Vol.2010-MPS-79 No.3
2010/7/12

線形分離オートマトンの高速な状態数最小化アルゴリズム

沼 井 裕 二[†1] 小 林 聡[†1]

本論文では，著者らが過去に提案した線形分離オートマトン（LSA）の状態数最小化アルゴリズムを高速化する．LSA は有限オートマトンを拡張したモデルで，実ベクトル系列を受理する能力を持ち，各状態には重み関数と閾値系列が付随する．この二つによって，各時点でのある状態からの遷移先状態が決定する．

過去の著者らの論文では，与えられた LSA M の状態数を最小化する理論を確立し， $O(n^2)$ 時間の状態数最小化アルゴリズムを提出した．n は M の状態数である．本論文では， $O(n\log n)$ 時間にまで高速化した状態数最小化アルゴリズムを与える．

Efficient State Minimization Algorithm of Linear Separation Automata

Yuji Numai[†1] and Satoshi Kobayashi [†1]

In this paper, we present a faster state minimization algorithm of a linear separation automaton (LSA for short) than the algorithm we recently proposed. An LSA is an extended model of a finite automaton. It accepts a sequence of real vectors, and has a weight and a threshold sequence at every state, which determine the transition from the current state to the next at each step.

We established a theory of minimizing the number of states of a given LSA M, and proposed an $O(n^2)$ time state minimization algorithm of M, where n is the number of states of M. This paper gives an $O(n\log n)$ time efficient state minimization algorithm of M.

†1 電気通信大学大学院電気通信学研究科情報工学専攻
Department of Computer Science, Graduate School of Electro-Communications, The University of Electro-Communications

1. Introduction

The theory of computational models that can deal with sequences of real valued vectors is an important research topic if we consider problem domains including weather forecasting[5], motion recognition[3),4], and time-sequential image analysis[9].

We proposed a computational model, called a Linear Separation Automaton (LSA for short). The LSA is an extended model of a finite automaton. It accepts a sequence of real vectors, and has a weight and a threshold sequence at every state, which determine the transition from the current state to the next at each step. If we consider the learning problem of automata, it is essentially important to establish the theory of minimizing the number of states of a given automaton. It is often the case that learning algorithms try to find a minimum state automaton consistent with a given set of examples.

In our previous works, we established a theory of minimizing the number of states of a given LSA[7], and proposed an $O(n^2)$ time state minimization algorithm[6]. This paper gives an $O(n\log n)$ time efficient state minimization algorithm of a given LSA.

2. Preliminaries

In this section, we introduce some basic definitions and notation, and introduce a linear separation automaton.

2.1 Basic Definitions and Notation

By R, we denote the set of real numbers. For a positive integer d, by R^d we denote d-dimensional vector space over R. For $x, y \in \mathrm{R}^d$, $x \otimes y$ denotes the inner product of x and y. We define $(\mathrm{R}^d)^*$ as the set of all finite sequences of elements in R^d. For a sequence $\alpha = \langle x_1, \ldots, x_n\rangle \in (\mathrm{R}^d)^*$, we denote the length of α by $|\alpha|$, i.e., $|\alpha| = n$. An element in $(\mathrm{R}^d)^*$ of length 0 is called an *empty sequence*, and is denoted by λ. For sequences $\alpha, \beta \in (\mathrm{R}^d)^*$, we denote the concatenation of sequences α and β by $\alpha\beta$. For $\alpha = \langle x_1, \ldots, x_n\rangle \in (\mathrm{R}^1)^*$, α is said to be *increasing* if the inequality $x_i < x_{i+1}$ holds for every i.

Let S be a set. A *partition* $\pi = \{S_1, \ldots, S_k\}$ of S is the set of mutually disjoint non-empty subsets S_i of S for $1 \leq i \leq k$ such that $\cup_{i=1,\ldots,k} S_i = S$. A partition $\pi = \{S_1, \ldots, S_k\}$ of R^d is said to be *linearly separable* iff there exist $w \in \mathrm{R}^d$ and an increasing $h = \langle h_1, \ldots, h_{k-1} \rangle \in (\mathrm{R}^1)^*$ such that, for any $x \in \mathrm{R}^d$,

$$h_{i-1} < w \otimes x \leq h_i \quad \Leftrightarrow \quad x \in S_i \quad (i = 1, \ldots, k)$$

holds, where $h_0 = -\infty$ and $h_k = \infty$.

