New Algorithms for Manipulating Sequence BDDs

Shuhei Denzumi(✉)

The University of Tokyo, Hongo 7-3-1, Bunkyo, Tokyo 113-8656, Japan
denzumi@mist.i.u-tokyo.ac.jp

Abstract. Sequence binary decision diagram (SeqBDD) is a data structure to represent and manipulate sets of strings. This is a variant of zero-suppressed binary decision diagram (ZDD) that manipulates combinatorial sets. Nowadays, binary decision diagrams (BDDs) and its family have been recognized as an important data structure to manipulate discrete structures. SeqBDD has some set manipulation operations inherited from ZDD, but the number of the operations is not enough to deal with a wide variety of requests in string processing area. In this paper, we propose 50 new algorithms for manipulating SeqBDDs. We divide the operations into three categories and list up them. We also analyzed the time and space complexities of some new algorithms.

Keywords: Manipulation algorithm · Operation · Sequence binary decision diagram · Data structure · Complexity

1 Introduction

Constructing indices that store sets of strings in compact space is a fundamental problem in computer science, and have been extensively studied in the decades [4,8–10,12,19]. Examples of compact string indices include: tries [1,9], finite automata and transducers [10,13]. By the rapid increase of massive amounts of sequential data such as biological sequences, natural language texts, and sensing data stream, these compact string indices have attracted much attention and gained more importance in many string processing applications [9,12]. In such applications, an index not only has to compactly store sets of strings for *searching*, but also has to efficiently manipulate them with various set operations. For example, the most basic operations are *union, intersection*, *difference*, and *concatenation*. *Minimal acyclic deterministic finite automata* (*minimal ADFAs*) [9,10,13] are one of such index structures that fulfill the above requirements based on finite automata theory, and have been used in many sequence processing applications [15,19]. However, the algorithms to manipulate them is complicated because of the multiple branching of the underlying directed acyclic graph structure.

To overcome this problem, Loekito *et al.* [14] proposed *sequence binary decision diagrams* (*SeqBDDs*), which is a compact representation of finite sets of

M. Hospodár and G. Jirásková (Eds.): CIAA 2019, LNCS 11601, pp. 108–120, 2019.
https://doi.org/10.1007/978-3-030-23679-3_9

strings along with algorithms for manipulation operations. A SeqBDD is a vertex-labeled graph structure, which resembles an acyclic DFA in binary form (left-child, right-sibling representation [6]) with associated minimization rules for sharing siblings as well as children that are different from ones for a minimal ADFA. Due to these minimization rules, a SeqBDD can be more compact than an equivalent ADFA [11]. Novel features of the SeqBDDs are their abilities to share equivalent subgraphs and reuse results of intermediate computation between different multiple SeqBDDs. These characteristics allow us to avoid redundant generation of vertices and computation. In 2014, SeqDD, a variant of SeqBDD, was proposed by Alhakami, Ciardo and Chrobak [2]. However, they did not propose manipulating algorithms.

SeqBDD is a member of decision diagram family. Binary decision diagram (*BDD*) [5] is proposed by Bryant to manipulate Boolean functions. There are some studies about relationships between BDDs and Automata [7,17]. The most fundamental operations for string sets, such as union, intersection, and difference, are implemented by the almost same algorithms on *zero-suppressed BDD* (*ZDD*) [18] which is a variant of BDD and manipulates sets of combinations. ZDD has much more operations to manipulate sets of combinations. Since SeqBDD can be said as a child of ZDD, it inherits some operations from ZDD. However, it is not enough to manipulate sets of combinations because we can define much more operations for string sets than sets of combinations due to the differences between combinations and strings. SeqBDD did not have even fundamental operations such as concatenation. Size of a combination is bounded by the size of the universal set, but length of a string is not bounded by the size of the alphabet. A combination does not have order between its elements, but a string has order between its symbols. For example, a combination $\{a, b, c\}$ equals to $\{b, c, a\}$, $\{c, b, a\}$, and $\{a, b, c, b, a\}$, but a string abc is not equal to bca, cba, and $abcba$. In addition, we can distinguish substrings such as prefixes, suffixes, substrings, and subsequences even though they are the same as string. In this paper, we propose 50 new operations on SeqBDD. Almost all algorithms can be implemented as simple recursive algorithms. The collection of manipulation operations will be useful to implement various string applications on the top of SeqBDDs. The organization of this paper is as follows. In Sect. 2, we introduce our notation and data structures, operations, and techniques used throughout this paper. In Sect. 3, we propose new operations and analyze their complexities.

2 Preliminary

Let $\Sigma = \{a, b, \ldots\}$ be a countable alphabet of *symbols*. We assume that the symbols of Σ are ordered by a precedence $\prec_\Sigma$ such as $a \prec_\Sigma b \prec_\Sigma \cdots$ in a standard way. Let $s = a_1 \cdots a_n$, $n \geq 0$, be a *string* over Σ. For every $i = 1, \ldots, n$, we denote by $\alpha[i] = a_i$ the i-th symbol of α. We denote by $|\alpha| = n$ the *length* of α. The *empty string*, a string of length zero, is denoted by ε. We denote by Σ^* the set of all strings of length $n \geq 0$. For two strings α and β, we denote the *concatenation* of α and β by $\alpha \cdot \beta$ or $\alpha\beta$. If $\zeta = \alpha\beta\gamma$ for some possibly empty

Attribute	Terminal	Nonterminal
zero	null	$zero(v)$
one	null	$one(v)$
label	$\top$	$label(v)$
val	$value(v)$	null

Fig. 1. The attribute values for a vertex v.

