Variable Shift SDD: A More Succinct Sentential Decision Diagram

Kengo Nakamura (NTT)
Shuhei Denzumi (The University of Tokyo)
Masaaki Nishino (NTT)

18th Symposium on Experimental Algorithms (**SEA2020**)
2020/6/18 @ online (Catania, Italy)

Contents

Boolean function

$$f: \{true, false\}^n \rightarrow \{true, false\}$$

Input: n Boolean values, **Output**: a Boolean value

There are **useful queries** for Boolean function:

- **Model count**: How many patterns of input make output true?
- **Equivalency**: Are given two Boolean functions equivalent?

etc.

However, these queries are **difficult to solve**

Ex.) Using truth table, they take $O(2^n)$ time

$$(A \vee \bar{B}) \wedge (B \vee \bar{C})$$

A	B	C	f
false	false	false	true
false	false	true	false
false	true	false	false
false	true	true	false
true	false	false	true
true	false	true	false
true	true	false	true
true	true	true	true

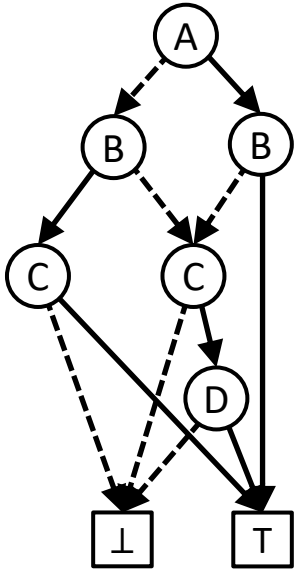
Truth table

Ordered Binary Decision Diagram (OBDD) [Bryant '86]

Directed Acyclic Graph (DAG) representation of Boolean function

- Size of OBDD is generally exponential, but is **significantly smaller than 2^n** for many practical cases
- OBDD **supports many queries** in **polytime w.r.t. OBDD size**
  - Model count, Equivalency, etc.

$$(A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$$



OBDD

OBDD and Apply operation [Bryant '86]

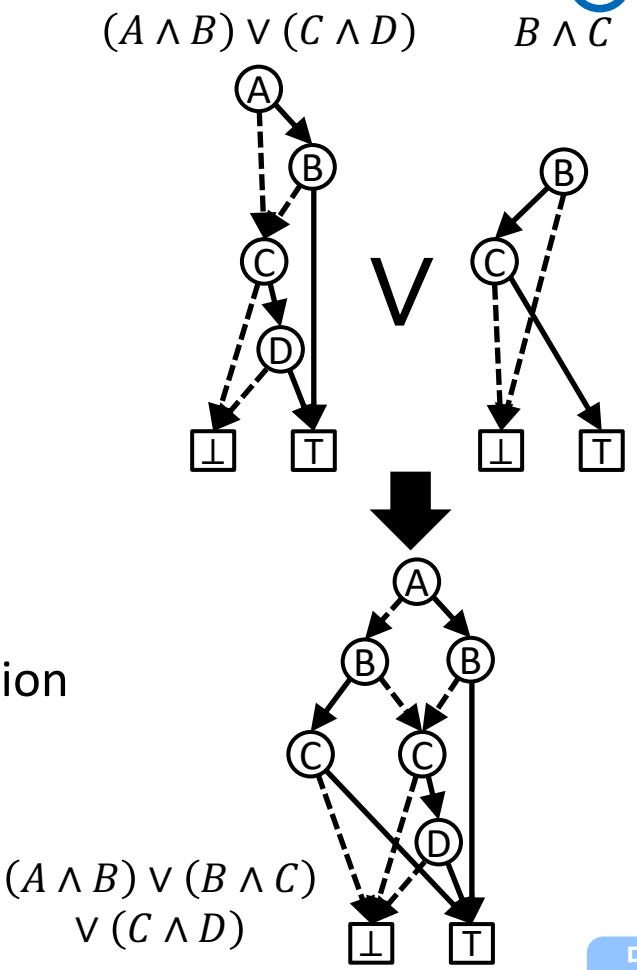
Size of α

OBDDs support **Apply operation** in $\boldsymbol{O(|\alpha||\beta|)}$ **time**

- **Input**: Two OBDDs α, β and binary operator $\circ$
  - conjunction($\wedge$), disjunction($\vee$), etc.
- **Output**: OBDD representing $f \circ g$
  - α (β) represents f (g resp.)

