On the Sizes of Decision Diagrams Representing the Set of All Parse Trees of a Context-free Grammar

Kei Amii K.AMII@IIP.IST.I.KYOTO-U.AC.JP
Kyoto University
Kyoto (Japan)

Masaaki Nishino NISHINO.MASAAKI@LAB.NTT.CO.JP
NTT Communication Science Laboratories
Kyoto (Japan)

Akihiro Yamamoto AKIHIRO@IIP.IST.I.KYOTO-U.AC.JP
Kyoto University
Kyoto (Japan)

Abstract

In this paper, we analyze the size of decision diagrams (DD) representing the set of all parse trees of a context-free grammar (CFG). CFG is widely used in the field of natural language processing and bioinformatics to estimate the hidden structures of sequence data. A decision diagram is a data structure that represents a Boolean function in a concise form. By using DDs to represent the set of all parse trees, we can efficiently perform many useful operations over the parse trees, such as finding trees that satisfy additional constraints and finding the best parse tree. Since the time complexity of these operations depends on DD size, selecting an appropriate DD variant is important. Experiments on a simple CFG show that the Zero-suppressed Sentential Decision Diagram (ZSDD) is better than other DDs; we also give theoretical upper bounds on ZSDD size.

Keywords: context-free grammar, decision diagram, ZSDD

1. Introduction

A context-free grammar (CFG) is a model often used to estimate the hidden structure of sequence data. It is widely used for syntax analysis in natural language processing [Manning and Schütze (1999)], RNA secondary structure analysis in bioinformatics [Durbin et al. (1998)], and so on. The CYK (Cocke-Younger-Kasami) algorithm is a well-known technique for determining how a given sequence data is generated from a given CFG, that is, to obtain parse trees. If N is sequence data length and $|P|$ is the size of a CFG grammar, the set of all parse trees can be obtained in $O(|P|N^3)$ time by using the CYK algorithm. Unfortunately, the CYK algorithm cannot be used to find the set of all parse trees that satisfy some additional constraints, e.g., a restriction on the number of times a rule can be used or prohibition of the use of a particular pair of rules at the same time. By using additional constraints, we can analyze structures with the utilization of background knowledge. For example, in natural language processing, we can restrict the number of times a specific part of speech can appear or make a particular pair of words take a common part of speech.

Similarly, in bioinformatics, constraints are derived from knowledge of bond distance. In this paper, we introduce a method based on decision diagrams (DDs) to handle such constraints.

Decision Diagrams (DDs) are data structures that represent Boolean functions as directed acyclic graphs. A DD can support various operations on a Boolean function in time polynomial with DD size. By representing the set of all possible parse trees as a Boolean function, we can perform various operations efficiently, such as finding trees that satisfy additional constraints and finding the best parse tree.

Since the efficiency of performing DD operations strongly depends on DD size, it is important to select a succinct DD representation. There are several variants of decision diagram, such as Binary Decision Diagram (BDD) [Bryant (1986)], Zero-suppressed Binary Decision Diagram (ZDD) [Minato (1993)], Sentential Decision Diagram (SDD) [Darwiche (2011)], and Zero-suppressed Sentential Decision Diagram (ZSDD) [Nishino et al. (2016)]. Which DD variant is the smallest depends on targeted Boolean functions. Moreover, each DD has configurable parameters, namely variable ordering and vtrees, which influence DD size. Thus, in order to treat a set of parse trees efficiently, we need (1) to choose the most appropriate DD, and (2) to optimize DD parameters.

This paper compares DD size of the above 4 kinds of DDs given some orders of variables and some vtrees. The result show that ZSDD has smaller size than the others. Moreover, we give the theoretical upper bound of ZSDD and a vtree which makes the size smallest in our experiment.

We define the context-free grammar and CYK algorithm as technical preliminaries in Section 2. Next, Section 3 introduces our method to convert the set of all parse trees of a context-free grammar into a Boolean function. We illustrate DDs in Section 4, and we describe our experiment on 4 DDs, and show that ZSDD has the smallest size when representing the set of all parse trees of a simple context-free grammar in Section 5. ZSDD size is theoretically analyzed in Section 6, and we provide a summary and future work in Section 7.

2. Technical Preliminaries

2.1 Context-Free Grammar

Definition 1 *A context-free grammar (CFG) G is the 4-tuple (V, Σ, P, S), where V is a finite set of non-terminal characters, Σ is a finite set of terminal characters, P is a finite set of production rules in the form $A \to \alpha$ $(A \in V, \alpha \in (\Sigma \cup V)^*)$, and S is a start symbol in V.*

In this paper, we treat only context-free grammars that are Chomsky normal form. A context-free grammar, G, is a Chomsky normal form (CNF) if all of its production rules take the following forms.

