Sequence Decision Diagrams*

Hind Alhakami[1], Gianfranco Ciardo[2], and Marek Chrobak[1]

[1] Dept. of Computer Science and Engineering, University of California, Riverside
[2] Dept. of Computer Science, Iowa State University

Abstract. Compact encoding of finite sets of strings is a classic problem. The manipulation of large sets requires compact data structures that allow for efficient set operations. We define *sequence decision diagrams* (SeqDDs), which can encode arbitrary finite sets of strings over an alphabet. SeqDDs can be seen as a variant of classic decision diagrams such as BDDs and MDDs where, instead of a fixed number of levels, we simply require that the number of paths and the lengths of these paths be finite. However, the main difference between the two is the target application: while MDDs are suited to store and manipulate large sets of constant-length tuples, SeqDDs can store arbitrary finite languages and, as such, should be studied in relation to finite automata. We do so, examining in particular the size of equivalent representations.

1 Introduction

Many data structures have been introduced to compactly encode finite sets of finite strings. *Substring indices* data structures, such as *tries*, suffix trees, suffix arrays, and DAWGs, exploit prefix sharing, suffix sharing, or both to achieve efficient storage of large sets. Beside compactness, the main purpose of *substring indices* data structures is to solve substring matching problem for multiple patterns in a given text with a time complexity proportional to the pattern size, not the whole text. These data structures allow for efficient matching, but updating them to add or delete strings is hard [1]. Additionally, the lack of efficient set manipulation algorithms for such data structures motivates work that leverages the benefits of *substring indices* while enabling efficient set manipulation.

In 2009, Loekito [7] introduced a new data structure, *sequence* BDD, SeqBDD, for short, that offers compact storage of finite languages. SeqBDDs are a half-relaxed variation of ZBDDs [8] where variables along *one-paths* may appear multiple times in any order. SeqBDDs inherit ZBDDs' efficient set manipulations, and also support algorithms to solve the substring matching problem.

Size complexity is crucial to decision diagrams, including SeqBDDs, due to two factors: first, decision diagrams are used to store efficiently an enormous amount of data; second, the time complexity of decision diagram algorithms is proportional to the size of the arguments, which is in turn sensitive to variable ordering. Since optimal variable ordering is an NP-complete problem [3], heuristics can only achieve a "good "variable ordering. Moreover, while sharing

* This work is supported in part by Ministry of Higher Education - Saudi Arabia, and National Science Foundation under grants CCF-1217314 and CCF-1442586.

E. Moura and M. Crochemore (Eds.): SPIRE 2014, LNCS 8799, pp. 149–160, 2014.

common suffixes as well as common prefixes contributes to the compactness of SeqBDDs, embracing a binary representation degrades compactness [9].

We define *sequence decision diagrams* (SeqDDs) to encode arbitrary finite languages. SeqDDs are somewhat analogous to a multi-valued variation of SeqBDDs, but are insensitive to variable ordering; in fact, they do not even associate variables or levels to nodes. Instead, they simply require that the number of paths and the lengths of these paths be finite. We introduce two canonical SeqDD definitions and discuss their compactness in relation to finite automata. Canonical SeqDD promotes efficient algorithms for set manipulations and substring manipulations by exploiting node sharing and *memoization.* The rest of the paper is organized as follows: Section 2 provides preliminaries. Section 3 introduces non-canonical and canonical SeqDDs. Section 4 discusses the relative compactness of canonical SeqDDs. Section 5 introduces set and string manipulation algorithms. Section 6 provides preliminary applications of SeqDDs. Section 7 presents conclusions and future work.

2 Preliminaries

Finite automata are a well known data structure to describe regular languages. While finite automata are memory efficient, their manipulation algorithms are not guaranteed to provide minimized outputs even if their inputs are minimized. On the other hand, decision diagrams have efficient manipulation algorithms but most, for example BDDs [4] and MDDs [6], only target fixed-length languages.

2.1 Finite Automata

A finite automaton (FA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, with a finite set of states, a finite alphabet, a transition function, a start state, and a set of accepting states. Depending on the transition function, the FA is a *deterministic* FA (DFA, with $\delta : Q \times \Sigma \to Q$) or a *non-deterministic* FA (NFA, with $\delta : Q \times \Sigma \cup \{\epsilon\} \to 2^Q$). We also consider a *partial* DFA [2], a minimized DFA with partial transition function $\delta : Q \times \Sigma \to Q \cup \{\emptyset\}$, obtained from the equivalent DFA by deleting all states with no path to accepting states, as well as their incoming transitions.

2.2 Decision Diagrams

Binary decision diagrams (BDDs) are directed acyclic graph where each node is associated with a boolean variable and encodes boolean functions over a structured boolean domain. Multi-valued decision diagrams (MDDs) generalize BDDs by allowing nodes to have more than two outgoing edges, and provide a canonical representation of boolean functions over structured finite domains (we use "MDDs" from now on, since BDDs are just a special case).