Fundamental in

- **constructing OBDD** representing any Boolean function
- **supporting useful queries**



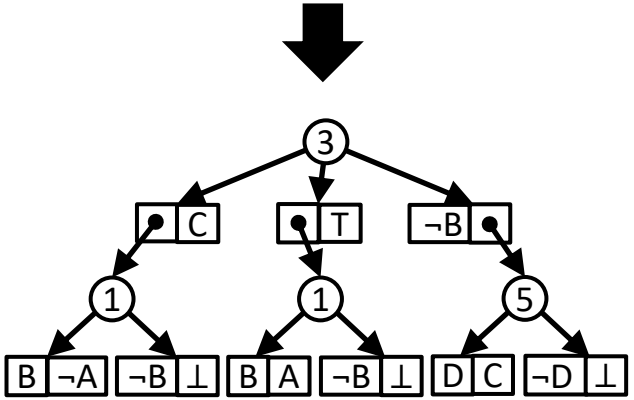
Sentential Decision Diagram (SDD) [Darwiche '11]

- **DAG representation** of Boolean function
- **Generalization of OBDD structure**

- **Tighter bound on size** than OBDD
  - In some cases, SDD can make size **exponentially smaller** than OBDD [Bova '16]
- Supporting **useful queries** in **polytime w.r.t. SDD size**
- Supporting **Apply operation** in $\boldsymbol{O(|\alpha||\beta|)}$ **time**

Input: Two **SDDs** α, β and oper. $\circ$
Output: **SDD** representing $f \circ g$

$$(A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$$



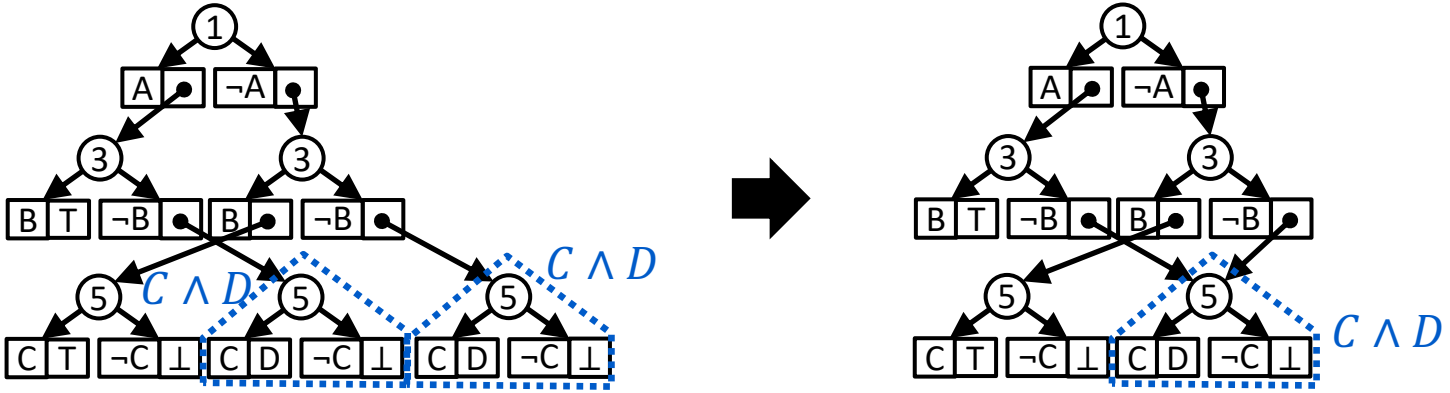
SDD

Substructure sharing

One of the reason why OBDD/SDD is succinct:

- Boolean function is **recursively decomposed** into subfunctions that are also represented by DAGs (substructures)
- When **same subfunctions** emerge, **only one DAG is needed** instead of having multiple (identical) DAGs

… **Share of substructure <u>representing the same function</u>**



Proposal: Variable Shift SDD (VS-SDD)

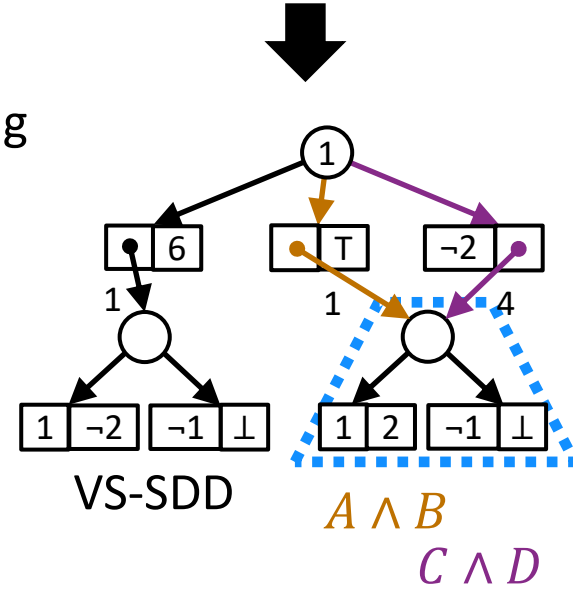
Can we share more substructures to achieve more succinct representation? -> **Variable Shift SDD**

… SDD-based representation of Boolean function

- VS-SDD can share substructures of subfunctions that are **equivalent under specific type of variable substitution**
  - Ex.) $C \wedge D$ can be obtained from $A \wedge B$ by substituting A with C and B with D
- Representing Boolean function of substructure is **dependent on the path from root to it**

Typical example of emerging such sets of subfunctions: modeling time-evolving systems;

$$(A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$$



Proposal: Variable Shift SDD (VS-SDD)

