Enumerating All Subgraphs under Given Constraints Using Zero-suppressed Sentential Decision Diagrams

Yu Nakahata[1], Masaaki Nishino[2], Jun Kawahara[1], Shin-ichi Minato[1]

[1]Graduate School of Informatics, Kyoto University, Japan

[2]NTT Communication Science Laboratories, NTT Corporation, Japan

June 16, SEA 2020

Outline

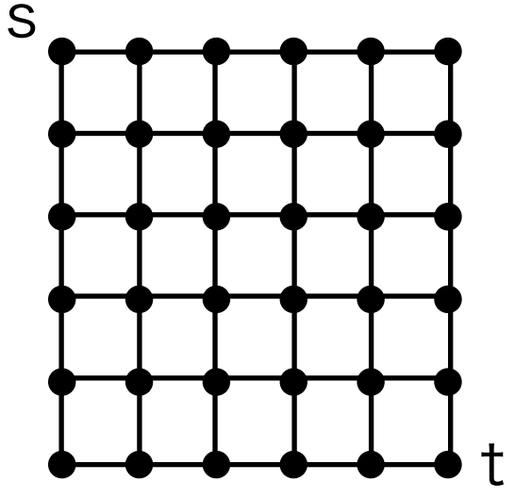
- Introduction
  - Subgraph enumeration with decision diagrams
- Our target representation: ZSDDs
- Proposed algorithms
- Experiments and remarks

Subgraph enumeration

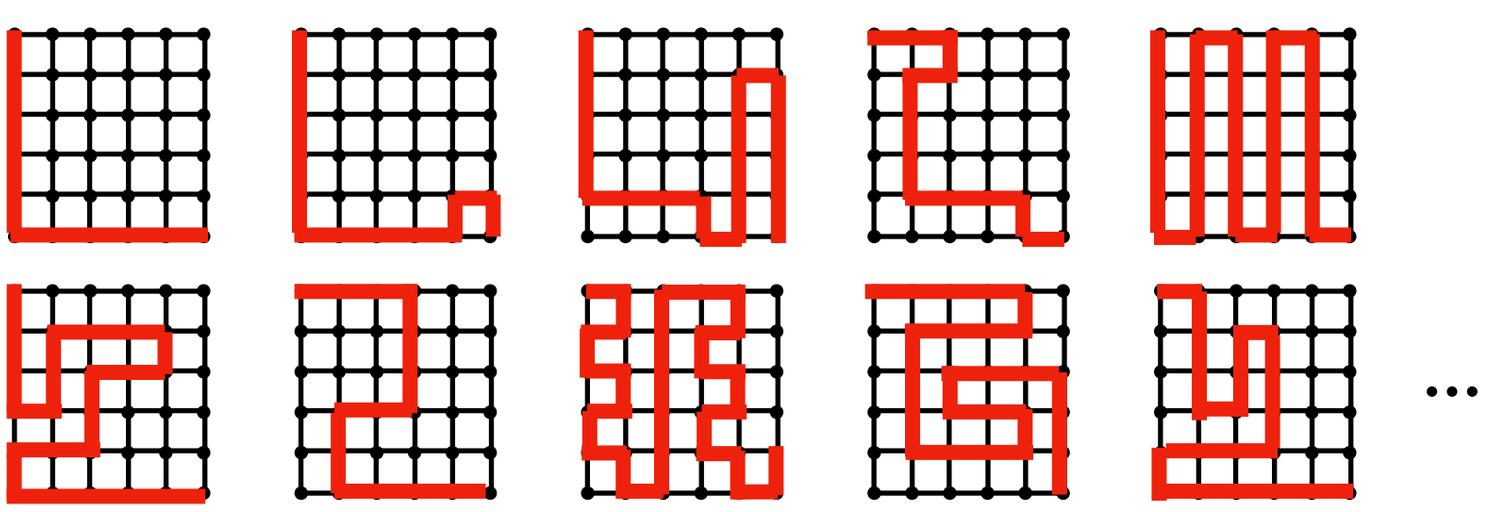
Input: A graph G

Output: All specific subgraphs (e.g., paths and cycles) of G

- Important in many areas of computer science
- Output can be exponentially larger than the input size



6x6 grid graph
(36 vertices
60 edges)



1,262,816 paths [1]

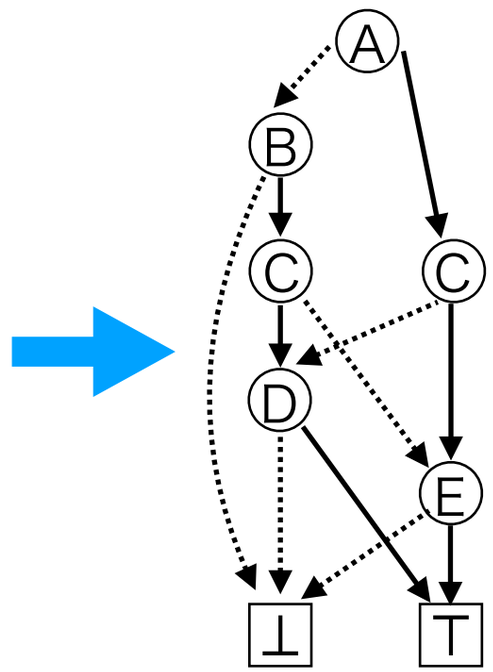
[1] https://oeis.org/A007764

ZDD (Zero-suppressed Binary Decision Diagram) [Minato, DAC '93]

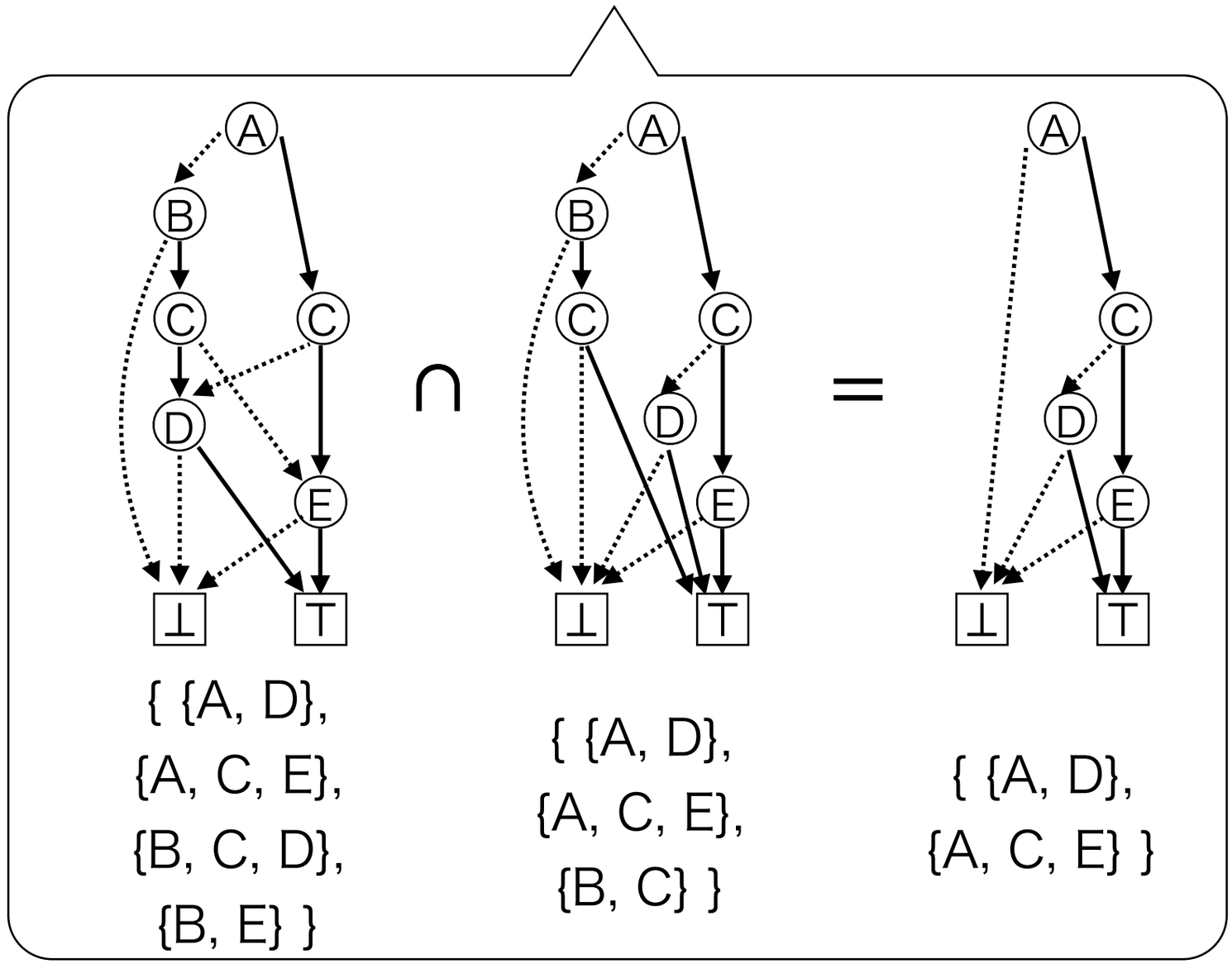
- ZDDs are compact representations of set families
- ZDDs support several queries on set families
  - Counting, random sampling, **Apply operations**