An *ordering* rule is enforced: assuming k domain variables $\{x_1, ..., x_k\}$, all paths respect the order $x_k \prec x_{k-1} \prec \cdots \prec x_1 \prec x_0$, where x_0 is the range variable associated with terminal nodes. Then, canonicity requires choosing a

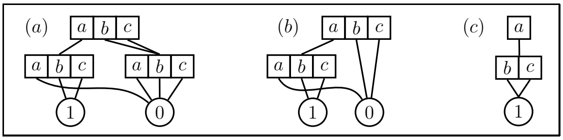

Fig. 1. Quasi (a), fully (b), and sparsely (c) reduced MDDs encoding $\mathcal{Y} = \{ab, ac\}$

reduction: *quasi*-reduced, only merge duplicate (i.e., isomorphic) nodes; *fully*-reduced, merge duplicate nodes and skip redundant (i.e., with identical children) nodes; or *sparsely*-reduced, merge duplicate nodes and omit nodes not reaching the **1**-terminal, and any edge pointing to them (Fig.1).

Decision diagrams excel at encoding sets that share many subsets, and their recursive structure enables effective use of dynamic programming through an *operation cache*, which virtually eliminates the need to recompute subproblems.

2.3 Notation

Given alphabet $\Sigma = \{s_1, \cdots, s_m\}$, with $m \in \mathbb{N}$, let Σ^* be the set of strings over Σ, i.e., $\Sigma^* = \{a_1 \cdots a_k : k \geq 0, \forall h, 1 \leq h \leq k, a_h \in \Sigma\}$. We introduce the following notation to discuss SeqDDs encoding a finite language $\mathcal{Y} \subset \Sigma^*$:

- If $\mathcal{Y} = \emptyset$, then $height(\mathcal{Y}) = \bot$, "undefined". Otherwise, the height of $\mathcal{Y}$ is the length of the longest string in it, $height(\mathcal{Y}) = \max\{|\sigma| : \sigma \in \mathcal{Y}\}$.
- $lengths(\mathcal{Y}) = \{k \in \mathbb{N} : \exists \sigma \in \mathcal{Y}, |\sigma| = k\}$, the set of all string lengths in $\mathcal{Y}$.
- For $k \in lengths(\mathcal{Y})$, $\mathcal{Y}_k = \{\sigma \in \mathcal{Y} : |\sigma| = k\}$, the strings of length k in $\mathcal{Y}$, and $\mathcal{Y}_{<k} = \{\sigma \in \mathcal{Y} : |\sigma| < k\}$, the strings of length less than k in $\mathcal{Y}$.
- For $a \in \Sigma$, $\mathcal{Y}/a = \{\sigma \in \Sigma^* : a \cdot \sigma \in \mathcal{Y}\}$, the strings that, preceded by a, form a string in $\mathcal{Y}$.
- For $k \in lengths(\mathcal{Y})$ and $a \in \Sigma$, $\mathcal{Y}_k/a = \{\sigma \in \Sigma^{k-1} : a \cdot \sigma \in \mathcal{Y}_k\}$, the strings that, preceded by a, form a string of length k in $\mathcal{Y}$.
- $||\mathcal{Y}|| = \sum_{\sigma \in \mathcal{Y}} |\sigma|$, the total number of symbols in $\mathcal{Y}$, not to be confused with $|\mathcal{Y}|$, the number of strings in $\mathcal{Y}$.

3 Definition of Sequence Decision Diagrams

We now define a class of decision diagrams to encode any finite subset of Σ^*.

Definition 1. A *sequence decision diagram* (SeqDD) is a directed acyclic finite graph with two *terminal* nodes, **0** and **1**, and such that each *nonterminal* node p has $m + 1$ outgoing edges, each labeled with a different element from $\Sigma \cup \{\epsilon\}$; we write $p[a] = q$ to indicate that the outgoing edge labeled with $a \in \Sigma \cup \{\epsilon\}$ points to node q, which can be a terminal or nonterminal node. □

Definition 2. The set of strings $\mathcal{X}(p)$ encoded by a SeqDD node p is:

$$\mathcal{X}(p) = \begin{cases} \emptyset, \text{ the empty set} & \text{if } p = \mathbf{0}, \\ \{\epsilon\}, \text{ the set containing only the empty string} & \text{if } p = \mathbf{1}, \\ \bigcup_{a \in \Sigma \cup \{\epsilon\}} \{a \cdot \sigma : \sigma \in \mathcal{X}(p[a])\} & \text{otherwise.} \end{cases} \qquad \square$$

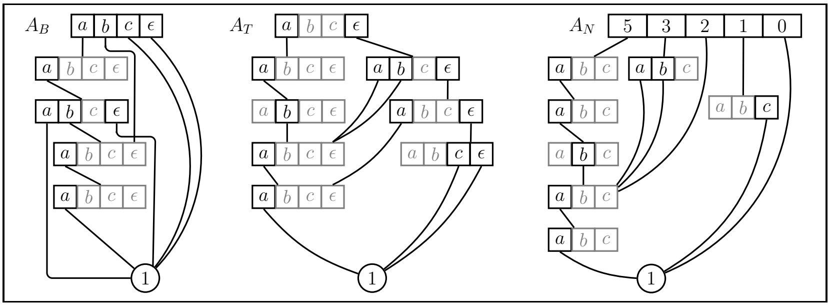

Fig. 2. A SeqDD_B, a SeqDD_T, and a SeqDD_N encoding $\mathcal{Y} = \{aa, aaa, aabaa, baa, c, \epsilon\}$. Indices in gray point to terminal **0** (not represented for clarity).

