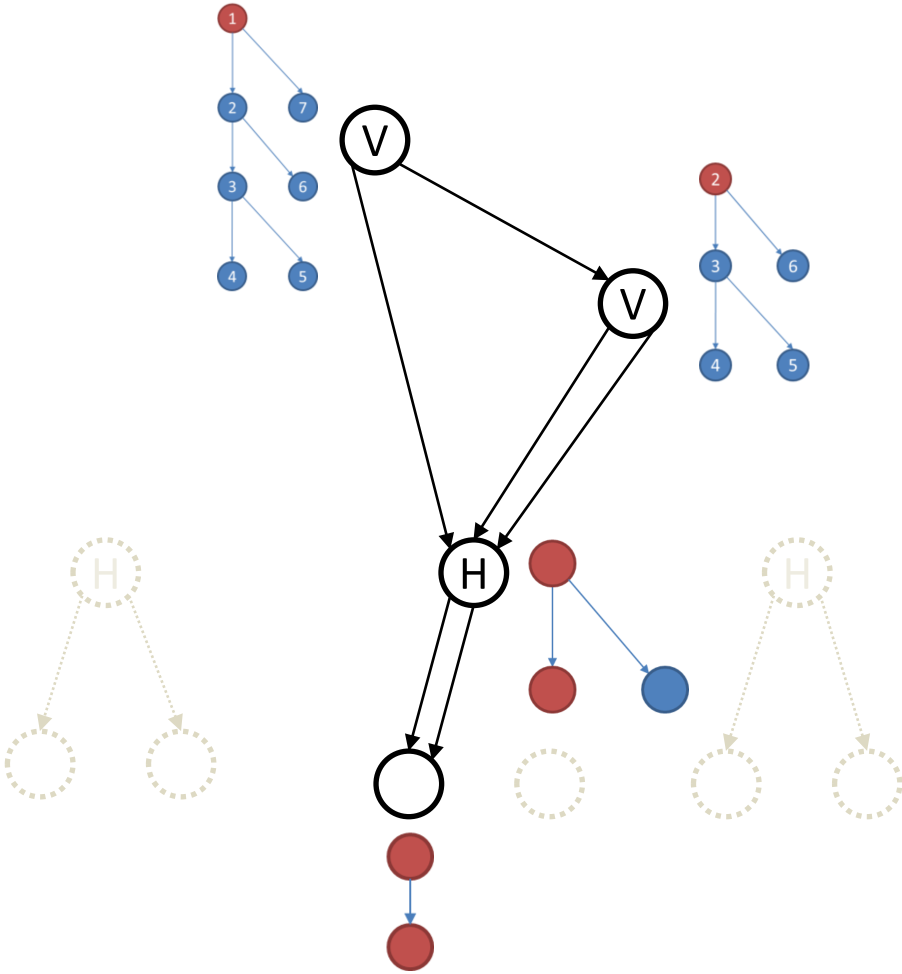
top tree $\mathcal{T}$

Top tree

- Example:



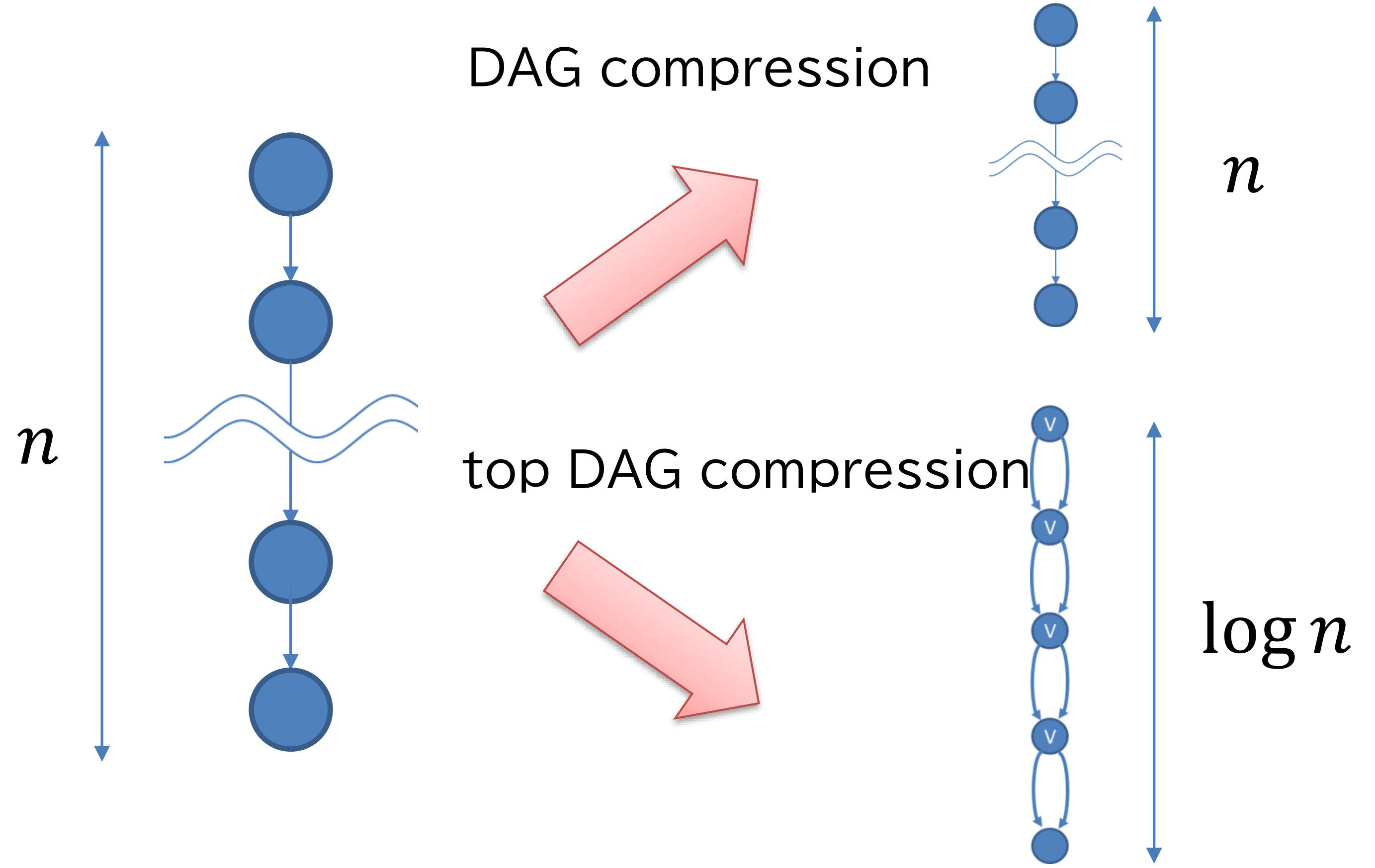
Input tree T



top DAG $\mathcal{TD}$

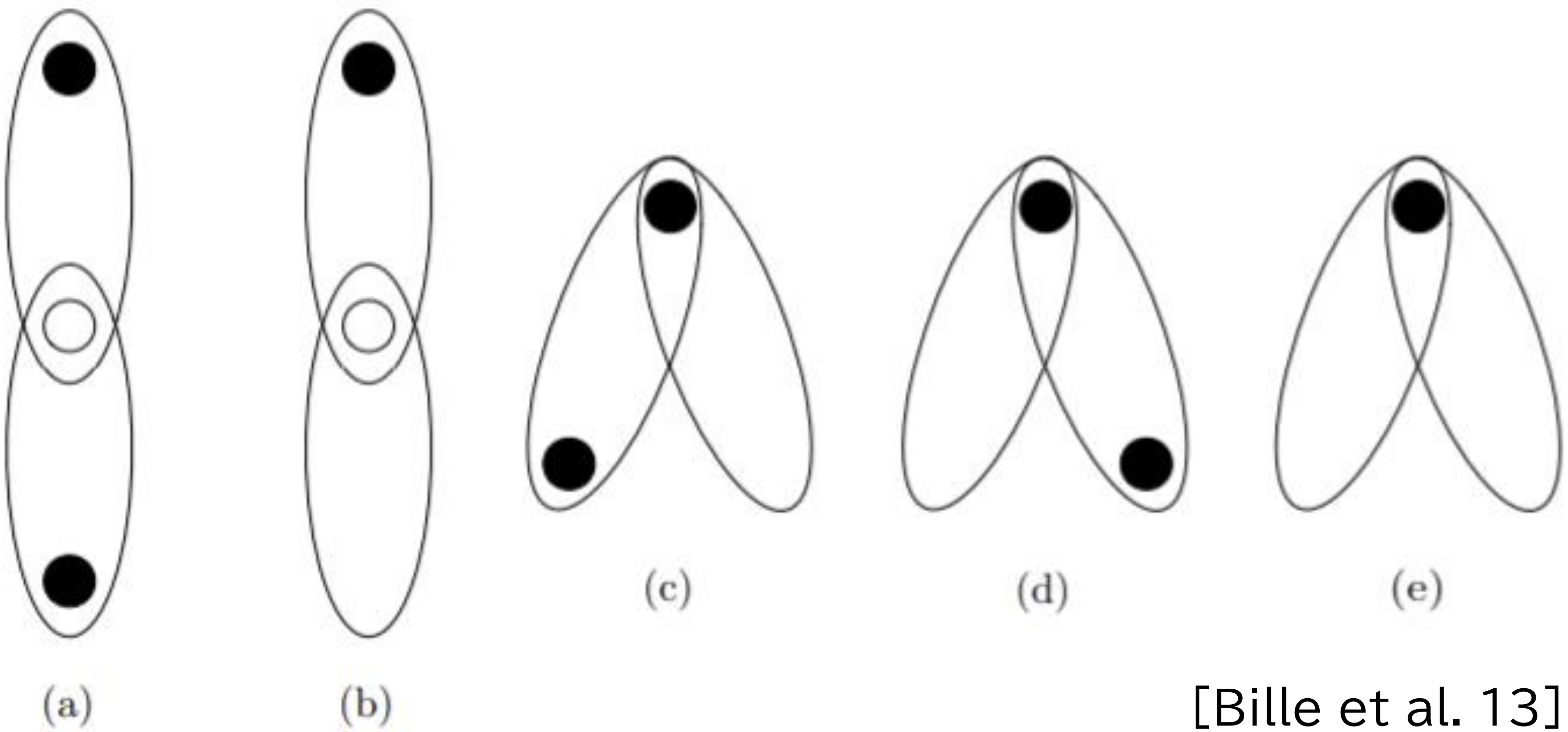
Advantage of top DAG

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



Two types of merging

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

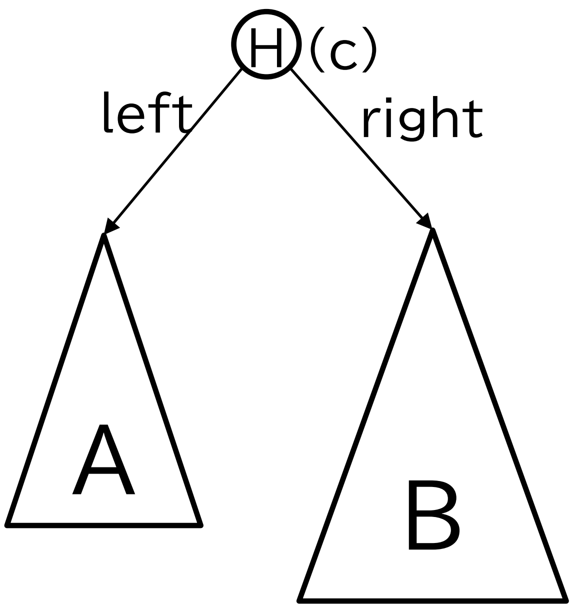


[Bille et al. 13]