Consider equivalence relations $\equiv, \equiv_1$, and $\equiv_2$ over $(\mathrm{R}^d)^*$. The number of the equivalence classes of $\equiv$ is called the *index* of $\equiv$. An equivalence relation $\equiv_1$ is *finer* than an equivalence relation $\equiv_2$ (or $\equiv_2$ is *coarser* than $\equiv_1$) iff $x \equiv_1 y$ implies $x \equiv_2 y$ for any x and y. An equivalence relation $\equiv$ is *right invariant* iff $\alpha \equiv \beta$ implies $\alpha\gamma \equiv \beta\gamma$ for any α, β and γ.

Consider partitions π_1 and π_2 of R^d. A partition π_1 is *finer* than a partition π_2 (or π_2 is *coarser* than π_1) iff for any block $B_1 \in \pi_1$, there exists a block $B_2 \in \pi_2$ such that $B_1 \subseteq B_2$. We say that π_1 is a *refinement* of π_2 iff π_1 is finer than π_2.

2.2 Linear Separation Automata

A *linear separation automaton (LSA)* M is formally defined as an 8-tuple

$$M = (d, Q, q_0, F, w, h, s, \delta) \ ,$$

where d is a positive integer specifying the dimension of input vectors to M; Q is a finite set of states; q_0 is an initial state ($q_0 \in Q$); F is a finite set of final states ($F \subseteq Q$); w is a weight function from Q to R^d such that $w(q)$ is a *unit vector* for any $q \in Q$; h is a threshold function from Q to $(\mathrm{R}^1)^*$ such that $h(q)$ is increasing for every $q \in Q$, and is denoted by $h(q) = \langle h(q)_1, \ldots, h(q)_{|h(q)|} \rangle$; and s is a sub-transition function from Q to Q^*, and is denoted by $s(q) = \langle s(q)_1, \ldots, s(q)_{|s(q)|} \rangle$. If $|s(q)| \geq 1$, then the equality $|h(q)| = |s(q)| - 1$ holds for every $q \in Q$.

In order to improve the readability, we write $i_q = |h(q)|$ for any $q \in Q$.

δ is a state transition function from $Q \times \mathrm{R}^d$ to Q; and is defined in the following way by using w, h, and s. Consider any state $q \in Q$ and vector $x \in \mathrm{R}^d$. The definition of δ is separated into three components. First, in the case of $|s(q)| = 0$, the value $\delta(q, x)$ is undefined. Secondly, suppose that $|s(q)| = 1$. The value $\delta(q, x)$ is defined as $\delta(q, x) = s(q)_1$. Finally, assume that $|s(q)| \geq 2$. The value $\delta(q, x)$ is defined as follows:

$$\delta(q, x) = \begin{cases} s(q)_1 & \text{if} \quad w(q) \otimes x \leq h(q)_1 \\ s(q)_2 & \text{if} \quad h(q)_1 < w(q) \otimes x \leq h(q)_2 \\ \vdots & \qquad \vdots \\ s(q)_{i_q} & \text{if} \quad h(q)_{i_q - 1} < w(q) \otimes x \leq h(q)_{i_q} \\ s(q)_{i_q+1} & \text{if} \quad h(q)_{i_q} < w(q) \otimes x \ . \end{cases}$$

Consider a state transition diagram as in Fig. 1. Suppose that $\delta(q, x) = p$ holds if $h(q)_i < w(q) \otimes x \leq h(q)_{i+1}$. In the diagram, the transition from q to p is associated with the interval $(h(q)_i, h(q)_{i+1}]$.

For $\alpha = \langle x_1, \ldots, x_l \rangle \in (\mathrm{R}^d)^*$, we write $\delta(p, \alpha) = q$ if there exists a sequence $p_1 (= p), p_2, \ldots, p_{l+1} (= q)$ of states such that $\delta(p_i, x_i) = p_{i+1}$ holds for any i. We define the set of sequences accepted by an LSA M, denoted by $L(M)$, as $L(M) = \{\alpha \in (\mathrm{R}^d)^* \mid \delta(q_0, \alpha) \in F\}$. A subset L of $(\mathrm{R}^d)^*$ is said to be *regular* if there exists an LSA M such that $L = L(M)$. We define the size of M as the cardinality $|Q|$ of Q.