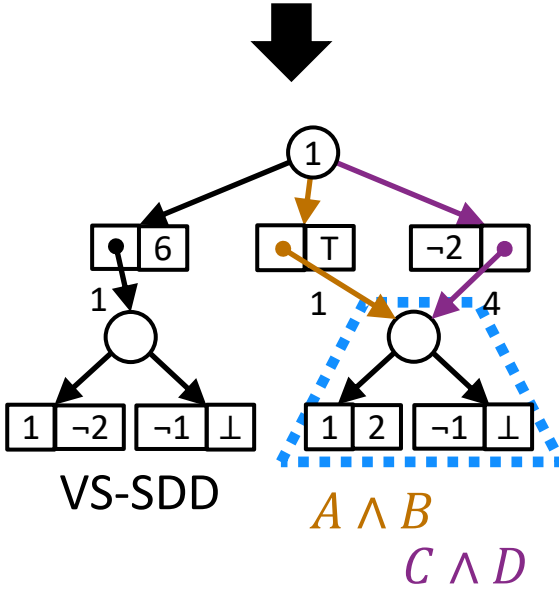
Features:

- VS-SDD is **never larger than SDD** of the same function
  - There exists a class of function for which VS-SDD is **exponentially smaller than SDD**
- VS-SDD supports **many useful queries as SDD does**
  - Supports **Apply operation** in $\boldsymbol{O(|\alpha||\beta|)}$ **time**

Input: Two **VS-SDDs** α, β and oper. $\circ$
Output: **VS-SDD** representing $f \circ g$

VS-SDDs incur **no additional overhead over SDDs** while **being potentially smaller than SDDs**

$(A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$



Contents

Related works

Attempts to share substructures representing **“equivalent”**
Boolean functions up to conversion

For OBDDs:

- **Attributed edge** [Madre & Billon ’88]
- **Complement edge** [Minato et al. ’90] … up to taking negation $f(\boldsymbol{X}) \leftrightarrow \neg f(\boldsymbol{X})$
  - Solvability of useful operations is not considered
- **↑ΔBDD** [Anuchitanukul et al. ’95] … up to “shifting” of variables $f(X_1, X_2, X_3) \leftrightarrow f(X_4, X_5, X_6)$
  - Operation like Apply is supported, but its time complexity is not polynomial of DAG sizes

Related works

$f(X_1, X_2, X_3)$
$\leftrightarrow f(X_5, X_4, X_6)$

For other DAG structures:

- **Sym-DDG/Sym-FBDD** [Bart et al. '14] … up to **any** variable substitution
  - Based on DDG [Fragier & Marquis '06] and FBDD [Gergov & Meinel '94]
  - Fail to support some important operations such as Apply and Conditioning, as the based structures do

For SDDs:

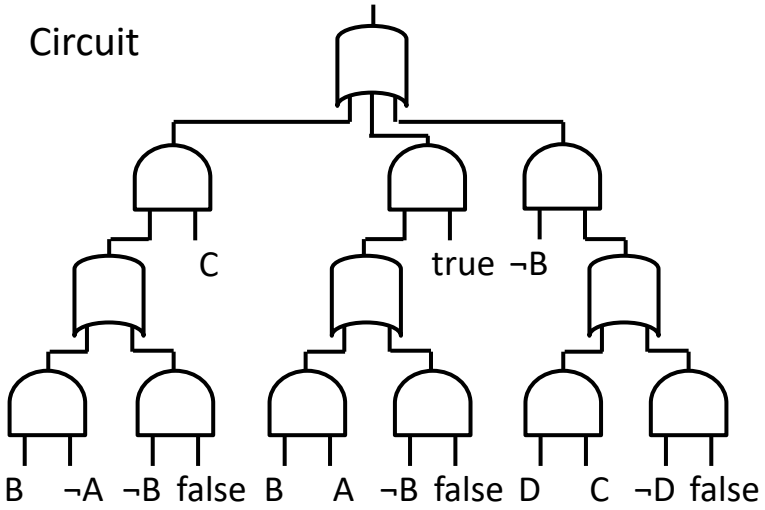
- **No such previous study is known**
- VS-SDD **cannot** be obtained by straightforward application of techniques for OBDDs (like ↑ΔBDD)

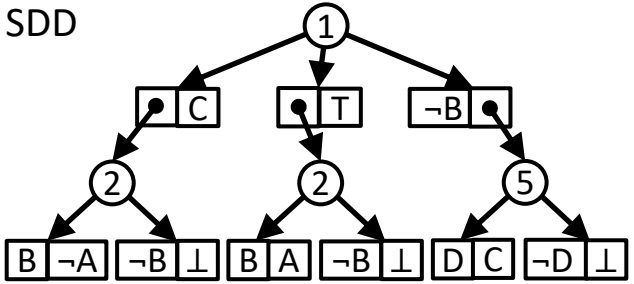
Contents

SDD: Boolean circuit view

SDD = **Boolean circuit** with **structured decomposability** and **strong determinism**