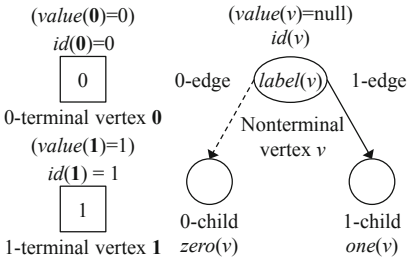

Fig. 2. The 0-terminal, 1-terminal and nonterminal vertices.

strings α, β, and γ, we refer to α, β, and γ as a *prefix*, *factor*, and *suffix* of ζ, respectively. For a string ζ of length n and $1 \leq i_1 < i_2 < \cdots < i_k \leq n$, we refer to $\zeta[i_1]\zeta[i_2]\cdots\zeta[i_k]$ as a *subsequence* of ζ. A *reverse* of ζ is $\zeta^R = \zeta[|\zeta|]\cdots\zeta[1]$.

A *language* on an alphabet Σ is a set $L \subseteq \Sigma^*$ of strings on Σ. A *finite language* of size $m \geq 0$ is just a finite set $L = \{\alpha_1, \ldots, \alpha_m\}$ of m strings on Σ. A finite language L is referred to as a *string set*. We define the *cardinality* of L by $|L| = m$, the *total length* of L by $||L|| = \sum_{\alpha \in L} |\alpha|$, and the *maximal string length* of L by $\mathsf{maxlen}(L) = \max\{\, |\alpha| \mid \alpha \in L \,\}$. The *empty language* of cardinality 0 is denoted by $\emptyset$. For languages $L, M \subseteq \Sigma^*$, we define the following binary operations, called *Boolean set operations*: the *union* $L \cup M$, the *intersection* $L \cap M$, the *difference* $L \backslash M$, the *symmetric difference* $L \oplus M = (L \backslash M) \cup (M \backslash L)$, the *concatenation* $L \cdot M = \{\, \alpha\beta \mid \alpha \in L, \beta \in M \,\}$ as usual.

2.1 Sequence Binary Decision Diagrams

In this subsection, we give the SeqBDD, introduced by Loekito *et al.* [14], as our graphical representation of a finite language. Then, we show its canonical form. A vertex v in a SeqBDD is represented by a structure with the attributes *id*, *label*, *zero*, *one*, and *value*. We have two types of vertices, called *nonterminal* and *terminal vertices*, both of which are represented by the same type of struct, but the attribute values for a vertex v depend on its vertex type, as given in Fig. 1. A graphical explanation of the correspondence between the attribute values and the vertex type is given in Fig. 2.

Definition 1 (Sequence BDD) [14]. *A* sequence binary decision diagram *(a SeqBDD) is a multi-rooted, vertex-labeled, directed graph $G = (V, E)$ with $R \subseteq V$ satisfying the following:*

- *V is a vertex set containing two types of vertices known as terminal and nonterminal vertices. Each has certain attributes, id, label, zero, one, and value. The respective attributes are shown in Fig. 1.*
- *There are two types of terminal vertices, called* 1-terminal *and* 0-terminal *vertices, respectively. A SeqBDD may have at most one 0-terminal and at most one 1-terminal: (1) A* terminal vertex *v has as an attribute value$(v) \in$*

$\{0,1\}$, *indicating whether it is a 1-terminal or a 0-terminal, denoted by* **1** *or* **0**, *respectively. v has an attribute $label(v) = \top$ the special* null symbol $\top \notin \Sigma$, *which is larger than any symbol in Σ, i.e., $c \prec_\Sigma \top$ for any $c \in \Sigma$. The equality $=_\Sigma$ and the strict total order $\prec_\Sigma$ are defined on $\Sigma \cup \{\top\}$; (2) A* nonterminal vertex v *has as attributes a symbol $label(v) \in \Sigma$ called the* label, *and two children, $one(v)$ and $zero(v) \in V$, called the 1-child and 0-child. We refer to the pair of corresponding outgoing edges as the 1-edge and 0-edge from v. We define the* attribute triple *for v by $triple(v) = \langle label(v), zero(v), one(v)\rangle$. For distinct vertices u and v, $id(u) \neq id(v)$ holds.*

- *We assume that the graph is* acyclic *in its 1- and 0-edges. That is, there exists some partial order $\prec_V$ on vertices of V such that $v \prec_V zero(v)$ and $v \prec_V one(v)$ for any nonterminal v.*
- *Furthermore, we assume that the graph must be* ordered *in its 0-edges, that is, for any nonterminal vertex v, if $zero(v)$ is also nonterminal, we must have $label(v) \prec_\Sigma label(zero(v))$, where $\prec_\Sigma$ is the strict total order on symbols of $\Sigma \cup \{\top\}$. The graph is not necessarily ordered in its 1-edges.*
- *R is a set of roots. All vertices in V are reachable from at least one vertex in R.*

For any vertex v in a SeqBDD G, the *subgraph rooted by v* is defined as the graph consisting of v and all its descendants. A SeqBDD is called *single-rooted* if it has exactly one root, and *multi-rooted* otherwise. We define the *size* of the graph rooted by a vertex v, denoted by $|v|$, as the number of its nonterminals reachable from v. By definition, the graph consisting of a single terminal vertex, **0** or **1**, is a SeqBDD of size zero. A graph G is called *single-rooted*. In this paper, we identify a single-rooted SeqBDD and its root vertex name. Multi-rooted graphs are useful in the shared SeqBDD environment described in Subsect. 2.2.