- $A \to BC$ $(A, B, C \in V)$
- $A \to \alpha$ $(\alpha \in \Sigma)$
- $S \to \epsilon$

An arbitrary context-free grammar can be converted into Chomsky normal form, and the CYK (Cocke-Younger-Kasami) algorithm yields the set of all parse trees of the Chomsky normal form.

2.2 CYK Algorithm

The CYK algorithm can determine whether given sequence data $\alpha_1\alpha_2\dots\alpha_N$ is generated from CFG G. If the data is generated, it also finds which production rules generated the sequence data. We use α_{ij} to represent subsequence $\alpha_i\dots\alpha_N$ $(i \le j)$, and q_k to represent the k-th production rule $(q_k \in P)$.

Let $S_{ij}^{(X)}$ be the set of indices used to generate α_{ij}, where the start symbol is X. It follows that $S_{ii}^{(X)}$ is defined as $S_{ii}^{(X)} = \{k|q_k = (X \to \alpha), \alpha = \alpha_i\}$. $S_{ij}^{(X)}$ can be found recursively as $S_{ij}^{(X)} = \bigcup_{m=i}^{j-1}\{k|q_k = (X \to YZ), S_{im}^{(Y)} \neq \emptyset, S_{m+1,j}^{(Z)} \neq \emptyset\}$, where $i < j$. If $S_{1N}^{(S)} \neq \emptyset$, the given sequence data can be generated and parse trees can be obtained by tracing indices in $S_{ij}^{(X)}$. It is known that calculating all $S_{ij}^{(X)}$ takes $O(|P|N^3)$ time. If all $S_{ij}^{(X)}$ are obtained by dynamic programming, we can use them to enumerate all possible parse trees.

3. Transforming a parse tree set into a Boolean function

In order to define a Boolean function that represents the set of all parse trees, we use Boolean variable $b_{i,j,k}$ to represent the appearance of production rules in a parse tree. $b_{i,j,k} = 1$ when α_{ij} is generated from the k-th production rule q_k, where α_{ij} is subsequence $\alpha_i\dots\alpha_j$.

We define a Boolean function that returns $true$ when given a set of rules that corresponds to a valid parse tree by imposing constraints as follows. Each subsequence, α_{ij}, is generated from at most one production rule. Therefore, we need a constraint that makes multiple variables among b_{ij*} not true. This is written as

$$H_{ij} = \bigwedge_{b_{ijk},b_{ijl}:k\neq l} \neg b_{ijk} \vee \neg b_{ijl}. \tag{1}$$

Next, we define the following constraint to make ensure one of the rules in $S_{ij}^{(X)}$ is used.

$$F_{ij}^{(X)} = \bigvee_{k\in S_{ij}^{(X)}} b_{ijk}. \tag{2}$$

If $q_k = (X \to YZ)$ is used, the rules that correspond to Y and Z must be used.

$$D_{ijk} = \neg b_{ijk} \vee \left(\bigvee_{m=i}^{j-1} F_{i,m}^{(Y)} \wedge F_{m+1,j}^{(Z)}\right) \tag{3}$$

Finally, a constraint on arbitrary pairs b_{ijk}, b_{mnl} such that $i < m \le j < n$ is defined as

$$C = \bigwedge_{b_{ijk},b_{mnl}:i<m\le j<n} \neg b_{ijk} \vee \neg b_{mnl}. \tag{4}$$

By using these constraints, we can represent a Boolean function that returns $true$ when given a set of rules that corresponds to correct generation as

$$\left(\bigwedge_{i,j:1\le i<j\le N} H_{ij}\right) \wedge \left(\bigwedge_{b_{ijk}} D_{ijk}\right) \wedge C. \tag{5}$$

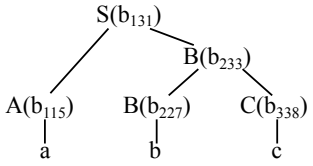


Figure 1: A correct parse tree. The variables in parentheses denote Boolean variables corresponding to each production rule.

As an example, we define a Boolean function representing the set of all parse trees of the following CFG $G = (V, \Sigma, P, S)$. Let $V = \{A, B, C\}$, $\Sigma = \{a, b, c\}$, and P be the following.

- 1. $S \rightarrow A\ B$
- 2. $A \rightarrow A\ A$
- 3. $B \rightarrow B\ C$
- 4. $B \rightarrow A\ C$
- 5. $A \rightarrow a$
- 6. $A \rightarrow b$
- 7. $B \rightarrow b$
- 8. $C \rightarrow c$

We consider the set of parse trees for sequence data *abc*. Using the CYK algorithm, we can find $S_{ij}^{(X)}$ as follows (other than empty set).