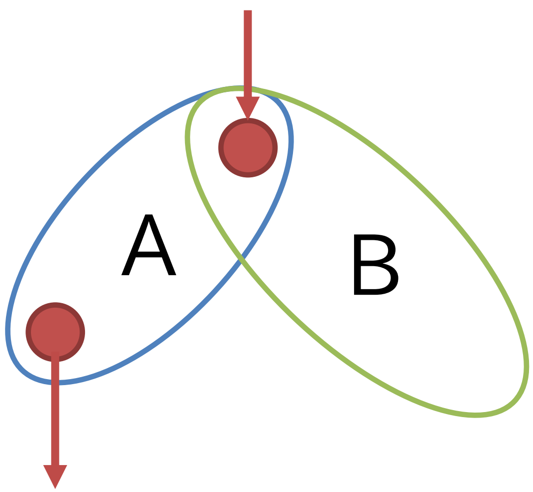
Horizontal merge

- Merge two clusters
  that have the same node
  as their top boundary nodes

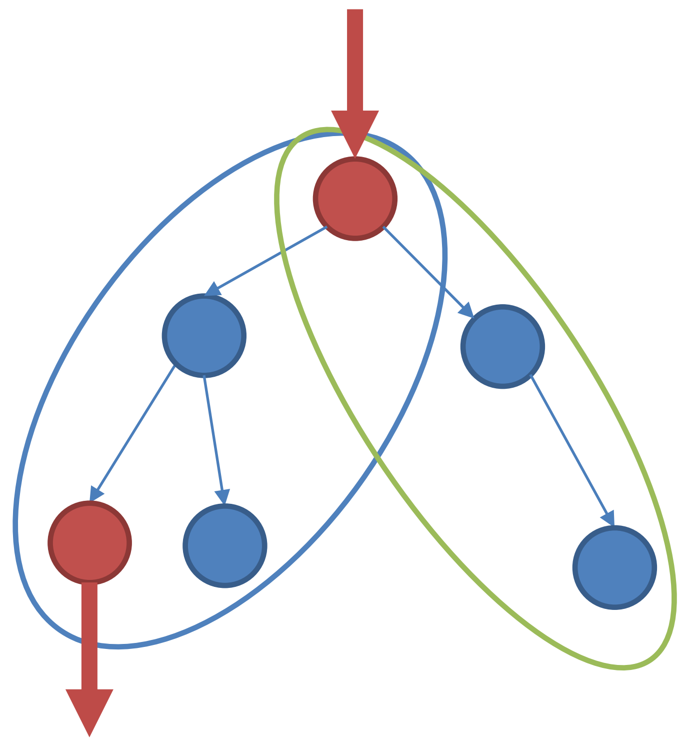
top tree



Corresponding clusters

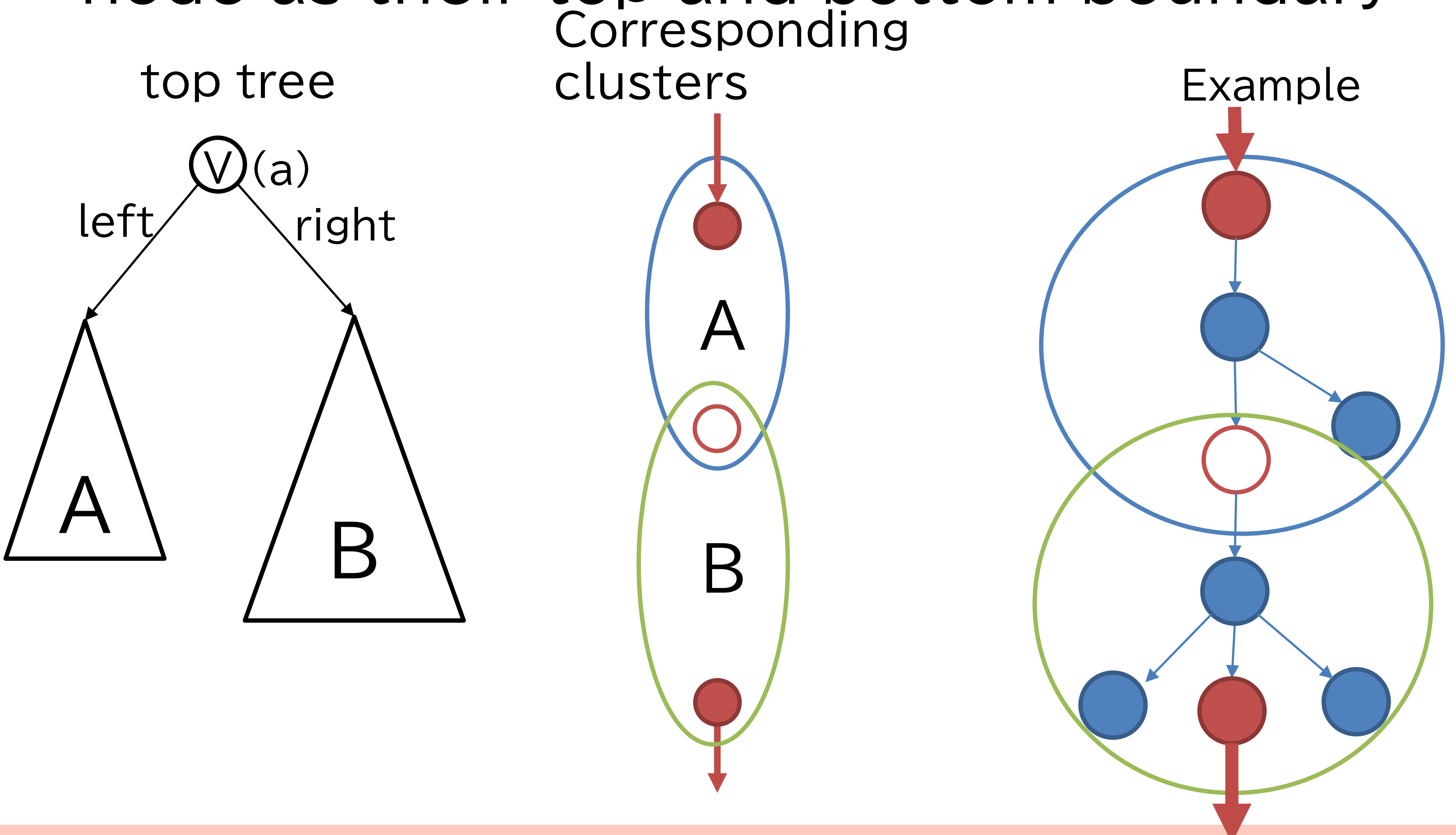


Example



Vertical merge

- Merge two clusters that have the same node as their top and bottom boundary

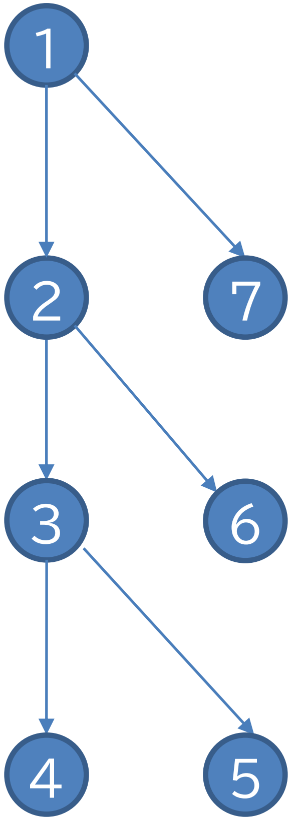


Top tree construction

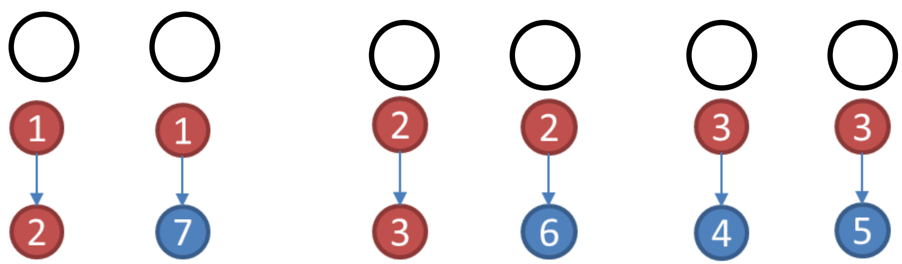
- Top tree is not uniquely determined from the input tree
- Greedy construction
  - Repeat 1—3 until the tree T become 1 edge
  - 1. Choose pairs of clusters that
    can be horizontally merged as much as possible
  - 2. Choose pairs of clusters that
    can be vertically merged from remaining nodes as much as possible
  - 3. Merge the all pairs chosen at 1 and 2