A state $q \in Q$ is said to be *reachable* if there exists $\alpha \in (\mathrm{R}^d)^*$ such that $\delta(q_0, \alpha) = q$. A state $q \in Q$ is said to be *unreachable* if q is not reachable.

Example 1. *Consider an LSA M_1 in Fig. 1. Let $\alpha = \langle x_1, x_2, x_3 \rangle$ be an input sequence of vectors in R^2 with $x_1 = (3\sqrt{10}, 2\sqrt{10})$, $x_2 = (-\sqrt{5}, 2\sqrt{5})$, and $x_3 = (-3\sqrt{10}, -2\sqrt{10})$. It holds that $\delta(q_1, x_1) = q_6$, $\delta(q_6, x_2) = q_4$, and $\delta(q_4, x_3) = q_4 \in F$. Hence, the sequence α is accepted by M_1.*

3. Basic State Minimization Algorithm of LSA

In this section, we give some theoretical results for LSAs, showed in 7), and review a basic algorithm to minimize the number of states of a given LSA, presented in 6).

3.1 Theoretical Results

For any subset S of $(\mathrm{R}^d)^*$, we define an equivalence relation $\approx_S$ over $(\mathrm{R}^d)^*$ as follows:

$$\alpha \approx_S \beta \ \overset{\text{def}}{\Leftrightarrow} \ \forall \gamma \in (\mathrm{R}^d)^* \ \ (\alpha\gamma \in S \text{ iff } \beta\gamma \in S) \ .$$

Let $M = (d, Q, q_0, F, w, h, s, \delta)$ be an LSA accepting S with no unreachable states.

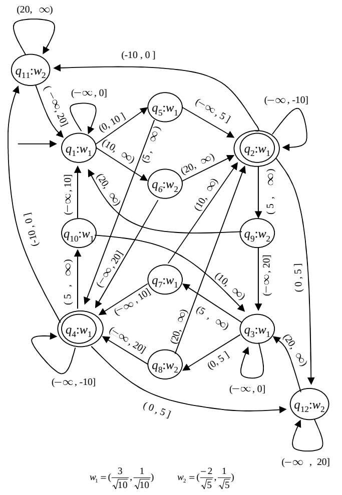

Fig. 1 LSA M_1.

For any $p, q \in Q$, there exists $\alpha, \beta \in (\mathrm{R}^d)^*$ such that $\delta(q_0, \alpha) = p$ and $\delta(q_0, \beta) = q$. We define the equivalence relation $\sim$ over Q as follows:

$$p \sim q \overset{\text{def}}{\Leftrightarrow} \alpha \approx_S \beta \ .$$

The states p and q are said to be *indistinguishable* iff $p \sim q$. The states p and q are said to be *distinguishable* iff $p \not\sim q$.

For any $p \in Q$, by $r(p)$ we denote a *representative element* of $[p]_\sim$. We define an LSA

$$M/\sim = (d, Q', q'_0, F', w', h', s', \delta') \ ,$$

where

$$Q' = Q/\sim \ , \ q'_0 = [q_0]_\sim \ , \ F' = \{[q]_\sim \mid q \in F\} \ ,$$
$$\delta'([q]_\sim, x) = [\delta(r(q), x)]_\sim \ , \ w'([q]_\sim) = w(r(q)) \ , \ h'([q]_\sim) = h(r(q)) \ .$$

Theorem 1 (Characterization of Minimum State LSA)**.** *Let M be an LSA. The LSA $M/\sim$ is a minimum state LSA for M such that $L(M/\sim) = L(M)$.*

3.2 Basic State Minimization Algorithm

Let $M = (d, Q, q_0, F, w, h, s, \delta)$ be an LSA. For a state $q_1, q_2 \in Q$, we write $q_1 \sim_w q_2$ if $w(q_1) = w(q_2)$. A state q_1 is *preceding to* a state q_2 *with respect to* a state q, denoted by $q_1 \prec_q q_2$, if there exists an integer i such that $s(q)_i = q_1$ and $s(q)_{i+1} = q_2$. For $q \in Q$, we define

