Notes on Sequence Binary Decision Diagrams: Relationship to Acyclic Automata and Complexities of Binary Set Operations

Shuhei Denzumi[1], Ryo Yoshinaka[2], Hiroki Arimura[1], and Shin-ichi Minato[1,2]

[1] Graduate School of IST, Hokkaido University, Japan
[2] ERATO MINATO Discrete Structure Manipulation System Project, JST, Japan
{denzumi, ry, arim, minato}@ist.hokudai.ac.jp

Abstract. Manipulation of large sequence data is one of the most important problems in string processing. Recently, Loekito *et al.* (Knowl. Inf. Syst., 24(2), 235–268, 2009) have introduced a new data structure, called *Sequence Binary Decision Diagrams (SeqBDDs, or SDDs)*, which are descendants of both acyclic DFAs (ADFAs) and binary decision diagrams (BDDs). SDDs can compactly represent sets of sequences as well as minimal ADFAs, while SDDs allow efficient set operations inherited from BDDs. A novel feature of the SDDs is that different SDDs can share equivalent subgraphs and duplicated computation in common to save the time and space in various operations. In this paper, we study fundamental properties of SDDs. In particular, we first present non-trivial relationships between sizes of minimum SDDs and minimal ADFAs. We then analyze the complexities of algorithms for Boolean set operations, called the binary synthesis. Finally, we show experimental results to confirm the results of the theoretical analysis on real data sets.

1 Introduction

1.1 Background

Compact string indexes for storing sets of strings are fundamental data structures in computer science, and have been extensively studied in the decades [2,4,5,6,9,19]. Examples of compact string indexes include: tries [1,5], finite automata and transducers [6,10], suffix trees [15], suffix arrays [14], DAWGs [2], and factor automata (FAs) [19]. By the rapid increase of massive amounts of sequence data such as biological sequences, natural language texts, and event sequences, these compact string indexes have attracted much attention and gained more importance [5,9]. In such applications, an index have not only to compactly store sets of strings for *searching*, but also have to efficiently manipulate them with various set operations, e.g., *merge, intersection*, and *subtraction*.

Minimal acyclic deterministic finite automata (*ADFAs*) [5,6,10] are one of such index structures that fulfill the above requirement based on finite automata theory, and have been used in many sequence processing applications [13,18]. However, they have drawback of complicated procedures for minimization and various set operations caused by multiple branching of the underlying directed acyclic graph structure. To overcome this problem, Loekito *et al.* [12] proposed the class of *sequence binary decision diagrams* (*sequence BDDs*, or abbreviated as *SDDs* in this paper), which is a compact representation for sets of strings that allows a variety of operations for sets of strings. An SDD is a node-labeled graph structure, which resembles to an acyclic DFA in binary form, but with the minimization rule which is different from one for

Shuhei Denzumi, Ryo Yoshinaka, Hiroki Arimura, Shin-ichi Minato: Notes on Sequence Binary Decision Diagrams: Relationship to Acyclic Automata and Complexities of Binary Set Operations, pp. 147–161.
Proceedings of PSC 2011, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-04870-2

a minimal DFA. A novel feature of the SDDs is their ability to share equivalent subgraphs and results of similar intermediate computation between different SDDs, which avoids redundant generation of nodes and computation.

1.2 Main results

In this paper, we present theoretical analysis of two fundamental problems on sequence binary decision diagrams, which have not been studied before: the relationship to acyclic automata and the complexities of binary set operations as follows.

The relationship to acyclic automata. The structure of SDDs apparently resembles that of Acyclic Deterministic Finite Automata (ADFAs), which are a classical model for representing string sets. While a state of an ADFA may have many outgoing edges, a node of an SDD always has two outgoing edges, which can be seen as just the "first-child next-sibling" representation of a branching with many edges. Indeed one can find a straightforward translation from an ADFA to an SDD and vice versa. However, there are subtle differences between those data structures and actually an SDD can be even more compact than the corresponding ADFA. We show that the minimum SDD is never larger than the minimum ADFA but the minimum ADFA can be $|\Sigma|$ times larger than the minimum SDD for the same language over Σ.

The computational complexities of binary set operations. Next, we study the complexity of the *binary synthesis*, which are binary operations for minimal SDDs, such as *union*, *intersection*, and *subtraction*, which directly construct a minimal SDD. Specifically, we study upper and lower bounds for the time complexity of the binary synthesis algorithms. Loekito *et al.* [12] have proposed algorithms for union and subtraction, which are similar to the apply algorithm Bryant [3], and have conjectured that they run in input-output linear-time. We generalize their algorithms into an algorithm $\mathsf{Meld}_\diamond$ which uniformly implements eight set operations in the style of Knuth's *melding* operation for BDDs [11]. We show an upper bound that its time complexity is quadratic in the input size, and linear in the size of non-reduced version of the output size. Moreover, we show a lower bound that $\mathsf{Meld}_\diamond$ actually requires quadratic time in input size for some infinite series of inputs using a technique recently devised for BDD [3], giving matching upper bound.

Experimental results. Finally, we run experiments on real data sets. We first observed that minimal SDDs were superior to minimal DFAs when large subgraphs were shared in inputs and outputs due to the node-sharing across multiple SDDs. We also observed that each binary synthesis operation took less than seconds to take set operations $\cup$, $\cap$, and $\setminus$ of two input SDDs with around three to four thousands of nodes each, which were relatively smaller than the running time for set construction [7].

1.3 Related works

There have been a number of researches on manipulation of finite automata in automata theory and string algorithms. The textbook [10] gives classic examples of a quadratic-time algorithm for computing the union, intersection, and subtraction of two DFAs, and a state-minimization algorithm for a given DFA. Daciuk, Mihov, Watson, and Watson [6] presented an incremental algorithm for constructing the minimal ADFA for a set of strings. Blumer *et al.* [2] and Crochemore [4] gave linear-time algorithms for construction of the minimal state ADFAs for the set of all factors of an input string. Compared with a straightforward two-stage algorithm for binary synthesis for ADFAs using product followed by state-minimization [10], the advantages of the proposed $\mathsf{Meld}_\diamond$ are its simplicity and efficiency that it directly computes the output by applying the *on-the-fly minimization* [7]. Using binary synthesis, Denzumi *et al.* [7] presented a simple linear-time algorithm for incremental construction of SDDs, and a recursive top-down algorithm for construction of factor SDDs.

SDDs inherit many of their features from *binary decision diagrams* (*BDDs*), which are compact representation for storing and manipulating combinatorial structures developed in logic design community [3,11,16,21]. Especially, BDDs equipped with binary synthesis operation were invented by Bryant [3] in the 80s for dealing with Boolean functions, while their variant with node-sharing and zero-suppress rules, called *zero-suppressed BDDs* (*ZDDs*), were proposed by Minato [16] in the 90s for sparse combinatorial sets. On their early history, reduced BDDs were constructed from tree-like circuits through offline minimization. After the invention of the *binary synthesis* algorithm by Bryant [3], it became popular to build large BDDs on-the-fly in real applications. Loekito *et al.* [12] discovered that if we remove the ordering constraint on the 1-edges from ZDDs, the resulting variant of ZDDs, which actually are SDDs, has a similar structure to ADFAs in binary form and suitable to storing and manipulating sets of strings. This observation led to the invention of SDDs [12].