{ {A, D},
{A, C, E},
{B, C, D},
{B, E} }

Set family



ZDD



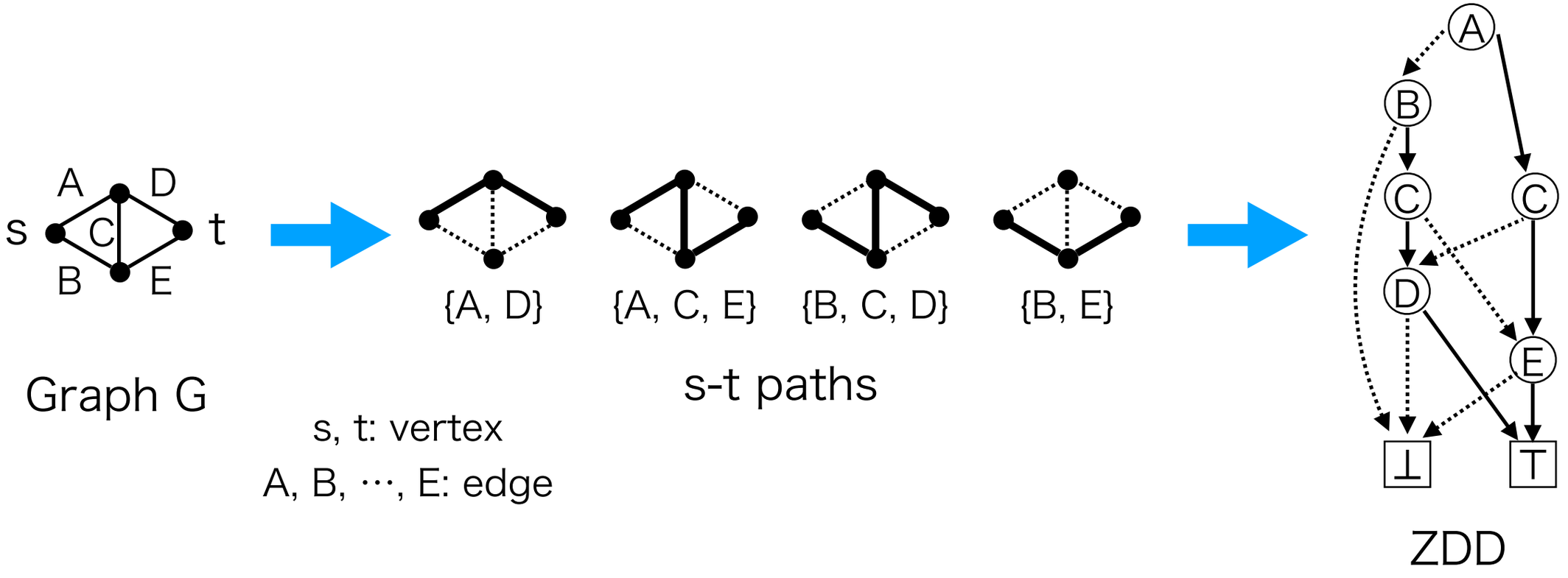
{ {A, D},
{A, C, E},
{B, C, D},
{B, E} }

{ {A, D},
{A, C, E},
{B, C} }

{ {A, D},
{A, C, E} }

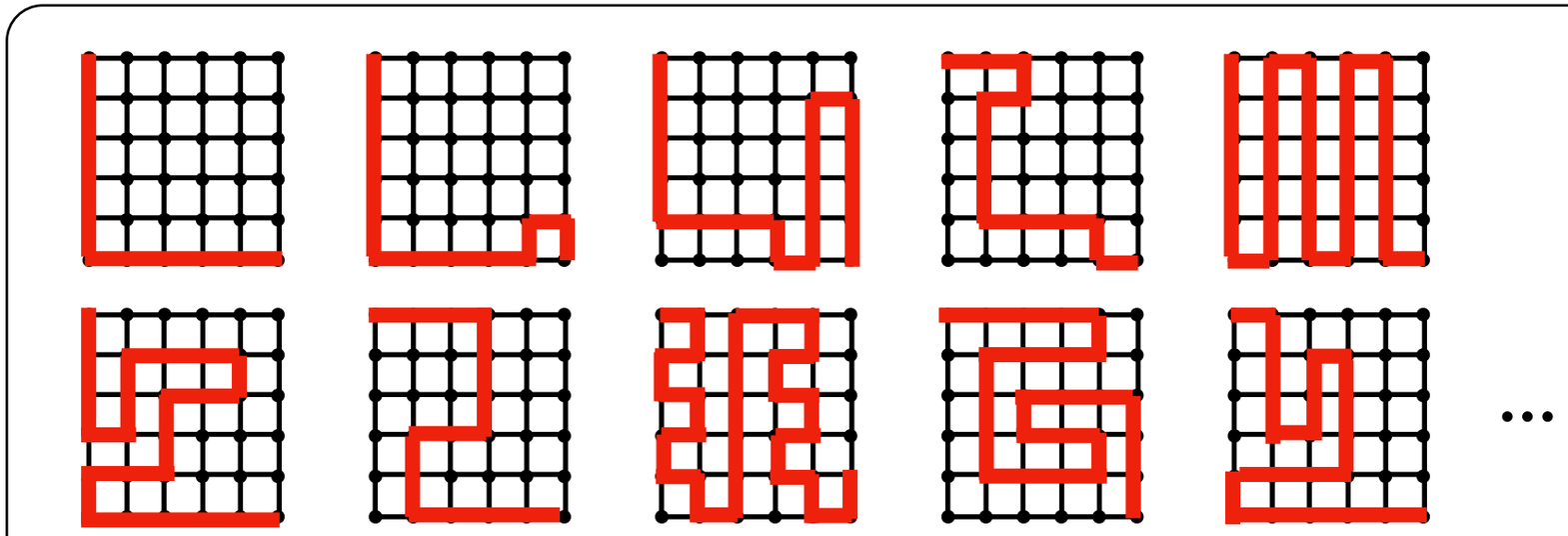
Subgraph enumeration with ZDDs

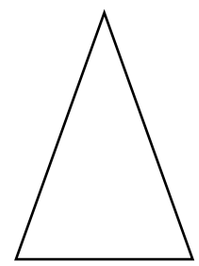
- An (edge-induced) subgraph <=> its edge set
- A set of subgraphs <=> a familiy of edge sets
- => A ZDD can represent a set of subgraphs



Merit of subgraph enumeration with ZDDs

- A ZDD can represent a set of subgraphs compactly
- Applied for several graph-related problems
  (e.g., [Inoue et al., IEEE Trans. Smart Grid, '14], [Nakahata et al., SEA '18])



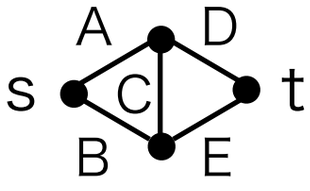


(Calculated by Graphillion*)

*https://github.com/takemaru/graphillion

Top-down construction of ZDDs

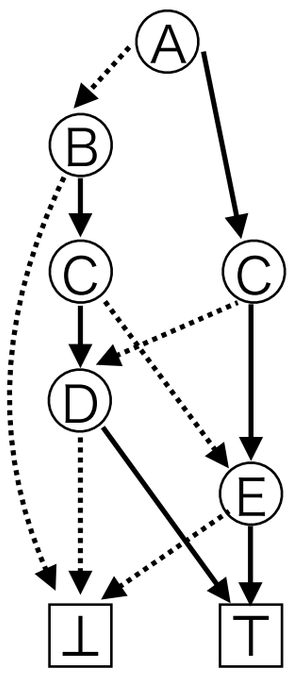
Input: A graph G
Output: A **ZDD** representing a set of all specific subgraphs (e.g., paths and cycles) of G



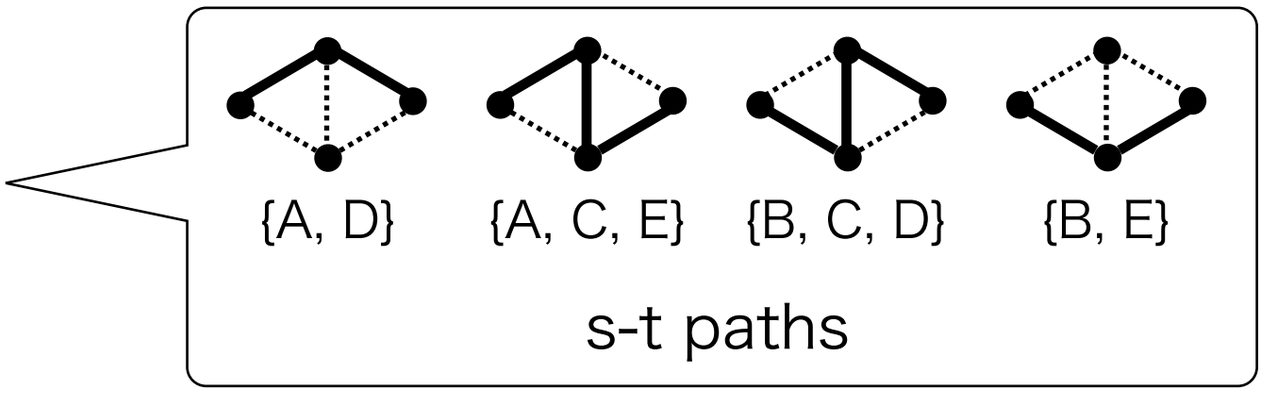
Graph G

- Construct a ZDD directly without explicitly enumerating subgraphs
- The size of the output ZDD is bounded by the path-width of G

[Inoue and Minato, TCS-TR-A-16-80. Hokkaido University, '16]