$$\Delta(q) = \{\, p \mid p \in s(q) \,\} \ .$$

For a subset X of Q, we define

$$\Delta(X) = \{\, p \mid q \in X, \, p \in \Delta(q) \,\} \ .$$

For a partition π of Q and $q_1, q_2 \in Q$, we write $q_1 \sim_{(\pi)} q_2$ if there exists $B \in \pi$ such that $q_1, q_2 \in B$. For a subset X of Q, we define

$$W(X) = \{\, w(q) \mid q \in X \,\} \ .$$

For a subset X of Q and $\omega \in W(Q)$, we define

$$X_\omega = \{\, q \in X \mid w(q) = \omega \,\} \ .$$

For any $\omega \in W(Q)$, we also define

$$H(\omega) = \{\, h(q)_i \mid q \in Q_\omega, \, 1 \le i \le i_q, \, s(q)_i \ne s(q)_{i+1} \,\} \cup \{\infty\} \ .$$

For $\omega \in W(Q)$ and $v \in H(\omega)$, we define the function $\delta_{\omega,v}$ from Q_ω to Q as follows:

$$\delta_{\omega,v}(q) = \delta(q, x) \ \text{ for some } \ x \in \mathrm{R}^d \ \text{ with } \ \omega \otimes x = v \ .$$

We define the set of functions $\overline{\Delta}$ as follows:

$$\overline{\Delta} = \{\, \delta_{\omega,v} \mid \omega \in W(Q), \, v \in H(\omega) \,\} \ .$$

In the sequel, for simple description of the algorithm, we often use graph representation of mappings $f \in \overline{\Delta}$ and $\Delta : Q \to 2^Q$, i.e., f is represented as a graph containing edges between q_1 and q_2 such that $q_2 = f(q_1)$, written $q_1 f q_2$; and Δ is represented as a graph containing edges between q_1 and q_2 such that $q_2 \in \Delta(q_1)$, written $q_1 \Delta q_2$.

Theorem 2 (Characterization of Partition $Q/\sim$)**.** *Let $M = (d, Q, q_0, F, w, h, s, \delta)$ be an LSA. The partition $Q/\sim$ is a coarsest refinement π of $\pi_0 = \{F, Q - F\}$ which satisfies the following conditions:*
(C1) $\forall B \in \pi \ \forall f \in \overline{\Delta} \ \exists B' \in \pi$ *such that* $f(B) \subseteq B'$,
(C2) $\forall B \in \pi \ (|W(B)| > 1 \ \Rightarrow \exists B' \in \pi$ *such that* $\Delta(B) \subseteq B' \ )$.

Now, let us describe the basic state minimization algorithm of a given LSA. It uses two primitive refinement operations *split*$_1$ and *split*$_2$; the former is for the condition **(C1)**, and the latter is for **(C2)**.

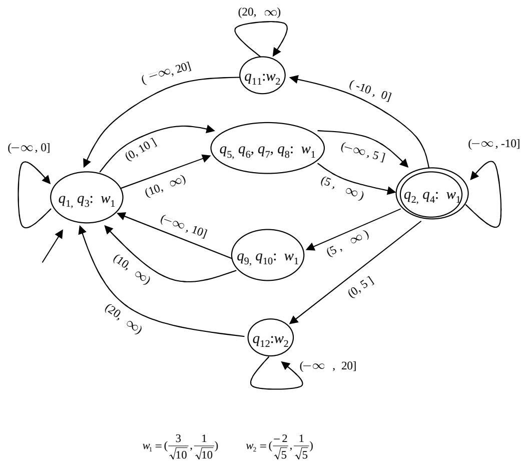

Fig. 2 Minimum state LSA of M_1.

For a set $S \subseteq Q$, $f \in \overline{\Delta}$, and a partition π of Q, the operation $split_1(S, f, \pi)$ is defined as follows:

---

find all blocks $B \in \pi$ such that $f(B) \cap S \neq \emptyset$ and $f(B) \not\subseteq S$. Define $B_1 = B \cap f^{-1}(S)$ and $B_2 = B - B_1$. Then, split $B \in \pi$ into the blocks B_1 and B_2, which results in the refinement of π.