- $S_{11}^{(A)} = \{5\}$
- $S_{22}^{(A)} = \{6\}$
- $S_{22}^{(B)} = \{7\}$
- $S_{33}^{(C)} = \{8\}$
- $S_{12}^{(A)} = \{2\}$
- $S_{23}^{(B)} = \{3, 4\}$
- $S_{13}^{(S)} = \{1\}$

We use Boolean variables $b_{115}, b_{226}, b_{227}, b_{338}, b_{122}, b_{233}, b_{234}, b_{131}$ and impose constraints. From (1), $H_{22} = \neg b_{226} \vee \neg b_{227}$, $H_{23} = \neg b_{233} \vee \neg b_{234}$.

From (2), $F_{23}^{(B)} = b_{233} \vee b_{234}$. With regard to the other variables, $F_{11}^{(A)} = b_{115}$, $F_{22}^{(A)} = b_{226}$, ...

From (3),

- $D_{122} = \neg b_{122} \vee (F_{11}^{(A)} \wedge F_{22}^{(A)})$
- $D_{233} = \neg b_{233} \vee (F_{22}^{(B)} \wedge F_{33}^{(C)})$
- $D_{234} = \neg b_{234} \vee (F_{22}^{(A)} \wedge F_{33}^{(C)})$
- $D_{131} = \neg b_{131} \vee (F_{11}^{(A)} \wedge F_{23}^{(B)}) \vee (F_{12}^{(A)} \wedge F_{33}^{(B)})$.

From (4), $C = (\neg b_{122} \vee \neg b_{233}) \wedge (\neg b_{122} \vee \neg b_{234})$.

We can represent the set of all parse trees as (5) by using H_{ij}, D_{ijk}, C. This Boolean function returns 1 only when $b_{115} = 1, b_{226} = 0, b_{227} = 1, b_{338} = 1, b_{122} = 0, b_{233} = 1, b_{234} = 0, b_{131} = 1$, which corresponds to a correct parse tree as indicated in Fig. 1.

4. Decision Diagram

Due to space limits, we focus on ZSDD. Although ZSDD was invented to represent families of sets, ZSDD can be also used to represent a Boolean function because a family of sets is equivalent to a Boolean function.

4.1 $(\mathbf{X}, \mathbf{Y})$-decomposition

$(\mathbf{X}, \mathbf{Y})$-decomposition is a method that divides a given family of sets into smaller subsets, a significant step in composing DDs. Let f be a family of sets, and $\mathbf{X}$, $\mathbf{Y}$ be groups of variables that compose a partition of the variables of f. It follows that the function can be

(a) A vtree (b) A ZSDD

Figure 2: (a) An example of a vtree, (b) ZSDD representing the family of sets $\{\{A, B\}, \{B\}, \{B, C\}, \{C, D\}\}$ given the vtree in (a).

decomposed as

$$f = [p_1(\mathbf{X}) \sqcup s_1(\mathbf{Y})] \cup \cdots \cup [p_n(\mathbf{X}) \sqcup s_n(\mathbf{Y})], \tag{6}$$

where $p_i(\mathbf{X}), s_i(\mathbf{Y})$ are subfunctions whose variables are $\mathbf{X}$ and $\mathbf{Y}$, respectively. Operations $\cup$ and $\sqcup$ are union and join, respectively. They are defined as $f \cup g = \{a \mid a \in f \text{ and } a \in g\}$ and $f \sqcup g = \{a \cup b \mid a \in f \text{ and } b \in g\}$. We call $p_1, \ldots, p_n$ primes and $s_1, \ldots, s_n$ subs. If $p_i \cap p_j = \emptyset$ for all $i \neq j$, $\bigcup_{i=1}^{n} p_i$ is equal to the power set of the universal set. If $p_i \neq \emptyset$ for all i, we say the decomposition is an $(\mathbf{X}, \mathbf{Y})$-partition, and denote it as $(p_1, s_1), \ldots, (p_n, s_n)$. Moreover, if $s_i \neq s_j$ for all $i \neq j$ is satisfied, we say the $(\mathbf{X}, \mathbf{Y})$-partition is compressed. For example, given $f = \{\{A, B\}, \{B\}, \{B, C\}, \{C, D\}\}$, $\mathbf{X} = \{A, B\}$, and $\mathbf{Y} = \{C, D\}$, the $(\mathbf{X}, \mathbf{Y})$-partition is

$$[\{\{A, B\}\} \sqcup \{\emptyset\}] \cup [\{\{B\}\} \sqcup \{\emptyset, \{C\}\}] \cup [\{\emptyset\} \sqcup \{\{C, D\}\}], \tag{7}$$

where $\{\{A, B\}\}, \{\{B\}\}, \{\emptyset\}$ are primes and $\{\emptyset\}, \{\emptyset, \{C\}\}, \{\{C, D\}\}$ are subs.