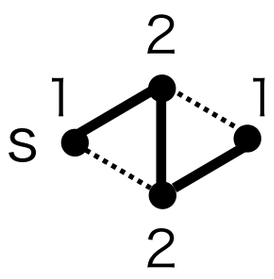
ZDD



s, t: vertex
A, B, ⋯, E: edge

Top-down construction algorithms for ZDDs

- General framework [Kawahara et al., IEICE Trans. '17] can deal with several fundamental constraints for subgraphs
- By combining the fundamental constraints, we can specify several types of subgraphs -> many applications

e.g., 

s-t path <=>

- connected (ignoring isolated vertices),
- s and t have degree 1, and
- the other vertices have degree 0 or 2

the number of edges
degrees of vertices
connectivity of vertices

Fundamental constraints for subgraphs

matching
path
cycle
spanning tree
spanning forest
…

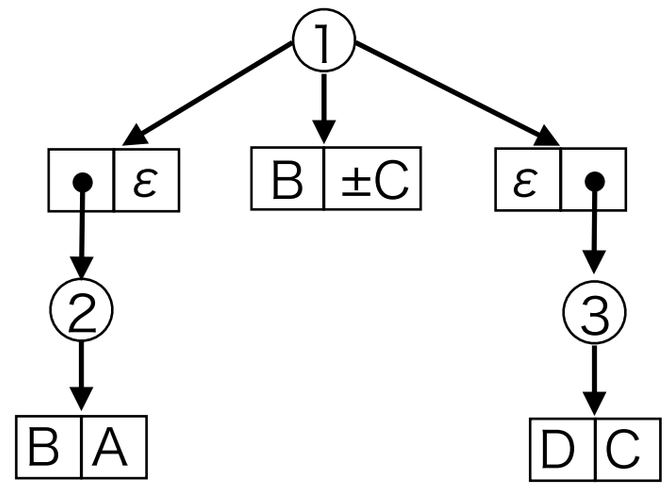
Subgraphs

electrical network [Inoue+14]
network reliability evaluation [Hardy+07]
text summarization [Nishino+15]
political redistricting [Kawahara+17]
evacuation planning [Nakahata+18]
…

Applications

ZSDD (Zero-suppressed Sentential Decision Diagram) [Nishino et al., AAAI '16]

- ZSDDs are compact representations of set families and generalizations of ZDDs
- Merits of ZSDDs
  - Theoretically, there exist set families that have poly-size ZSDD but exp-size ZDD [Bova et al., IJCAI '16]
  - Several poly-time queries like ZDDs
    - Counting, random sampling, Apply operations
- Are ZSDDs are useful for subgraph enumeration?



A ZSDD

(We explain how to read the figure later)

Subgraph enumeration with ZSDDs

- Existing method: Algorithms for matchings and paths
  [Nishino et al., AAAI '17]
  - ☺The sizes of output ZSDDs are bounded by the branch-width of the input graph, which are smaller than bounds of ZDDs by the path-width
  - ☺Experimentally faster than methods for ZDDs and the output ZSDDs are smaller than ZDDs
  - 😥It seems difficult to extend the algorithms to other types of subgraphs
  - 😥The algorithms are explained in a procedural way, which makes theoretical analysis difficult

the number of edges
degrees of vertices
connectivity of vertices

Fundamental constraints
for subgraphs used in ZDDs

Our contribution (1)

- We propose a novel framework of top-down construction algorithms for ZSDDs
- We apply our framework to the three fundamental constraints used in ZDDs: the number of edges, degrees, and connectivity
  - By combining these constraints, we can specify several types of subgraphs (e.g., paths, cycles, and spanning trees)
- To design an algorithm using our framework, one only has to show a recursive formula for the desired set of subgraphs
  -> makes theoretical analysis easier (e.g., correctness and complexity)

Our contribution (2)

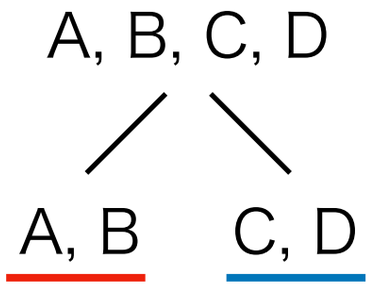
- We show that the sizes of output ZSDDs are bounded by the branch-width of the input graph (not only for matchings and paths)
- Experimental results show that the proposed method can construct ZSDDs faster than the existing method for ZDDs and that the output ZSDDs are smaller than ZDDs
- Our method extends types of subgraphs that ZSDDs can be constructed
  -> ZSDD can be applied for problems that ZDDs has been applied for

Outline

- Introduction
  - Subgraph enumeration with decision diagrams
  - Our target representation: ZSDDs
  - Proposed algorithms
  - Experiments and remarks

ZSDDs

- ZSDDs are obtained by recursively decomposing a set family into sub-families



$$\{\{A,B\},\{A,C\},\{B,C\},\{C,D\}\}$$

$$\left[\{\{\}\}\times\{\{C,D\}\}\right]\cup\left[\{\{A\}\}\times\{\{C\}\}\right]\cup\left[\{\{B\}\}\times\{\{C\}\}\right]\cup\left[\{\{A,B\}\}\times\{\{\}\}\right]$$

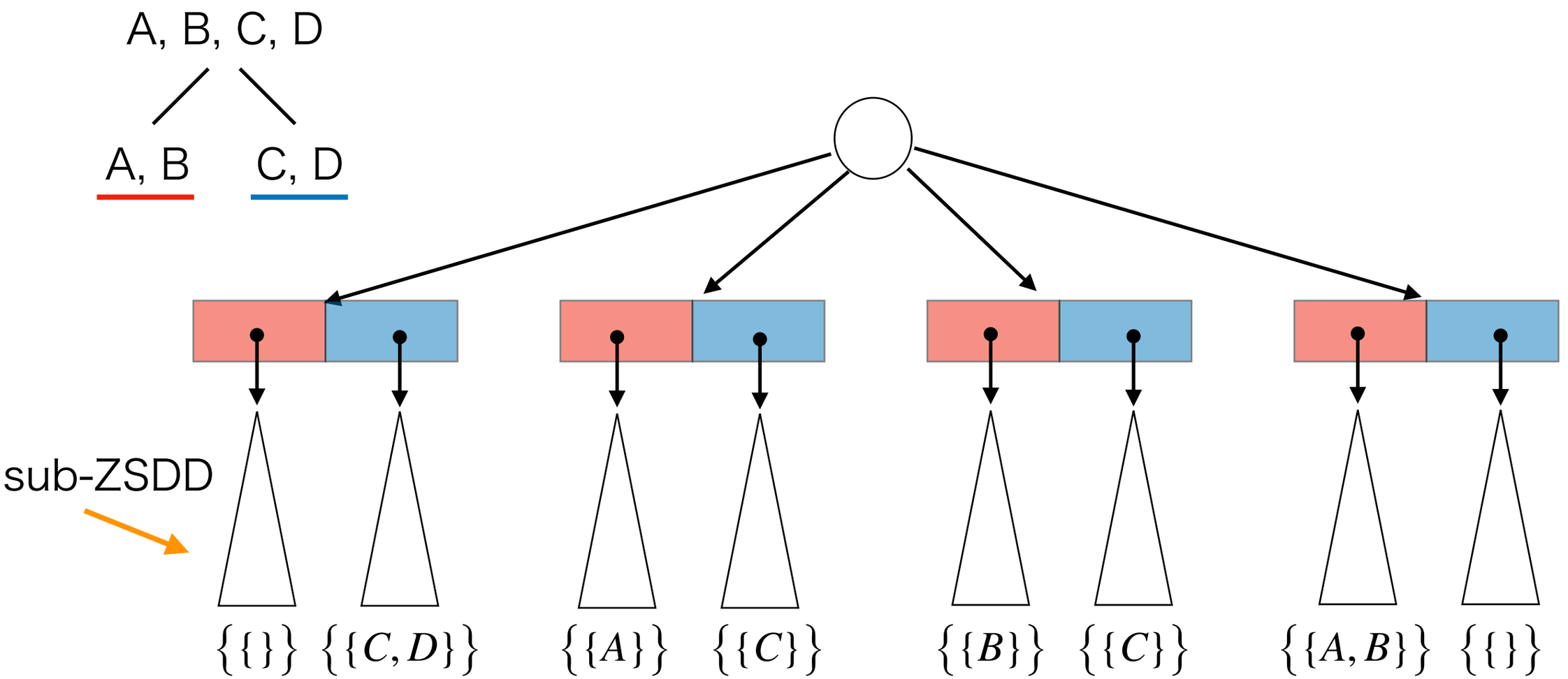
$\{\{C,D\}\}$ $\{\{A,C\}\}$ $\{\{B,C\}\}$ $\{\{A,B\}\}$