For a set $S \subseteq Q$ and a partition π of Q, the operation $split_2(S, \pi)$ is defined as follows:

---

find all blocks $B \in \pi$ such that $\Delta(B) \cap S \neq \emptyset$, $\Delta(B) \not\subseteq S$ and $|W(B)| > 1$, and split B into some smaller blocks defined in the following way; Define $B' = \{\, q \in B \mid \Delta(q) \cap S \neq \emptyset$ and $\Delta(q) \not\subseteq S \,\}$, $B_1 = \{q \in B - B' \mid \Delta(q) \subseteq S\}$, and $B_2 = (B - B') - B_1$. For each $\omega \in W(B')$, consider B'_ω. Then, split $B \in \pi$ into B_1, B_2 and B'_ω's for all $\omega \in W(B')$, which results in the refinement of π.

Algorithm 1 is below.

Algorithm 1 Basic State Minimization Algorithm for LSA

Input: An LSA $M = (d, Q, q_0, F, w, h, s, \delta)$

Output: π

```
1:  let $\pi = \{F, Q - F\}$;
2:  loop
3:    if $\exists B \in \pi$ such that $split_2(B, \pi) \neq \pi$ then
4:      replace $\pi$ with $split_2(B, \pi)$;
5:    else if $\exists B \in \pi$, $\exists f \in \overline{\Delta}$ such that $split_1(B, f, \pi) \neq \pi$ then
6:      replace $\pi$ with $split_1(B, f, \pi)$;
7:    else
8:      output $\pi$ and halt;
9:    end if
10: end loop
```

Example 2. *The output of Algorithm 1 for the input LSA M_1 in Fig. 1 is in Fig. 2.*

3.3 Correctness of Algorithm 1

We give some basic properties of the split operations:

Lemma 1. *A partition π satisfies* **(C1)** *if and only if $split_1(B, f, \pi) = \pi$ for every block $B \in \pi$ and $f \in \overline{\Delta}$. A partition π satisfies* **(C2)** *if and only if $split_2(B, \pi) = \pi$ for every block $B \in \pi$.*

Lemma 2. *If π_2 is a refinement of π_1 and $split_1(S, f, \pi_1) = \pi_1$ holds, then $split_1(S, f, \pi_2) = \pi_2$ holds. If π_2 is a refinement of π_1 and $split_2(S, \pi_1) = \pi_1$ holds, then $split_2(S, \pi_2) = \pi_2$ holds.*

Lemma 3. *The equalities $split_1(S_1, f, \pi) = \pi$ and $split_1(S_2, f, \pi) = \pi$ imply $split_1(S_1 \cup S_2, f, \pi) = \pi$. The equalities $split_2(S_1, \pi) = \pi$ and $split_2(S_2, \pi) = \pi$ imply $split_2(S_1 \cup S_2, \pi) = \pi$.*

Lemma 4. *If π_1 is a refinement of π_2 and $split_2(S, \pi_2) = \pi_2$ holds, then $split_1(S, f, \pi_1)$ is a refinement of $split_1(S, f, \pi_2)$.*

Lemma 5. *Let π_1 be a partition satisfying* **(C1)** *and S be a union of some blocks in*

π_1. If π_1 is a refinement of π_2, then $split_2(S, \pi_1)$ is a refinement of $split_2(S, \pi_2)$.

Lemma 6. *Algorithm 1 maintains the invariant that any coarsest refinement of the initial partition $\{F, Q-F\}$ satisfying* **(C1)** *and* **(C2)** *is also a refinement of the current partition π.*

Finally, we have the following theorem.

Theorem 3 (Correctness of Algorithm 1)**.** *Let $M = (d, Q, q_0, F, w, h, s, \delta)$ be an LSA, and $n = |Q|$. Algorithm 1 for the input M is correct and terminates after at most $n-1$ refinement steps, having computed the coarsest refinement of $\{F, Q-F\}$ satisfying* **(C1)** *and* **(C2)**.

Proof. Since the number of blocks of a partition of Q is less than or equal to n, and since the number of blocks increases at each refinement step, the algorithm terminates at most $n-1$ refinement steps. Lemma 1 implies that the final partition π_f satisfies **(C1)** and **(C2)**. Moreover, Lemma 6 implies that π_f should be the coarsest refinement of $\{F, Q-F\}$ satisfying **(C1)** and **(C2)**. □

Let us discuss the time complexity of Algorithm 1. We define

$$K = \max\{\, |H(\omega)| \mid \omega \in W(Q)\,\}$$

and

$$k = \max\{\, |\Delta(q)| \mid q \in Q\,\} \ .$$

The following theorem holds.