Theorem 1. Given a finite set of strings $\mathcal{Y} \subset \Sigma^*$, there exists a SeqDD with a root (i.e., a node with no incoming edges) p satisfying $\mathcal{X}(p) = \mathcal{Y}$.
Proof. The proof is trivial and left to the reader. □

As defined, SeqDDs are general non-canonical encoding of finite languages. Any set $\mathcal{Y} \subset \Sigma^*$ can be encoded by infinitely many SeqDDs because, if a node r encodes $\mathcal{Y}$, any node r' with $r'[a] = \mathbf{0}$ for each $a \in \Sigma$ and $r'[\epsilon] = r$ also encodes $\mathcal{Y}$, and the "insertion" of such "useless nodes" can be repeated at will (indeed, not just above the root, but anywhere along any path in the SeqDD). Thus, we now describe possible sets of restrictions to ensure canonicity. In any case:

– No *duplicate nodes* are allowed: the SeqDD cannot contain two nonterminal nodes p and q such that $p[a] = q[a]$ for every $a \in \Sigma \cup \{\epsilon\}$.
– No *empty nodes* are allowed: the SeqDD cannot contain a nonterminal node p such that $p[a] = \mathbf{0}$ for every $a \in \Sigma \cup \{\epsilon\}$.
– No ϵ-*nodes* are allowed: the SeqDD cannot contain a nonterminal node p such that $p[a] = \mathbf{0}$ iff $a \in \Sigma$.

Then, informally, canonicity is achieved by additionally "pushing" ϵ-edges (not pointing to **0**) toward the bottom, or toward the top, of the diagram (Fig. 2).

3.1 Definition of Canonical SeqDDs with ϵ at the Bottom

Definition 3. A SeqDD_B is a SeqDD with no duplicate, empty, or ϵ-nodes where, for any nonterminal node p, either $p[\epsilon] = \mathbf{0}$ or $p[\epsilon] = \mathbf{1}$. □

Theorem 2. Given a finite set of strings $\mathcal{Y} \subset \Sigma^*$, there exists a unique single-root SeqDD_B whose root p satisfies $\mathcal{X}(p) = \mathcal{Y}$.
Proof. If $height(\mathcal{Y}) = \bot$, then $\mathcal{Y} = \emptyset$, and the canonicity restrictions imply that $p = \mathbf{0}$ is the only SeqDD_B node encoding $\mathcal{Y}$. If $height(\mathcal{Y}) = 0$, then $\mathcal{Y} = \{\epsilon\}$, and the same restrictions imply that $p = \mathbf{1}$ is the only SeqDD_B node encoding $\mathcal{Y}$. If $height(\mathcal{Y}) = k > 0$, assume the theorem holds for any $\mathcal{Y}'$ with $height(\mathcal{Y}') < k$. Clearly, $height(\mathcal{Y}/a) < k$ and, if $\epsilon \in \mathcal{Y}$, then $\mathcal{Y} = \{\epsilon\} \cup \bigcup_{a\in\Sigma} a \cdot \mathcal{Y}/a$, otherwise $\mathcal{Y} = \bigcup_{a\in\Sigma} a \cdot \mathcal{Y}/a$. Then, if $\epsilon \in \mathcal{Y}$, we can define node p, with $p[\epsilon] = \mathbf{1}$ and,

for each $a \in \Sigma$, $p[a] = q_a$, where q_a is the unique node encoding $\mathcal{Y}/a$ (by induction, q_a exist since $height(\mathcal{Y}/a) < k$). Note that we might have $\mathcal{Y}/a = \mathcal{Y}/b$ for $a \neq b$, this simply means that the two corresponding edges in p point to the same $\mathrm{SeqDD_B}$ node (indeed nodes are shared across any of the descendants of p, to avoid duplicates). No other node q encoding $\mathcal{Y}$ can exist because it would have to differ from p in at least one index $a \in \Sigma$, while we must have $p[\epsilon] = q[\epsilon] = \mathbf{1}$. By inductive assumption, $\mathrm{SeqDD_B}$'s $p[a]$ and $q[a]$ cannot encode the same set, that is, $\mathcal{X}(p[a]) = \mathcal{Y}/a \neq \mathcal{X}(q[a])$, thus there is a string $a \cdot \sigma'$ in $\mathcal{X}(p)$ and not in $\mathcal{X}(q)$, or vice versa. The case where $\epsilon \notin \mathcal{Y}$ is analogous, except that $p[\epsilon] = \mathbf{0}$. □

3.2 Definition of Canonical SeqDDs with ϵ at the Top

For the alternative definition where we allow "ϵ at the top", it is easier to recast the definition of quasi-reduced MDDs [5] as a special case of SeqDDs.

Definition 4. A k-level MDD is the terminal node $\mathbf{1}$, if $k = 0$, or, if $k > 0$, it is a single-root SeqDD without duplicate, empty, or ϵ-nodes where the root p is such that $p[\epsilon] = \mathbf{0}$ and, for $a \in \Sigma$, $p[a]$ is a $(k-1)$-level MDD or $\mathbf{0}$. □

Thus, the root of a k-level MDD encodes a nonempty set of strings of length k.

