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

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

\[ f : \{true, false\}^n \to \{true, false\} \]

**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

<table>
<thead>
<tr>
<th></th>
<th>(A)</th>
<th>(B)</th>
<th>(C)</th>
<th>(f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>f</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>f</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>f</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>f</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>f</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>f</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>f</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

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.
OBDD and Apply operation [Bryant ’86]

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

- **Input**: Two OBDDs $\alpha$, $\beta$ and **binary operator** $\circ$
  - conjunction($\land$), disjunction($\lor$), etc.
- **Output**: OBDD representing $f \circ g$
  - $\alpha$ ($\beta$) represents $f$ ($g$ resp.)

Fundamental in

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

![Diagram](image)
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 $O(|\alpha||\beta|)$ time

Input: Two SDDs $\alpha, \beta$ and oper. $\circ$
Output: SDD representing $f \circ g$
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 representing the same function
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.) $C \land D$ can be obtained from $A \land B$ 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;
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 $O(|\alpha||\beta|)$ time

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

Input: Two VS-SDDs $\alpha$, $\beta$ and oper. $\circ$
Output: VS-SDD representing $f \circ g$
Contents

- Introduction
- Related works
- SDD
- VS-SDD and its property
- Experiments
- Conclusion
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
  – Solvability of useful operations is not considered
• **\(\mathbin{\triangleleft}\Delta\)BDD** [Anuchitanukul et al. ’95] ... up to “shifting” of variables
  – Operation like Apply is supported, but its time complexity is not polynomial of DAG sizes
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

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

\[
\begin{align*}
B & \quad \neg A \quad \neg B \quad \text{false} \\
A & \quad \neg B \quad \text{false} \\
D & \quad \neg D \quad \text{false}
\end{align*}
\]

Circuit

\[
\begin{align*}
C & \\
\neg B & \\
\text{true} &
\end{align*}
\]

SDD

\[
\begin{align*}
1 & \\
2 & \\
5 &
\end{align*}
\]
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 $\alpha, \beta$ and there is an internal node in $v$ s.t. used variables of $\alpha$ ($\beta$) 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, \( p_i \land p_j = \text{false} \) (\( i \neq j \)) 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
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 \( f = (A \land B) \lor (B \land C) \lor (C \land D) \)
• Vtrees rooted at 2 and 5 are identical
  – Exchanged by following substitution: \( B \leftrightarrow D, \ A \leftrightarrow C \)
• \( A \land B \) and \( C \land D \) 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 $f = (A \land B) \lor (B \land C) \lor (C \land D)$

**Marked substructure** represents $A \land B$ when traversing yellow path, and $C \land D$ 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 \( \Omega(2^j) \) given any vtree
  - VS-SDD size is \( O(j) \) 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

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
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++
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 \textit{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 \textit{true} iff constituting matching
  ... For \textit{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
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 ...?
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