Greedy construction

- Example



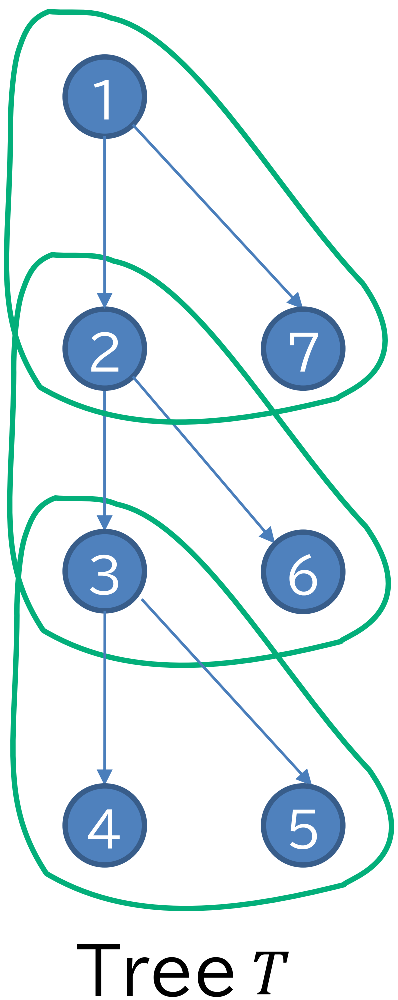
Tree T

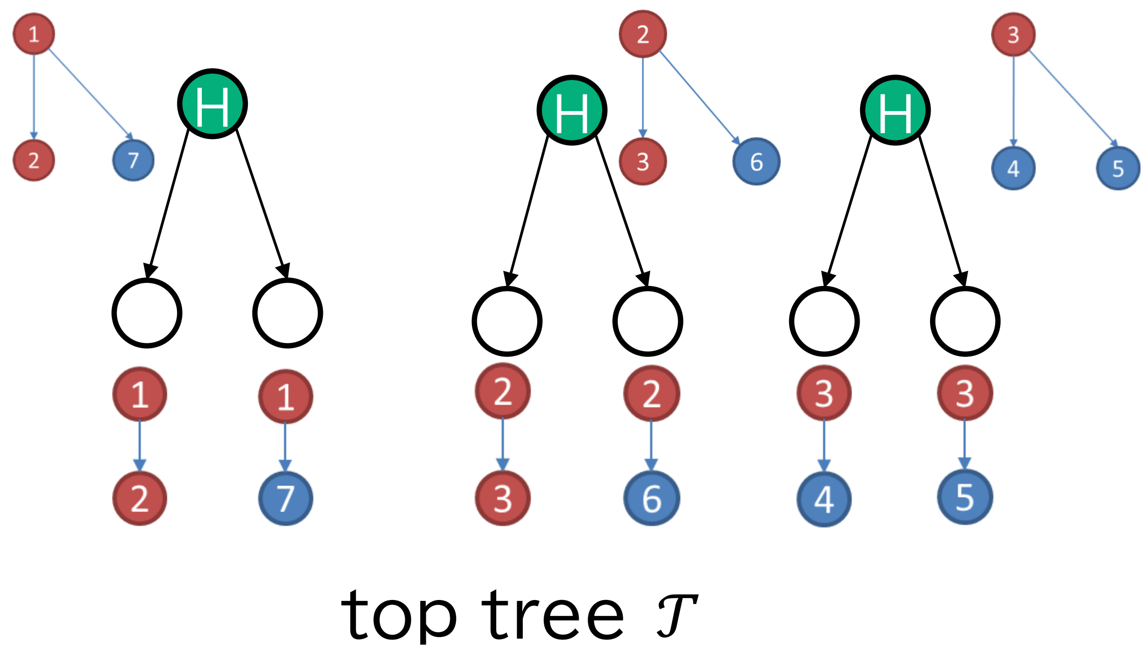


top tree $\mathcal{T}$

Greedy construction

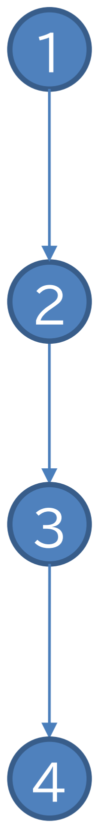
- Example



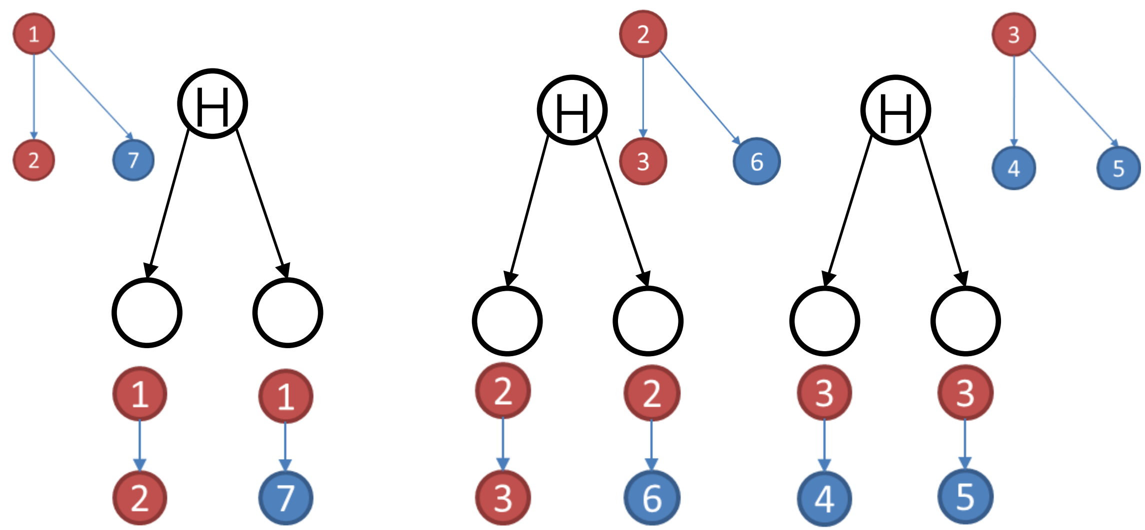


Greedy construction

- Example



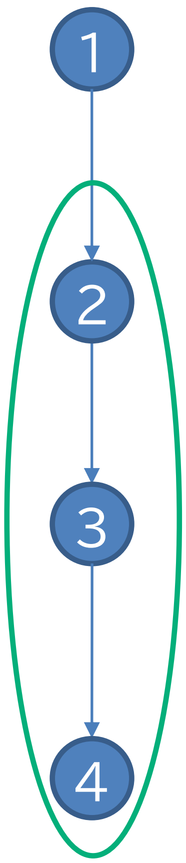
Tree T



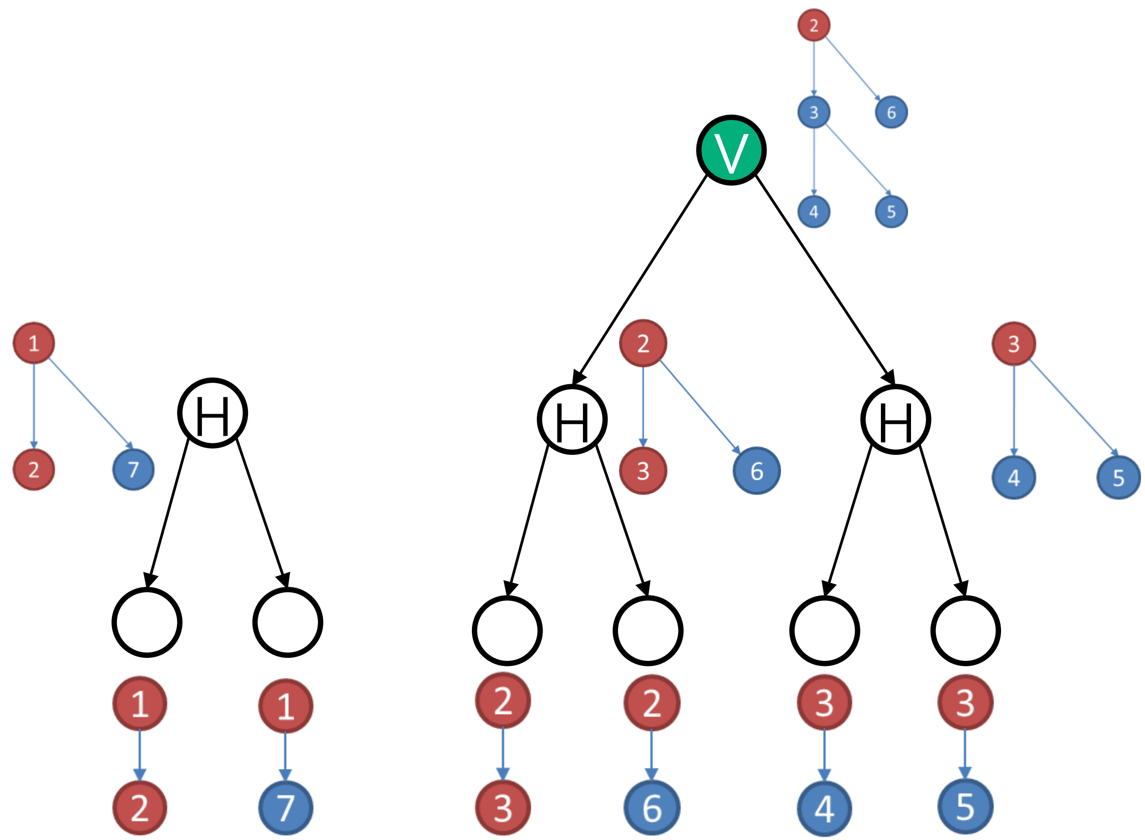
top tree $\mathcal{T}$

Greedy construction

- Example



Tree T

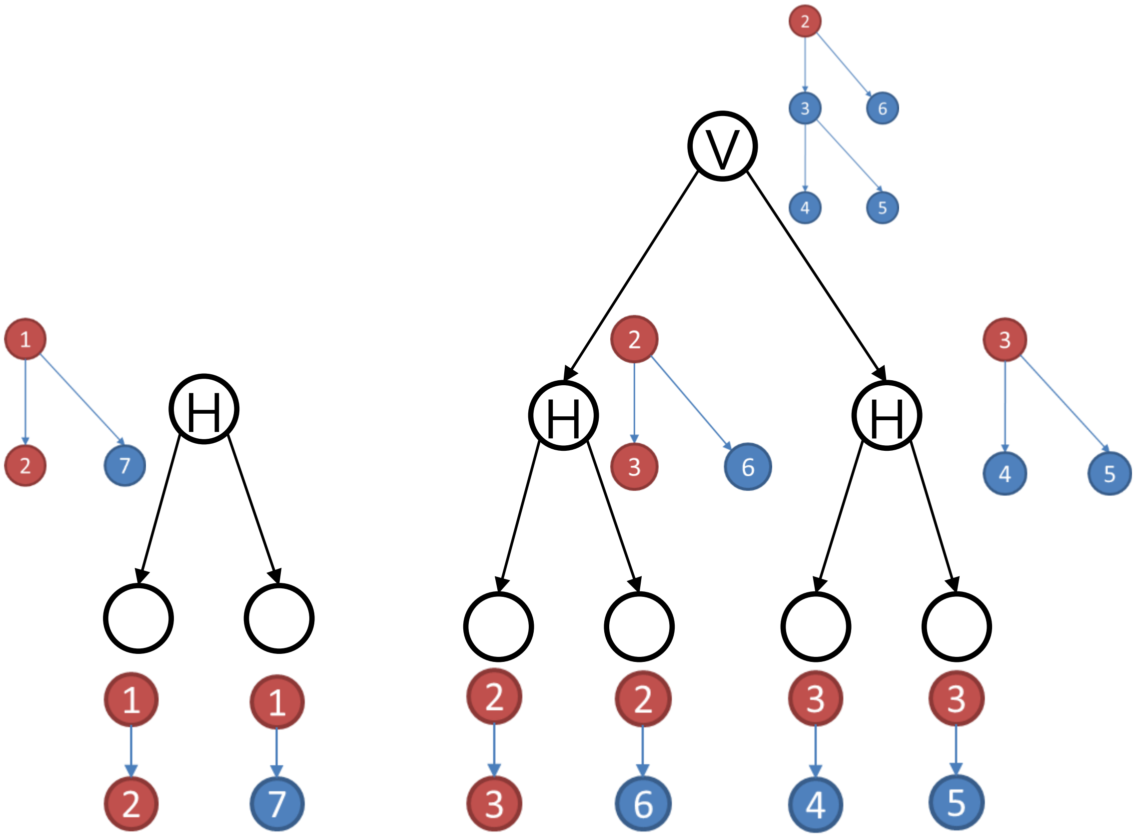


top tree $\mathcal{T}$

Greedy construction

- Example

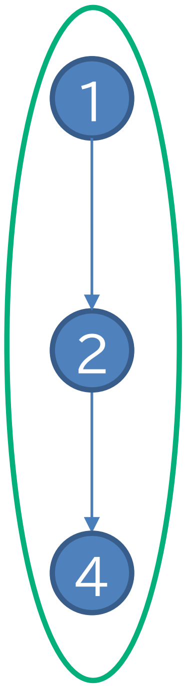
Tree T



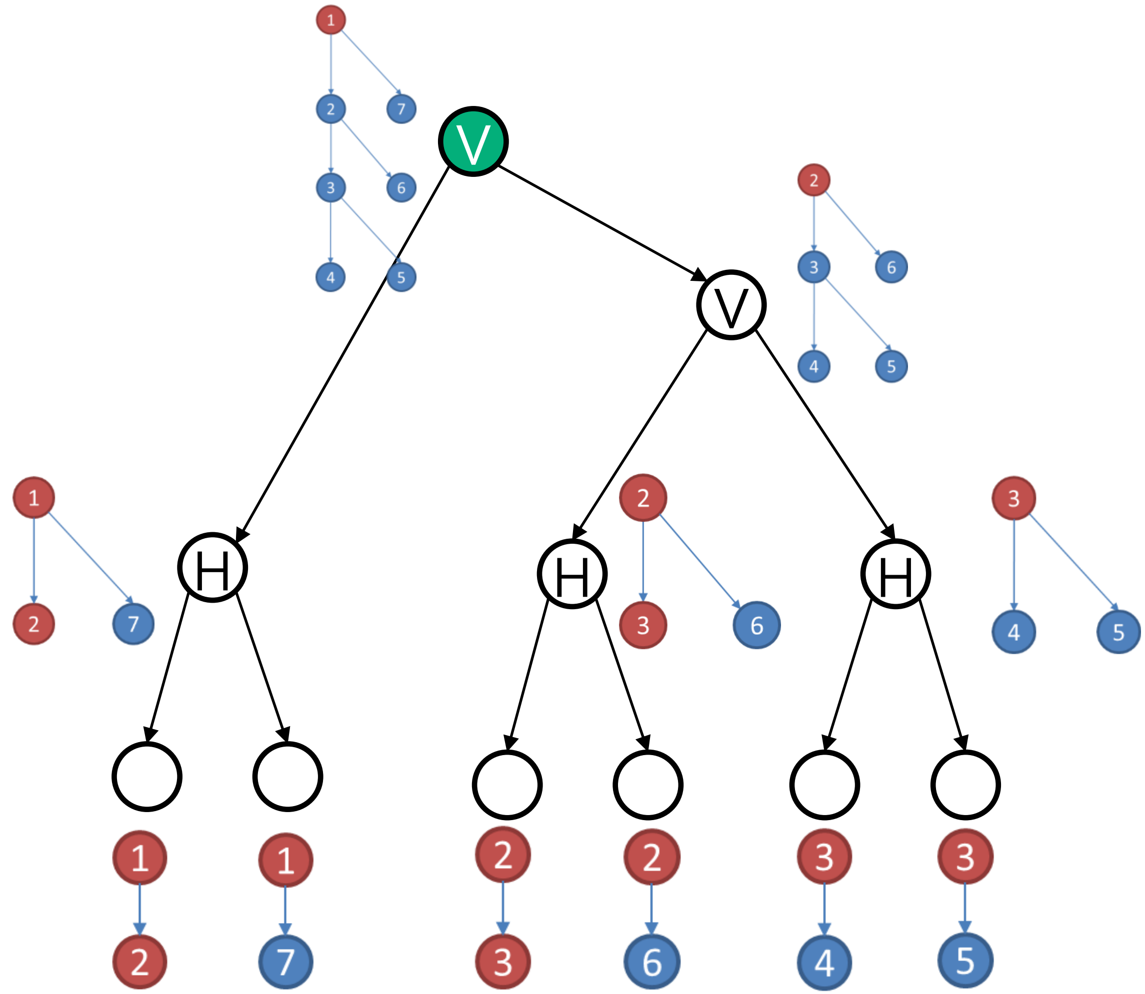
top tree $\mathcal{T}$

Greedy construction

- Example



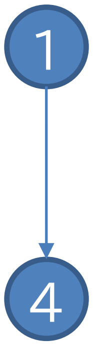