4.2 vtree

We introduce vtree, another significant concept in composing ZSDDs. A ZSDD represents a family of sets as a directed acyclic graph by applying $(\mathbf{X}, \mathbf{Y})$-partition recursively. That is, ZSDD divides a given family of sets into $p_1, \ldots, p_n$ and $s_1, \ldots, s_n$ by applying the $(\mathbf{X}, \mathbf{Y})$-partition. A ZSDD represents a family of sets by recursively applying $(\mathbf{X}, \mathbf{Y})$-partitions, where partition order is determined by the vtree. We can make a unique ZSDD given a family of sets and a vtree. A vtree is a binary tree whose leafs correspond to variables. We show a vtree example in Fig. 2(a).

The vtree root represents the partition of variables into two groups: variables that appear in the left subtree and those that appear in the right subtree. In this figure, root node $v = 3$ represents $(\mathbf{X}, \mathbf{Y})$-partition where $\mathbf{X} = \{A, B\}$ and $\mathbf{Y} = \{C, D\}$. Similarly, node $v = 1$ represents a partition where $\mathbf{X} = \{B\}$ and $\mathbf{Y} = \{A\}$. In this way, every non-leaf vtree node represents a partitioning. We use v^l, v^r to represent the left and the right child vtree nodes of v, respectively. We say a vtree is right-linear if each left-child of an internal node is a leaf. To avoid confusion, we use the term vnode to refer to nodes in the vtree and denote them as $v, v^l, v^r, v_1, v_2, \ldots$.

4.3 Zero-suppressed Sentential Decision Diagram (ZSDD)

We recursively define the Zero-suppressed Sentential Decision Diagram (ZSDD) as follows. Let α be a ZSDD and $\langle\alpha\rangle$ be the set that α represents.

Definition 2 *α is a ZSDD that respects vtree v iff:*

- $\alpha = \epsilon$ *or* $\alpha = \bot$.
  Semantics: $\langle\epsilon\rangle = \{\emptyset\}$ *and* $\langle\bot\rangle = \emptyset$
- $\alpha = X$ *or* $\alpha = \pm X$ *and v is a leaf with variable X.*
  Semantics: $\langle X\rangle = \{\{X\}\}$ *and* $\langle\pm X\rangle = \{\{X\}, \emptyset\}$
- $\alpha = \{(p_1, s_1), \ldots, (p_n, s_n)\}$, *$v$ is internal, $(p_1, \ldots p_n)$ are ZSDDs that respect subtrees of v^l, $s_1, \ldots s_n$ are ZSDDs that respect subtrees of v^r, and $\langle p_1\rangle, \ldots \langle p_n\rangle$ is a partition.*
  Semantics: $\langle\alpha\rangle = \bigcup_{i=1}^{n} \langle p_i\rangle \sqcup \langle s_i\rangle$

$\epsilon, \bot, X, \pm Z$ are called terminal ZSDDs. Other ZSDDs represent $(\mathbf{X}, \mathbf{Y})$-partition $\{(p_1, s_1), \ldots, (p_n, s_n)\}$ corresponding to vnode v.

Fig. 2(b) is an example of a ZSDD that represents the family of sets $\{\{A, B\}, \{B\}, \{B, C\}, \{C, D\}\}$ given the vtree in Fig. 2(a). We represent $(\mathbf{X}, \mathbf{Y})$-partition as a circle node, and call it a decision node. A decision node has child nodes, and each child node is represented as a pair of boxes. The left box of a child node corresponds to prime p, while the right box corresponds to sub s. We call the ZSDD generated by $(\mathbf{X}, \mathbf{Y})$-partition a subfunction on $\mathbf{X}$. We define the size of a decision node as the number of pairs of primes and subs covered by the node, and the size of ZSDD as the sum of the sizes of all decision nodes.

We show the how DDs can be used to efficiently perform probabilistic context free grammar (PCFG) inferences, including the computation of the probability of a sequence data and the problem of finding the most likely parse tree. The computation of probability data can be formulated as the weighted model counting (WMC) problem, whose solution time is known to be linear with DD size [Chavira and Darwiche (2008)]. The problem of finding the most likely parse tree can be solved by an algorithm that is similar to the WMC algorithm for DDs.