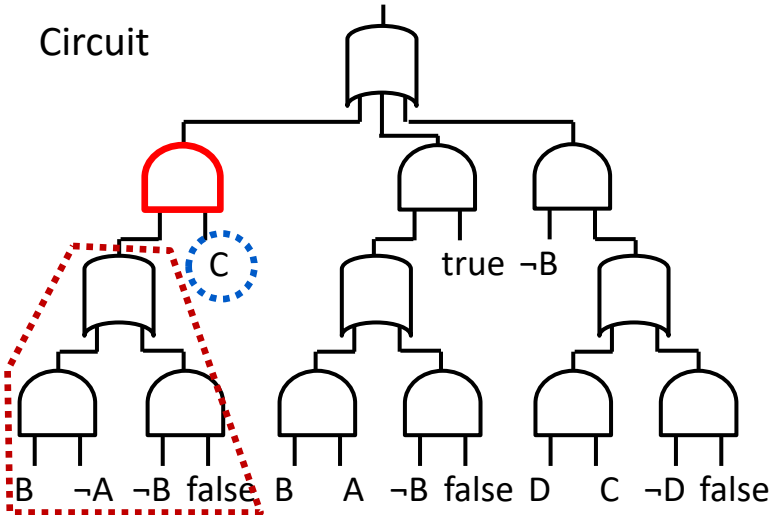
- Represents Boolean function by recursive application of **OR** and **AND**
  - **OR** gate corresponds to circle node
  - **AND** gate corresponds to box pair

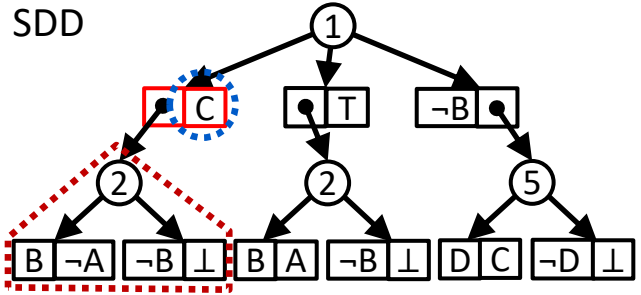




Decomposability

- **Decomposable**: for each **AND** gate, used variables are non-overlapping for any two inputs of it

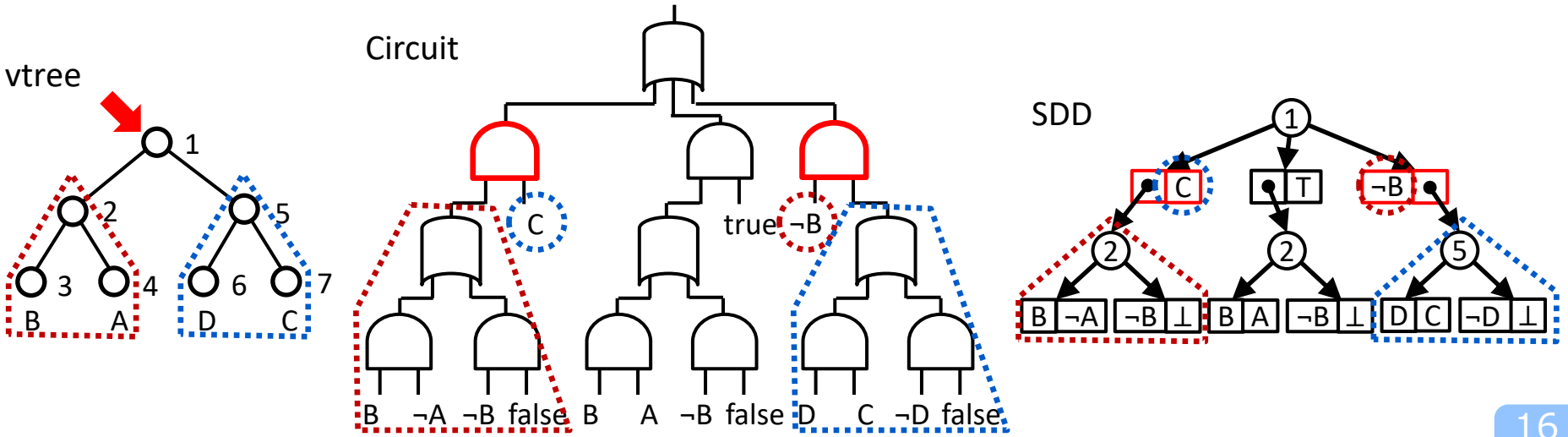




Vtree and structured decomposability

Decomposable circuit whose **decomposition of variables is structured** with vtree

- **Vtree**: full binary tree whose leaves are labeled with variables
- **Structured decomposable** (given vtree v): for each **AND** gate, it has two inputs α, β and there is an internal node in v s.t. used variables of α (β) are all left (right resp.) descendants