Theorem 4 (Time Complexity of Algorithm 1)**.** *Let $M = (d, Q, q_0, F, w, h, s, \delta)$ be an LSA, and $n = |Q|$. The time complexity of Algorithm 1 for the input M is $O((K+k)\,n^2)$.*

Proof. Let $m = (K+k)n$, i.e., m is the upper bound of the total number of edges contained in the graphs $f \in \overline{\Delta}$ and in the graph Δ. It is straightforward to see that finding a block B satisfying the if-conditions (at lines 3 and 5) and refining π afterwards can be done in time $O(m)$.

Moreover, the upper bound of the number of refining π is $n-1$.

Hence the time complexity of Algorithm 1 is $O(mn) = O((K+k)\,n^2)$. □

4. Efficient State Minimization Algorithm

In this section, we will show a faster algorithm than Algorithm 1, called Algorithm 2, which works in time $O(m \log n) = O(Kn \log n)$. This is based on the "process the smaller half" idea used by Hopcroft[1),2)] and Paige[8)].

In Algorithm 2, we keep two partitions π and π' such that π is a refinement of π' and $split_1(B, f, \pi) = \pi$ and $split_2(B, \pi) = \pi$ hold for every block $B \in \pi'$ and $f \in \overline{\Delta}$. A block $B \in \pi'$ is said to be *compound* if it contains more than one blocks of π.

Algorithm 2 Efficient State Minimization Algorithm for LSA

Input: An LSA $M = (d, Q, q_0, F, w, h, s, \delta)$

Output: π

```
1:  let $\pi = \{F, Q-F\}$;
2:  let $\pi' = \{Q\}$;
3:  while $\pi \neq \pi'$ do
4:      select a compound block $B \in \pi'$;
5:      let $B_1, B_2$ be the first two blocks of $\pi$ contained in $B$, and let $B_1$ be the smaller;
6:      let $\pi' := (\pi' - \{B\}) \cup \{B_1, B - B_1\}$;
7:      if $split_2(B_1, \pi) \neq \pi$ then
8:          replace $\pi$ with $split_2(B_1, \pi)$;
9:      end if
10:     for all $f \in \overline{\Delta}$ such that $split_1(B_1, f, \pi) \neq \pi$ do
11:         replace $\pi$ with $split_1(B_1, f, \pi)$;
12:     end for
13: end while
14: output $\pi$;
```

Note that at lines 8 and 11 of Algorithm 2, π might be refined by split operations, and that at line 6 π' might be refined by decomposing a compound block. Therefore, at every while loop, π is a refinement of π', and there exists a compound block in π' unless $\pi = \pi'$.

The correctness of Algorithm 2 can be proved almost in a similar manner as in the case of Algorithm 1. An important difference is that in Algorithm 2, we apply only $split_1$ and $split_2$ operations based only on B_1, and do not apply those operations based on $B - B_1$. However, we can show that the latter operations are not necessary for the construction of the coarsest refinement.

Lemma 7. *Let S be any subset of Q, π be a partition of Q, B be a block in π with $B \subseteq S$, and f be a function in $\overline{\Delta}$.*

(1) *If $split_1(S, f, \pi) = \pi$ holds, $split_1(B, f, \pi) = split_1(S - B, f, split_1(B, f, \pi))$ holds.*

(2) *If $split_2(S, \pi) = \pi$ holds, $split_2(B, \pi) = split_2(S - B, split_2(B, \pi))$ holds.*

Proof. We will show the proof of **(1)**. Let $\pi' = split_1(B, f, \pi)$. Let D be any block in π'. We will prove that either $f(D) \subseteq S - B$ or $f(D) \cap (S - B) = \emptyset$ holds.

Note that $\pi' = split_1(B, f, \pi')$ holds. Therefore, for any $D \in \pi'$, $f(D) \subseteq B$ or $f(D) \cap B = \emptyset$ holds.

In the case of $f(D) \subseteq B$, we have $f(D) \cap (S - B) = \emptyset$. Let us consider the case of $f(D) \cap B = \emptyset$. By $split_1(S, f, \pi) = \pi$ and Lemma 2, $split_1(S, f, \pi') = \pi'$ holds. Therefore, $f(D) \subseteq S$ or $f(D) \cap S = \emptyset$ holds. In the case of $f(D) \cap S = \emptyset$, we have $f(D) \cap (S - B) = \emptyset$. In the case of $f(D) \subseteq S$, by combining with $f(D) \cap B = \emptyset$, we obtain $f(D) \subseteq S - B$.