We obtain the most likely parse tree that satisfies constraints with the following steps.

1. Construct a DD that represents the set of all valid parse trees.
2. Perform binary operation to obtain the DD that represents the set of parse trees that satisfy the constraints.
3. Perform WMC with the obtained DDs by dynamic programming.

5. Experiment

We use BDD, SDD, ZDD, and ZSDD to represent the set of all parse trees of the context-free grammars described below and compare them in terms of size. The experiment uses context-free grammars G_1 and G_2 because they are simple and typical.

$$G_1: \quad A \to AA\,, \quad A \to a$$
$$G_2: \quad A \to AA\,, \quad A \to AB, \quad B \to BA, \quad A \to a, \quad B \to b$$

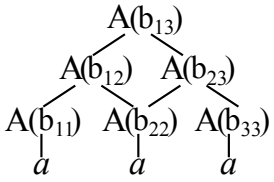


Figure 3: The correspondence of production rules of G_1 to Boolean variables

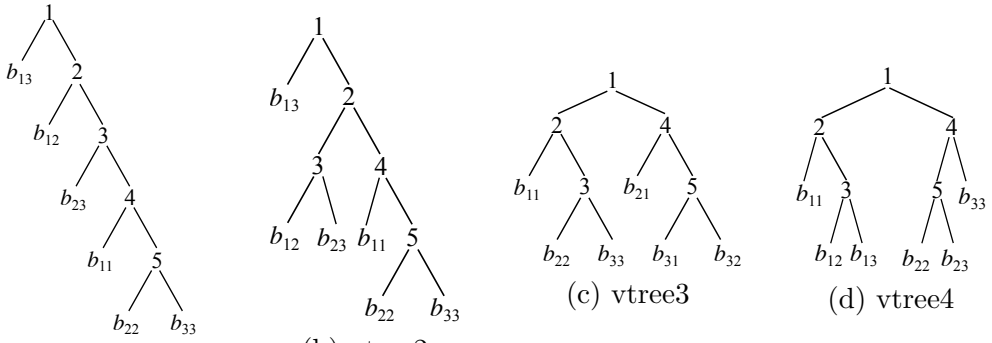


Figure 4: vtrees for grammar G_1: sequence length is 3

Let A, B be a non-terminal symbol, and a, b be a terminal symbol. We show the DD size versus the length of sequence data n. In grammar G_1, the number of terminal and non-terminal symbols is only one, so we use $b_{i,j}$ instead of $b_{i,j,l}$. We parse sequence like $aaaa$. In grammar G_2, the sequence is composed of a repetition of ab as in $ababab$. The correspondence of production rules to Boolean variables is shown in Fig. 3.

The problem of finding optimal variable ordering for a BDD is NP-complete [Tani et al. (1996)]. However, it is known BDDs tend to be small if the variable order places related variables close [Fujita et al. (1988)]. Thus, we examine the following 4 typical vtrees that place related variables close. We show examples of each in Fig. 4; sequence length is 3. BDD and ZDD are limited to decomposition by a single variable, and are equivalent to SDD and ZSDD given right-linear vtrees.

vtree1 A right-linear vtree decomposing in descending order of the value $j - i$. If the values are equal, it does in descending order of value i. This corresponds to the seriate variables from root to leaves in Fig. 4(a).

vtree2 A vtree combining right-linear vtrees composed of variables whose values, $j - i$, are equal. This decomposes variables that have the same height, Fig. 4(b), at once.

vtree3 A vtree combining two right-linear vtrees. One corresponds to terminal symbols, and the other corresponds to non-terminal symbols. At root vnode v, v^l corresponds to terminal symbols and v^r corresponds to non-terminal symbols, see Fig. 4(c).

vtree4 A vtree combining right-linear vtrees composed of the variables that have equal value, i. This corresponds to making the right-linear vtree from left in Fig. 4(d).

		G_1								G_2				
n		3	4	5	6	7	8	9	10	2	4	6	8	10
BDD	vtree1	14	47	92	223	416	901	1164	2087	4	52	304	955	2637
ZDD	vtree1	6	15	34	78	177	395	874	1914	2	13	68	330	1519
SDD	vtree2	12	44	106	202	419	803	1346	2558	4	56	284	886	2281
	vtree3	10	40	112	227	458	929	2086	2174	6	59	376	1252	3448
	vtree4	12	42	96	218	425	723	1443	2735	4	58	302	1027	2989
ZSDD	vtree2	6	15	34	78	173	**381**	**825**	**1772**	2	13	64	303	1398
	vtree3	6	14	32	75	173	390	868	1907	2	11	60	297	1390
	vtree4	8	22	53	120	263	566	1205	2548	2	21	113	511	2171

Table 1: DD size versus sequence length.