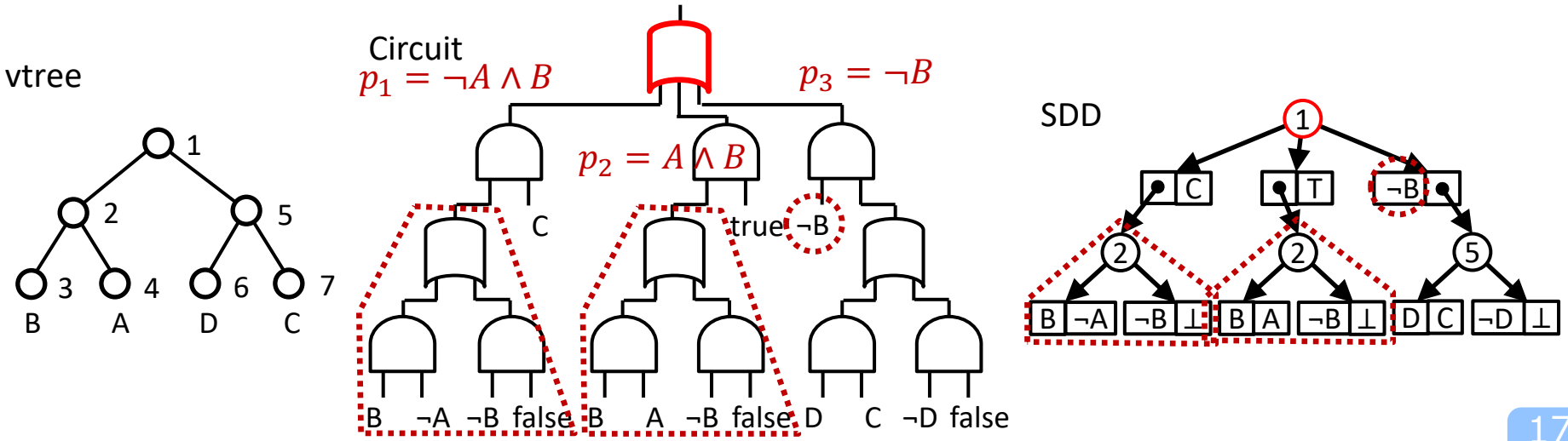


Strong determinism

For structured decomposable Boolean circuits:

- **Strongly deterministic**: for each **OR** gate, let p_i be a Boolean function of the left input of i-th child **AND** gate. Then, $p_i \wedge p_j = false\ (i \neq j)$ holds.

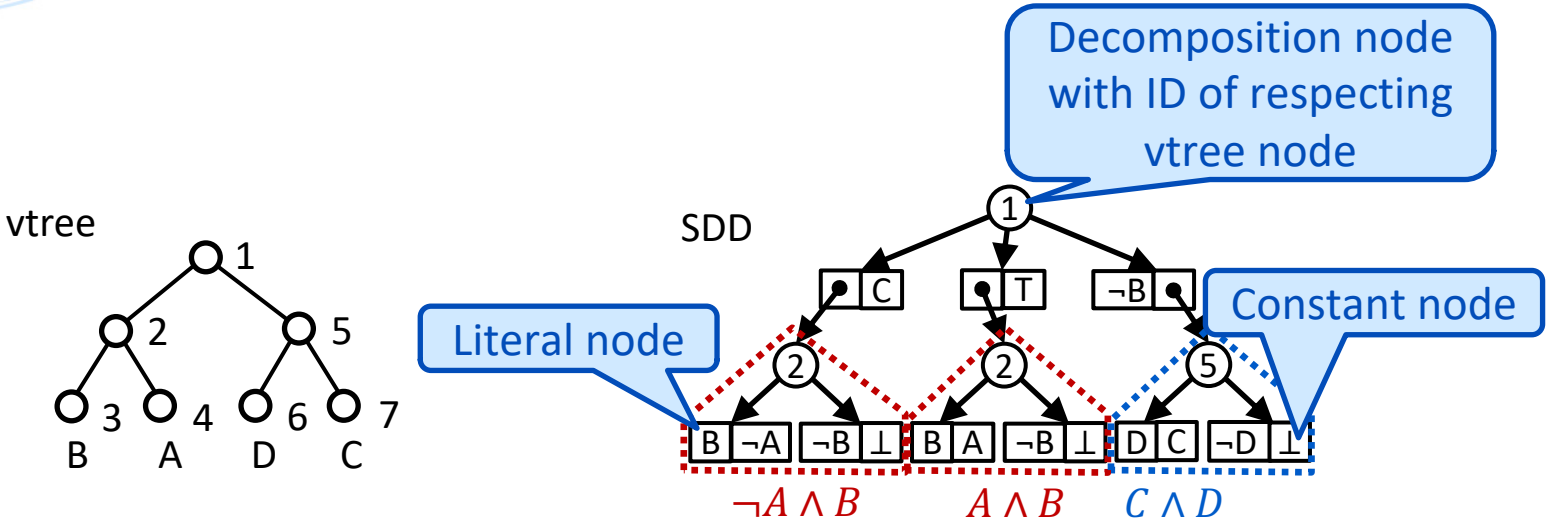
Due to structured decomposability and strong determinism, **SDD supports polytime Apply** and has preferable properties such as canonicity (uniqueness)