Definition 5. A k-level $\mathrm{SeqDD_T}$ is a SeqDD without duplicate, empty, or ϵ-nodes whose root node p is such that, for $a \in \Sigma$, $p[a]$ is $\mathbf{0}$ or the root of a $(k-1)$-level MDD, while $p[\epsilon]$ is $\mathbf{0}$ or the root of an h-level $\mathrm{SeqDD_T}$, $h < k$. □

Thus, it is easy to prove by induction that the root p of a k-level $\mathrm{SeqDD_T}$ encodes a nonempty set of strings of length k, $\bigcup_{a \in \Sigma} \mathcal{X}(q[a])$, plus a possibly empty set of strings of length less than k, $\mathcal{X}(q[\epsilon])$.

Theorem 3. Given a finite language $\mathcal{Y} \subset \Sigma^*$, there exists a unique single-root $\mathrm{SeqDD_T}$ with root p such that $\mathcal{X}(p) = \mathcal{Y}$.
Proof. If $height(\mathcal{Y}) = \bot$, then $\mathcal{Y} = \emptyset$, and the canonicity restrictions imply that $p = \mathbf{0}$ is the only $\mathrm{SeqDD_T}$ encoding $\mathcal{Y}$. If $height(\mathcal{Y}) = 0$, then $\mathcal{Y} = \{\epsilon\}$, and the same restrictions imply that $p = \mathbf{1}$ is the only $\mathrm{SeqDD_T}$ encoding $\mathcal{Y}$. If instead $height(\mathcal{Y}) = k > 0$, assume that the theorem holds for any set $\mathcal{Y}'$ with $height(\mathcal{Y}') < k$. Since $\mathcal{Y} = \mathcal{Y}_{<k} \cup \bigcup_{a \in \Sigma} a \cdot \mathcal{Y}_k/a$, we can define node p such that, for $a \in \Sigma$, $p[a] = q_a$ with $\mathcal{X}(q_a) = \mathcal{Y}_k/a$, while $p[\epsilon] = q_\epsilon$ with $\mathcal{X}(q_\epsilon) = \mathcal{Y}_{<k}$. By inductive hypothesis, nodes q_a and q_ϵ are unique, as they all encode sets of height less than k and, since $\mathcal{Y}_k/a$ contains only strings of length $k-1$, q_a is in particular the root of an MDD, i.e., $q_a[\epsilon] = \mathbf{0}$. Then, node p is also the only node encoding $\mathcal{Y}$ since any other node p' would have to differ from p in at least one child. If $p[\epsilon] \neq p'[\epsilon]$, there must exists a string σ of length less than k in $\mathcal{X}(p[\epsilon])$, thus $\mathcal{X}(p)$, and not in $\mathcal{X}(p'[\epsilon])$, thus $\mathcal{X}(p')$, or vice versa. If there is an $a \in \Sigma$ with $p[a] \neq p'[a]$, there must exists a string σ in $\mathcal{X}(p[a])$ and not in $\mathcal{X}(p'[a])$, so that $a \cdot \sigma$ is in $\mathcal{X}(p)$ and not in $\mathcal{X}(p')$, or vice versa ($a \cdot \sigma$ cannot possibly be in $\mathcal{X}(p'[\epsilon])$ as it is of length k). Either way, p' cannot encode the same set as p. □

A $\mathrm{SeqDD_T}$ relies on some concept of level for the nodes of the decision diagram. More specifically, a $\mathrm{SeqDD_T}$ node encodes all the maximum-length strings in its children corresponding to elements of Σ and delegates the encoding of the shorter strings to its ϵ-child. A similar encoding for set $\mathcal{Y}$ partitions its strings according to their length, and uses a top node to make a decision based on the length of the string σ being searched, not on the first symbol of σ (Fig. 2). This leads us to a third, different in spirit but essentially equivalent, definition.

Definition 6. A SeqDD_N is a set of "sparse" root nodes, each root r having a finite set $\mathcal{R}$ of outgoing edges labeled with different elements $k \in \mathbb{N}$, such that $r[k]$ points to a k-level MDD. The set encoded by r is $\bigcup_{k\in\mathcal{R}} \mathcal{X}(r[k])$. □

4 Compactness of Canonical SeqDD Definitions

We now discuss the size of our SeqDDs, where the size of a SeqDD A is the number of edges it contains, *edges*(A), rather than the number of nodes. Given the structural differences between a SeqDD_B and a SeqDD_T, we compare them by thinking of them as finite automata. A closer look at a SeqDD_B shows that it can be easily converted into a DFA (Theorem 4). On the other hand, a SeqDD_T can be converted into a restricted type of NFA.

4.1 DFA Representation of SeqDD_B

Given a SeqDD_B A_B encoding a finite language $\mathcal{Y} \subset \Sigma^*$, we can build an equivalent DFA $M = (Q, \Sigma, \delta, q_0, F)$. If $A_B = \mathbf{0}$ then $M = (\{q_0\}, \Sigma, \delta, q_0, \emptyset)$. Otherwise, we first define the states Q in terms of the nodes in A_B: every nonterminal node q in A_B corresponds to a state $q \in Q$, while node $\mathbf{1}$ in A_B corresponds to new state $f \in Q$ and node $\mathbf{0}$ corresponds to a new trap state $t \in Q$.

The initial state q_0 corresponds to A_B's root while the transition function $\delta : Q \times \Sigma \to Q$ is such that, for every $a \in \Sigma$ and edge $q[a] = p$ in A_B, there is a corresponding transition $\delta(q, a) = p$ and, if $q[\epsilon] = \mathbf{1}$, no transition is added, but q is added to the accepting states F. Lastly, state f is also added to F.