Organization of this paper. In Section 2, we prepare basic notions and notations on SDDs. In Section 3, we give the size bounds for SDDs and DFAs. In Section 4, we give the time and space complexities of binary synthesis procedures for SDDs. In Section 5, we show some experimental results. In Section 6, we conclude this paper. For details of basic properties and algorithms related to SDDs not described in this paper, please consult the companion paper [7].

2 Preliminaries

In this section, we give basic definitions and notations in strings and sequence BDDs according to [11,12,16]. For the details of results not found here, please consult the companion paper [7]. An *ordered alphabet* is a pair $\langle \Sigma, \prec \rangle$ where Σ is a finite alphabet and $\prec$ is a total order on Σ. The order $\prec$ associated with Σ is often denoted by $\prec_\Sigma$ and the ordered alphabet is simply written Σ for legibility. A *string* on Σ is a sequence $s = s_1 \cdots s_n$ of letters $s_i \in \Sigma$ $(1 \le i \le n)$, where $|s| = n$ denotes the *length*. If $s = xyz$ for some $x, y, z \in \Sigma^*$, then we say that x is a *prefix*, y is a *factor*, and z is a *suffix* of s. A *string set* (or a *language*) is any finite $S \subseteq \Sigma^*$. We denote by $|S|$ the cardinality. For any $x \in \Sigma$, we define $x \cdot S = \{\, xy \mid y \in S \,\}$.

Sequence BDDs. Let dom be a countable domain of the nodes. A *sequence binary decision diagram* or a *sequence BDD* (abbreviated as *SDD*[1] here) is a directed acyclic graph (DAG) $B = \langle \Sigma, V, \tau, \mathbf{r}, \mathbf{0}, \mathbf{1} \rangle$ where $V = V(B) \subseteq \mathsf{dom}$ is a finite set of nodes, $\mathbf{r} \in V$ is called the *root* of B and $\mathbf{0}$ and $\mathbf{1} \in V$ are distinct nodes called the *0-* and *1-terminals*, resp. The nodes in $V_\mathrm{N} = V \backslash \{\mathbf{0}, \mathbf{1}\}$ are called *nonterminals*. Each node $v \in V_\mathrm{N}$ of B is labeled by a symbol $v.lab$ in Σ and has two children, the *0-child* and the *1-child*, denoted by $v.0$ and $v.1$, resp, which can be identical. We call the

[1] Note that the abbreviation SeqBDD is used to denote sequence BDD in the original paper by *Loekito et al.* [12]. We also note that the abbreviation SDD was also used for the *set decision diagrams* (Couvreur, Thierry-Mieg, Proc. FORTE 2005, LNCS 3731, 443–457, 2005) and the *spectral decision diagrams* (Thornton, Drechsler, Proc. DATE'01, IEEE, 713–719, 2001).

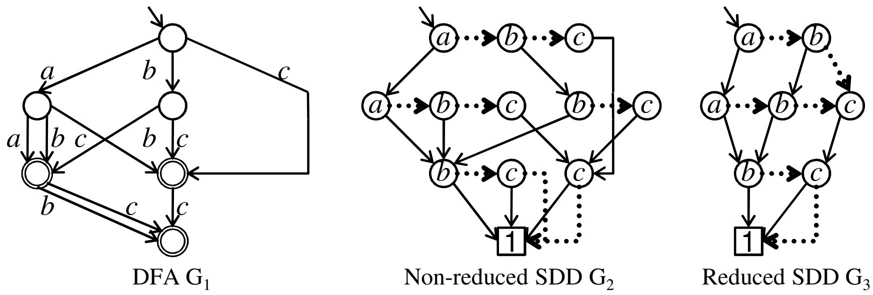

Figure 1. Examples of three index structures on $\Sigma_1 = \{a, b, c\}$ for the same string set $S_1 = \{aa, aab, aac, ab, abb, abc, ac, acc, bb, bbb, bbc, bc, bcc, c, cc\}$: a minimal DFA G_1 (left), a minimal DFA as a non-reduced SDD G_2 (middle), and a reduced SDD G_3 (right). In the figure, solid and dotted arrows indicate the 1- and 0-edge . The edges to the 0-terminal are omitted.

edge from v to $v.0$ and $v.1$ the *0-* and *1-edge* of v, resp. The information is formally described by a function $\tau : V_{\mathrm{N}} \to \Sigma \times V^2$ that assigns the triple $\tau(v) = \langle v.lab, v.0, v.1\rangle$ to each $v \in V_{\mathrm{N}}$. An SDD must be *acyclic*, that is, one may assume a strict partial order $\succ_V$ on V such that $v \succ_V v.0$ and $v \succ_V v.1$ hold for any $v \in V_{\mathrm{N}}$. The 1-child and 0-child of a node correspond to the *leftmost-child* and the *right-sibling* in a DAG in *binary form* [1,11]. Siblings are deterministically ordered from *left* to *right* according to the order $\prec_\Sigma$. That is, we always have $v.lab \prec_\Sigma (v.0).lab$ unless $v.0$ is a terminal node. We assume that any SDD B is *well-defined* meaning that B is both acyclic and deterministic. We define the *size* of B by $|B| = |V_{\mathrm{N}}| = |V| - 2$, the number of non-terminals in B.

To each node $v \in V$, we inductively (w.r.t. $\succ_V$) assign a language $L_B(v)$ as follows:

(i) $L_B(\mathbf{0}) = \emptyset$;
(ii) $L_B(\mathbf{1}) = \{\varepsilon\}$;
(iii) $L_B(v) = L_B(v.0) \cup (v.lab) \cdot L_B(v.1)$.

Equivalently, $s \in L_B(v)$ iff there is a path from v to $\mathbf{1}$ such that one obtains s by concatenating the labels of the nodes whose 1-edges appear in the path. The language $L(B)$ of B is defined to be $L_B(\mathbf{r})$. We say that two SDDs B and B' are *equivalent* if $L(B) = L(B')$. An SDD B is said to be *minimal* if it has the smallest number of nodes among the equivalent SDDs, i.e., $|B| \le |B'|$ for any SDD B' such that $L(B') = L(B)$. Figure 1 illustrates examples of SDDs together with the minimum deterministic finite automaton for the same language.

Reduced SDDs. A reduced SDD is a normal form of SDDs. An SDD is said to be *reduced* if it satisfies the following two conditions:

1. For any $u, v \in V_{\mathrm{N}}$, $\tau(u) = \tau(v)$ implies $u = v$ (*node-sharing rule*).
2. For any $v \in V_{\mathrm{N}}$, $v.1 \ne \mathbf{0}$ holds (*zero-suppress rule*).

The above rules say that no distinct non-terminal nodes have the same triple, and the 1-child of any non-terminal node v is not the 0-terminal. For any finite set of strings $L \subseteq \Sigma^*$, we can construct the *canonical SDD* for L [7] in a way similar to the minimal DFA in Myhill-Nerode theorem (e.g.,[10,20]). Actually, the next theorem gives a characterization of minimal SDDs in terms of a reduced SDD and the canonical SDD. See the companion paper [7] for the details.

Theorem 1 (Denzumi *et al.* [7]). *For any SDD B with the language $L = L(B)$, the following (1)–(3) are equivalent to each other.*

Global variable: *uniqtable*: hash table for triples.