Now, we give the semantics of a SeqBDD.

Definition 2 (The Language Represented by a Single-Rooted Seq BDD). *In a single-rooted SeqBDD G, a vertex v in G denotes a finite language $L_G(v)$ on Σ defined recursively as:*

1. *If v is a terminal vertex, $L_G(v)$ is the trivial language defined as: (i) if $value(v) = 1$, $L_G(v) = \{\varepsilon\}$, and (ii) if $value(v) = 0$, $L_G(v) = \emptyset$.*
2. *If v is a nonterminal vertex, $L_G(v)$ is the finite language $L_G(v) = (label(v) \cdot L_G(one(v))) \cup L_G(zero(v))$.*

For example, the SeqBDD in Fig. 3 represents languages $L(r_1) = \{$aaba, aabc, aac, abba, abbc, abc, acc, adc, bba, bbc, bc, cc, dc$\}$ and $L(r_2) = \{$abba, abbc, abc, acc, adc, bbba, bbbc, bbc, bcc, bdc, cc, dc$\}$.

We write $L(v)$ for $L_G(v)$ if the underlying graph G is clearly understood. Moreover, if G is a SeqBDD with the single root $r \in R$, we write $L(G)$ for $L_G(r)$. We say that G is a *SeqBDD for* L if $L = L(G)$.

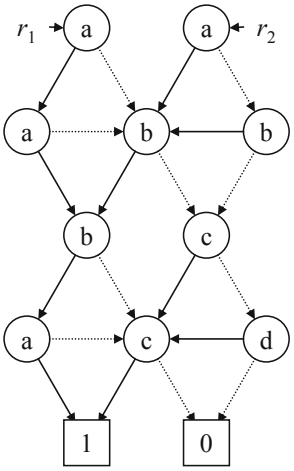

Fig. 3. An example of a SeqBDD in a shared environment. Nonterminal vertices are drawn as cercles with their labels. Terminal vertices are drawn as squares with their values. 1-edges and 0-edges are drawn as solid arrows and dotted arrows, respectively.

2.2 Shared SeqBDDs

We can use a multi-rooted SeqBDD G as a persistent data structure for storing and manipulating a collection of more than one set of strings on an alphabet Σ. In an environment, we can create a new subgraph by combining one or more existing subgraphs in G in an arbitrary way. As an invariant, all subgraphs in G are maintained as minimal. A *shared SeqBDD environment* is a 4-tuple $\mathcal{E} = (G, R, \mathit{uniqtable}, \mathit{memocache})$ consisting of a multi-rooted SeqBDD G with a vertex set V, a root vertex set R, and two hash tables *uniqtable* and *memocache*, explained below. If two tables are clearly understood from context, we identify $\mathcal{E}$ with the underlying graph G by omitting tables.

The first table *uniqtable*, called the *unique vertex table*, assigns a nonterminal vertex $v = \mathit{uniqtable}(a, v_0, v_1)$ of G to a given triple $\tau = \langle a, v_0, v_1 \rangle$ of a symbol and a pair of vertices in G. This table is maintained such that it is a function from all triples τ to the nonterminal vertex v in G such that $\mathit{triple}(v) = \tau$. If such a node does not exist, *uniqtable* returns *null*. When we want to get a vertex with a certain attribute triple $\langle a, v_0, v_1 \rangle$, we first check whether such a vertex already exists or not by querying the *uniqtable*. If such a vertex exists, we use the vertex returned from the *uniqtable*. Otherwise, we create a new vertex with the attribute triple and register it to the *uniqtable*. The operation $\mathsf{Getvertex}(a, v_0, v_1)$ executes this process. Due to this process, we can avoid generating redundant equivalent vertices. Consequently, the SeqBDD is kept minimal even though it is multi-rooted.

The second table *memocache*, called the *operation cache*, is used for a user to memorize the invocation pattern "$op(x_1, \ldots, x_k)$" of a user-defined operation *op*

and the associated return value $u = op(v_1, \ldots, v_k)$, where each v_i, $i = 1, \ldots, n$ is an argument of the operation. An argument can be a symbol, a natural number, or an existing vertex in G. We assume that the hash tables *uniqtable* and *memocache* are global variables in $\mathcal{E}$, and initialized to the empty tables when $\mathcal{E}$ is initialized.

Figure 3 shows that two SeqBDDs rooted by r_1 and r_2 share their equivalent subgraphs. In a shared environment, we can deal with multiple SeqBDDs in minimal form by using *uniqtable*, and reuse computational results of operations for some vertex when we want to execute the same operation for the same vertex by referring *memocache*. For example, assume that we compute $\mathsf{Card}(r_1)$, $\mathsf{Card}(v)$ is called for each descendant of r_1 during the recursive process. (Note that $|L(v)| = |L(zero(v))| + |L(one(v))|$.) If we compute $\mathsf{Card}(r_2)$ after obtaining the result of $\mathsf{Card}(r_1)$, during the computation of the cardinality of r_2, we need to continue recursive calls $\mathsf{Card}(zero(v))$ and $\mathsf{Card}(one(v))$ only at each nonterminal vertex v that is a descendant of r_2 but not a descendant of r_1 because the *memocache* remembers the result $\mathsf{Card}(v)$.