Theorem 4. Given a SeqDD_B A_B encoding a finite language $\mathcal{Y} \subset \Sigma^*$, building an equivalent minimized DFA M requires linear time in the size of A_B.
Proof. The proof is direct from the translation algorithm above. □

For memory efficiency, decision diagrams can be stored in a sparse form. In the case of a sparse SeqDD_B, this corresponds to a *partial* DFA, and the translation is analogous to the non-sparse version just discussed. From now on, we consider sparse representations for all canonical forms of SeqDD and for partial DFAs.

4.2 NFA Representation of SeqDD_T

To discuss the translation of a SeqDD_T into an equivalent NFA, we first define RNFAs, a restricted version of NFAs, keeping in mind that our goal is to facilitate size comparisons between a SeqDD_B and a SeqDD_T. To that end, our RNFA definition resembles the structure of SeqDD_T while respecting the key characteristics of ordinary NFAs when encoding a finite language.

Definition 7. A restricted NFA (RNFA) is an acyclic NFA $N = (Q, \Sigma, \delta, Q_I, Q_F)$, where both Q_I and Q_F are singletons sets and, for each state $q \in Q$, the following condition holds: at most one outgoing ϵ-transition is allowed, and if $k = \max(lengths(L(q)))$ then all strings in $\bigcup_{a \in \Sigma} L(\delta(q, a))$ have length equal $k - 1$ and all strings in $L(\delta(q, \epsilon))$ have length at most $k - 1$. This value k is called the *level of* q. □

A *minimized* RNFA enforces the following restriction rules.

- No *duplicate states* are allowed: An RNFA cannot contains q and p such that $L(q) = L(p)$.
- No *empty states* are allowed: An RNFA cannot contain a state $q \in Q \setminus Q_I$ such that $L(q) = \emptyset$.
- No ϵ-*states* are allowed: An RNFA cannot contain a state $q \in Q \setminus Q_F$ such that $L(q) = \{\epsilon\}$.

Any RNFA can be converted to an equivalent minimized RNFA by adapting the bucket-sort based OBDD reduction algorithm proposed in [10]. The minimized RNFA for a given language is unique, the proof is omitted due to lack of space.

The following lemma affirms that RNFAs, like DFAs, can recognize any finite language (unlike DFAs, they obviously cannot accept any infinite language).

Lemma 1. If $\mathcal{Y} \subset \Sigma^*$ is a finite language, there exists an RNFA N to accept $\mathcal{Y}$.
Proof. The proof of existence is analogous to the one of Theorem 3. □

If SeqDD$_{\mathrm{T}}$ A_T with a single root node r encodes a finite language $\mathcal{Y} \subset \Sigma^*$, the equivalent RNFA $T = (Q, \Sigma, \delta, Q_I, Q_F)$ is built as follows. Each nonterminal node q of A_T corresponds to a state $q \in Q$; terminal node $\mathbf{1}$ of A_T corresponds to a new state $\mathbf{1} \in Q$, and $F = \{\mathbf{1}\}$; finally, $Q_I = \{r\}$ (note that, if $r = \mathbf{0}$, we also must add r to Q). The transition function $\delta : Q \times \Sigma \cup \{\epsilon\} \to Q$ is such that, for every edge $q[a] = p$ in A_T with $a \in \Sigma \cup \{\epsilon\}$, there is a corresponding transition $\delta(q, a) = p$. Thus, in particular, if $r = \mathbf{0}$, then $T = (\{\mathbf{0}\}, \Sigma, \emptyset, \{\mathbf{0}\}, \{\mathbf{1}\})$, and the encoded language is $\mathcal{Y} = \emptyset$, while, if $A_T = \mathbf{1}$, then $T = (\{\mathbf{1}\}, \Sigma, \emptyset, \{\mathbf{1}\}, \{\mathbf{1}\})$ and the encoded language is $\mathcal{Y} = \{\epsilon\}$.

From the conversion process, it is easy to conclude that a canonical SeqDD size is bounded by the size of the corresponding FA in terms of number of transitions, plus the number of accepting states.

4.3 SeqDD Compactness Comparison by Means of Finite Automata

To study the relative compactness of canonical SeqDDs, we first discussed bounds on the number of states for equivalent DFAs and RNFAs; these are trivially reflected in similar bounds for SeqDD$_{\mathrm{B}}$'s and SeqDD$_{\mathrm{T}}$'s. To obtain bounds on the number of transitions, one could just multiply the state bounds by the alphabet size, but we are really interested in the actual number of edges for equivalent SeqDDs, thus partial FAs. This section shows that bounds similar to those for states hold also for edges.