SDD: structure and respecting vtree node

Seeing SDD as data structure representing **Boolean function (given vtree)**

- **OR** gate of circuit is represented as **decomposition node** (circle)
  - It has corresponding vtree node called **respecting node**
- Bottom literal (constant) of circuit is represented as **literal** (**constant** resp.) **node**
- **Size of SDD** is defined as sum of #(inputs) of all OR gates



Contents

- Introduction
- Related works
- SDD
- **VS-SDD and its property**
- Experiments
- Conclusion

Observation

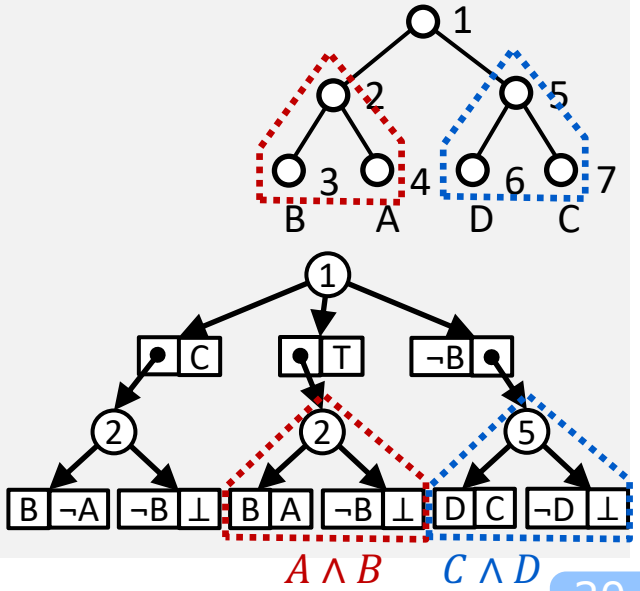
Equivalent substructures (except for labels) emerges if:

- their respecting vtrees are identical (same shape) except for labels
- their representing functions are equivalent under variable substitution induced from vtree isomorphism

Ex.) SDD representing $f = (A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$

- Vtrees rooted at 2 and 5 are identical
  - Exchanged by following substitution: $B \leftrightarrow D$, $A \leftrightarrow C$
- $A \wedge B$ and $C \wedge D$ are equivalent under above substitution

-> Bottom-middle and bottom-right nodes have **same shape**



VS-SDD: key idea

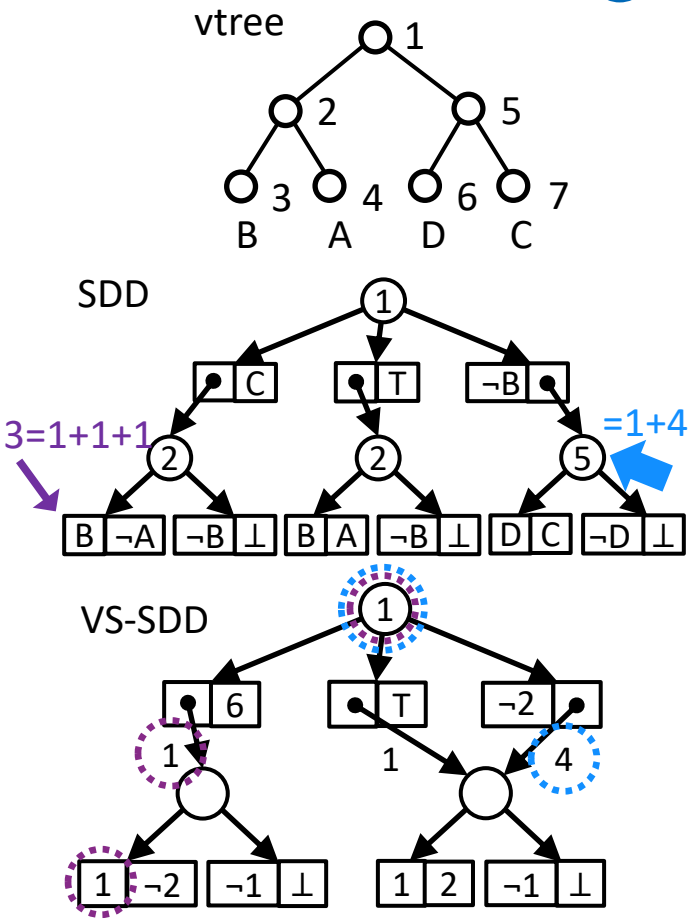
- SDD retains **respecting vtree node (ID)** of each decomposition and literal **explicitly**

-> Substructures representing Boolean functions with different variables cannot be shared

- VS-SDD retains it **differentially** in edges

… It is represented as **sum of numbers from root to decomposition/literal node**

- To facilitate this idea, **vtree nodes are numbered following preorder traversal of vtree**