We gave vtree1 to BDD and ZDD, and vtrees 2,3,4 to SDD and ZSDD. For grammar G_2, we gave right-linear vtree composed of b_{ij*} instead of b_{ij}.

Tab. 1 shows the results. We can find that the combination of ZSDD and vtree2 yields the smallest size when n is large in G_1. In G_2, the combination of ZSDD and vtree3 does, but there is virtually no difference compared to vtree2. ZSDD is known to be smaller than SDD when the model is small [Nishino et al. (2016)], which might be why ZSDD is smaller than SDD in our experiment.

6. ZSDD Size

Given the above result, we discuss here the upper bound of ZSDD size respecting vtree2 given n on G_1. Before analyzing the size, we define the underlying concepts. $b_{i,j}$ is a Boolean variable corresponding to the initial rule used to generate $j-i+1$ symbols $\alpha_i \dots \alpha_j$. Let $j-i+1$ be the *height* of $b_{i,j}$, and B_r be the set of variables whose height is r. In general, $|B_r| = n-r+1$. For the first $n-1$ vnodes that are reachable by tracing only right edges from the root, the set of variables appearing in their left children equals B_r. For example, the left child of root vnode is $B_n = b_{1,n}$, the left child of its right child is $B_{n-1} = b_{1,n-1}, b_{2,n}$. In this way, vtree2 decomposes the variables by height. In the following, we investigate ZSDD size by finding the upper bound of (1) decision nodes and (2) child nodes of each decision node.

Each decision node in ZSDD represents a subfunction. It is known that no two decision nodes that correspond to equivalent subfunctions can coexist in a ZSDD. Therefore, we identify the upper bound of the size of decision nodes by discussing the number of different subfunctions that can appear in a ZSDD. If vnode v's left child corresponds to B_r, we call v a height-decompose vnode. We use a different analysis approach depending on whether the vnode that the decision node respects is a height-decompose vnode or not. In Fig. 4(b), vnode1,2 are height-decompose vnodes. Similarly, we find the upper bound of child nodes using different methods according to vnode kinds.

6.1 Decision nodes corresponding to height-decompose vnode

The decision nodes associated with height-decompose vnodes denote subfunctions representing the set of partial parse trees whose height is smaller than r, where every partial

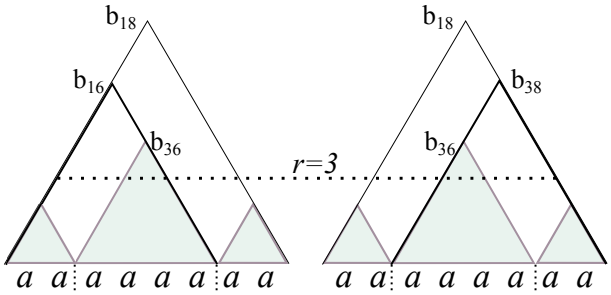


Figure 5: Example of subfunctions whose heights are smaller than r that are equivalent on two assignments($n = 8$). Left figure denotes $\{b_{18}, b_{16}, b_{36}\}$, right figure denotes $\{b_{18}, b_{38}, b_{36}\}$ when $r = 3$. The assignments are different if we consider heights larger than $r + 1$, but at heights of smaller than r we consider assignment only in the shaded triangles so the subfunctions are equivalent.

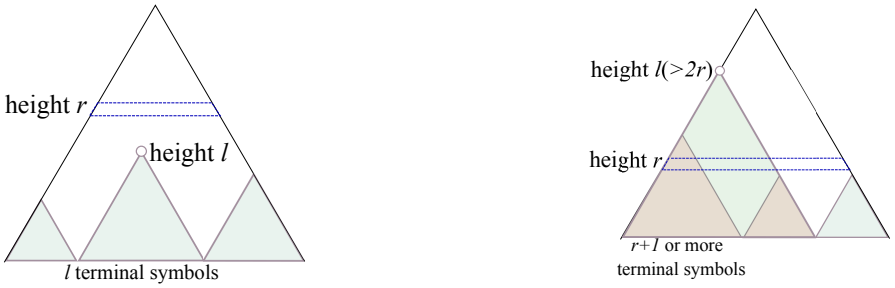
parse forms a valid parse tree in combination with some partial tree whose height is larger than $r + 1$. A subfunction represents a set of Boolean variables whose height is smaller than r, and the set forms a parse tree in combination with variables whose height is larger than $r + 1$. Before finding the upper bound of the number of subfunctions, we define the *grouping* of symbols in a sequence as follows. Given a set of variables whose height is larger than $r + 1$, we assign label b_{jk} to each terminal symbol α_i, when $j \leq i \leq k$ and the height of b_{jk} is the smallest. If the set is valid, namely it can compose valid parse trees with variables whose height is r or less, each terminal symbol always has a label. For a grouping of terminal symbols based on the label, the following theorem holds.