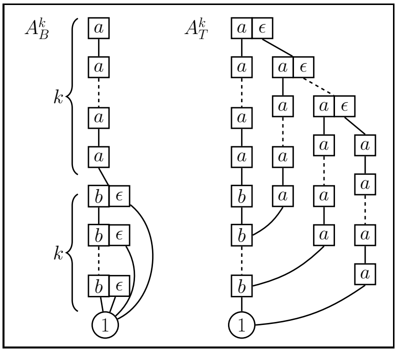

Fig. 3. Example of quadratic growth when translating SeqDD_B into SeqDD_T

Theorem 5. Given a DFA $M = (Q, \Sigma, \delta_D, q_0, F)$ with n states encoding a finite language $\mathcal{Y} \subset \Sigma^*$, an equivalent minimized RNFA N has $O(n^2)$ states.
Proof. For each state $q \in Q$ and $k = 0, \ldots, \mathit{height}(\mathcal{Y})$, let $L(q, k) = L(q) \cap \Sigma^k$. Then, we build an equivalent RNFA N with states organized by level:

- Level 0 of the RNFA contains a single accepting state f.
- Level k contains a state $\langle q,k \rangle$ for each nonempty $L(q, k)$.
- The initial state of N is $\langle q_0, \max \mathit{lengths}(\mathcal{Y}) \rangle$.
- The transition function δ_N of N satisfies
  - For each state $\langle q,k \rangle$ with $k > 0$ in N and for each $a \in \Sigma$: $\langle p,k-1 \rangle \in \delta_N(\langle q,k \rangle, a)$ iff $\delta_D(q, a) = p$.
  - For each state $\langle q,k \rangle$ in N, let h be the largest integer less than k such that state $\langle q,h \rangle$ exists in N; if such state exists, then $\langle q,h \rangle \in \delta_N(\langle q,k \rangle, \epsilon)$.

Note that the resulting RNFA might not be minimized, in the sense that it is possible that $\langle q,k \rangle$ and $\langle p,k \rangle$ encode the same language, in which case they should be merged. In any case, however, the number of states of the RNFA is at most equal to the number of states of the DFA times the maximum length of a string in $\mathcal{Y}$, which, again, is at most equal to the number of states. Thus the number of RNFA states is at most quadratic the number of DFA states. As the two automata obviously accept the same language $\mathcal{Y}$, the proof is complete. $\square$

To show that the growth of of Theorem 5 is indeed possible, consider the family of languages $\mathcal{G} = \{\mathcal{G}_k : k \in \mathbb{N}\}$ over $\{a, b\}$. Let $\mathcal{G}_k = \{a^k b^k, a^k b^{k-1}, \cdots, a^k b, a^k\}$, so that $||\mathcal{G}_k|| = 3(k+1)k/2$. Then, the SeqDD_T A_T^k encoding $\mathcal{G}_k$ contains k^2+3k edges, while the SeqDD_B A_B^k encoding $\mathcal{G}_k$ contains $3k$ edges (see Fig. 3).

Theorem 6. Given a minimized RNFA N with n states encoding a finite language $\mathcal{Y} \subset \Sigma^*$, an equivalent minimized DFA has at most $O(2^n)$ states.
Proof. The proof is immediate given the well known fact that an NFA-to-DFA conversion may result in an exponential increase in the number of states. $\square$

Since RNFAs are a restricted form of NFAs, however, one may wonder whether an exponential growth can actually occur. To show that this is the case, consider the

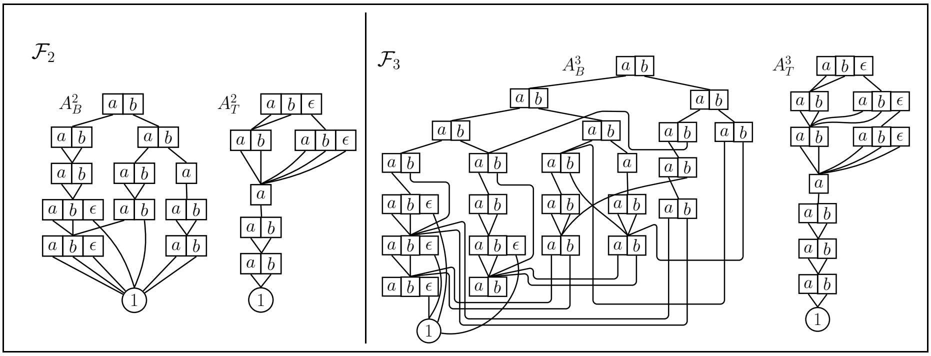

Fig. 4. Example of exponential growth when translating $\mathrm{SeqDD_T}$ into $\mathrm{SeqDD_B}$

family of languages $\{\mathcal{F}_k : k \in \mathbb{N}\}$ with $\mathcal{F}_k = \{xay : x, y \in \{a,b\}^*, |x| \leq k, |y| = k\}$. Then, the $\mathrm{SeqDD_T}$ A_T^k encoding $\mathcal{G}_k$ contains $7k-1$ edges while the $\mathrm{SeqDD_B}$ A_B^k encoding $\mathcal{G}_k$ contains $\Omega(2^k)$ edges (see Fig. 4). This is similar to the well-known construction that demonstrates the proof of Theorem 6.

5 Manipulation Algorithms for SeqDDs

We now consider two types of algorithms: *set manipulation algorithms* and *substring manipulation algorithms*. Those of the first type take two or more canonical SeqDDs with the same canonicity rule and perform set operations such as *union* or *intersection*. Those of the second type input a canonical SeqDD and a string, and select strings satisfying a criterion for matching a substring, changing a substring into another, or shorten or lengthen a string.