2.3 Operations

We view a symbolic manipulation program as executing a sequence of commands that build up representations of languages and that determine various properties about them. For example, suppose we wish to construct the representation of the language computed by a data mining program. At this point, we can test various properties of the language, such as to list some member, to list all members, and to test some string for membership.

Here, we will present algorithms to perform basic operations on sets of strings represented as SeqBDD. Table 1 summarizes operations of SeqBDDs. This table contains some new operations. Rev, LRotate, and RRotate are new ones and can be used to find palindromes and Lyndon words [16]. These basic operations can be combined to perform a wide variety of operations on sets of strings. We can construct a SeqBDD that represents a given language in $O(||L||)$ time. Aoki *et al.* proposed a more efficient algorithm to construct a SeqBDD representing a set of reversed strings [3]. Our algorithms utilize techniques commonly used in BDD and ZDD algorithms such as ordered traversal, table look-up, and vertex encoding. As the table shows, most of the algorithms have time complexity proportional to the size of the SeqBDDs being manipulated. Hence, as long as the languages of interest can be represented by reasonably small SeqBDD such that used for speech recognition [19], our algorithms are quite efficient.

These algorithms are implemented as simple recursive algorithms. Such style of algorithms are commonly used on other decision diagrams because it has the following nice properties:

– Speeding up by memoization: Sharing intermediate results between different execution of operations for SeqBDDs rooted by different vertices.

Table 1. SeqBDD basic operations.

Name	Output	Time & space complexity
0	The 0-terminal vertex	O(1)
1	The 1-terminal vertex	O(1)
Getvertex(a, v_0, v_1)	A SeqBDD vertex r such that $label(r) = a, zero(r) = v_0, one(v) = v_1$	O(1)
Build(α)	A SeqBDD vertex r such that $L(r) = \{\alpha\}$	O$(\|\alpha\|)$
Onset(u, a)	A SeqBDD vertex r such that $L(r) = \{\alpha \| a\alpha \in L(u)\}$	O$(\|\Sigma\|)$
Offset(u, a)	A SeqBDD vertex r such that $L(r) = \{b\alpha \| b\alpha \in L(u), b \neq a\}$	O$(\|\Sigma\|)$
Member(u, α)	$\alpha \in L(v)$?	O$(\|\Sigma\|\|\alpha\|)$
AddStr(v, α)	A SeqBDD vertex r such that $L(r) = L(v) \cup \{\alpha\}$	O$(\|\Sigma\|\|\alpha\|)$
DelStr(v, α)	A SeqBDD vertex r such that $L(r) = L(v) \backslash \{\alpha\}$	O$(\|\Sigma\|\|\alpha\|)$
Union(u, v)	A SeqBDD vertex r such that $L(r) = L(u) \cup L(v)$	O$(\|u\|\|v\|)$
Intersection(u, v)	A SeqBDD vertex r such that $L(r) = L(u) \cap L(v)$	O$(\|u\|\|v\|)$
Difference(u, v)	A SeqBDD vertex r such that $L(r) = L(u) \backslash L(v)$	O$(\|u\|\|v\|)$
SymDiff(u, v)	A SeqBDD vertex r such that $L(r) = L(u) \oplus L(v)$	O$(\|u\|\|v\|)$
Equal(u, v)	$L(u) = L(v)$?	O(1)
IsSubset(u, v)	$L(u) \subseteq L(v)$?	O$(\|u\|\|v\|)$
Count(u)	the number of nodes $\|u\|$	O$(\|u\|)$
Card(u)	$\|L(u)\|$	O$(\|u\|)$
TotalLen(u)	$\sum_{\alpha \in L(u)} \|\alpha\|$	O$(\|u\|)$
MinLen(u)	$\min_{\alpha \in L(u)}\{\|\alpha\|\}$	O$(\|u\|)$
MaxLen(u)	$\max_{\alpha \in L(u)}\{\|\alpha\|\}$	O$(\|u\|)$
Print1(v)	Some string of $L(v)$	O(maxlen$(L(v))$)
PrintAll(v)	$L(v)$	O$(\|L(v)\|$maxlen$(L(v)))$
Random(u)	A string $\alpha \in L(u)$ chosen uniformly at random	O$(\|u\|)$
All1()	$\{a \| a \in \Sigma\}$	O$(\|\Sigma\|)$
AlIn()	A SeqBDD vertex r such that $L(r) = \{a_1 \cdots a_n \| a_1, \ldots, a_n \in \Sigma\}$	O$(n\|\Sigma\|)$
HeadRemove(u)	$\{\alpha \| a \in \Sigma, a\alpha \in L(u)\}$	O$(2^{2\|u\|})$
TailRemove(u)	$\{\alpha \| a \in \Sigma, \alpha a, \in L(u)\}$	O$(\|u\|)$
Rev(u)	A SeqBDD vertex r such that $L(r) = \{\alpha^R \| \alpha \in L(u)\}$	N/A
LRotate(u)	A SeqBDD vertex r such that $L(r) = \{\alpha a \| a\alpha \in L(u), a \in \Sigma\} \cup (L(u) \cap \{\epsilon\})$	O$(\|u\|^2)$
RRotate(u)	A SeqBDD vertex r such that $L(r) = \{a\alpha \| \alpha a \in L(u), a \in \Sigma\} \cup (L(u) \cap \{\epsilon\})$	N/A

– Easy implementation: From a long history of research of automata, we can find a more efficient algorithm for each operation. However, implementing the best algorithms is generally difficult. Simple algorithms are valuable to make libraries in order to accept various requests for manipulation on sets of strings.