For set families f and g,

$f \times g = \{a \cup b \mid a \in f,\ b \in g\}$

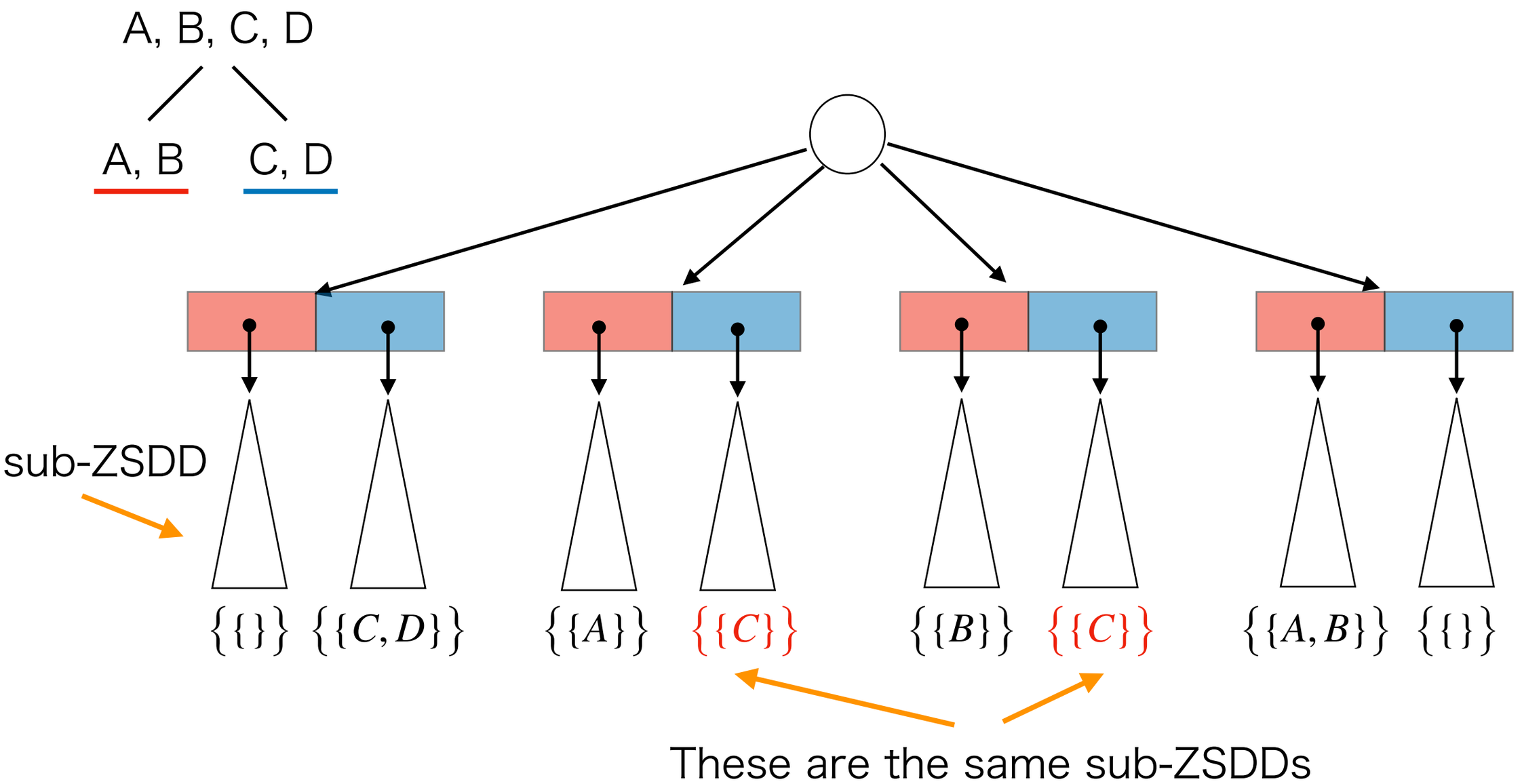
ZSDDs

- ZSDDs are obtained by recursively decomposing a set family into sub-families



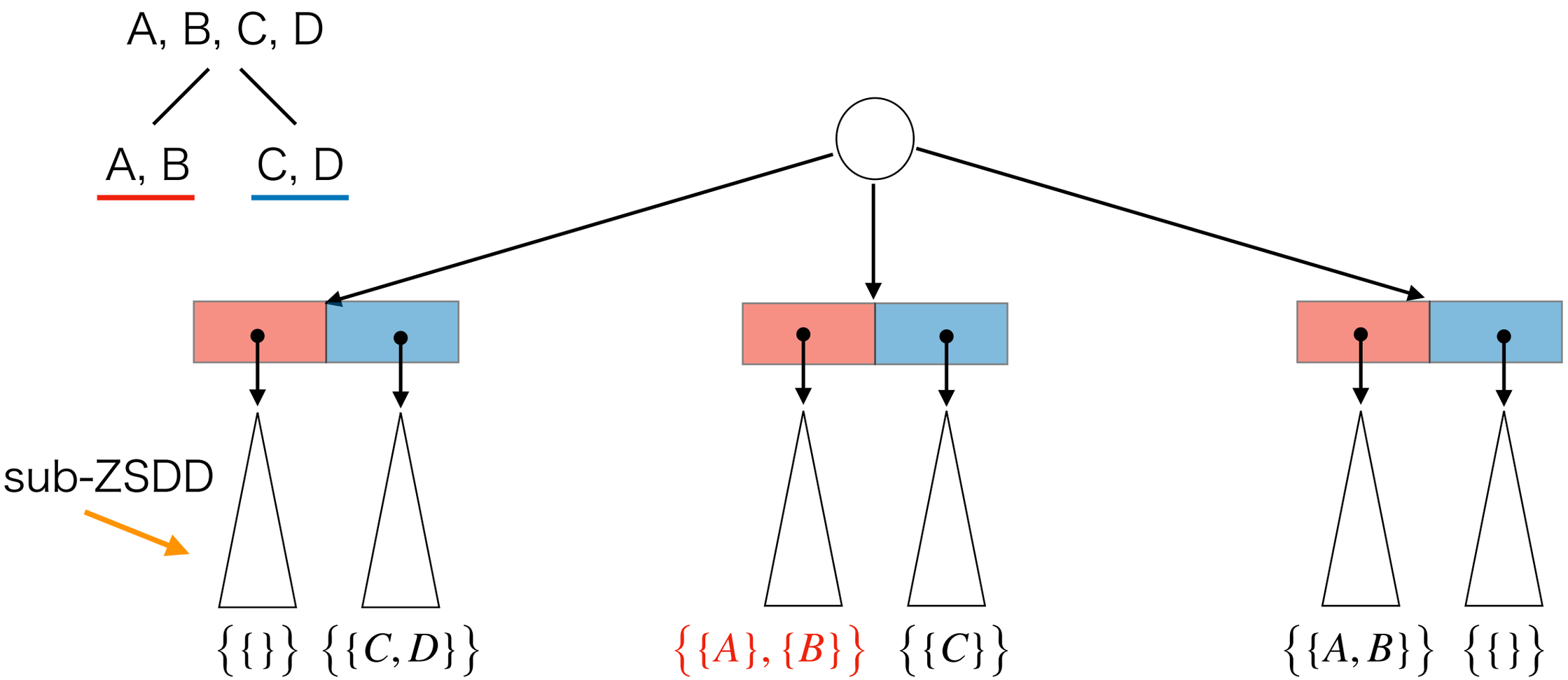
ZSDDs

- ZSDDs are obtained by recursively decomposing a set family into sub-families



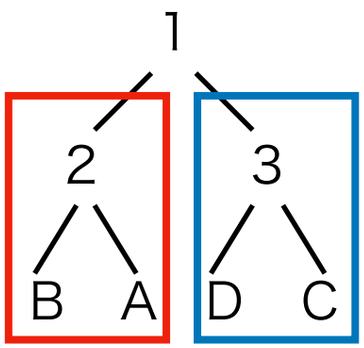
ZSDDs

- ZSDDs are obtained by recursively decomposing a set family into sub-families

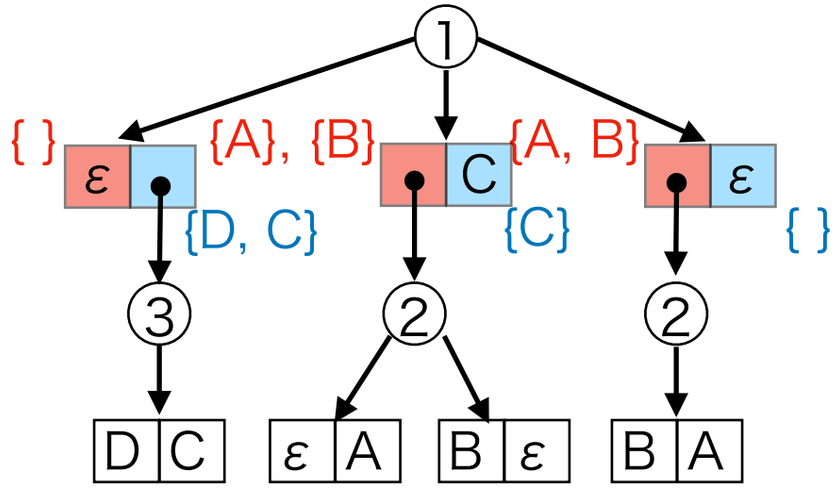


Vtree and ZSDD

- A ZSDD is obtained by recursively decomposing a set family into sub-families
- The order of decomposition is defined by a vtree