In conclusion, for any $D \in \pi'$, we have either $f(D) \subseteq S - B$ or $f(D) \cap (S - B) = \emptyset$. Therefore, $split_1(B, f, \pi) = split_1(S - B, f, \pi')$ holds. The proof of **(2)** can be obtained in a similar way. □

Theorem 5 (Correctness of Algorithm 2). *Let $M = (d, Q, q_0, F, w, h, s, \delta)$ be an LSA, and $n = |Q|$. Algorithm 2 for the input M is correct and terminates after at most $n - 1$ refinement steps, having computed the coarsest refinement of $\{F, Q - F\}$ satisfying* **(C1)** *and* **(C2)**.

Proof. We can show by induction and with Lemma 7 that, at each iteration of the while loop, $split_1(B, f, \pi) = \pi$ and $split_2(B, \pi) = \pi$ hold for any $B \in \pi'$ and $f \in \overline{\Delta}$, and that at each step, the current partition π is a refinement of the coarsest one. Then, we can show that Algorithm 2 outputs the coarsest refinement after at most $n - 1$ refinement steps. □

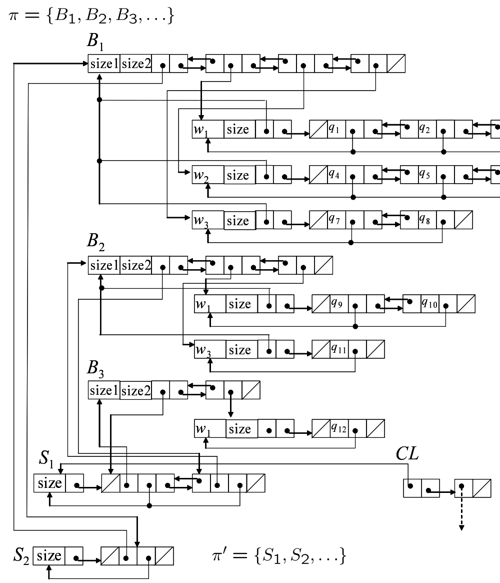

Fig. 3 Data Structure.

In the case of Algorithm 2, each state always moves to a block which is smaller than half of the current one, therefore, each state can move from a block to a block at most $O(\log n)$ times. Furthermore, using the data structure in Fig. 3, we can implement the inner blocks of the while loop in $O(m + |B_1|)$ time. If we notice that each edge is processed at most $O(\log n)$ times, the total time complexity is bounded by $O(m \log n)$ time.

We will describe several data structures necessary for the implementation of Algorithm 2. We will describe each element x by a record, which we shall not distinguish from the element itself. We represent each pair x, y such that $x\,f\,y$ or $x\,\Delta\,y$ by a record, which is called an edge. Each edge $x\,f\,y$ and $x\,\Delta\,y$ points to element x. Each element y points to a list of the edges $x\,f\,y$ and a list of edges $x\,\Delta\,y$. This allows the scanning of the set $f^{-1}(\{y\})$ or $\Delta^{-1}(\{y\})$ in time proportional to its size.

Concerning partitions π and π', we need data structures described in Fig. 3. A block B of π is represented as a record containing its sizes $|B|$ and $|W(B)|$ (indicated by $size1$ and $size2$, respectively, in Fig. 3), and a doubly linked list containing pointers to records of B_ω's for all $\omega \in W(B)$. Each B_ω is represented as a record containing its weight ω, its size $|B_\omega|$, a pointer to the record B and a doubly linked list of states contained in it. Each member (state) in the list also points to the record B_ω. For a block S of π', we define $Blk(S) = \{B \in \pi \mid B \subseteq S\}$. A block S of π' has a record containing its size $|Blk(S)|$ and a doubly linked list of pointers to records of blocks in $Blk(S)$, where the records of blocks in $Blk(S)$ also points to the corresponding elements of this list. Each element (pointer) of the list also points to the record S. We also maintain a doubly linked list CL of pointers of compound blocks of π'.

We can implement Algorithm 2 in the following way.