Proc Getnode(x: letter, P_0, P_1: SDD):

if ($P_1 = 0$) **return** P_0; /* zero-suppress rule */
else if (($R \leftarrow$ *uniqtable*$[\langle x, P_0, P_1 \rangle]$) exists) **return** R; /* node-sharing rule */
else
  $R \leftarrow$ a new node with $\tau(R) = \langle x, P_0, P_1 \rangle$; *uniqtable*$[\langle x, P_0, P_1 \rangle] \leftarrow R$;
  return R;

Figure 2. The Getnode procedure for on-the-fly minimization.

(1) B is a reduced SDD.
(2) B is a canonical SDD for L up to isomorphism.
(3) B is a minimal SDD.

Due to Theorem 1, for a fixed language L, we may call a reduced SDD for L *the* reduced SDD for L when we work modulo isomorphism. In order to satisfy the node-sharing rule, we maintain a hash table, called *uniqtable*, which is the inverse of τ. That is, it gives the unique node v such that $\tau(v) = \langle x, v_0, v_1 \rangle$ (if exists) for the key $\langle x, v_0, v_1 \rangle$. As is the case for BDDs and ZDDs, usually we consider only *reduced* SDDs. Hereafter we assume that all SDDs are reduced unless otherwise noted.

While an SDD represents a set of strings, we often would like to manipulate two or more sets of strings. In our *shared* SDD environment, the terminals **0** and **1**, the function τ and thus the hash *uniqtable* : $\Sigma \times \mathsf{dom} \times \mathsf{dom} \to \mathsf{dom}$ are shared by more than one SDD in common so that we can have a compact representation of a family of sets of strings. One may think of the shared SDD environment as a single SDD with multiple roots. By picking up a node v as the root, one can extract a subgraph as an SDD consisting of all the nodes that are reachable from v. For convenience, we often identify a node v with the SDD rooted by v extracted from the shared SDD environment. Hence $|v|$ represents the number of nonterminal nodes reachable from the node v.

Figure 2 shows the node allocation procedure Getnode, which is used as a subroutine in algorithms on SDDs. Throughout this paper, we assume that the hash table *uniqtable* is a global variable, and a look-up for it takes $O(1)$ time; We have to add additional $O(\log n)$ term if we use balanced binary tree dictionary [1]. In the shared and reduced SDD environment studied here and in [7,12], we use write-only construction, similarly to [8], such that any new SDD is constructed by adding a new node on the top of already constructed SDDs using a call of Getnode given existing nodes as its arguments. The next lemma guarantees that we always have reduced SDDs as long as we solely use Getnode to obtain a new node.

Lemma 2 (Denzumi *et al.* [7]). *Let B be any reduced SDD. For any symbol $x \in \Sigma$ and nodes $v_0, v_1 \in V(B)$ in B such that $v_0 \notin V_{\mathrm{N}}$ or $x \prec_{\Sigma} v_0.lab$, if we invoke $v =$ Getnode(x, v_0, v_1) on B and add the result v to $V(B)$, then the resulting SDD B' with root v obtained from B is well-defined and reduced, too.*

Based on the procedure Getnode above, for example, we can implement an off-line minimization (i.e., reduction) algorithm Reduce for SDDs in linear time and space [7][2].

[2] The complexity analysis assumes that a look-up for the hash table *uniqtable* takes $O(1)$ time. A precise worst-case time complexity is $O(n \log n)$ if the hash table does not work efficiently.

Factually it simply makes a copy of an input SDD using Getnode, which merges equivalent nodes in *uniqtable*. See [7,12] for details.

3 Space-Bounds for Sequence Binary Decision Diagrams and Acyclic Automata

The structure of SDDs apparently resembles that of acyclic deterministic finite automata (ADFAs). There is a straightforward translation from an ADFA to an SDD and the other way around. However, we should note subtle differences between those formalisms. Actually SDD can be even more compact. This section discusses their relationship in detail.

3.1 Finite Automata

We presume a basic knowledge of the automaton theory. For a comprehensive introduction to the automaton theory, see [10] for example. A (partial) DFA is represented by a tuple $A = \langle \Sigma, \Gamma, \delta, q_0, F \rangle$, where Σ is the input alphabet, Γ is the state set, δ is the partial transition function from $\Gamma \times \Sigma$ to Γ, $q_0 \in \Gamma$ is the initial symbol and $F \subseteq \Gamma$ is the set of acceptance states. The partial function δ can be regarded as a subset $\delta \subseteq \Gamma \times \Sigma \times \Gamma$. We define the *size* of a DFA A, denoted by $|A|$, as the number of labeled edges in A, i.e., $|A| = |\delta|$.

The set of strings that lead the automaton A from a state q to an accept state is denoted by $L_A(q)$. The language $L(A)$ accepted by A is $L_A(q_0)$. A minimum DFA has no state q such that $L_A(q) = \emptyset$ and no distinct states q' and q'' such that $L_A(q') = L_A(q'')$. Since we are concerned with finite sets of strings, all DFAs discussed in this section are *acyclic* (ADFA). We say that A and B, which can be an ADFA or an SDD, are *equivalent* if $L(A) = L(B)$.

3.2 From ADFAs to SDDs

We first give a straightforward translation from an ADFA to an equivalent SDD, which may be non-reduced, and compare the sizes of them.

Theorem 3. *For any ADFA $A = \langle \Sigma, \Gamma, \delta, q_0, F \rangle$, there is an equivalent SDD $B = \langle \Sigma, V, \tau, \mathbf{0}, \mathbf{1}, \boldsymbol{r} \rangle$ such that $|V_{\mathrm{N}}| \le |\delta|$. Moreover, for every positive integer $n \ge 1$, there is an ADFA A that admits no equivalent SDD B such that $|V_{\mathrm{N}}| < |\delta| = n$.*

Proof. For an ADFA $A = \langle \Sigma, \Gamma, \delta, q_0, F \rangle$, we construct an equivalent SDD $\mathbf{B}(A)$. Let

$$\deg(q) = |\{\, a \in \Sigma \mid \delta(q, a) \text{ is defined} \,\}|\,.$$

We define $\mathbf{B}(A) = \langle \Sigma, V, \tau, \mathbf{0}, \mathbf{1}, \mathbf{r} \rangle$ as follows. The set of nodes is given by

$$V = \{\mathbf{0}, \mathbf{1}\} \cup \{\, [q, i] \mid q \in \Gamma \text{ and } 1 \le i \le \deg(q) \,\}.$$

For each $q \in \Gamma$ with $\deg(q) = k \ge 1$, let $a_1, \dots, a_k \in \Sigma$ and $q_1, \dots, q_k \in \Gamma$ be such that

- $\delta(q, a_i) = q_i$ for $i = 1, \dots, k$,
- $a_1 \prec a_2 \prec \cdots \prec a_k$.

Define τ by

$$\tau([q,i]) = \begin{cases} \langle a_i, [q, i+1], \widehat{q_i} \rangle & \text{if } i < k, \\ \langle a_k, \mathbf{1}, \widehat{q_k} \rangle & \text{if } i = k \text{ and } q \in F, \\ \langle a_k, \mathbf{0}, \widehat{q_k} \rangle & \text{if } i = k \text{ and } q \notin F, \end{cases}$$

where

$$\widehat{q'} = \begin{cases} [q', 1] & \text{if } \deg(q') > 0, \\ \mathbf{1} & \text{if } \deg(q') = 0 \text{ and } q' \in F, \\ \mathbf{0} & \text{if } \deg(q') = 0 \text{ and } q' \notin F. \end{cases}$$

The root $\mathbf{r}$ of $\mathbf{B}(A)$ is $\widehat{q_0}$.