Tree T



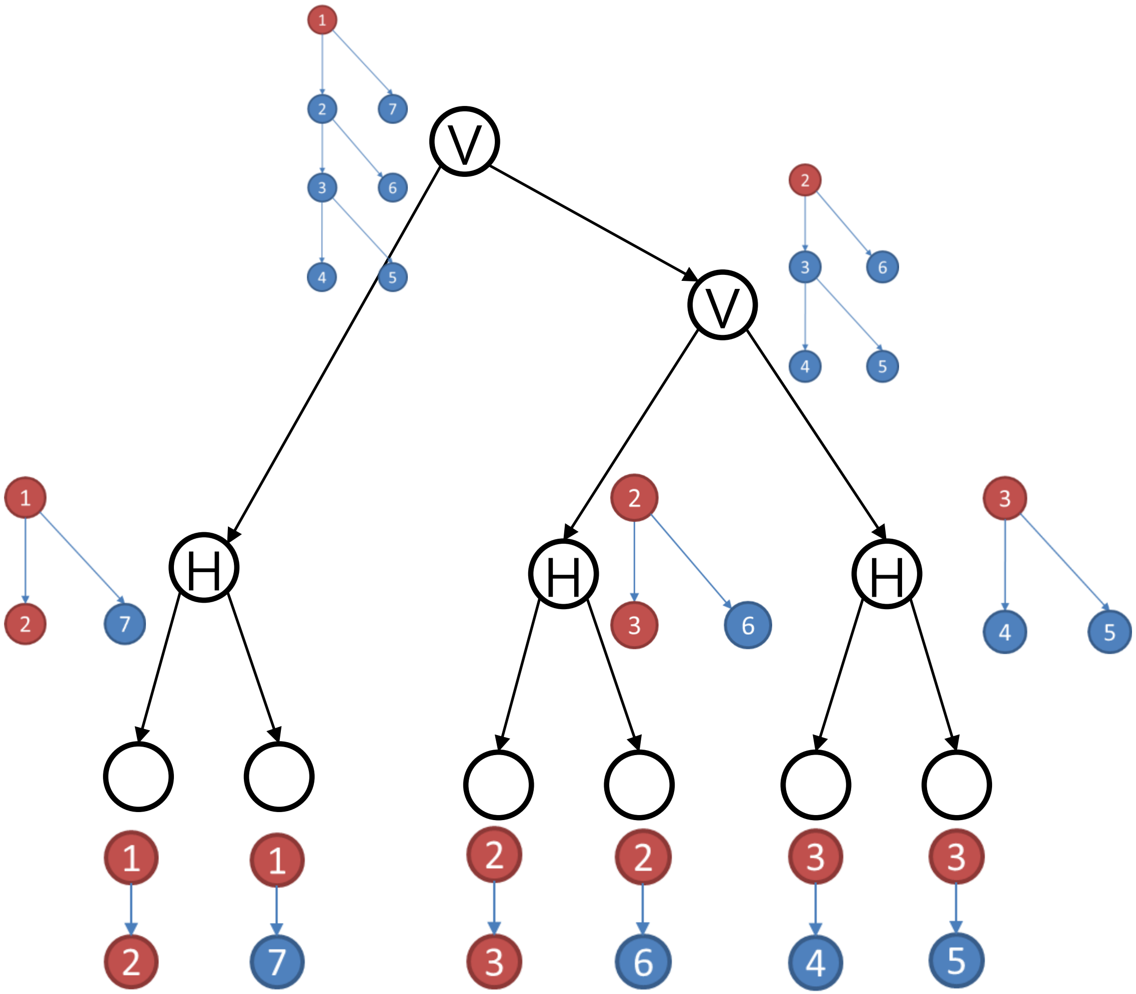
top tree $\mathcal{T}$

Greedy construction

- Example



Tree T



top tree $\mathcal{T}$

Complexity: Greedy method

- n: #node of an input tree, σ: #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)
  - $\mathrm{access}(x)$: label of x
  - $\mathrm{parent}(x)$: preorder of the parent of x
  - $\mathrm{depth}(x)$: depth of x
  - $\mathrm{height}(x)$: height of x
  - $\mathrm{size}(x)$: number of nodes in $T(x)$
  - $\mathrm{firstchild}(x)$: preorder of the first child of x
  - $\mathrm{nextsibling}(x)$: preorder of the next sibling of x
  - $\mathrm{la}(x, i)$: preorder of i-th ancestor of x
  - $\mathrm{nca}(x, y)$: preorder of nearest common ancestor of x and y