Theorem 3 *If two valid assignments I_1, I_2 to Boolean variables whose height is larger than $r + 1$ give equivalent groupings on the sequence data, two subfunctions f, g over variables whose height is lower than r that give valid parse trees when combined with each assignment I_1, I_2 are equivalent.*

Proof The above definition of grouping makes terminal symbols belonging to the same group always appear as a substring. If terminal symbols α_i, α_{i+1} are in different groups, we cannot use b_{jk} $(j \leq i < k)$ whose height is r or less to obtain valid parse trees. Therefore, a subfunction whose height is r or less is defined as a set of partial parse trees on successive terminal symbols in each group. Partial parse trees on a group whose height is r or less are consistent regardless of how the group is made. From the above, if groupings are equivalent, their subfunctions are also equivalent. ■

Fig. 5 shows an example of equivalent subfunctions. The parse trees and assignments are different if we consider heights larger than $r+1$, but the subfunctions are equivalent. From Theorem 3, we find the number of subfunctions can be bounded by using the number of groupings. Let maximum grouping number be the maximum group size of groupings. The following lemma shows that possible maximum grouping size number is bounded.

Lemma 4 *Maximum grouping number l satisfies $r + 1 \leq l \leq 2r$.*



(a) $l < r + 1$ (b) $2r < l$

Figure 6: An example of a grouping that does not satisfy Lemma 4.

Proof When the maximum grouping number is l, the largest height among label b_{jk} of each terminal symbol α_i is l. Since we consider assignment with height $r + 1$ or more, $r + 1 \leq l$ must be satisfied.

On the other hand, we will show the above is invalid when $2r < l$. We assume assignment I that satisfies $2r < l$ and whose height is $r + 1$ or more. If we divide the largest group into two, the length of one side is $r + 1$ or more. Since the length of the group is equal to the height of the label, the height of the label b_{jk} of terminal symbols in this group is $r + 1$ or more. That is, the maximum grouping number of I is less than $r + 1$, which is invalid. ■

Lemma 5 *Let $a_{l,n}$ be the number of groupings whose maximum grouping number is l or less on n symbols, then*

$$a_{l,n} = \begin{cases} 2^{n-1} & (n \leq l) \\ \sum_{i=n-l}^{n-1} a_{k,i} & (n > l). \end{cases} \tag{8}$$

Proof In the case that $n \leq l$, an arbitrary grouping satisfies the condition, so $a_{l,n} = 2^{n-1}$. Otherwise, the length of head group is $1, \ldots, l$. We consider the remaining groupings; their lengths are $n - 1, \ldots, n - l$. Therefore, $\sum_{i=n-l}^{n-1} a_{k,i}$ groupings exist in total. ■

Theorem 6 *The upper bound of the number of subfunctions on B_r is taken to be $a_{2r,n} - a_{r,n} = O(2^n)$.*

Proof The number of subfunctions on B_r is equal to the number of groupings of assignments whose height is $r + 1$ or more, and the maximum grouping number l satisfies $r + 1 \leq l \leq 2r$ from Lemma 4. Since $a_{r,n}$ is the number of groupings whose maximum number is r or less from Lemma 5, the upper bound is given by $a_{2r,n} - a_{r,n}$. ■

6.2 Child nodes corresponding to height-decompose vnode

In the following, we give the upper bound of child nodes that a decision node of B_r may have. The number of child nodes is equal to the number of elements of $(\mathbf{X}, \mathbf{Y})$-partition on a subfunction. Since the number of elements of $(\mathbf{X}, \mathbf{Y})$-partition is bounded by the number of possible assignments that can form a valid parse tree, it corresponds to the number of possible assignments over B_r for a decision node that respects a height-decompose vnode.

Theorem 7 *Let $f(g)$ be the number of assignments over B_r in a group whose length is g. Then, $f(g) = 1$ when $g < r+1$, $f(g) = 3$ when $g = r+1$, and $f(g) \leq 3$ when $r+1 < g \leq 2r$.*

Proof When $g < r + 1$, all variables in B_r take 0, so $f(g) = 1$.