vtree

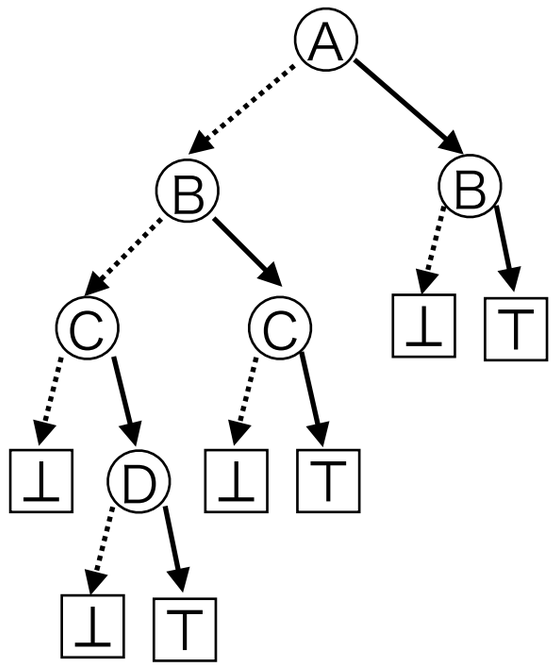


ZSDD representing

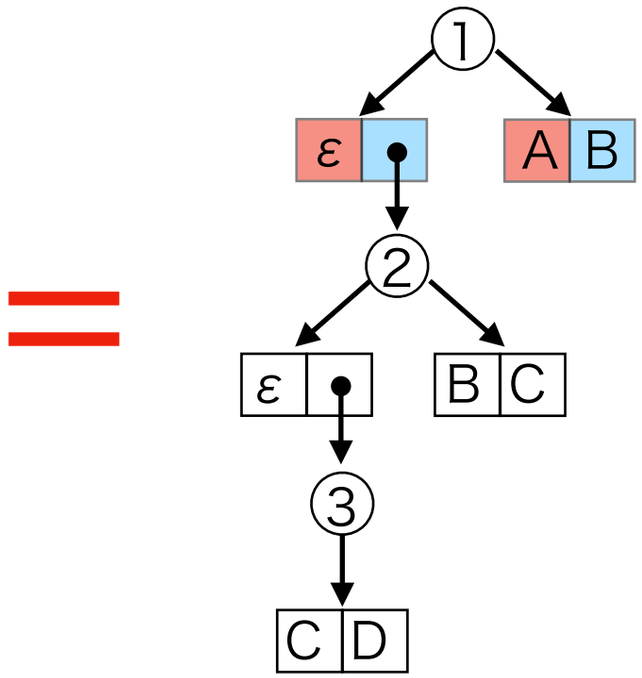
$$f = \{\{A, B\}, \{A, C\}, \{B, C\}, \{C, D\}\}$$

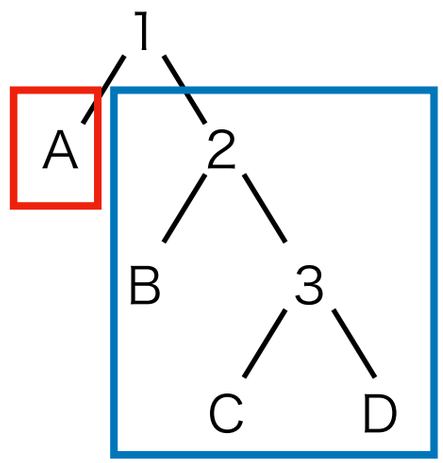
ZDDs are special cases of ZSDDs

- A ZSDD with a right-linear vtree topologically corresponds to a ZDD



=





ZDD

ZSDD

right-linear vtree

{ {A, B}, {B, C}, {C, D} }

Outline

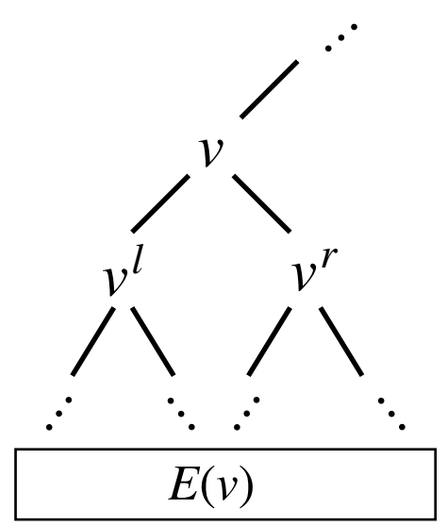
- Introduction
  - Subgraph enumeration with decision diagrams
- Our target representation: ZSDDs
- Proposed algorithms
- Experiments and remarks

Problem 1: Cardinality constraint

Input: vtree T, non-negative integer k
Output: a ZSDD representing the family of sets with **exactly k** elements

With a small modification,
we can deal with at most/least k
(details are omitted)

- We show a recursive formula for the desired set family
- Definitions:
  - v: vnode (a node of a vtree)
  - v^l, v^r: left/right children of v
  - E(v): the set of elements correspond to the leaf vnodes of the sub-vtree rooted at v
  - For vnode v and non-negative integer i, we define $f(v, i) := \{S \subseteq E(v) \mid |S| = i\}$
    - The desired set family is $f(v^{\mathrm{root}}, k)$ v^{root}: the root vnode of T



vtree

Recursive formula for the cardinality constraint

- For a vnode v and a non-negative integer k,
- If v is a leaf vnode:

$$f(v,k) = \begin{cases} \{\varnothing\} & (k=0) \\ \{\{\ell(v)\}\} & (k=1) \\ \{\} & (k \geq 2) \end{cases}$$

$\ell(v)$: an element corresponding to a leaf vnode v

Definition

$f(v,k) := \{S \subseteq E(v) \mid |S| = k\}$

Recursive formula for the cardinality constraint

- For a vnode v and a non-negative integer k,
- If v is a leaf vnode:

$$f(v,k) = \begin{cases} \{\varnothing\} & (k=0) \\ \{\{\ell(v)\}\} & (k=1) \\ \{\} & (k \geq 2) \end{cases}$$

$\ell(v)$: an element corresponding to a leaf vnode v

Definition

$f(v,k) := \{S \subseteq E(v) \mid |S| = k\}$

- If v is internal:

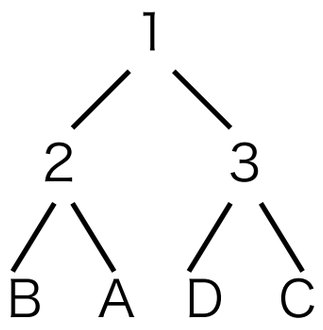
$$f(v,k) = \bigcup_{i=0}^{k} \left(f(v^l, i) \times f(v^r, k-i) \right)$$

If we take i elements from $E(v^l)$,

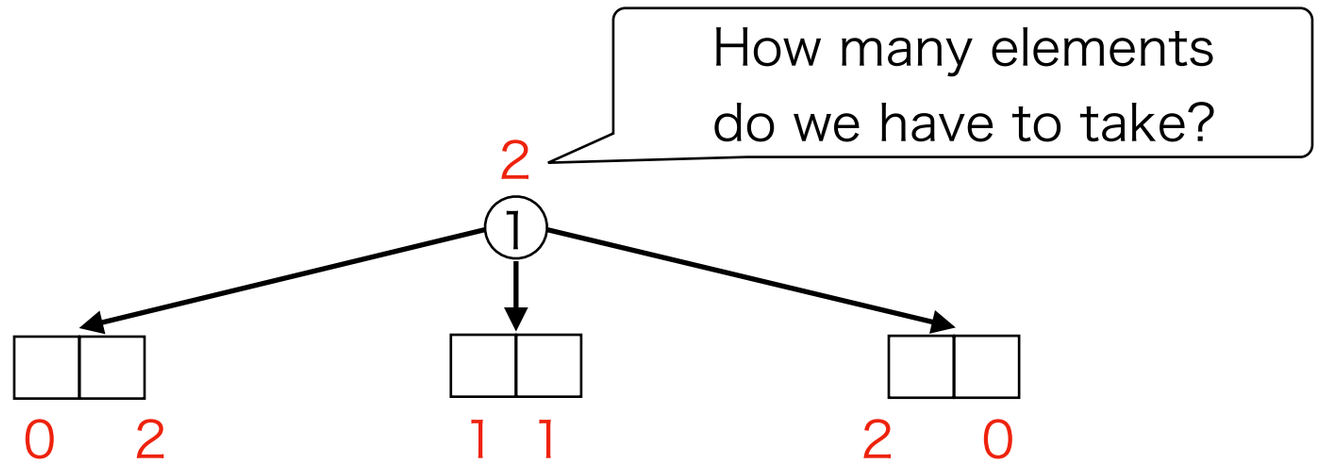
we have to take k - i elements from $E(v^r)$

Example of ZSDD construction

Let k = 2.