It is easy to see that $L_A(q) = L_{\mathbf{B}(A)}(\hat{q})$ for all $q \in \mathit{\Gamma}$. We note that the above construction can be done in linear time in $|\delta|$.

The first claim of the theorem can be verified by the above construction of $\mathbf{B}(A)$. The second claim is established by observing the minimum ADFA and the reduced SDD that accept the singleton $\{a^n\}$ for each positive integer n. For the detail of construction of the reduced (canonical) SDD from a string set, consult [7]. □

We remark that $\mathbf{B}(A)$ in the proof is not necessarily reduced for a minimum ADFA A.

Example 4. Let us compare the minimum ADFA A and the constructed SDD $\mathbf{B}(A)$ for the set $\{ab, b\}$ with $a \prec b$:

transition rules of A	corresponding nodes of $\mathbf{B}(A)$
$\delta(q_0, a) = q_1$	$\tau([q_0, 1]) = \langle a, [q_0, 2], [q_1, 1] \rangle$
$\delta(q_0, b) = q_2$	$\tau([q_0, 2]) = \langle b, \mathbf{0}, \mathbf{1} \rangle$
$\delta(q_1, b) = q_2$	$\tau([q_1, 1]) = \langle b, \mathbf{0}, \mathbf{1} \rangle$

$\mathbf{B}(A)$ is not reduced since $\tau([q_0, 2]) = \tau([q_1, 1])$ for $[q_0, 2] \neq [q_1, 1]$.

In this example, A has two distinct edges that are labeled with b and come into q_2, which should be merged into the same node in a reduced SDD. Hence the reduced SDD can be more compact than the minimum ADFA for the same language. We next discuss how much an SDD can be smaller than an ADFA through a translation from an SDD into an ADFA.

3.3 From SDDs to ADFAs

We next discuss how much an SDD can be smaller than an ADFA through a translation from an SDD into an ADFA. Let an SDD $B = \langle \mathit{\Sigma}, V, \tau, \mathbf{0}, \mathbf{1}, \mathbf{r} \rangle$ be given. We construct an ADFA $\mathbf{A}(B) = \langle \mathit{\Sigma}, \mathit{\Gamma}, \delta, q_0, F \rangle$ such that $L(\mathbf{A}(B)) = L(B)$. We assume that $\mathbf{r} \neq \mathbf{0}$. Otherwise, the translation is trivial.

For each $P \in V_{\mathrm{N}}$, let $\widetilde{P} = [P_1, \ldots, P_k]$ be such that $P_1 = P$ and $\tau(P_i) = \langle a_i, P_{i+1}, R_i \rangle$ for some $R_i \in V$ for $i \leq k$ and $P_{k+1} \in \{\mathbf{0}, \mathbf{1}\}$. We define $\tilde{\mathbf{1}}$ to be $[\mathbf{1}]$. Let

- $\mathit{\Gamma} = \{\widetilde{\mathbf{r}}\} \cup \{\, \widetilde{P_1} \mid P_1 \neq \mathbf{0} \text{ is the 1-child of some } P \in V_{\mathrm{N}} \,\}$,
- $q_0 = \widetilde{\mathbf{r}}$,
- $F = \{\, \widetilde{P} \in \mathit{\Gamma} \mid \widetilde{P} = [P_1, \ldots, P_k] \text{ and } P_k = \mathbf{1} \,\}$,
- $\delta(\widetilde{P}, a_i) = \widetilde{R_i}$ if $\widetilde{P} = [P_1, \ldots, P_k]$ and $\tau(P_i) = \langle a_i, P_{i+1}, R_i \rangle$.

It is easy to see that $L_B(P) = L_{\mathbf{A}(B)}(\widetilde{P})$ for all $\widetilde{P} \in \Gamma$. This implies that if B is reduced, $\mathbf{A}(B)$ is minimum. Contrary to the translation from an ADFA into an equivalent SDD, this construction takes $O(|\Sigma||V_{\mathrm{N}}|)$ time. In fact, this is optimal. The following theorem implies that the reduced SDD can be about $|\Sigma|$ times more compact than the minimum ADFA for the same set of strings.

Theorem 5. *For any SDD $B = \langle \Sigma, V, \tau, \boldsymbol{r}, \mathbf{0}, \mathbf{1} \rangle$, one can construct in $O(|\Sigma||V_{\mathrm{N}}|)$ time the equivalent minimum ADFA $A = \langle \Sigma, \Gamma, \delta, q_0, F \rangle$ such that $|\Gamma| \leq |V_{\mathrm{N}}| + 1$ and*

$$|\delta| \leq \begin{cases} |V_{\mathrm{N}}|(|V_{\mathrm{N}}| + 1)/2 & \text{if } |V_{\mathrm{N}}| \leq |\Sigma|; \\ |\Sigma|(2|V_{\mathrm{N}}| - |\Sigma| + 1)/2 & \text{if } |V_{\mathrm{N}}| > |\Sigma|. \end{cases}$$

Moreover, there is an SDD B that admits no equivalent ADFA A for which the strict inequality holds.

Proof. The first claim, $|\Gamma| \leq |V_{\mathrm{N}}| + 1$, clearly holds by the conversion.

In order to establish the second part of the theorem, we give a variant of the construction of $\mathbf{A}(B)$. We define $\mathbf{C}(B)$ from B by replacing the definition of Γ in $\mathbf{A}(B)$ with $\Gamma = \{ \widetilde{P} \mid P \in V - \{\mathbf{0}\} \}$. For $\mathbf{C}(B) = \langle \Sigma, \Gamma, \delta, q_0, F \rangle$, we prove the inequality by induction on $|V_{\mathrm{N}}|$. Clearly $\mathbf{A}(B)$ is not bigger than $\mathbf{C}(B)$, and thus this claim implies the theorem. In the following discussion, we ignore the root of B and the initial state of $\mathbf{C}(B)$, because it does not affect the discussion of their description size. For $|V_{\mathrm{N}}| = 1$, it is easy to see that the claim holds. Suppose that $|V_{\mathrm{N}}| > 1$. Let B' be obtained from B by deleting an arbitrary nonterminal node P that has no incoming edge.

If $|V_{\mathrm{N}}| \leq |\Sigma|$, we have $|\delta'| \leq (|V_{\mathrm{N}}| - 1)|V_{\mathrm{N}}|/2$ by the induction hypothesis, where δ' denotes the transition set of $\mathbf{C}(B')$. By definition, $\mathbf{C}(B)$ can be obtained from $\mathbf{C}(B')$ by adding one state $\widetilde{P}$ and at most $|V_{\mathrm{N}}|$ outgoing edges from it. Hence

$$|\delta| \leq |\delta'| + |V_{\mathrm{N}}| \leq (|V_{\mathrm{N}}| - 1)|V_{\mathrm{N}}|/2 + |V_{\mathrm{N}}| = |V_{\mathrm{N}}|(|V_{\mathrm{N}}| + 1)/2.$$

If $|V_{\mathrm{N}}| > |\Sigma|$, we have $|\delta'| \leq |\Sigma|(2|V_{\mathrm{N}}| - |\Sigma| - 1)/2$ by the induction hypothesis. By definition, $\mathbf{C}(B)$ can be obtained from $\mathbf{C}(B')$ by adding one state $\widetilde{P}$ and at most $|\Sigma|$ outgoing edges from it. Hence

$$|\delta| \leq |\delta'| + |\Sigma| \leq |\Sigma|(2|V_{\mathrm{N}}| - |\Sigma| - 1)/2 + |\Sigma| = |\Sigma|(2|V_{\mathrm{N}}| - |\Sigma| + 1)/2.$$

We have proven the inequality.