Proposed method

Top ZDD

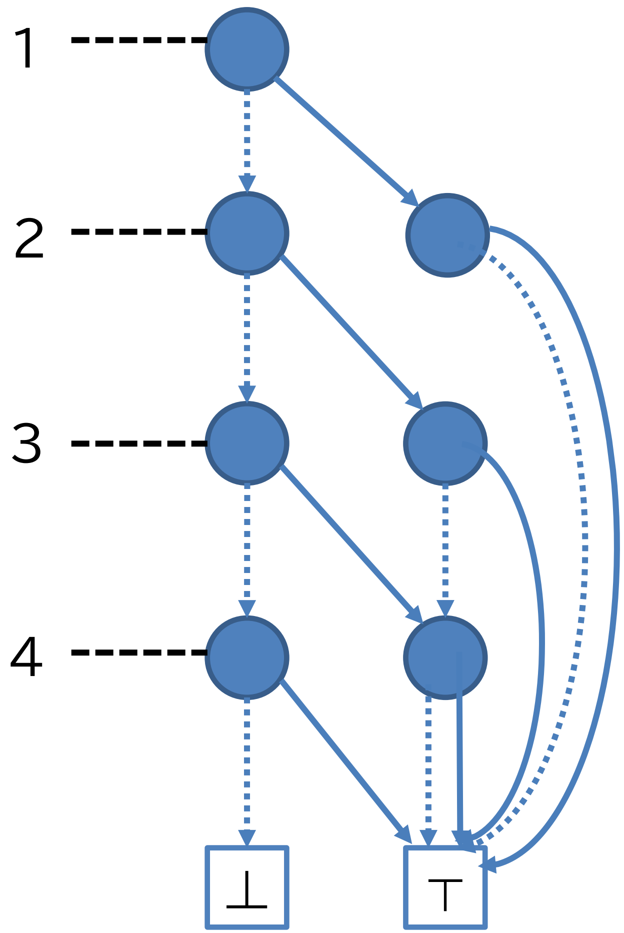
Construction algorithm
Experiment

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

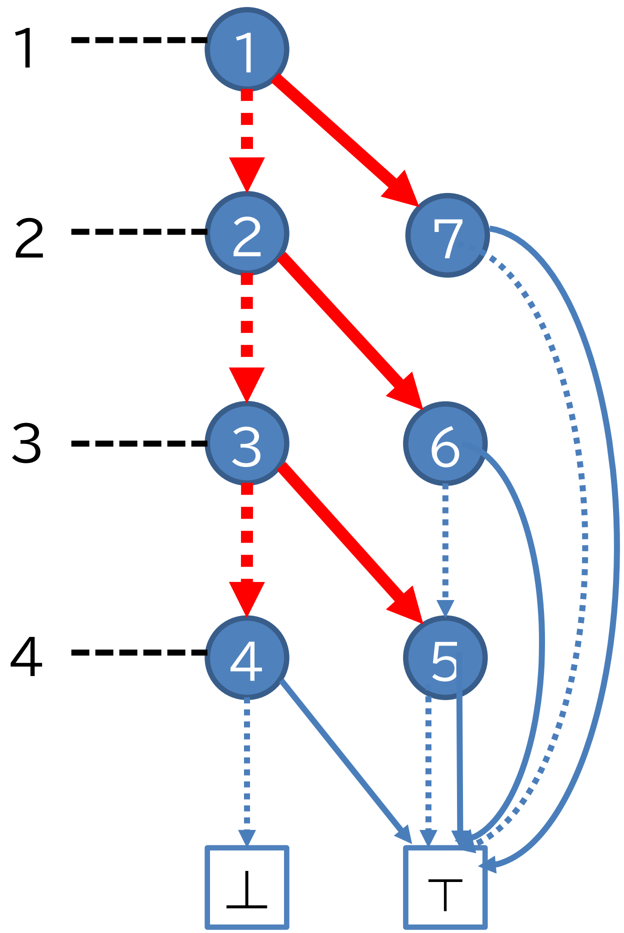
- Step 0. Input



Original ZDD

Example of construction

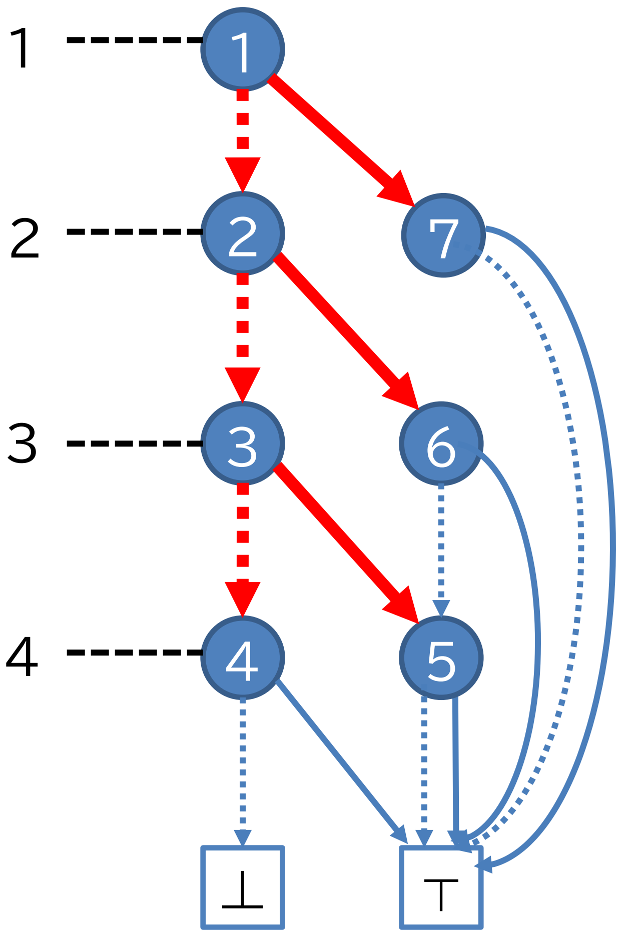
- Step 1. Find a spanning tree



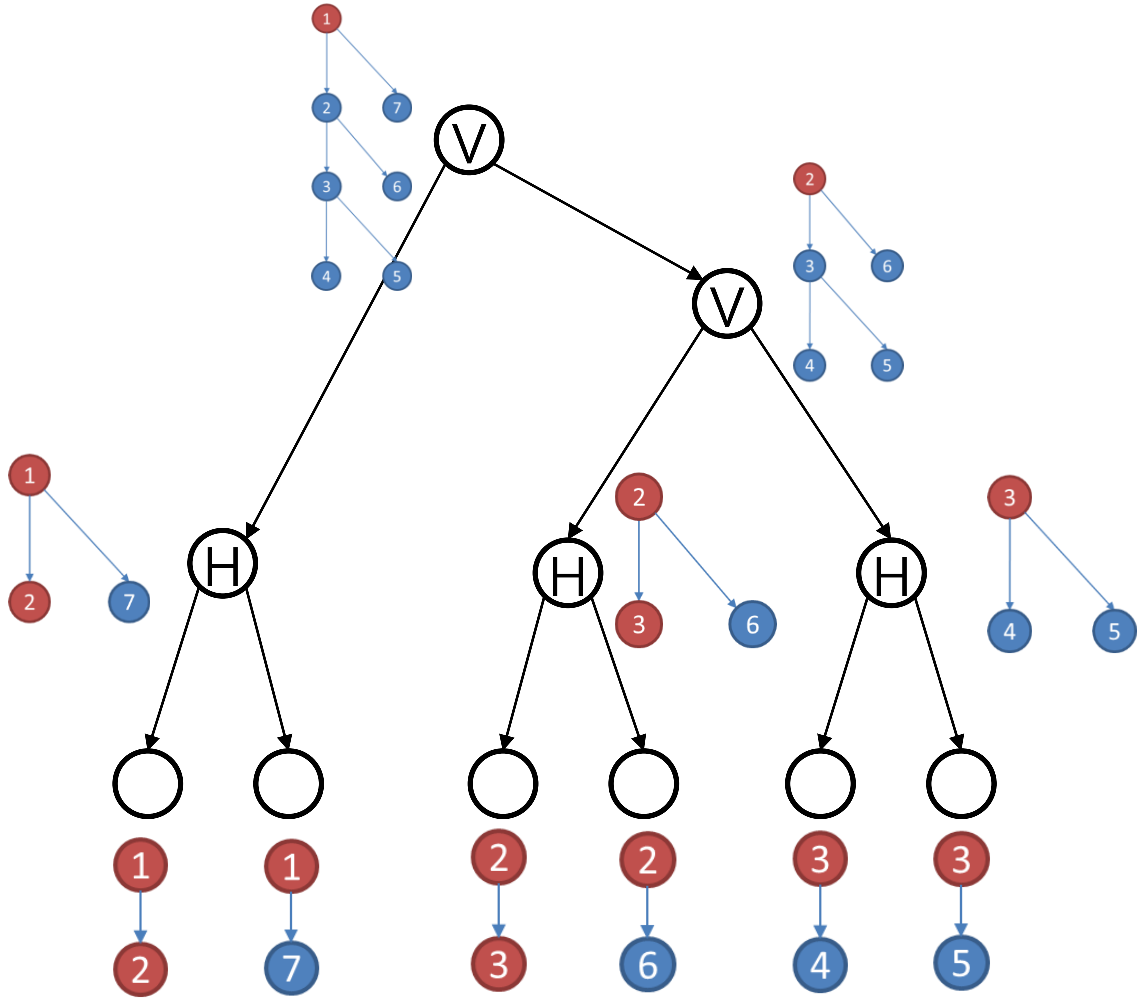
Original ZDD

Example of construction

- Step 2. Construct a top tree

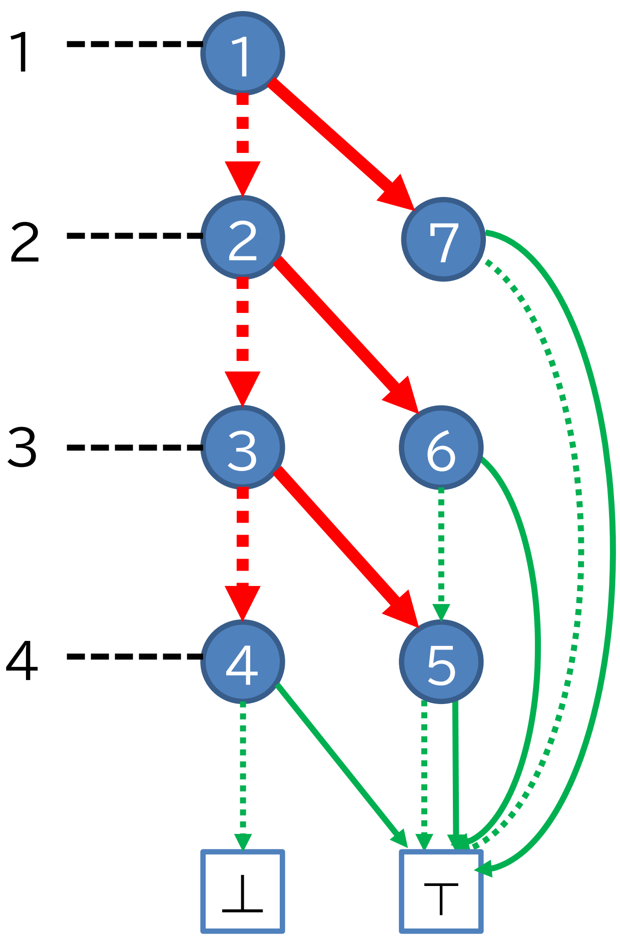


Original ZDD

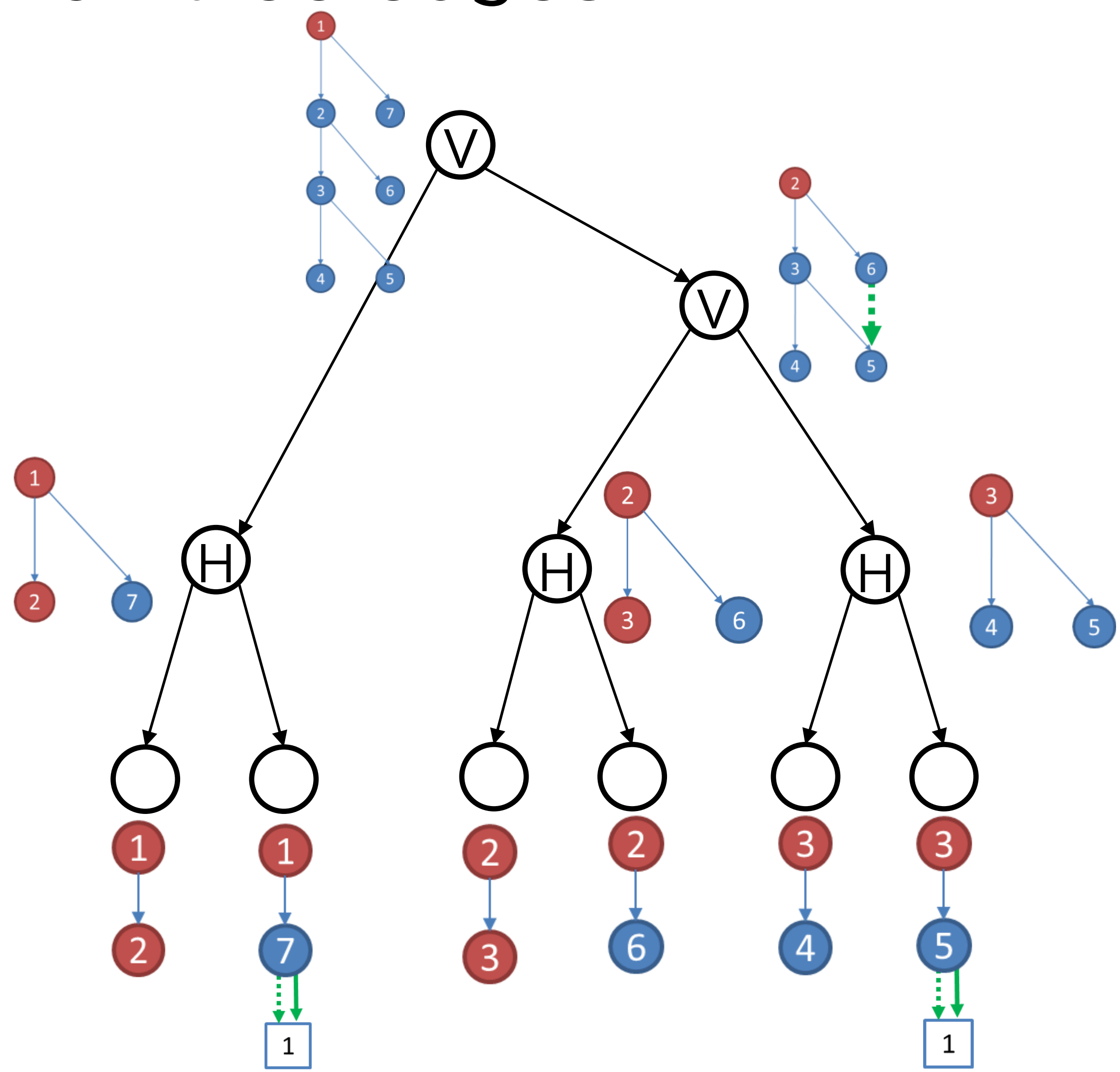


Example of construction

- Step 3. Store non tree edges

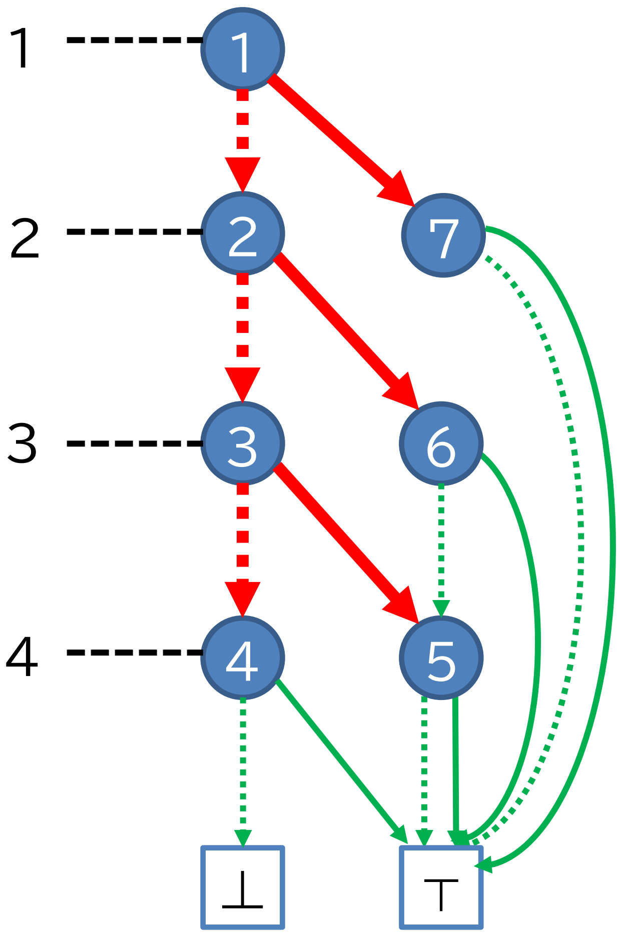


Original ZDD

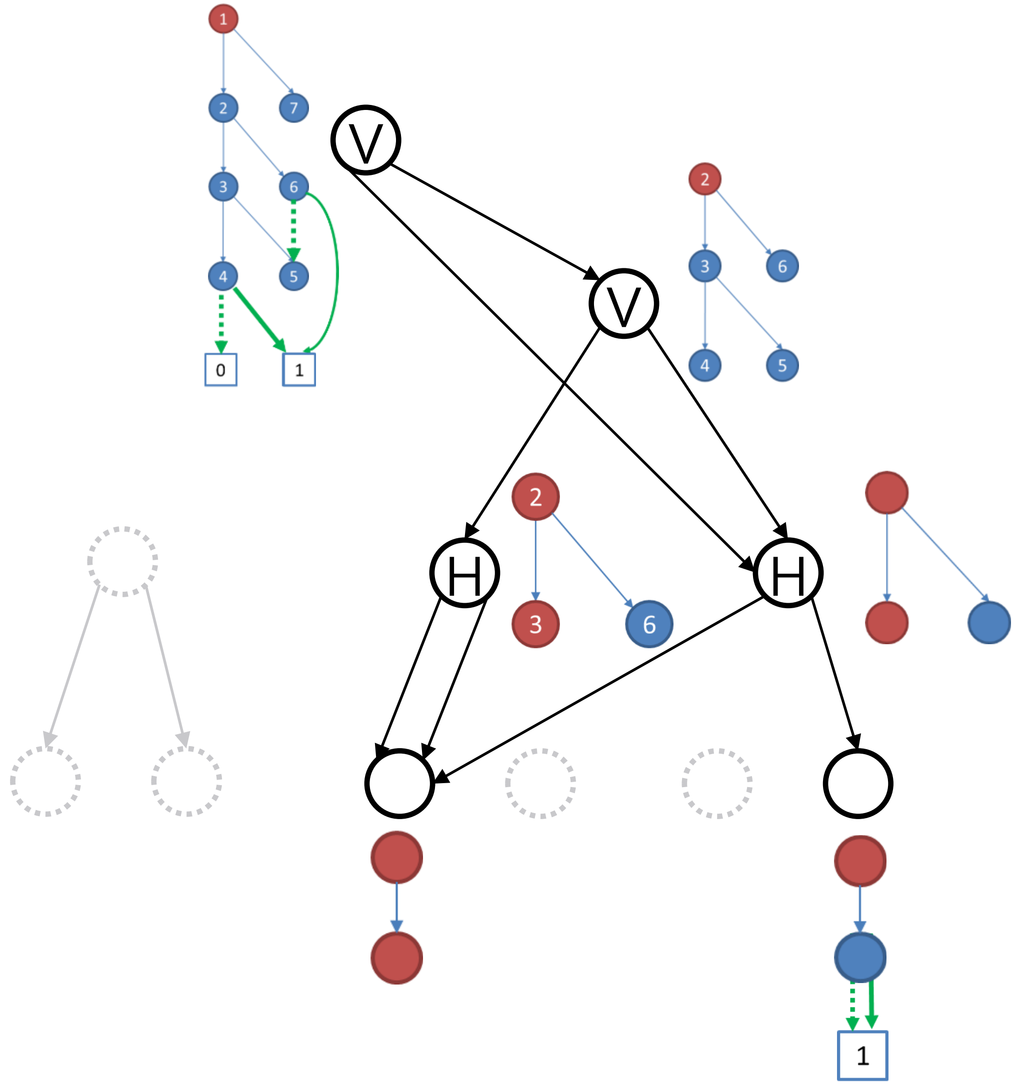


Example of construction

- Step 4. Share equivalent subtrees



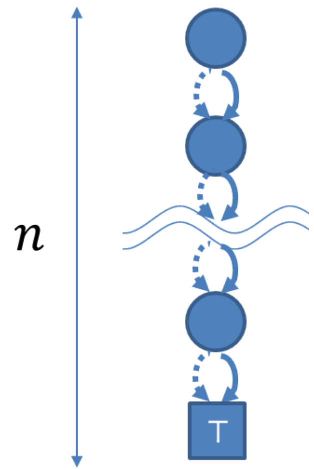
Original ZDD

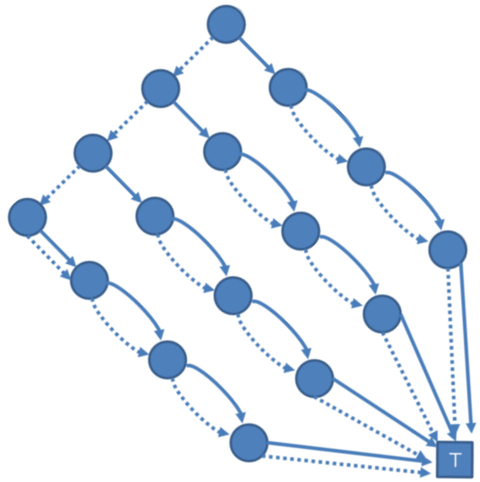


Theoretical results

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, σ: #label) (byte)