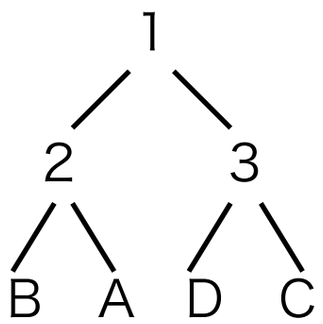


vtree

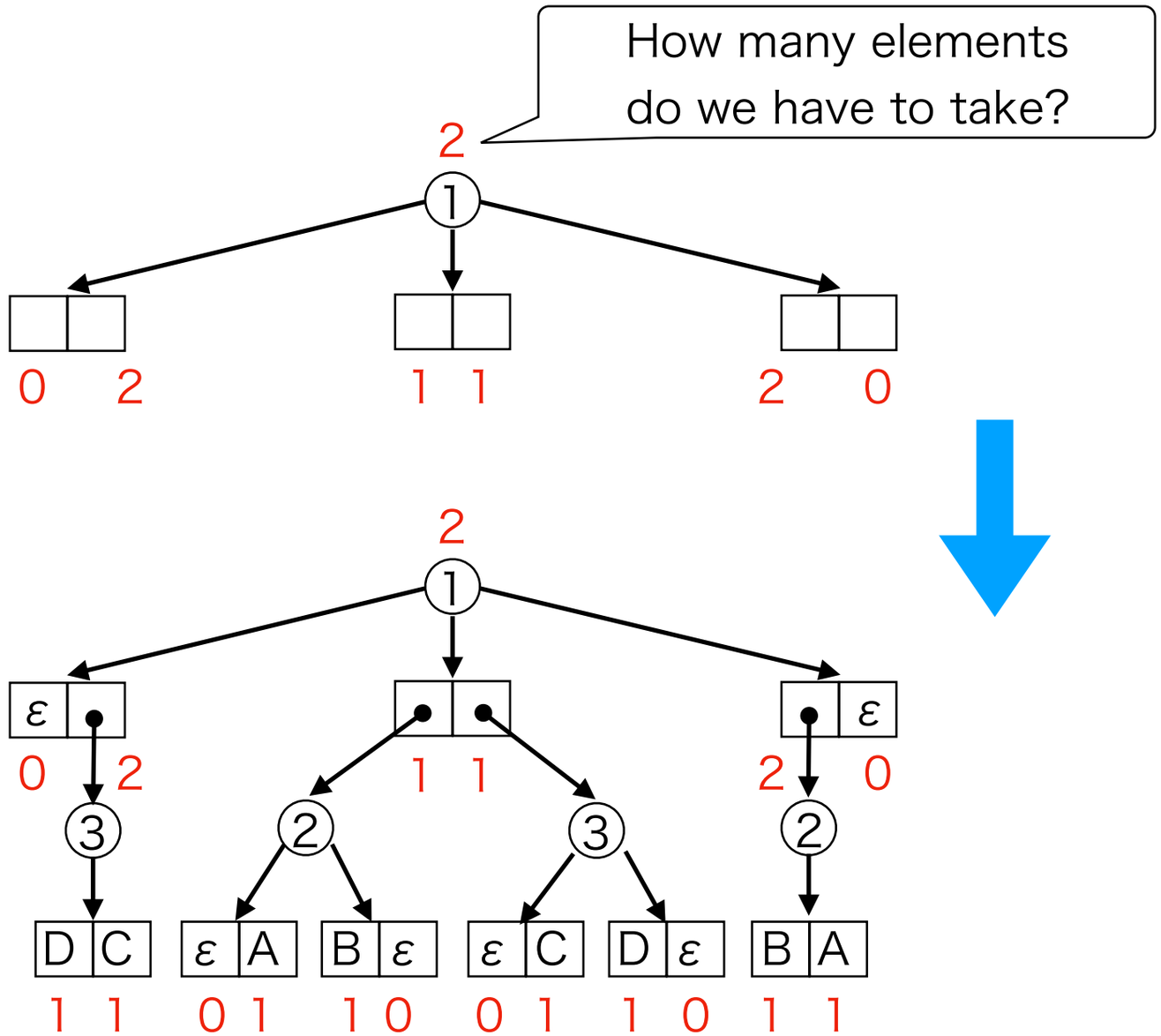


Example of ZSDD construction

Let k = 2.

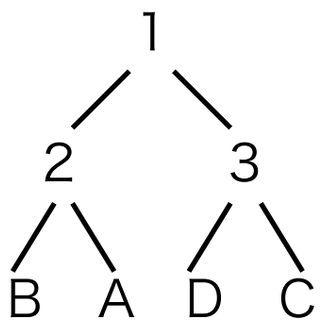


vtree

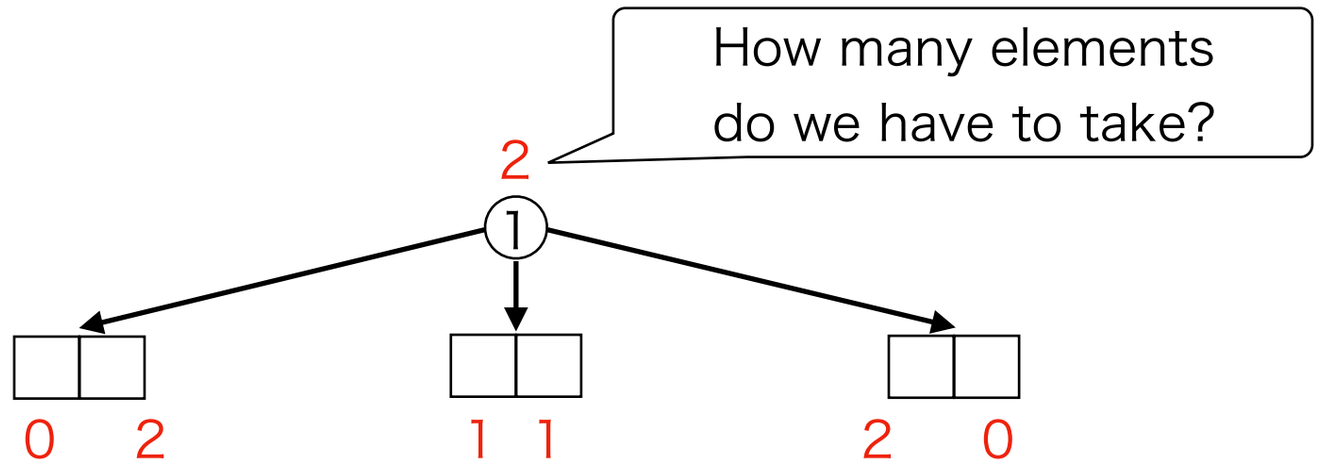


Example of ZSDD construction

Let k = 2.



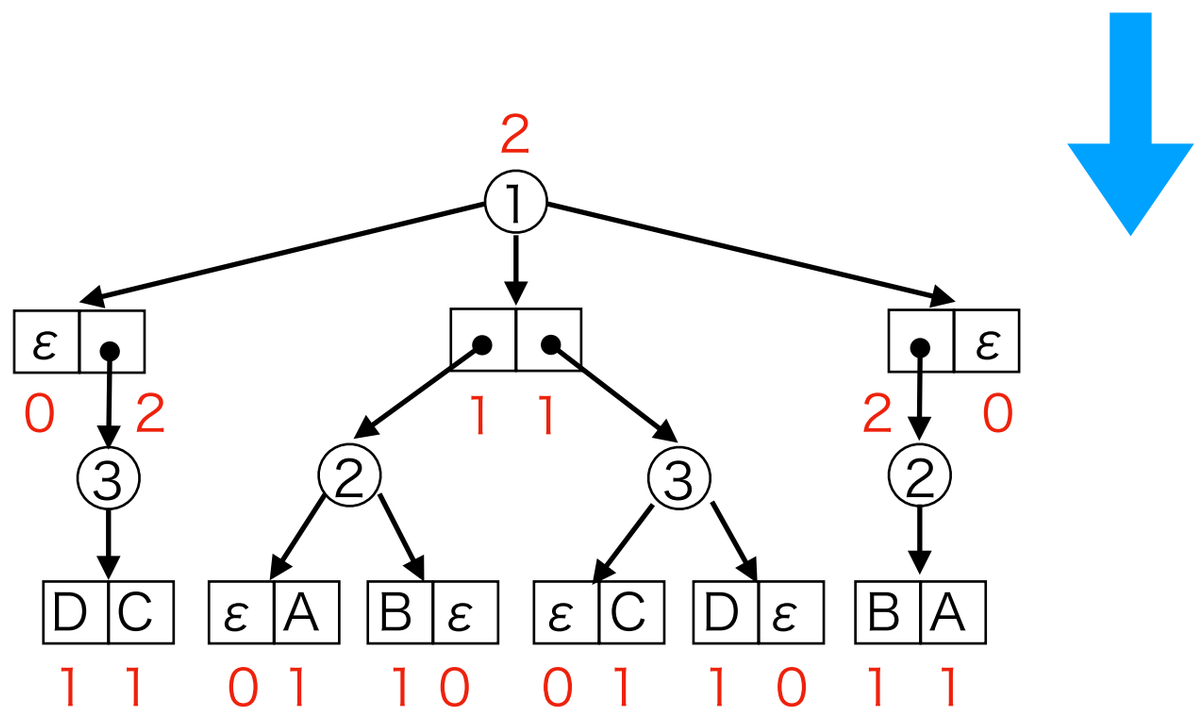
vtree



By the recursive formula,
we can show the correctness of
the algorithm and analyze the size
of the output ZSDD

Theorem 1

The size of the output ZSDD is $O(|E|k^2)$

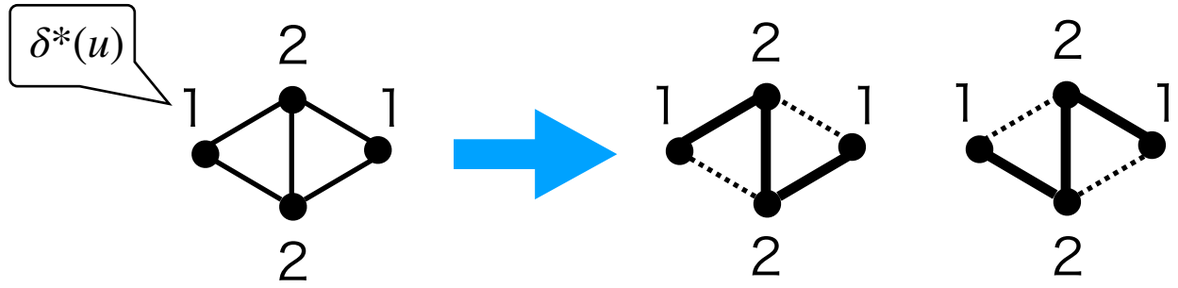


Problem 2: Degree constraint

Input: graph G, vtree T, function $\delta^*: V(G) \to \mathbb{N}$
Output: a ZSDD representing the set of subgraphs s.t., for all $u \in V(G)$, the degree of u equals $\delta^*(u)$