In order to see that the above bound is tight, consider the reduced SDD and the minimum ADFA for the language $L_n = \{ a_0^k a_{i_1} \dots a_{i_j} \mid 0 \leq k \leq n - |\Sigma|, 0 \leq j \leq \min\{m, n\}, 1 \leq i_1 < \dots < i_j \leq m \}$ over $\Sigma = \{a_0, \dots, a_m\}$ with $a_0 \prec a_1 \prec \dots \prec a_m$. □

We note that if $a_m \prec \dots \prec a_1 \prec a_0$, we have $|V'_{\mathrm{N}}| = |\delta|$ for the node set V' of the reduced SDD B' for L_n and the transition set δ of the minimum ADFA A for L_n in the proof of Theorem 5. Hence an order on Σ induces a reduced SDD that has asymptotically $|\Sigma|$ times more nodes than the one induced by another order on Σ.

Corollary 6. *For an order π on Σ and a finite language L over Σ, let $\mathbf{B}^\pi(L) = \langle\langle \Sigma, \pi \rangle, V^\pi, \tau^\pi, S, \mathbf{0}, \mathbf{1} \rangle$ be the reduced SDD for L that respects the order π over Σ. For any order π, ρ on Σ, we have $|V^\pi_{\mathrm{N}}| \leq |\Sigma||V^\rho_{\mathrm{N}}|$.*

Proof. Let δ be the transition set of the minimum automaton for L. By Theorem 3, $|V_{\mathrm{N}}^{\pi}| \leq |\delta|$. By Theorem 5, $|\delta| \leq |\Sigma||V_{\mathrm{N}}^{\rho}|$. Hence $|V_{\mathrm{N}}^{\pi}| \leq |\Sigma||V_{\mathrm{N}}^{\rho}|$. □

Through the conversion techniques presented above between ADFAs and SDDs and/or by Theorems 3 and 5, most known results on the size of minimum ADFAs can be translated into those on SDDs. A special case is where the set of all factors of a string is in concern. Let

$$Fact(w) = \{\, y \in \Sigma^* \mid w = xyz \text{ for some } x, z \in \Sigma^* \,\}.$$

The literature has intensively studied the *factor automata* for the set $Fact(w)$.

Theorem 7 (Blumer et al. [2], Crochemore [4]). *For $w \in \Sigma^*$, let Γ and δ be the state set and the transition set of the minimum ADFA for $Fact(w)$. Then $|\Gamma| \leq 2|w| - 2$ and $|\delta| \leq 3|w| - 4$.*

Corollary 8. *For $w \in \Sigma^*$, let V be the node set of the reduced SDD for $Fact(w)$. Then $|w| \leq |V_{\mathrm{N}}| \leq 3|w| - 4$.*

Proof. By Theorem 7 and Theorem 5. □

For $w = cb^n a$ with $a \prec b \prec c$, we have $|V_{\mathrm{N}}| = 3|w| - 4$.

Corollary 9. *For $w \in \Sigma^*$ and order π on Σ, let V^{π} be the node set of the reduced SDD for $Fact(w)$. Then $|V_{\mathrm{N}}^{\pi}| \leq |V_{\mathrm{N}}^{\rho}| + |w| - 1$. Moreover, there are w, π and ρ for which the equality holds.*

Proof. Let δ be the transition set of the minimum automaton for $Fact(w)$. We have $|V_{\mathrm{N}}^{\pi}| \leq |\delta|$ and $|\Gamma| \leq |V_{\mathrm{N}}^{\rho}| + 1$ by Theorems 3 and 5, respectively. Blumer et al. [2, Lemma 1.6] show that $|\delta| \leq |\Gamma| + |w| - 2$. Hence

$$|V_{\mathrm{N}}^{\pi}| \leq |\delta| \leq |\Gamma| + |w| - 2 \leq |V_{\mathrm{N}}^{\rho}| + |w| - 1.$$

In fact for $w = a^n b$, $\pi = \langle b \prec a \rangle$, $\rho = \langle a \prec b \rangle$, we have $|V_{\mathrm{N}}^{\pi}| = 2n - 1$ and $|V_{\mathrm{N}}^{\rho}| = n - 1$. □

4 Input- and Output-Sensitive Time-bounds for Binary Synthesis Operations

In this section, we consider time complexity of set operations on SDDs. In particular, given a binary set operation $\diamond \in \{\cup, \cap, \setminus, \ldots\}$, we consider the *synthesis problem* that receives two reduced SDDs P, Q and computes $R = P \diamond Q$, where $P \diamond Q$ denotes the reduced SDD such that $L(P \diamond Q) = L(P) \diamond L(Q)$. Bryant [3] presented in his seminal paper on BDDs, a recursive synthesis algorithm for all Boolean operations. Loekito *et al.* [12] gave its string-counterpart for union $\cup$ and difference $\setminus$. Below, we generalize the algorithm in [12] for a family of set operations, called *melding*, in the style of Knuth [11].

Global variable: *uniqtable*, *cache*: hash tables for triples and operations.

Algorithm $\mathsf{Meld}_\diamond(P, Q\text{: SDDs})$:
Output: The reduced SDD for the melding $P \diamond Q$ given $F_\diamond : \{0,1\}^2 \to \{0,1\}$;

if $(P = \mathbf{0}$ **or** $Q = \mathbf{0}$ **or** $P = Q)$
  if $(F_\diamond[\mathsf{sign}(P), \mathsf{sign}(Q)] = 0)$ **return** $\mathbf{0}$; /* See text for $F_\diamond$. */
  else if $P \neq \mathbf{0}$ **return** P;
  else if $Q \neq \mathbf{0}$ **return** Q;
else if $((R \leftarrow cache[\text{“}Meld_\diamond(P, Q)\text{”}])$ exists) **return** R;
else
  $x \leftarrow P.lab$; $y \leftarrow Q.lab$;
  if $(x \prec_\Sigma y)$ $R \leftarrow \mathsf{Getnode}(x, \mathsf{Meld}_\diamond(P.0, Q), \mathsf{Meld}_\diamond(P.1, \mathbf{0}))$;
  else if $(x \succ_\Sigma y)$ $R \leftarrow \mathsf{Getnode}(y, \mathsf{Meld}_\diamond(P, Q.0), \mathsf{Meld}_\diamond(\mathbf{0}, Q.1))$;
  else if $(x = y)$ $R \leftarrow \mathsf{Getnode}(x, \mathsf{Meld}_\diamond(P.0, Q.0), \mathsf{Meld}_\diamond(P.1, Q.1))$;
  $cache[\text{“}Meld_\diamond(P, Q)\text{”}] \leftarrow R$;
  return R;

For convenience, we assume $\mathbf{1}.lab$ to be a symbol larger than any symbols in Σ.

Figure 3. An algorithm $\mathsf{Meld}_\diamond$ for built-in binary set operations $\diamond \in \{\cup, \cap, \setminus, \oplus, \ldots\}$.