- Data sets
  - $\{S \subseteq U \mid |S| \leq B\}$, where $|U| = A$
  - $\{S \subseteq U \mid {}^{\forall}e \in S, {}^{\exists}f \in S \text{ s.t. } |e - f| \leq B\} \cup U$, where $U = \{1, \ldots, A\}$
  - 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,

$$\{S \subseteq U \mid |S| \leq B\}$$

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

(bytes)

- Top ZDDs are 2—20 times smaller than DenseZDDs

Experimental results 2/6

- Data: For an underlying set U of size A,

$$\{S \subseteq U \mid {}^\forall e \in S, {}^\exists f \in S \text{ s.t. } |e - f| \leq B\} \cup U$$

	top ZDD	DenseZDD	$(2n \log n + n \log c)/8$
$A = 500, B = 250$	2,431	227,798	321,594
$A = 1000, B = 500$	2,511	321,594	1,440,375

(bytes)

- Top ZDDs are 100—125 times smaller than DenseZDDs

Experimental results 3/6

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

	top ZDD	DenseZDD	$(2n \log n + n \log c)/8$
A =1000	2,254	4,185	3,750
$A = 50000$	2,464	178,764	300,000

(bytes)

- Top ZDDs are highly effective because the inputs have simple structure

Experimental results 4/6

- Data: Solutions of knapsack problems
  - $w_i \in [1, W]$: random weight ($w_i \geqq w_{i+1}$), C: capacity

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

- Top ZDDs are better than DenseZDDs

Experimental results 5/6

- Data: Sets of matching edges of graphs

	top ZDD	DenseZDD	$(2n\log n + n\log c)/8$
8×8 grid graph	12,196	16,150	18,014
Perfect graph K_{12}	23,038	16,304	25,340
"Interroute"	30,780	39,831	50,144

(bytes)

(*"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)

	top ZDD	DenseZDD	$(2n \log n + n \log c)/8$
"mushroom" ($p = 0.001$)	104,580	91,757	123,576
"retail" ($p = 0.00025$)	59,854	65,219	62,766
T40I10D100K" ($p = 0.005$)	224,378	188,400	248,656

(bytes)

(Data are from 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!