As with all decision diagram algorithms, we adopt a recursive style. SeqDD nodes are stored in a *unique table* to ensure canonicity. An *operation cache* ensures efficiency by virtually eliminating repeated computations. Each of the following *set manipulation algorithms* has been developed for $\mathrm{SeqDD_B}$ and $\mathrm{SeqDD_N}$ representations: union, intersection, set difference, symmetric set difference, and concatenation. For instance, the *Intersection* algorithm for two $\mathrm{SeqDD_B}$'s traverses them top-down and builds the resulting $\mathrm{SeqDD_B}$ bottom-up (see the pseudo-code in Fig. 5). $\mathrm{SeqDD_N}$ set manipulation algorithms can be considered as shared MDD algorithms, since a $\mathrm{SeqDD_N}$ is organized by the length of the strings encoded.

Various string manipulations can be performed. For example, the classical membership problem can be solved by a single trace, no longer than the *query size* $+1$, starting from the root and ending in either terminal **1** or **0**. Set manipulation algorithms can also become handy in performing string manipulations; for instance, the membership problem is solved by a set intersection, and string replacement can be solved using a combination of set difference, intersection, and union. However, if we want to perform substring manipulations, the use of set manipulation algorithms becomes inefficient, hence we developed specific substring manipulation algorithms.

SeqDD$_B$ *Intersection*(*SeqDD*$_B$ p, *SeqDD*$_B$ q) • *returns* $\mathcal{X}(r) = \mathcal{X}(p) \cap \mathcal{X}(q)$
declare local SeqDD$_B$ r;
declare local int $count$;
if $p = \mathbf{0}$ or $q = \mathbf{0}$ then return $\mathbf{0}$; • *base case: empty set*
if $p = q$ then return p; • *base case: Intersection of two equivalent sets*
if $p = \mathbf{1}$ then if $q[\epsilon] = \mathbf{1}$ then return $\mathbf{1}$; else return $\mathbf{0}$; • *base case:* ϵ
if $q = \mathbf{1}$ then if $p[\epsilon] = \mathbf{1}$ then return $\mathbf{1}$; else return $\mathbf{0}$; • *base case:* ϵ
if $Cache$ contains $\langle$*Intersection*,$\{p, q\}$:$r\rangle$ then return r; • *check if already computed*
$count \leftarrow 0$; • *initialize counter*
foreach $a \in \Sigma$ do • *if not, recursively call Intersection for each* $a \in \Sigma$
    $r[a] \leftarrow Intersection(p[a], q[a])$;
    if $r[a] = \mathbf{0}$ then $count \leftarrow count + 1$; • *count edges pointing to terminal-*$\mathbf{0}$
if $count = |\Sigma|$ then $r \leftarrow \mathbf{0}$; • *potential empty-node or* ϵ*-node*
if $p[\epsilon] = \mathbf{1}$ and $q[\epsilon] = \mathbf{1}$ then • *deal with* ϵ *case*
    if $r = \mathbf{0}$ or $r = \mathbf{1}$ then $r \leftarrow \mathbf{1}$;
    else $r[\epsilon] \leftarrow \mathbf{1}$;
$UniqueTableInsert(r)$; • *insert to* unique table *to ensure canonicity*
$Cache \leftarrow \langle$*Intersection*,$\{p, q\}$:$r\rangle$; • *record result in cache to avoid recomputation*
return r;

Fig. 5. SeqDD$_B$ *Intersection* operation

The main advantage of using SeqDDs for substring manipulation lies in the ability to search or modify a set of strings at once, thanks to node sharing and *memoization*. For example, in a SeqDD$_B$, replacing the first occurrence of a substring t with t' is done once for all strings sharing a prefix that contains t. Moreover, a shared suffix is processed the first time we explore it; for other strings sharing that suffix the algorithm simply checks the *operation cache* for the result. A universal algorithm *replace* can replace, insert, or delete a specific substring: replacing ϵ by a string $t \neq \epsilon$ performs an insertion, while replacing t by ϵ performs a deletion. Of course, this can be refined by additionally providing to the algorithm specific substrings that must be found before and after the replacement location.

6 Applications of Sequence Decision Diagrams

Advancements in genome sequencing techniques along with their affordability have resulted in an increasing number of sequenced genomes. As a consequence, a concise representation that allows for efficient data manipulation is required to query, analyze, and retrieve this information. These processes are essential in various molecular biology problems.

SeqDD$_B$ and SeqDD$_N$ provide simple indexing data structures. Their compactness in regards to sequence indexing is summarized in Table 1. Given a string w of size x, it is well known that the size of a DAWG that encodes the

Table 1. Summary of the upper bound size of a SeqDD_B or SeqDD_N encodingthe set of all prefixes, suffixes, or subwords of a certain string of size x

Encoded set	DAWG size	SeqDD_B size	SeqDD_N size
Suffixes	$3x-4$	$3x-4$	$2x+1$
Subwords	$3x-4$	$3x-4$	$(5x^2+3x+6)/4$
Prefixes	x	x	x^2+1

set of suffixes / subwords of w is at most $3x-4$ transitions, for $x > 2$ [2]. The size of a SeqDD_B encoding w's suffixes (subwords) is bounded by $4x-3$ $(5x-6)$ transitions. Technically, while a SeqDD_B ϵ-transitions are shown in the figures as edges, in reality they can be encoded by a single bit, since an ϵ-transition can only point to the terminal state. Thus, the size of a SeqDD_B is actually bounded by $3x-4$ transition plus $x+1$ bits when encoding the set of suffixes or $2x-2$ bits when encoding the set of subwords given that all states are accepting. On the other hand, the size of a SeqDD_N encoding subwords of w is bounded by $2x + \sum_{j=1}^{x} j + 3/2 \sum_{j=2}^{x-2} j$, which simplifies to $(5x^2+3x+6)/4$ transitions.