4.1 The Family of Melding Operations

We give a family of binary set operations $\diamond$ called *melding* below. A *terminal operation table* is a binary Boolean function $F : \{0,1\}^2 \to \{0,1\}$ such that $F[0,0] = 0$. Clearly, there are exactly eight such tables. Let $\mathcal{O} = \{\cup, \cap, \setminus, /, \oplus, \emptyset, LHS, RHS\}$ be a set of names of set operations $\diamond : 2^{\Sigma^*} \times 2^{\Sigma^*} \to 2^{\Sigma^*}$ on subsets Σ^*. We define $F_\diamond$ by: $F_\cup[x,y] = x \vee y$, $F_\cap[x,y] = x \wedge y$, $F_\setminus[x,y] = x \wedge \neg y$, $F_/[x,y] = \neg x \wedge y$, $F_\oplus[x,y] = x \oplus y$ (exclusive-or), $F_\emptyset[x,y] = 0$, $F_{LHS}[x,y] = x$, $F_{RHS}[x,y] = y$, where $x, y \in \{0,1\}$. For any SDD P, we define $\mathsf{sign}(P)$ to be 0 if $P = \mathbf{0}$ and 1 otherwise.

In Fig. 3, we give the algorithm $\mathsf{Meld}_\diamond$ that computes the reduced SDD $R = P \diamond Q$ for two SDDs P and Q given a terminal operation table $F_\diamond$. Clearly, the trivial operations $\emptyset$, LHS and RHS can be computed in constant time without $\mathsf{Meld}_\diamond$. Yet those are also uniformly described as $\mathsf{Meld}_\diamond$. A specified terminal operation table $F_\diamond$ uniquely determines melding operation $P \diamond Q$. In what follows, we assume that the inputs P and Q and the output R are built by using the same hash table *uniqtable*, where *uniqtable* is initialized with the empty relation before constructing P and Q. Moreover, our algorithm uses a hash table $cache : \mathsf{op} \times \mathsf{dom}^2 \to \mathsf{dom}$ that stores invocation patterns of operations for avoiding redundant computation, where op is the set of operation names. By a similar discussion in Knuth [11], we establish the following theorem. $\mathsf{Meld}_\diamond$ directly computes the output without producing redundant nodes.

Theorem 10 (correctness). *Let $\diamond \in \mathcal{O}$ be any of the eight operations. Given $F_\diamond$, the algorithm $\mathsf{Meld}_\diamond$ in Fig. 3 correctly computes the reduced SDD for $R = P \diamond Q$ exactly eight string set operations $P \diamond Q$, where the set operation $P \diamond Q$ is defined as follows:*

the union $P \cup Q$, the intersection $P \cap Q$,
the difference $P \setminus Q$, the inverse difference $P/Q = Q \setminus P$,
the symmetric difference $P \oplus Q = (P \setminus Q) \cup (Q \setminus P)$, the empty set $\emptyset$,
the left hand side $LHS(P,Q) = P$. the right hand side $RHS(P,Q) = Q$.

4.2 Input-Sensitive Complexity of Binary Synthesis

First, we start with input-sensitive analysis of the time complexity for the melding procedure. We prepare some necessary notations. Consider the algorithm $\mathsf{Meld}_\diamond$ of Fig. 3. Let us denote by $\mathsf{Meld}^0_\diamond$ and $\mathsf{Meld}^1_\diamond$ the first and second parts of the algorithm, that is, the top level if-clause and else-clause consisting of Lines 1 to 5 and Lines 6 to 12, respectively. For a procedure α, $\#\alpha(P, Q)$ denotes the number of times that α is executed during the computation of $\mathsf{Meld}_\diamond(P, Q)$. We assume that $|\mathbf{0}| = |\mathbf{1}| = 1$ for convenience.

Theorem 11 (input complexity of melding). *Let $\diamond$ be any melding operation. For reduced SDDs P and Q, the algorithm* $\mathsf{Meld}_\diamond$ *of Fig. 3 computes $R = P \diamond Q$ in $O(|P| \cdot |Q|)$ time and space.*

Proof. Consider the computation of $\mathsf{Meld}_\diamond(P, Q)$. Since the arguments P' and Q' of any subroutine call $\mathsf{Meld}_\diamond(P', Q')$, resp., are subgraphs of P and Q, the number of *distinct* calls for $\mathsf{Meld}_\diamond(P, Q)$ is at most $|P| \cdot |Q|$ (*Claim 1*). It also follows that *cache* has $O(|P| \cdot |Q|)$ entries. Since the table-lookup with *cache* at Line 5 eliminates duplicated calls, the $\mathsf{Meld}^1_\diamond$ can be executed at most once for each (P', Q'), and thus, we have $\#\mathsf{Meld}^1_\diamond \le |P|\cdot|Q|$ (*Claim 2*). We observe that $\mathsf{Meld}_\diamond$ is called either (i) at the top-level or (ii) within $\mathsf{Meld}^1_\diamond$. Since exactly one of Line 8, 9, and 10 is executed in $\mathsf{Meld}^1_\diamond$, which contains at most two calls for $\mathsf{Meld}_\diamond$, we have $\#\mathsf{Meld}_\diamond \le 2{\cdot}\#\mathsf{Meld}^1_\diamond{+}1$ (*Claim 3*). Combining Claims 2 and 3, we have that $\#\mathsf{Meld}_\diamond \le 2 \cdot |P| \cdot |Q| + 1 = O(|P| \cdot |Q|)$. If each call of $\mathsf{Meld}_\diamond$ takes $O(1)$ time, then the time complexity is $O(|P| \cdot |Q|)$. On the other hand, each $\mathsf{Meld}_\diamond(P', Q')$ makes exactly one call for $\mathsf{Getnode}$ by adding a new node. Thus, the algorithm adds at most $|R| \le \#\mathsf{Getnode} \le \#\mathsf{Meld}_\diamond = O(|P| \cdot |Q|)$ nodes. Since the number of *cache*-entries is $O(|P| \cdot |Q|)$ and the function stack has depth no more than $\#\mathsf{Meld}_\diamond$, the space complexity is $O(|P| \cdot |Q|)$. □

From the proof of the above theorem, we have the following corollary.

Corollary 12 *For any melding operation $\diamond \in \mathcal{O}$, the reduced output size $|R|$ is bounded from above by $O(|P| \cdot |Q|)$.*

4.3 Pseudo Output Sensitive Complexity of Binary Synthesis

Next, we present output-sensitive analysis of the time complexity of the melding in the style of Wegener [21], which analyzed the time complexity of Boolean operations for BDDs based on the size of non-reduced BDDs. We define $R^* = P \diamond_* Q$ to be the (possibly non-reduced) SDD computed by $\mathsf{Meld}_\diamond$ equipped with the modification of $\mathsf{Getnode}$ in Fig. 3 by removing Line 1 and 2 for node-sharing and zero-suppress rules. Clearly, the non-reduced output size $|R^*|$ is bounded from above by $O(|P| \cdot |Q|)$.