Therefore, implementing various operations in a simple recursive manner is important to obtain the above properties even though there are more efficient problem-specific algorithms.

3 SeqBDD Manipulation Operations

In this section, we list up new SeqBDD operations. We categorize the algorithms into the following three groups:

- Combination: Combine multiple SeqBDDs.
- Enumeration: Enumerate all strings that satisfy some condition.
- Retrieval: Retrieve strings which satisfy some conditions from a given set.

All of the following algorithms use *uniqtable* and *memocache* as global variables. We can define operations not included in the following tables, but we only consider operations that we can provide their algorithms.

3.1 Combination Operations

Combination operations are listed in Table 2. Basic Boolean set operations are in the table of basic operations. Concat, OverlapConcat, LDiv, RDiv, CDiv, LRem, RRem, and CRem are algebraic operations. These operations can be used to construct SeqBDDs for given regular languages and decompose SeqBDDs into several languages. LExistDiv, RExistDiv, CExistDiv, LExistRem, RExistRem, and CExistRem is variants of LDiv, RDiv, CDiv, LRem, RRem, and CRem that can be obtained by switching the quantifiers from $\forall$ to $\exists$ in the definition of operations. PrefAssign, SuffAssign, and FactAssign construct SeqBDDs by replacing prefixes, suffixes, and factors of $L(u)$ that is included in $L(v)$ by the language $L(w)$, respectively. Separate computes all factors of $L(u)$ that can be obtained by considering strings in $L(v)$ as delimiters of strings in $L(u)$.

3.2 Enumeration Operations

Enumeration operations are listed in Table 3. For given a string $\alpha \in \Sigma^*$, we define $Prefix(\alpha)$ is the set of all prefixes of α, $Suffix(\alpha)$ is the set of all suffixes of α, $Factor(\alpha)$ is the set of all factors of α, and $Subseq(\alpha)$ is the set of all subsequence of α. Also, we define $PropPrefix(\alpha) = Prefix(\alpha)\backslash\{\alpha\}$, $PropSuffix(\alpha) = Suffix(\alpha)\backslash\{\alpha\}$, $PropFactor(\alpha) = Factor(\alpha)\backslash\{\alpha\}$, and $PropSubseq(\alpha) = Subseq(\alpha)\backslash\{\alpha\}$. We use "?" as a wild card in the algorithm HammDistWild and EditDistWild. Pref, Suff, Fact, Subseq. PropPref, PropSuff, PropFact, PropSubseq construct SeqBDDs that can be used as indices. HammDist, EditDist, HammDistWild. EditDistWild are applied to approximate indices and matching problems. These algorithms are useful to generate all candidates to be processed explicitly.

Table 2. SeqBDD combination operations.