Using SeqDD_B or SeqDD_N for indexing sequences allows for efficient manipulations. For instance, the membership problem requires time linear in the size of the query when handled one sequence at a time. Querying a large set of sequences at once could lead to substantial improvement in time complexity because decision diagrams exploit node sharing and *memoization*, if we build a SeqDD that encodes the query set and perform a simple intersection.

The longest common substring can be retrieved by intersecting the SeqDDs encoding the set of subwords of each sequence. Using SeqDD_N's allows early pruning, but consumes space. To achieve better space efficiency, SeqDDs encoding the set of suffixes can be used along with a non-commutative variation of the intersection algorithm in Fig. 5, so that, when $p = \mathbf{1}$, the algorithm returns q. In this case, the longest common substring for more than two sequences is solved incrementally, thus SeqDD_N's lose the advantages of early pruning. Note that both SeqDD intersection and its variation have time complexity proportional to the size of the smallest argument. A generalization of this problem is the DNA contamination problem.

The all-pairs suffix-prefix matching problem can be solved with multi-terminal SeqDD, a simple tweak to our original definition. Let $\mathcal{G} = \{s_1, s_2, \cdots, s_k\}$ be a set of strings, all pairs with matching prefix-suffix can be obtained by performing a prefix intersection between $\mathcal{Q}$ and p, where $\mathcal{Q}$ is a shared SeqDD with k handles, each pointing to a SeqDD q_i encoding the set of suffixes of s_i and p is a multi-terminal SeqDD encoding $\mathcal{G}$ with $k+1$ terminal nodes corresponding to the $\mathbf{0}$-terminal and the k strings.

7 Conclusion

We introduced SeqDDs, multi-valued sequence decision diagrams, which can be seen as MDDs with no variable ordering but are nevertheless canonical. In fact, our SeqDDs do not have a notion of variables, hence any "size explosion" exclusively depends on the specific set to be encoded and on the canonization rule (we introduce two possibilities, $\mathrm{SeqDD_B}$ and $\mathrm{SeqDD_T}$). More importantly, SeqDDs are ideal for encoding finite sets of strings of arbitrary finite (but possibly different) lengths, that is, finite languages. $\mathrm{SeqDD_T}$'s are analogous to shared MDDs, and may be best implemented by adding special nodes at the top level that makes a choice based on the string length; we call this version $\mathrm{SeqDD_N}$. We study the compactness of our representations in terms of finite automata and show that there is no winner between the two versions: a $\mathrm{SeqDD_T}/\mathrm{SeqDD_N}$ can be quadratically larger than a $\mathrm{SeqDD_B}$ for certain languages, but exponentially more compact for others; therefore, we are implementing algorithms for both versions. SeqDDs are useful for applications requiring compact storage and efficient manipulation of large sets of strings with high sharing rate. As future work, an edge-valued variation is a must for many applications, such as symbolic generation of probabilistic witnesses in CSL model checking.

References

1. Aoki, H., Yamashita, S., Minato, S.: An efficient algorithm for constructing a sequence binary decision diagram representing a set of reversed sequences. In: 2011 IEEE International Conference on Granular Computing (GrC), pp. 54–59 (2011)
2. Blumer, A., Blumer, J., Ehrenfeucht, A., Haussler, D., McConnell, R.: Building the minimal DFA for the set of all subwords of a word on-line in linear time. In: Paredaens, J. (ed.) ICALP 1984. LNCS, vol. 172, pp. 109–118. Springer, Heidelberg (1984)
3. Bollig, B., Wegener, I.: Improving the variable ordering of OBDDs is NP-complete. IEEE Trans. Comput., 993–1002 (1996)
4. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Comput., 677–691 (1986)
5. Ciardo, G., Lüttgen, G., Siminiceanu, R.I.: Saturation: An efficient iteration strategy for symbolic state space generation. In: Margaria, T., Yi, W. (eds.) TACAS 2001. LNCS, vol. 2031, pp. 328–342. Springer, Heidelberg (2001)
6. Kam, T., Villa, T., Brayton, R.K., Sangiovanni-Vincentelli, A.: Multi-valued decision diagrams: Theory and applications. Multiple-Valued Logic, 9–62 (1998)
7. Loekito, E., Bailey, J., Pei, J.: A binary decision diagram based approach for mining frequent subsequences. Knowledge and Information Systems, 235–268 (2010)
8. Minato, S.: Zero-suppressed BDDs for set manipulation in combinatorial problems. In: 30th Conference on Design Automation, pp. 272–277 (1993)
9. Requeno, J.I., Colom, J.M.: Compact representation of biological sequences using set decision diagrams. In: Rocha, M.P., Luscombe, N., Fdez-Riverola, F., Rodríguez, J.M.C. (eds.) 6th International Conference on PACBB. AISC, vol. 154, pp. 231–240. Springer, Heidelberg (2012)
10. Sieling, D., Wegener, I.: Reduction of OBDDs in linear time. Information Processing Letters, 139–144 (1993)