Theorem 13 (output-sensitive complexity w.r.t. non-reduced output). *The reduced SDD for $R = P \diamond_* Q$ can be computed in $O(|R^*|)$ time and space by the algorithm* $\mathsf{Meld}_\diamond$ *in Fig. 3, where R^* is the non-reduced SDD for $P \diamond_* Q$.*

Proof. Consider the computation of $\mathsf{Meld}_\diamond$ of Fig. 3 equipped with $\mathsf{Getnode}^*$. Since each call of $\mathsf{Getnode}^*$ increases the output size by at least one, we have $\#\mathsf{Getnode}^* \le |R^*|$ (*Claim 4*). Since exactly one of Line 8, 9, and 10 is executed in $\mathsf{Meld}^1_\diamond$ and it contains at least one call for $\mathsf{Getnode}$, we have $\#\mathsf{Meld}^1_\diamond \le \#\mathsf{Getnode}^*$ (*Claim 5*). From the proof for Theorem 11, we have $\#\mathsf{Meld}_\diamond \leq 2 \cdot \#\mathsf{Meld}^1_\diamond + 1$ (*Claim 3*). Combining Claims 3, 4, and 5 above, we now have $\#\mathsf{Meld}_\diamond \leq 2{\cdot}\#\mathsf{Meld}^1_\diamond + 1 \leq 2{\cdot}\#\mathsf{Getnode}^* + 1 \leq 2{\cdot}|R^*| + 1 = O(|R^*|)$., and thus, we have the time complexity $O(|R^*|)$. Since *uniqtable* and *cache* contain at most $\#\mathsf{Getnode}^*$ and $\#\mathsf{Meld}_\diamond$ entries, resp., the space complexity follows from a similar argument to the proof for Theorem 11. □

4.4 A Lower Bound for the Time Complexity of Binary Synthesis

In the BDD community, there has been a strong belief that the quadratic input-sensitive complexities of the binary synthesis procedures for a number of variants of BDDs, including the BDDs and ZDDs, is output-linear time for most input instances, and there has been no super-linear lower bound for its time complexity. Recently, Yoshinaka et al. [22] show that this conjecture is not true for BDDs and ZDDs; They constructed an infinite sequence of input BDDs that demonstrated the quadratic lower bound for the time complexity of the melding for BDDs and ZDDs. Based on their discussion, below we show that the above quadratic input-sensitive complexity of the melding in terms of input size is optimal for SDDs in reality.

Theorem 14. *Let $\diamond$ be any melding operations. The algorithm* $\mathsf{Meld}_\diamond$ *of Fig. 3 requires $\Omega(|P| \cdot |Q|)$ time and space regardless of the output size, where P and Q are the input SDDs.*

Proof. Our example that the binary synthesis takes $O(|P| \cdot |Q|)$ time to compute $R = P \diamond Q$ where $|R|$ is linear in $|P| + |Q|$ is just a straightforward translation of the one from [22]. The theorem can be shown in a way similar to [22]. Here we give a rough sketch of the proof. Let $\Sigma = \{0, 1\}$. For a fixed positive integer n, we define

$$S = \{\, x_1y_1 \cdots x_ny_nz_1 \cdots z_m \in \{0,1\}^{2n+m} \mid x_{\beta(z_1 \cdots z_m)} = 1 \,\},$$
$$T = \{\, x_1y_1 \cdots x_ny_nz_1 \cdots z_m \in \{0,1\}^{2n+m} \mid y_{\beta(z_1 \cdots z_m)} = 1 \,\},$$

where $m = \lceil \log n \rceil$ and

$$\beta(z_1 \cdots z_m) = \begin{cases} 1 + \sum_{k=1}^{m} 2^{k-1} z_k & \text{if } \sum_{k=1}^{m} 2^{k-1} z_k < n; \\ 1 & \text{otherwise.} \end{cases}$$

We have

$$S \diamond T = \{\, x_1y_1 \cdots x_ny_nz_1 \cdots z_m \in \{0,1\}^{2n+m} \mid F_\diamond[x_{\beta(z_1 \cdots z_m)}, y_{\beta(z_1 \cdots z_m)}] = 1 \,\}.$$

Let P and Q be the reduced SDD for S and T, resp.

We first show that $|P|, |Q|, |R| = O(2^n)$. It is easy to see that every node in P and Q represents a set of strings of a fixed length, since all strings in S and T have the same length $2n + m$. We define the *level* of a node to be $2n + m - k$ if the node represents a set of strings of length k. Since the membership of $x_1y_1 \cdots x_ny_nz_1 \cdots z_m$ to S does not depend on any of y_i, it is not hard to see that there are at most $O(2^k)$ nodes of level $2k$ for $0 \leq k < n$. The number of nodes of level $2k + 1$ is at most twice as big as that of level $2k$. On the other hand, since there are at most 2^{2^k} distinct sets of strings of length k, there are at most $|\Sigma| \cdot 2^{2^k}$ nodes of level $2n + m - k$ for $0 \leq k \leq m = \lceil \log n \rceil$. All in all, $|P| = O(2^n)$. Similarly $|Q| = O(2^n)$. It is easy to see that for any $x_i, y_i, x'_i, y'_i \in \{0, 1\}$ such that $F_\diamond[x_i, y_i] = F_\diamond[x'_i, y'_i]$, we have

Data	Size (byte)	#line	#unique line	Ave. line len (byte)	|Σ|
BibleAll	4,047,392	30,383	30,129	133.2	62
BibleBi	7,793,268	767,854	154,479	10.1	27
Ecoli	4,638,690	1	1	4,638,690.0	4

Table 1. Outline of data sets

Data	Size (Kilo node)						Time (sec)			
SDD input	H1	H2	∪	∩	∖	∕	∪	∩	∖	∕
BibleAll(Fac)	3099	3082	6110	417	3415	3388	0.67	0.44	0.59	0.58
BibleBi	101	115	167	36	82	97	0.06	0.00	0.00	0.00
Ecoli(Fac)	4973	4970	9938	654	6346	6347	1.63	1.09	1.42	1.41

Table 2. Output size and running time of algorithms for binary synthesis

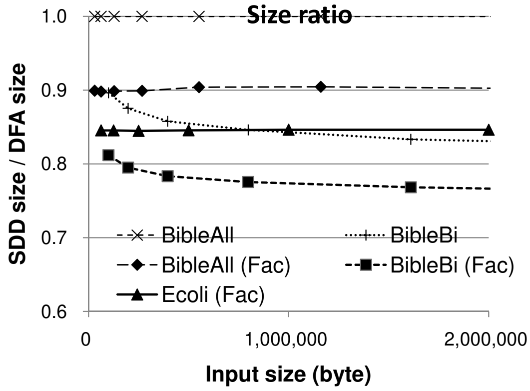

Figure 4. Ratio between the sizes SDDs and DFAs in binary format

$w_1x_iy_iw_2 \in S \diamond T$ iff $w_1x'_iy'_iw_2 \in S \diamond T$ for any $w_1 \in \{0,1\}^{2k}, w_2 \in \{0,1\}^{2n+m-k-2}$ with $k < n$. Hence we have $|R| = O(2^n)$ by a discussion similar to the one for $|P|, |Q| = O(2^n)$.

Second we show that $\#\mathsf{Meld}_\diamond \geq 2^{2n}$. For $w \in \{0,1\}^{2n}$, let P_w denote the node of P such that $L(P_w) = \{\, w' \mid ww' \in S \,\}$. In fact P has such a node for each w. Similarly we let Q_w be such that $L(Q_w) = \{\, w' \mid ww' \in T \,\}$. By definition, the algorithm calls $\mathsf{Meld}_\diamond(P_w, Q_w)$ for each w. Moreover, $P_{x_1 \cdots y_n} \neq P_{x'_1 \cdots y'_n}$ whenever $x_i \neq x'_i$ for some i and $Q_{x_1 \cdots y_n} \neq Q_{x'_1 \cdots y'_n}$ whenever $y_i \neq y'_i$ for some i. Therefore, for distinct $w, w' \in \{0,1\}^{2n}$, the pairs $\langle P_w, Q_w \rangle$ and $\langle P_{w'}, Q_{w'} \rangle$ are distinct. This means that $\#\mathsf{Meld}_\diamond \geq 2^{2n}$. □

5 Experiments

This section presents our experimental results on SDDs. Our first experiment has constructed SDDs and DFAs for the same sets of strings of real data and compared their sizes. Secondly we have implemented the binary synthesis algorithm $\mathsf{Meld}_\diamond$ and computed different binary operations on sets over SDDs.