Name	Output	Time & space complexity
$\mathsf{Concat}(u, v)$	A SeqBDD vertex r such that $L(r) = \{\alpha\beta \mid \alpha \in L(u), \beta \in L(v)\}$	$O(\|u\|^2 2^{2\|v\|})$
$\mathsf{OverlapConcat}_{i,j,k}\ (u, v)$	A SeqBDD vertex r such that $L(r) = \{\alpha\beta\gamma \mid \alpha\beta \in L(u), \beta\gamma \in L(v), \|\alpha\| \geq i, \|\beta\| \geq j, \|\gamma\| \geq k\}$	N/A
$\mathsf{LDiv}(u, v)$	A SeqBDD vertex r such that $L(r) = \{\gamma \mid \forall\beta \in L(v), \beta\gamma \in L(u)\}$	$O(\|v\|2^{2\|u\|})$
$\mathsf{RDiv}(u, v)$	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \forall\beta \in L(v), \alpha\beta \in L(u)\}$	$O(\|u\|\|v\|)$
$\mathsf{CDiv}(u, v)$	A SeqBDD vertex r such that $L(r) = \{\alpha\gamma \mid \forall\beta \in L(v), \alpha\beta\gamma \in L(u)\}$	$O(2^{2\|u\|\|v\|})$
$\mathsf{LRem}(u, v)$	A SeqBDD vertex r such that $L(r) = \{\zeta \mid \zeta \in L(u), \exists\beta \in L(v), \zeta \neq \beta\gamma, \forall\gamma \in \Sigma^*\}$	$O(\|u\|\|v\|^3 2^{2\|u\|})$
$\mathsf{RRem}(u, v)$	A SeqBDD vertex r such that $L(r) = \{\zeta \mid \zeta \in L(u), \exists\beta \in L(v), \zeta \neq \alpha\beta, \forall\alpha \in \Sigma^*\}$	$O(\|u\|\|v\|2^{2\|u\|})$
$\mathsf{CRem}(u, v)$	A SeqBDD vertex r such that $L(r) = \{\zeta \mid \zeta \in L(u), \exists\beta \in L(v), \zeta \neq \alpha\beta\gamma, \forall\alpha, \gamma \in \Sigma^*\}$	N/A
$\mathsf{LExistDiv}(u, v)$	A SeqBDD vertex r such that $L(r) = \{\gamma \mid \exists\beta \in L(v), \beta\gamma \in L(u)\}$	$O(\|v\|2^{2\|u\|})$
$\mathsf{RExistDiv}(u, v)$	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \exists\beta \in L(v), \alpha\beta \in L(u)\}$	$O(\|u\|\|v\|)$
$\mathsf{CExistDiv}(u, v)$	A SeqBDD vertex r such that $L(r) = \{\alpha_0 \cdots \alpha_n \mid \exists\beta_1, \ldots, \beta_n \in L(v), \alpha_0\beta_1\alpha_1 \cdots \beta_n\alpha_n \in L(u), L(v) \cap Factor(\alpha_i) = \emptyset, i = 0, \ldots, n\}$	N/A
$\mathsf{LExistRem}(u, v)$	A SeqBDD vertex r such that $L(r) = \{\zeta \mid \zeta \in L(u), \beta \in L(v), \zeta \neq \beta\gamma, \forall\gamma \in \Sigma^*\}$	$O(\|v\|2^{2\|u\|})$
$\mathsf{RExistRem}(u, v)$	A SeqBDD vertex r such that $L(r) = \{\zeta \mid \zeta \in L(u), \beta \in L(v), \zeta \neq \alpha\beta, \forall\alpha \in \Sigma^*\}$	$O(\|u\|\|v\|)$
$\mathsf{CExistRem}(u, v)$	A SeqBDD vertex r such that $L(r) = \{\zeta \mid \zeta \in L(u), \beta \in L(v), \beta \notin Factor(\zeta)\}$	N/A
$\mathsf{PrefAssign}(u, v, w)$	A SeqBDD vertex r such that $L(r) = \{\zeta\gamma \mid \exists\beta \in L(v), \beta\gamma \in L(u), \zeta \in L(w)$, **or** $\beta \in L(v), \zeta\gamma \in L(u), \beta \notin Prefix(\zeta\gamma)\}$	$O(\|v\|2^{2\|u\|\|w\|})$
$\mathsf{SuffAssign}(u, v, w)$	A SeqBDD vertex r such that $L(r) = \{\alpha\zeta \mid \exists\beta \in L(v), \alpha\beta \in L(u), \zeta \in L(w)$, **or** $\beta \in L(v), \alpha\zeta \in L(u), \beta \notin Suffix(\alpha\zeta)\}$	$O(\|v\|2^{2\|u\|\|w\|})$
$\mathsf{FactAssign}(u, v, w)$	A SeqBDD vertex r such that $L(r) = \{\alpha_0\zeta\alpha_1 \cdots \zeta\alpha_n \mid (\exists\beta_1, \ldots, \beta_n \in L(v), \alpha_0\beta_1\alpha_1 \cdots \beta_n\alpha_n \in L(u),\ L(v) \cap Factor(\alpha_i) = \emptyset, i = 0, \ldots, n, \zeta_1, \ldots, \zeta_n \in L(w))$, **or** $(\beta \in L(v), \alpha_0 \in L(u), \beta \notin Factor(\alpha_0))\}$	$O(\|v\|2^{2\|u\|\|w\|})$
$\mathsf{Separate}(u, v)$	A SeqBDD vertex r such that $L(r) = \{\alpha_i \mid \exists\beta_1, \ldots, \beta_n \in L(v), \alpha_0\beta_1\alpha_1 \cdots \beta_n\alpha_n \in L(u), L(v) \cap Factor(\alpha_i) = \emptyset, i = 0, \ldots, n$ $\alpha_0 \in L(u), \beta \in L(v), \beta \notin Factor(\alpha_0), i = 0\}$.	N/A

Table 3. SeqBDD enumeration operations.

Name	Output	Time & space complexity
Pref(u)	A SeqBDD vertex r such that $L(r) = \bigcup_{\alpha \in L(u)} Prefix(\alpha)$.	O($\|u\|$)
Suff(u)	A SeqBDD vertex r such that $L(r) = \bigcup_{\alpha \in L(u)} Suffix(\alpha)$.	O($\|u\|^2$)
Fact(u)	A SeqBDD vertex r such that $L(r) = \bigcup_{\alpha \in L(u)} Factor(\alpha)$.	O($\|u\|^2$)
Subseq(u)	A SeqBDD vertex r such that $L(r) = \bigcup_{\alpha \in L(u)} Subseq(\alpha)$.	O($2^{2\|u\|}$)
PropPref(u)	A SeqBDD vertex r such that $L(r) = \bigcup_{\alpha \in L(u)} PropPrefix(\alpha)$.	O($\|u\|$)
PropSuff(u)	A SeqBDD vertex r such that $L(r) = \bigcup_{\alpha \in L(u)} PropSuffix(\alpha)$.	O($\|u\|^2$)
PropFact(u)	A SeqBDD vertex r such that $L(r) = \bigcup_{\alpha \in L(u)} PropFactor(\alpha)$.	O($\|u\|^2$)
PropSubseq(u)	A SeqBDD vertex r such that $L(r) = \bigcup_{\alpha \in L(u)} PropSubseq(\alpha)$.	O($2^{2\|u\|}$)
HammDist(u, d)	A SeqBDD vertex r consists of strings within Hamming distance d from $\alpha \in L(u)$.	N/A
EditDist(u, d)	A SeqBDD vertex r consists of strings within edit distance d from $\alpha \in L(u)$.	N/A
HammDistWild(u, d)	A SeqBDD vertex r consists of strings within Hamming distance d from $\alpha \in L(u)$ allowing use of wild cards.	N/A
EditDistWild(u, d)	A SeqBDD vertex r consists of strings within edit distance d from $\alpha \in L(u)$ allowing use of wild cards.	N/A