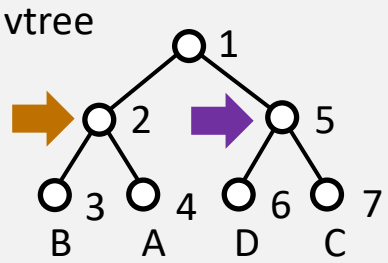
VS-SDD: example

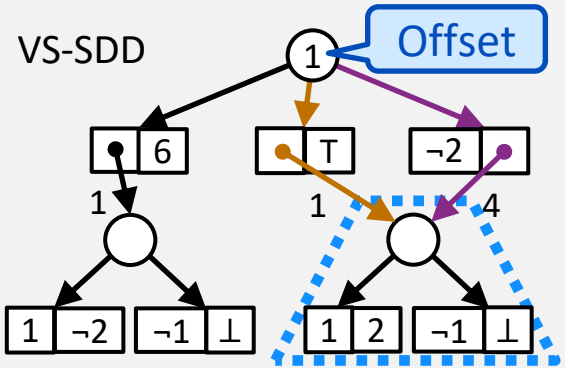
Given vtree, VS-SDD is also data structure to represent Boolean function

- Based on SDD structure
- Respecting vtree node ID of node is retained by **the sum of offset and the numbers from root to it**

Ex.) VS-SDD representing $f = (A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$

Marked substructure represents $A \wedge B$ when traversing yellow path, and $C \wedge D$ when traversing purple path



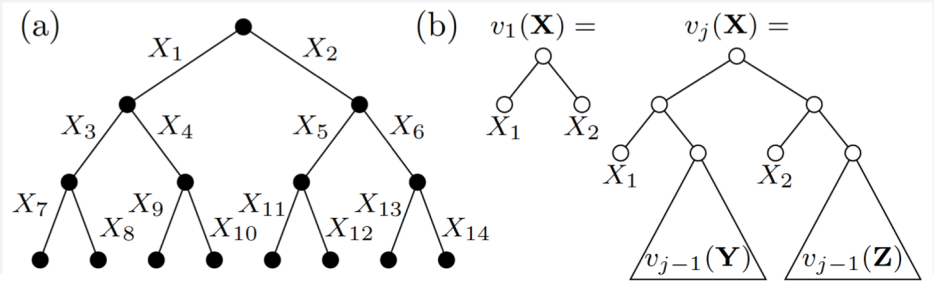


Size of VS-SDD

- The size of VS-SDD is **not larger than its SDD counterpart**
- There is a class of function for which VS-SDD is **exponentially smaller than SDD**

Ex.) Boolean function with high symmetry

- Attach a variable for each edge of complete binary tree of depth j
- Consider a Boolean function that evaluates $true$ iff the edges corresponding to $true$ variables constitute a matching
  - SDD size is $\boldsymbol{\Omega(2^j)}$ given any vtree
  - VS-SDD size is $\boldsymbol{O(j)}$ given vtree with recursive structure
    - For $j = 14$, SDD size is 278377 while VS-SDD size is only **259**



Other properties and operations of VS-SDD

NTT

Properties:

- VS-SDDs have **canonicity** (under some assumption), as SDDs do
  - Given vtree, there is **unique VS-SDD** for a Boolean function

Operations:

- VS-SDDs support **Apply operation** in $O(|\alpha||\beta|)$ **time**
- Useful queries listed in [Darwiche & Marquis '02] that are supported in polytime by SDDs are **also supported in polytime by VS-SDDs**
  - Including **model count**, **equivalency**, …

Contents

- Introduction
- Related works
- SDD
- VS-SDD and its property
- **Experiments**
- Conclusion

Experiment

Evaluating **how much our approach reduces the DAG size**

- For benchmark Boolean functions (in CNF form), we **compare the sizes of SDD and VS-SDD**, **given the same vtree**
  - Vtree is generated by SDD package version 2.0 [Choi & Darwiche '18] with balanced initial vtree

    … Meaning that vtree is searched to suit for SDDs
- Used benchmarks:
  - **Planning dataset** (used as benchmark for Sym-DDG [Bart et al. '14])
  - **N-queens problem** (used as benchmark for ZDD [Minato '93])
  - **Grid matching enumeration**
- Problems in which SDD compilation took more than 10 minutes are omitted
- Environment: 64-bit macOS, Intel Core i7@2.5GHz, 16GB RAM, Language: C++

Planning dataset

Given: **state variables** (Boolean) $f_1, \ldots, f_n$, **set of actions** $\{a_1, \ldots, a_k\}$

- Action changes assignment of state variables, depending on current assignment
- Given **initial and goal assignments**, we want to find out **a sequence of actions** that maps initial assignment to goal assignment

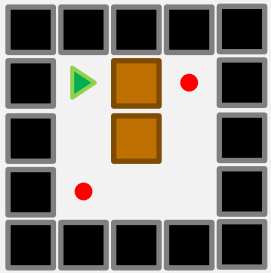
Modeling this as a Boolean function of following variables, it exhibits **high symmetry**

- $f_{t,i}$: f_i of timestamp t $(= 0, \ldots, T)$
- $a_{t,j}$: whether action a_j is performed at timestamp t $(= 1, \ldots, T)$

Ex.) Sokoban puzzle

- Player wants to move boxes to storage by pushing box
- Player can only push box when the square next to box is empty