$\mathbb{N}$: the set of non-negative integers

With a small modification, we can deal with at most/least k degree (details are omitted)



A graph G

Subgraphs satisfying the degree constraint

- $V(v)$: The set of endpoints of some edge in $E(v)$
- $\deg(S, u)$: The degree of vertex u in graph $(V(v), S)$

- Idea: The degree constraint = The cardinality constraint for each vertex
- For vnode v and function $\delta: V(v) \to \mathbb{N}$, we define

$$f(v, \delta) := \{S \subseteq E(v) \mid \forall u \in V(v), \deg(S, u) = \delta(u)\}$$

The set of subgraphs of (V(v), E(v)) satisfying the degree constraint in V(v)

The desired set family is $f(v^{\mathrm{root}}, \delta^*)$

Recursive formula for the degree constraint

- For vnode v and function δ: $V(v) \to \mathbb{N}$,
- If v is a leaf vnode, let u_1 and u_2 be the endpoints of the edge $\ell(v)$. Then,

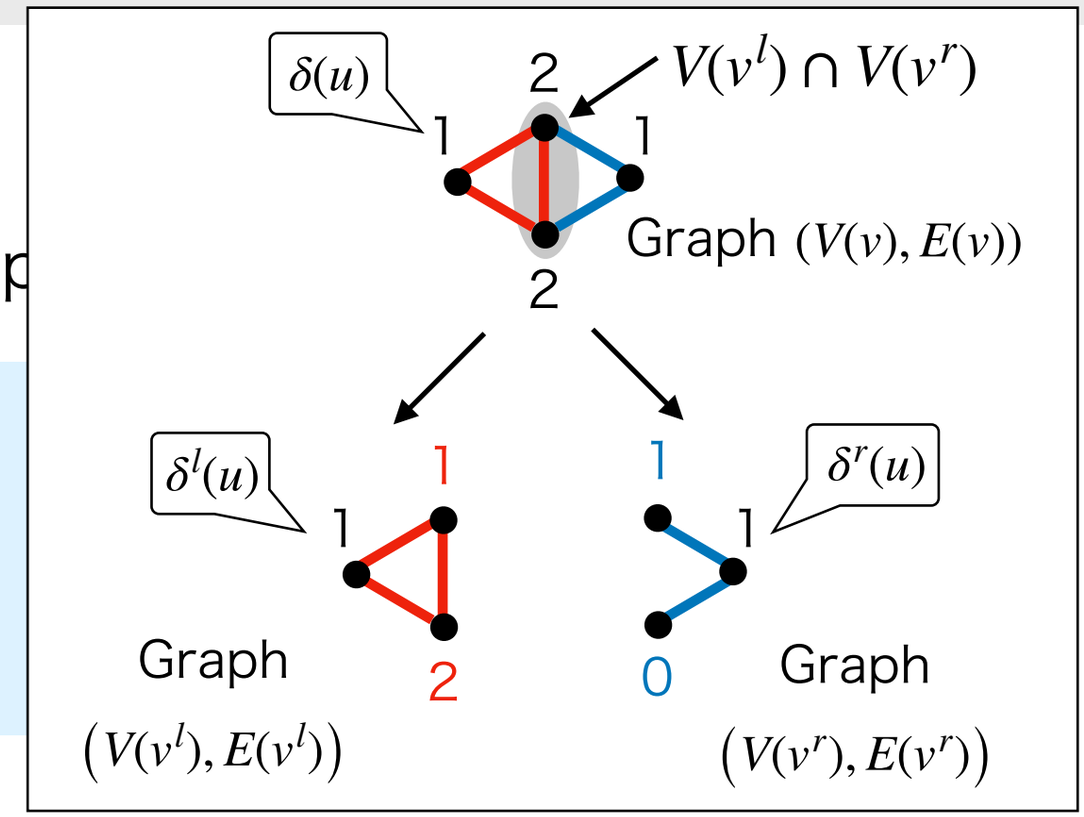
$$f(v,\delta) = \begin{cases} \{\varnothing\} & (\delta(u_1) = \delta(u_2) = 0\,) \\ \{\{\ell(v)\}\} & (\delta(u_1) = \delta(u_2) = 1\,) \\ \{\} & (\,\text{otherwise}\,) \end{cases}$$

$\ell(v)$: an element corresponding to a leaf vnode v

Recursive formula for the degree constraint

- For vnode v and function $\delta\colon V(v) \to \mathbb{N}$,
- If v is a leaf vnode, let u_1 and u_2 be the endp

$$f(v,\delta) = \begin{cases} \{\varnothing\} & (\delta(u_1) = \delta(u_2) = 0) \\ \{\{\ell(v)\}\} & (\delta(u_1) = \delta(u_2) = 1) \\ \{\} & (\text{otherwise}) \end{cases}$$



- If v is internal:

$$f(v,\delta) = \bigcup_{(\delta^l,\delta^r) \in \mathscr{P}(v,\delta)} \left(f(v^l,\delta^l) \times f(v^r,\delta^r) \right)$$

For each $u \in V(v)$, if we take i edges from $E(v^l)$,
we have to take k - i edges from $E(v^r)$

$\mathscr{P}(v,\delta)$ is the set of pairs of functions $\delta^l\colon V(v^l) \to \mathbb{N}$ and $\delta^r\colon V(v^r) \to \mathbb{N}$ s.t.

$$\begin{cases} u \in V(v^l) \cap V(v^r) \Rightarrow \delta^l(u) + \delta^r(u) = \delta(u) \\ u \in V(v^l) \setminus V(v^r) \Rightarrow \delta^l(u) = \delta(u) \\ u \in V(v^r) \setminus V(v^l) \Rightarrow \delta^r(u) = \delta(u) \end{cases}$$

The size of the output ZSDD for the degree constraint

- Bottleneck: When v is an internal vnode, for each $u \in V(v^l) \cap V(v^r)$, we decide the degree of u in $E(v^l)$
- $\left|V(v^l) \cap V(v^r)\right|$ is $O(|V|)$, but can be smaller depending on the vtree
- Let $w(T) := \max_{v \in \mathrm{in}(T)} \left|V(v^l) \cap V(v^r)\right|$ and $w(G) := \min_T w(T)$.

Then, the ZSDD size is $\mathrm{O}\left(|E| d^{2w(G)}\right)$

- $w(G)$ equals the branch-width $\mathrm{bw}(G)$

[Nishino et al., AAAI '17]

in(T): The set of internal vnodes
d: 1 + (the maximam value appearing in the degree constraint)

Theorem 2

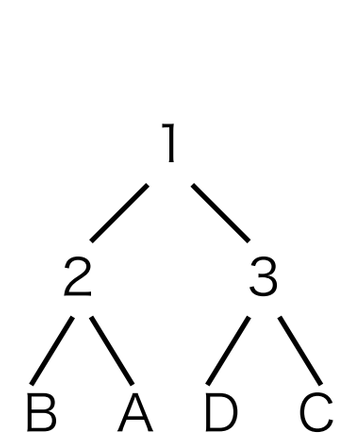
The size of the output ZSDD is $\mathrm{O}\left(|E| d^{2\mathrm{bw}(G)}\right)$

Outline

- Introduction
  - Subgraph enumeration with decision diagrams
- Our target representation: ZSDDs
- Proposed algorithms
- Experiments and remarks

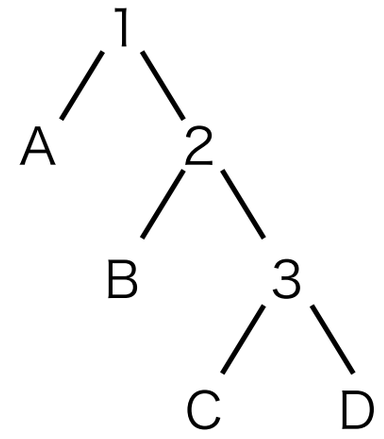
Experiments

- We compared our top-down algorithms for ZSDDs with the existing top-down algorithms for ZDDs in the same way as the existing paper [Nishino et al., AAAI '17]
- Benchmark graphs: TSPLIB and RomeGraph
- Types of subgraphs: Maximum degree ≤ 2 and spanning trees
- Three types of vtrees:
  - TD: Heuristics of branch decomposition
    [Cook and Seymour, INFORMS J. Comput., '03]
  - Z(b): Bredth-first ordering (used in Graphillion
    [Inoue et al., Int. J. Softw. Tools Technol. Transf., '16])
  - Z(v): Right-linear vtree obtained from TD
    [Xue et al., AAAI '12]



general vtree

(for ZSDDs)



right-linear vtree

(for ZDDs)

- All codes are written in C++ and compiled by g++-5.4.0 with -O3 option
- Machine: Intel Xeon W-2133 3.60 GHz CPU, 256 GB RAM

Result: Max. deg. ≤ 2

			Time (ms)			Size		
instance	$\|V\|$	$\|E\|$	TD	Z(b)	Z(v)	TD	Z(b)	Z(v)
att48	48	130	381	6801	2291	194786	1065745	507169
berlin52	52	145	1021	-	36354	807660	-	5229861
eil51	51	142	1012	247736	46524	774280	27277682	5974875
grafo10106	100	119	5	2617	16	2658	15461	7529
grafo10124	100	139	9237	-	40842	3060950	-	3283397
grafo10153	100	136	3784	-	4658	832943	-	561283
grafo10183	100	132	132	-	157837	80127	-	4088915
grafo10184	100	140	4981	-	119366	1006210	-	2002968
grafo10204	100	148	156529	-	303366	15712819	-	19847326
grafo10223	100	135	863	-	5956	330554	-	826121
			ZSDD	ZDD		ZSDD	ZDD	