3.3 Retrieval Operations

Retrieval operations are listed in Table 4. Shorter, Longer, Just, Shortest, and Longest derive languages consisting of strings of desired length. ExistPref, ExistSuff, ExistFact, and ExistSubseq retrieve strings that have some string in $L(v)$ as their prefixes, suffixes, factors, and subsequences, respectively. PrefMaximal, SuffMaximal, FactMaximal, SubseqMaximal, PrefMinimal, SuffMinimal, FactMinimal, and SubseqMinimal can find maximal or minimal strings among their prefixes, suffixes, factors, and subsequences, respectively.

3.4 Complexity Analyses

In this subsection, we describe how the complexities in the above tables are calculated. For basic set operations such as Union, Intersection, and Difference for

Table 4. SeqBDD retrieval operations.

Name	Output	Time & space complexity
Shorter(u, l)	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \alpha \in L(u), \lvert\alpha\rvert \leq l\}$.	$O(l\lvert u\rvert)$
Longer(u, l)	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \alpha \in L(u), l \leq \lvert\alpha\rvert\}$.	$O(l\lvert u\rvert)$
Just(u, l)	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \alpha \in L(u), l = \lvert\alpha\rvert\}$.	$O(l\lvert u\rvert)$
Shortest(u)	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \alpha \in L(u), \lvert\alpha\rvert = min_{\beta \in L(u)}\{\lvert\beta\rvert\}\}$.	$O(\lvert u\rvert)$
Longest(u)	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \alpha \in L(u), \lvert\alpha\rvert = max_{\beta \in L(u)}\{\lvert\beta\rvert\}\}$.	$O(\lvert u\rvert)$
ExistPref(u, v)	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \alpha \in L(u), \exists\beta \in L(v), \beta \in Prefix(\alpha)\}$.	$O(\lvert u\rvert\lvert v\rvert)$
ExistSuff(u, v)	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \alpha \in L(u), \exists\beta \in L(v), \beta \in Suffix(\alpha)\}$.	$O(\lvert u\rvert\lvert v\rvert)$
ExistFact(u, v)	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \alpha \in L(u), \exists\beta \in L(v), \beta \in Factor(\alpha)\}$.	$O(\lvert u\rvert\lvert v\rvert)$
ExistSubseq(u, v)	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \alpha \in L(u), \exists\beta \in L(v), \beta \in Subseq(\alpha)\}$.	$O(\lvert u\rvert 2^{2\lvert v\rvert})$
PrefMaximal(u)	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \alpha, \beta \in L(u), \alpha \neq \beta, \alpha \notin Prefix(\beta)\}$.	$O(\lvert u\rvert^2)$
SuffMaximal(u)	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \alpha, \beta \in L(u), \alpha \neq \beta, \alpha \notin Suffix(\beta)\}$.	$O(\lvert u\rvert^3)$
FactMaximal(u)	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \alpha, \beta \in L(u), \alpha \neq \beta, \alpha \notin Factor(\beta)\}$.	$O(\lvert u\rvert^3)$
SubseqMaximal(u)	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \alpha, \beta \in L(u), \alpha \neq \beta, \alpha \notin Subseq(\beta)\}$.	$O(\lvert u\rvert 2^{2\lvert u\rvert})$
PrefMinimal(u)	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \alpha, \beta \in L(u), \alpha \neq \beta, \beta \notin Prefix(\alpha)\}$.	$O(\lvert u\rvert^3)$
SuffMinimal(u)	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \alpha, \beta \in L(u), \alpha \neq \beta, \beta \notin Suffix(\alpha)\}$.	$O(\lvert u\rvert^3)$
FactMinimal(u)	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \alpha, \beta \in L(u), \alpha \neq \beta, \beta \notin Factor(\alpha)\}$.	$O(\lvert u\rvert^4)$
SubseqMinimal(u)	A SeqBDD vertex r such that $L(r) = \{\alpha \mid \alpha, \beta \in L(u), \alpha \neq \beta, \beta \notin Subseq(\alpha)\}$.	$O(\lvert u\rvert^2 2^{2\lvert u\rvert^2})$
ComnPref(u)	A SeqBDD vertex r such that $L(r) = \bigcap_{\alpha \in L(u)} Prefix(\alpha)$.	$O(\lvert u\rvert)$

vertices u and v, its time and space complexities are $O(|u||v|)$ because possible function calls with different pair of vertices are at most $O(|u||v|)$, and function calls with the same arguments are processed in constant time thanks to the memoization technique. As a result, the size of output SeqBDD is also $O(|u||v|)$ [11]. Consequently, if we continue Union or Intersection k times for vertices reachable from v, the complexity becomes $O(|v|^{k+1})$. Note that each vertex in the resultant