Setting. The data sets used in our experiments are summarized in Table 1. BibleAll and BibleBi are sets of all sentences and all word bi-grams drawn from an English text `bible.txt` and Ecoli is a single DNA string in `ecoli.txt` in Canterbury corpus[3]. We implemented our shared and reduced SDD environment on the top of

[3] `http://corpus.canterbury.ac.nz/resources/`

the *SAPPORO BDD package* [17] for BDDs and ZDDs written in C and C++, where each node is encoded in a 32-bit integer and a node triple occupies approximately 30 bytes in average including hash entries in *uniqtable*. We also used another implementation of SDD environment in functional language *Erlang*. Experiments were run on a PC (Intel Core i7, 2.67 GHz, 3.25 GB memory, Windows XP SP3). About 1.5 GB of memory was allocated to the SDD environment in maximum.

Exp 1: Comparison of the size of indexes. Figure 4 shows the sizes of SDDs and DFAs for different sets of strings, where BibleAll (Fac), BibleBi (Fac) and Ecoli (Fac) mean the sets of all factors of sequences in the respective input files. We see that a minimal SDD is 0 to 23 percent more succinct than the equivalent minimal ADFA in binary format. In particular, the size ratio for factor sets is even smaller than that for the original string data. SDDs can search strings as fast as DFAs where edges of DFAs are represented by linked list.

Exp 2: Binary synthesis. We divided the source texts into two parts, the first half H1 and the second H2, and then performed $\mathsf{Meld}_\diamond$ on those parts for $\diamond \in \{\cup, \cap, \setminus, /\}$. The results are presented in Table 2. It took less than seconds to compute set operations $\diamond$ on two SDDs with around three to four millions of nodes each. The output size of $\mathsf{H1} \cup \mathsf{H2}$ is much larger than that of $\mathsf{H1} \cap \mathsf{H2}$, but the running time is not different that much.

Overall, we conclude that the shared and reduced SDD environment with the above algorithms is a practical choice for storing and manipulating string sets in large-scale string applications.

6 Conclusion

In this paper, we consider the class of sequence binary decision diagrams (SDDs) proposed by Loekito *et al.* [12], and studied two fundamental problems on sequence binary decision diagrams: the relationship to acyclic automata and the complexities of the binary synthesis operation. In Sec. 4, we showed the quadratic time complexity of the $\mathsf{Meld}_\diamond$ algorithm. In [7], it is shown that the $\mathsf{Meld}_\diamond$ runs in input linear time if one of the argument is the minimal SDD of linear shape corresponding to a string. Therefore, it would be an interesting future problem to study special cases that $\mathsf{Meld}_\diamond$ has input linear time complexity. It would be another problem to apply SDDs for studying the dynamic versions of sequence analysis problems such as the maximal repeat problem and the consistent string problem [9].

Acknowledgements

The authors are grateful to anonymous referees for many useful comments and suggestions that improve the quality of this paper. They would like to thank Takashi Horiyama, Takeru Inoue, Jun Kawahara, Takuya Kida, Toshiki Saitoh, Yasuyuki Shirai, Kana Shimizu, Yasuo Tabei, Koji Tsuda, Takeaki Uno, and Thomas Zeugmann for their discussions and valuable comments. This research was partly supported by MEXT Grant-in-Aid for Scientific Research (A), 20240014, FY2008–2011, MEXT/ JSPS Global COE Program, FY2007–2011, and ERATO MINATO Discrete Structure Manipulation System Project, JST.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman: *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
2. A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M. T. Chen, and J. I. Seiferas: *The smallest automaton recognizing the subwords of a text. Theor. Comput. Sci.*, 40, 31–55, 1985.
3. R. E. Bryant: *Graph-based algorithms for boolean function manipulation. IEEE. Trans. Comput.*, C-35(8), 677–691, 1986.
4. M. Crochemore: *Transducers and repetitions. Theor. Comput. Sci.*, 45(1), 63–86, 1986.
5. M. Crochemore, C. Hancart, and T. Lecroq: *Algorithms on Strings*. Cambridge, 2007.
6. J. Daciuk, S. Mihov, B. W. Watson, R. E. Watson: *Incremental construction of minimal acyclic finite-state automata. Computational Linguistics*, 26(1), 2000.
7. S. Denzumi, R. Yoshinaka, S. Minato, and H. Arimura: *Efficient algorithms on sequence binary decision diagrams for manipulating sets of strings. Technical Report*, DCS, Hokkaido U., TCS-TR-A-11-53, April 2011. (submitting)
8. R. Giegerich, S. Kurtz, and J. Stoye: *Efficient implementation of lazy suffix trees. Software Practice and Experience*, 33, 1035–1049, 2003.
9. D. Gusfield: *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.
10. J. E. Hopcroft, R. Motwani, and J. D. Ullman: *Introduction to Formal Language Theory, 2nd edition*. Addison-Wesley, 2001.
11. D. E. Knuth: *The Art of Computer Programming, vo.4, Fascicle 1, Bitwise Tricks & Techniques; Binary Decision Diagrams*. Addison-Wesley, 2009.
12. E. Loekito, J. Bailey, and J. Pei: *A Binary decision diagram based approach for mining frequent subsequences. Knowl. Inf. Syst.*, 24(2), 235-268, 2009.
13. C. L. Lucchesi, and T. Kowaltowski: *Applications of finite automata representing large vocabularies. Software Practice and Experience*, 23(1), 15–30, 1993.
14. U. Manber and E. W. Myers: *Suffix arrays: a new method for on-line string searches. SIAM J. Comput.*, 22(5), 935–948, 1993.
15. E. M. McCreight: *A space-economical suffix tree construction algorithm. J. ACM*, 23, 262–272, 1976.
16. S. Minato: *Zero-suppressed BDDs and their applications. International Journal on Software Tools for Technology Transfer*, 3(2), 156-170, Springer, 2001.
17. S. Minato: *SAPPORO BDD package*. DCS, Hokkaido University, unreleased, 2011.
18. M. Mohri: *On some applications of finite-state automata theory to natural language processing. Natural Language Engineering*, 2(1), 61–80, 1996.
19. M. Mohri, P. Moreno, and E. Weinstein: *Factor automata of automata and applications. Proc. CIAA 2007*, 168-179, 2007.
20. D. Perrin: *Finite Automata. Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, J. van Leeuwen (Eds.), Elsevier and MIT Press, 1–57, 1990.
21. I. Wegener: *Branching Programs and Binary Decision Diagrams: Theory and Applications*. SIAM, 2000.
22. R. Yoshinaka J. Kawahara, S. Denzumi, H. Arimura, and S. Minato: *Counter examples to the long-standing conjecture on the complexity of BDD binary operations. Technical Report*, DCS, Hokkaido U., TCS-TR-A-11-52, April 2011. (submitting to international journal)