* In experiment, more simple planning problems are used

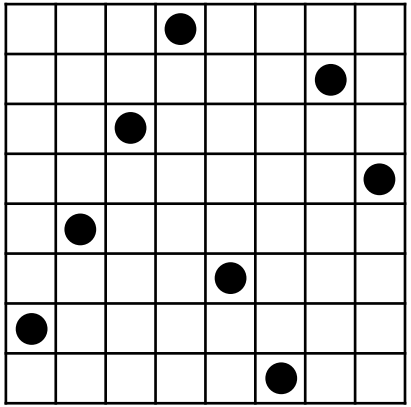


Other datasets

- **N-queens puzzle**

  … Place N queens on $N \times N$ chessboard s.t. no two queens attack each other

  – Associate variable for each square, and consider function that evaluates $true$ iff constituting solution of N-queens puzzle



- **Matching enumeration on graph** [Kawahara et al. ’17]

  – Associate variable for each edge, and consider function that evaluates $true$ iff constituting matching

  … For **grid graphs**, such function exhibits symmetry

Results

Planning dataset (Suffix "_tn" stands for $T = n$):

- VS-SDD reduces the sizes to **around 60-80% of original SDD**
  - This compression ratio is **competitive to (or for some cases better than) that of Sym-DDG compared to DDG**

Many nodes representing substitution-equivalent functions are found among bottom nodes of SDD, yielding substantial size decrease of VS-SDD

SDD size

VS-SDD size

NTT

Problem	#vars	S	V	ratio
blocks-2_t3	248	8811	7057	80.1%
blocks-2_t5	406	31861	28858	90.6%
bomb-5-1_t3	348	3798	2278	60.0%
bomb-5-1_t5	564	6327	3960	62.6%
bomb-5-1_t7	780	11212	7287	65.0%
bomb-5-1_t10	1104	16514	10426	63.1%
comm-5-2_t3	488	20584	18033	87.6%
emptyroom-4_t3	116	1822	1146	62.9%
emptyroom-4_t5	188	3090	1885	61.0%
emptyroom-4_t7	260	5073	3001	59.2%
emptyroom-4_t10	368	106737	103417	96.8%
emptyroom-8_t3	244	10511	8549	81.3%
safe-5_t3	54	567	441	77.8%
safe-5_t5	86	898	640	71.2%
safe-5_t7	118	1710	1314	76.8%
safe-5_t10	166	2506	1756	70.1%
safe-30_t3	304	5476	4067	74.3%
safe-30_t5	486	8710	6328	72.7%
safe-30_t7	668	14449	10371	71.8%
safe-30_t10	941	23469	17421	74.2%
8-Queens	64	2222	1624	73.1%
9-Queens	81	5559	4767	85.8%
10-Queens	100	10351	9159	88.5%
11-Queens	121	30611	28876	94.3%
Matching-6x6	60	13091	12671	96.8%
Matching-8x8	112	98200	97103	98.8%
Matching-6x18	192	36228	34241	94.5%

Results

Other datasets:

- N-queens data exhibits **better compression ratios**
- Matching data does not fit to VS-SDD

Due to the asymmetry in strong determinism …?

Problem	#vars	S	V	ratio
blocks-2_t3	248	8811	7057	80.1%
blocks-2_t5	406	31861	28858	90.6%
bomb-5-1_t3	348	3798	2278	60.0%
bomb-5-1_t5	564	6327	3960	62.6%
bomb-5-1_t7	780	11212	7287	65.0%
bomb-5-1_t10	1104	16514	10426	63.1%
comm-5-2_t3	488	20584	18033	87.6%
emptyroom-4_t3	116	1822	1146	62.9%
emptyroom-4_t5	188	3090	1885	61.0%
emptyroom-4_t7	260	5073	3001	59.2%
emptyroom-4_t10	368	106737	103417	96.8%
emptyroom-8_t3	244	10511	8549	81.3%
safe-5_t3	54	567	441	77.8%
safe-5_t5	86	898	640	71.2%
safe-5_t7	118	1710	1314	76.8%
safe-5_t10	166	2506	1756	70.1%
safe-30_t3	304	5476	4067	74.3%
safe-30_t5	486	8710	6328	72.7%
safe-30_t7	668	14449	10371	71.8%
safe-30_t10	941	23469	17421	74.2%
8-Queens	64	2222	1624	73.1%
9-Queens	81	5559	4767	85.8%
10-Queens	100	10351	9159	88.5%
11-Queens	121	30611	28876	94.3%
Matching-6x6	60	13091	12671	96.8%
Matching-8x8	112	98200	97103	98.8%
Matching-6x18	192	36228	34241	94.5%

Contents

- Introduction
- Related works
- SDD
- VS-SDD and its property
- Experiments
- **Conclusion**

Conclusion

- VS-SDD represents Boolean function succinctly as OBDD and SDD do
- VS-SDDs incur **no additional overhead over SDDs** while **being potentially smaller than SDDs**
- Experiments show that **VS-SDD effectively captures the symmetries of functions** especially for planning datasets

Future work:

- Suitable vtree search for VS-SDD
- Top-down construction of VS-SDD