- For comparison, we omit instances for which all the methods finished within a second or at most one method finished within 10 minutes
- For all graphs, TD was faster and memory-saving than Z(b) and Z(v)
- Time: TD was up to 245 (resp., 1195) times faster than Z(b) (resp., Z(v))
- Size: TD was up to 35 (resp., 51) times smaller than Z(b) (resp., Z(v))
- These results show the efficiency of our method

Result: Spanning trees

			Time (ms)			Size		
instance	$\|V\|$	$\|E\|$	TD	Z(b)	Z(v)	TD	Z(b)	Z(v)
att48	48	130	3494	103871	3005	279613	5098205	387715
berlin52	52	145	11826	-	62706	937746	-	3194017
eil51	51	142	25828	-	94272	838254	-	7178190
ulysses22	22	56	39	3391	65	3036	520035	16762
grafo10106	100	119	28	221161	53	1756	836212	4057
grafo10183	100	132	2866	-	538878	224373	-	16414697
grafo10223	100	135	48563	-	128097	1009299	-	7313087
grafo10248	100	126	301	195249	672	16524	1617024	47605
			ZSDD	ZDD		ZSDD	ZDD	

- For most graphs, TD was faster and memory-saving than Z(b) and Z(v)
- Time: TD was up to 7898 (resp., 188) times faster than Z(b) (resp., Z(v))
- Size: TD was up to 476 (resp., 73) times smaller than Z(b) (resp., Z(v))
- These results show the efficiency of our method
- Exception: For att48, Z(v) was faster than TD (due to the overhead of TD)

Concluding remarks

- We have proposed a novel framework of algorithms for top-down ZSDD construction
- We have applied our framework for three fundamental constraints: cardinality, degree, and connectivity
- We have shown that the sizes of output ZSDDs are bounded by the branch-width of the input graph
- Experiments confirmed the efficiency of our method
- We believe that our framework is useful for various problems
  - Using Apply operations, we can extract degree-constrained or connected subgraphs from ZSDDs storing set of subgraphs

Appendix

- Detailed description of ZSDDs
- Cardinality constraint: at most k
- Experimental results: max. deg. ≤ 3

(X, Y)-partitions

- ZSDDs are obtained by recursively applying (X, Y)-partitions to a set family
- Definition
  Let f be a set family and X, Y be a partition of the universe of f. Set family f can be written as

$$f = \bigsqcup_{i=1}^{h} [p_i \sqcup s_i], \quad (1)$$

"join"
$f \sqcup g = \{a \cup b \mid a \in f, b \in g\}$

where p_i and s_i are the set families whose universes are X and Y, respectively. We call $p_1, \ldots, p_h$ primes and $s_1, \ldots, s_h$ subs. If the primes are exclusive ($p_i \cap p_j = \varnothing$ for all $i \neq j$), Equation (1) is an (X, Y)-partition.

Example of an (X, Y)-partition

- Let $f = \{\{A, B\}, \{A, C\}, \{B, C\}, \{C, D\}\}$, $\mathbf{X} = \{A, B\}$, and $\mathbf{Y} = \{C, D\}$.

  An $(\mathbf{X}, \mathbf{Y})$-partition of f is

$$f = \left[\underline{\{\{\}\}} \sqcup \underline{\{\{C, D\}\}}\right] \cup \left[\underline{\{\{A\}, \{B\}\}} \sqcup \underline{\{\{C\}\}}\right] \cup \left[\underline{\{\{A, B\}\}} \sqcup \underline{\{\{\}\}}\right]$$

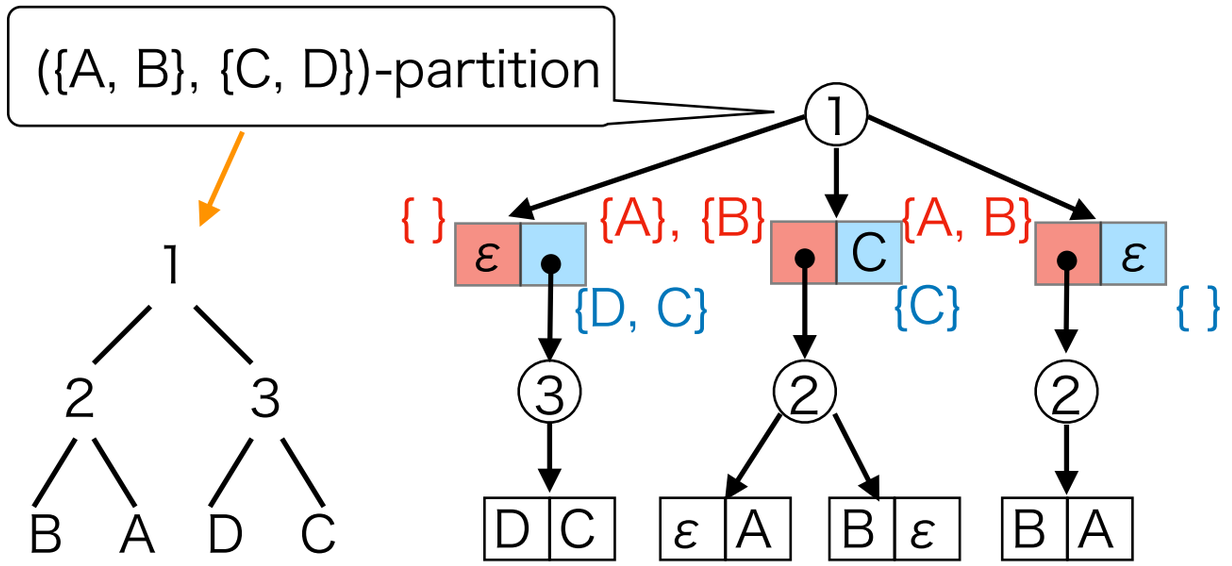
prime sub

$$\{\{C, D\}\} \qquad \{\{A, C\}, \{B, C\}\} \qquad \{\{A, B\}\}$$

- primes are exclusive

Vtree and ZSDD

- The order of (X, Y)-partitions is determined by a vtree
- A vtree is a rooted, ordered, and full binary tree whose leaves correspond to the elements of the universe
- The root ZSDD node (znode) respects the root vtree node
- From the root znode, a ZSDD is obtained by recursively applying (X, Y)-partitions to a set family



vtree ZSDD representing

$$f = \{\{A, B\}, \{A, C\}, \{B, C\}, \{C, D\}\}$$

Cardinality constraint: at most k

Input: vtree T, non-negative integer k
Output: a ZSDD representing the family of sets
with at most **k** elements

Similar results hold for
“at least k”

- If we define $g(v,k) := \{S \subseteq E(v) \mid |S| \leq k\}$,

  we can show a similar equation
  as “exactly k” constraint

$$g(v,k) = \bigcup_{i=0}^{k} \left(g(v^l, i) \sqcup g(v^r, k-i) \right)$$

- However, this equation is not
  an $(E(v^l), E(v^r))$-partition
  because the primes are not exclusive
  - For $i \leq j$, we have $g(v^l, i) \subseteq g(v^l, j)$

Cardinality constraint: at most k

Input: vtree T, non-negative integer k
Output: a ZSDD representing the family of sets
with at most **k** elements

Similar results hold for
"at least k"

Definition
$f(v,k) := \{S \subseteq E(v) \mid |S| = k\}$

- If we define $g(v,k) := \{S \subseteq E(v) \mid |S| \leq k\}$,
  we can show a similar equation
  as "exactly k" constraint

Use f instead of g
for the primes

$$g(v,k) = \bigcup_{i=0}^{k} \left(\cancel{g(v^l, i)} \sqcup g(v^r, k-i) \right) \quad (f(v^l, i))$$

- However, this equation is not
  an $(E(v^l), E(v^r))$-partition
  because the primes are not exclusive
  - For $i \leq j$, we have $g(v^l, i) \subseteq g(v^l, j)$

- By Combining the recursive formulas for f and g, we can construct a desired ZSDD
- The size of the output ZSDD is $O(|E|k^2)$ like "exactly k" constraint

Result: max. deg. ≤ 3

			Compilation time (ms)			Size		
instance	$\|V\|$	$\|E\|$	TD	Z(b)	Z(v)	TD	Z(b)	Z(v)
att48	48	130	2392	15791	22576	564163	1408493	994667
berlin52	52	145	7478	-	535530	2727435	-	11561690
eil51	51	142	17003	-	445662	3283534	-	14446615
grafo10106	100	119	14	3628	37	1565	1162	2504
grafo10124	100	139	186539	-	139582	1625041	-	589765
grafo10153	100	136	135821	-	10989	668892	-	51571
grafo10184	100	140	139648	-	398498	351873	-	212686
grafo10223	100	135	14332	-	18953	427096	-	115327

- For most graphs, TD was faster and memory-saving than Z(b) and Z(v)
  - For grafo(10124|10153), Z(v) was better than TD
- Time: TD was up to 259 (resp., 71) times faster than Z(b) (resp., Z(v))
- Size: TD was up to 5.5 (resp., 4.4) times smaller than Z(b) (resp., Z(v))