When $g = r + 1$, the number of variables in B_r is 2. Both variables cannot take 1 at the same time. If $r \geq 3$, only variables whose heights are smaller than r can make a valid parse tree, so both variables can take 0 at the same time (Impossible if $r = 2$). Since there are three kinds of sets of used variables, namely one variable or none, $f(g) = 3$. ($f(g) = 2$ if $r = 2$)

When $r + 1 < g \leq 2r$, $b_{1,r}, \ldots, b_{g-r+1,g}$ are the variables in B_r. If $b_{i,i+r} = 1$ ($1 < i < g - r + 1$), a variable whose height is larger than $r + 1$ must take 1. This, however, contradicts the grouping, so only $b_{1,r}$ or $b_{g-r+1,g}$ can take 1. (1)If $r + 1 < g < 2r$, neither variable can take 1 because Eq. (4). Therefore, $f(g) \leq 3$. (2)If $g = 2r$, both variables must take 1. Therefore, $f(g) = 1 \leq 3$. ■

Theorem 8 *The number of assignments over B_r is $3^{n/(r+1)}$ when $r \geq 3$, and $2^{n/(r+1)}$ when $r = 2$.*

Proof If g_i is the length of a group, the number of assignments over B_r is $\prod_i f(g_i)$. When $r \geq 3$, it is the solution of the following optimization problem.

$$\max \prod_i f(g_i) \ \text{ subject to } \sum_i g_i = n, \quad r + 1 \leq \max_i g_i \leq 2r \tag{9}$$

When $g_i = n/(r + 1)$, $\prod_i f(g_i)$ takes its maximum value, which is $3^{n/(r+1)}$. Similarly, when $r = 2$ we can obtain the maximum value, $2^{n/(r+1)}$. ■

Theorem 9 *The size of ZSDD when height-decompose vnode is $O(n2^n3^{n/4})$.*

Proof The number of child nodes is $O(2^n)(1+(2^{n/(r+1)})|_{r=2}+\sum_{r=3}^{n} 3^{n/(r+1)}) = O(n2^n3^{n/4})$. ■

6.3 Other vnodes

We consider vnodes in partial vtrees that are right-linear vtrees. In right-linear vtrees, each decision node corresponds to the case that one variable is used. Therefore, the size is given by the upper bound of assignments of variables and the number of variables.

Theorem 10 *The number of decision nodes on B_r is less than $O(n2^n)$.*

Proof Since $|B_r| = n-r+1$, the number of assignments is less than 2^{n-r+1} and $2^{n-r+1} < 2^n$. B_r has n or fewer variables; one variable corresponds to one decision node. Thus, the number of decision nodes is less than $O(n2^n)$. ■

Decision nodes of right-linear vtrees have two child nodes, so the size in B_r is less than $O(n2^n)$. Since the size of height-decompose vnodes is $O(2^n3^{n/4})$, it does not influence the order. Thus, the size of a ZSDD representing a set of parse trees on n symbols is $O(n2^n3^{n/4})$.

7. Conclusion

We compared the size of decision diagrams representing a set of parse trees of simple CFG, and gave the upper bound of ZSDD size, which was found to be the smallest in our experiment. This paper showed the upper bound of just one vtree, and it is not guaranteed to have the minimum size. Thus, future work is to consider other vtrees. Additionally, we should consider general grammars because we considered only a simple CFG consisting of only one terminal symbol and only one non-terminal symbol.

References

R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *Computers, IEEE Transactions*, 100.8:677–691, 1986.

M. Chavira and A. Darwiche. On Probabilistic Inference by Weighted Model Counting. *Artificial Intelligence*, 172(6-7):772–799, 2008.

A. Darwiche. SDD: A new canonical representation of propositional knowledge bases. *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 819–826, 2011.

R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. Biological sequence analysis: probabilistic models of proteins and nucleic acids. 1998.

M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and improvement of Boolean comparison method based on binary decision diagrams. *Proceedings of ACM/IEEE International Conf. on Computer-Aided Design (ICCAD)*, pages 2–5, 1988.

C. D. Manning and H. Schütze. *Foundations of statistical natural language processing.* Cambridge: MIT press, 1999.

S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. *Proceedings of the 30th international Design Automation Conference (DAC). ACM*, pages 272–277, 1993.

M. Nishino, N. Yasuda, S. Minato, and M. Nagata. Zero-Suppressed Sentential Decision Diagrams. *Proceedings of the 30th Conference on Artificial Intelligence (AAAI)*, pages 1058–1066, 2016.

S. Tani, K. Hamaguchi, and S. Yajima. The complexity of the optimal variable ordering problems of a shared binary decision diagram. *Proceedings of the 4th International Symposium on Algorithms and Computation (ISAAC)*, pages 271–281, 1996.