SeqBDD can be written as a combination of $k+1$ vertices of v's descendants. If the number of repetition k is not fixed, each vertex in the output SeqBDD can be written as combination of all v's descendants. Therefore, complexity becomes $O(2^{2^{|v|}})$ in such cases. Note that the size of output SeqBDD is $O(2^{|v|})$. If each output vertex can be written as a result of computing $|v|$ vertices combined by j non-commutative operations, its complexity is $O(\sum_{i=1}^{|v|} j^{i-1} i!)$. The complexities in the tables are obtained from the above observations. For operations with more complicated algorithms, we could not calculate complexities and the time & space complexity columns of such operations are N/A.

4 Conclusion

In this paper, we proposed 50 new algorithms for manipulating SeqBDDs. All of our algorithms are written as recursive algorithms with memoization. Due to intermediate results sharing caused by memoization technique, the total computation time of multiple executions of the same operation will be faster when dealing with multi-rooted SeqBDD in a shared environment. For future work, we implement the algorithms proposed in this paper. The complexity analyses can be improved. We should consider combining our algorithms with existing problem-specific efficient algorithms. We will be able to define more operations and give algorithms for them.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Boston (1974)
2. Alhakami, H., Ciardo, G., Chrobak, M.: Sequence decision diagrams. In: Moura, E., Crochemore, M. (eds.) SPIRE 2014. LNCS, vol. 8799, pp. 149–160. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11918-2_15
3. Aoki, H., Yamashita, S., Minato, S.: An efficient algorithm for constructing a sequence binary decision diagram representing a set of reversed sequences. In: Hong, T., et al. (eds.) Proceedings of 2011 IEEE International Conference on Granular Computing, pp. 54–59. IEEE Computer Society (2011). https://doi.org/10.1109/GRC.2011.6122567
4. Blumer, A., Blumer, J., Haussler, D., Ehrenfeucht, A., Chen, M.T., Seiferas, J.I.: The smallest automaton recognizing the subwords of a text. Theoret. Comput. Sci. **40**, 31–55 (1985). https://doi.org/10.1016/0304-3975(85)90157-4
5. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Trans. Comput. **C–35**(8), 677–691 (1986). https://doi.org/10.1109/TC.1986.1676819
6. Bubenzer, J.: Minimization of acyclic DFAs. In: Holub, J., Žďárek, J. (eds.) Proceedings of Prague Stringology Conference 2011, pp. 132–146. Czech Technical University (2011). http://www.stringology.org/event/2011/p12.html
7. Champarnaud, J.M., Pin, J.E.: A maxmin problem on finite automata. Discrete Appl. Math. **23**(1), 91–96 (1989). https://doi.org/10.1016/0166-218X(89)90037-1
8. Crochemore, M.: Transducers and repetitions. Theoret. Comput. Sci. **45**(1), 63–86 (1986). https://doi.org/10.1016/0304-3975(86)90041-1

9. Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on Strings. Cambridge University Press, Cambridge (2007)
10. Daciuk, J., Mihov, S., Watson, B.W., Watson, R.: Incremental construction of minimal acyclic finite state automata. Comput. Linguist. **26**(1), 3–16 (2000). https://doi.org/10.1162/089120100561601
11. Denzumi, S., Yoshinaka, R., Arimura, H., Minato, S.: Sequence binary decision diagram: minimization, relationship to acyclic automata, and complexities of Boolean set operations. Discrete Appl. Math. **212**, 61–80 (2016). https://doi.org/10.1016/j.dam.2014.11.022
12. Gusfield, D.: Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, Cambridge (1997)
13. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd edn. Addison-Wesley, Boston (2006)
14. Loekito, E., Bailey, J., Pei, J.: A binary decision diagram based approach for mining frequent subsequences. Knowl. Inf. Syst. **24**(2), 235–268 (2010). https://doi.org/10.1007/s10115-009-0252-9
15. Lucchesi, C.L., Kowaltowski, T.: Applications of finite automata representing large vocabularies. Softw. Pract. Exp. **23**(1), 15–30 (1993). https://doi.org/10.1002/spe.4380230103
16. Lyndon, R.C.: On Burnside's problem. Trans. Am. Math. Soc. **77**(2), 202–215 (1954)
17. Michon, J.-F., Champarnaud, J.-M.: Automata and binary decision diagrams. In: Champarnaud, J.-M., Ziadi, D., Maurel, D. (eds.) WIA 1998. LNCS, vol. 1660, pp. 178–182. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48057-9_15
18. Minato, S.: Zero-suppressed BDDs for set manipulation in combinatorial problems. In: Dunlop, A.E. (ed.) Proceedings of 30th Design Automation Conference, pp. 272–277. ACM Press (1993). https://doi.org/10.1145/157485.164890
19. Mohri, M., Moreno, P., Weinstein, E.: Factor automata of automata and applications. In: Holub, J., Žd'árek, J. (eds.) CIAA 2007. LNCS, vol. 4783, pp. 168–179. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76336-9_17