(1) Remove a compound block S from CL. If there exists no compound blocks, output π and halt. Otherwise, examine the first two blocks in S and let B_1 be the smaller.

(2) Remove B_1 from S and create a new block S' of π' containing only B_1. If S is still compound, put S back into CL.

(3) For all $D \in \pi$ such that $\Delta(D) \cap B_1 \neq \emptyset$, $\Delta(D) \not\subseteq B_1$ and $|W(D)| > 1$, apply the splitting operations defined in $split_2(B_1, \pi)$. Do this by scanning all the edges $x\Delta y$ such that $y \in B_1$. For this purpose, we prepare a counter $cnt(x)$ for each $x \in Q$, i.e., each state x has a counter $cnt(x)$ in its record, which is not described in Fig. 3 for its simplicity. To process an edge $x\Delta y$ with $y \in B_1$, the counter $cnt(x)$ is incremented by one. After scanning all the edges, we split D into D_1, D_2, and D'_ω 's, but, in the case of $|W(D)| = 1$, we do not split D. We can classify a state x into D'_ω if $0 < cnt(x) < deg(x)$ and $w(x) = \omega$, into D_1 if $cnt(x) = deg(x)$, into D_2 if $cnt(x) = 0$, respectively, in the $split_2$ operations, where size information is also updated appropriately. The updated counters should be linked together for later resetting. If $S'' \in \pi'$ containing D becomes compound, then add S'' to CL.

(4) For all $f \in \overline{\Delta}$, do the following procedure. For each $D \in \pi$ such that $f(D) \cap B_1 \neq \emptyset$ and $f(D) \not\subseteq B_1$, split D into $D_1 = D \cap f^{-1}(B_1)$ and $D_2 = D - D_1$. Do this by scanning edges $x\,f\,y$ such that $y \in B_1$. To process an edge $x\,f\,y$ with $y \in B_1$, determine $D \in \pi$ and $\omega \in W(D)$ such that D_ω contains x, create a temporary block D'_ω for D_ω and move x from D_ω to D'_ω. After scanning, a new block containing only D'_ω is added to π if D_ω is not empty, D_ω is just replaced by D'_ω otherwise (i.e., no change on D_ω), where size information is also updated appropriately. Temporary blocks D'_ω are linked together and have pointers to their original blocks D_ω for later process on them. If $S'' \in \pi'$ containing D becomes compound, then add S'' to CL.

The correctness of the implementation follows in a straightforward way from our discussion above.

Theorem 6 (Time Complexity of Algorithm 2)**.** *Let $M = (d, Q, q_0, F, w, h, s, \delta)$ be an LSA, and $n = |Q|$. The time complexity of Algorithm 2 for the input M is $O(Kn \log n)$.*

Proof. The preprocess for data structure at the initialization stage requires only $O(m+n)$ time. The time spent in a refinement step is $O(1)$ per edge scanned plus $O(1)$ per vertex of B_1, which results in the total time complexity $O(m \log n) = O(Kn \log n)$, since any element in Q can exist in at most $O(\log n)$ blocks in π. □

5. Further Improvement

In this section, we describe an implementation technique for improving the time complexity from $O(Kn \log n)$ to $O(kn \log n)$.

The idea comes from the observation that most of the edges are common to many graphs in $\overline{\Delta}$. Let us enumerate graphs in $\overline{\Delta}$ in the order $\delta_{\omega,v_1}, ..., \delta_{\omega,v_k}$ for each $\omega \in W(Q)$, where $H(\omega) = \{v_1, ..., v_k\}$ and $v_i < v_{i+1}$ for every $i = 1, ..., k-1$. Most edges might be common between adjacent graphs in this order. Let δ_{ω,v_0} be an empty graph for convenience. For each $i = 1, 2, ..., k$, define $E^+_{\omega,i}$ as a set of edges in the graph $\delta_{\omega,v_i} - \delta_{\omega,v_{i-1}}$, and $E^-_{\omega,i}$ as a set of edges in the graph $\delta_{\omega,v_{i-1}} - \delta_{\omega,v_i}$. Then, consider the following procedure: (1) let $f_{\omega,0}$ be an empty graph, (2) for each $i = 1, ..., k$, delete edges in $E^-_{\omega,i}$ from $f_{\omega,i-1}$ and add edges in $E^+_{\omega,i}$ to $f_{\omega,i-1}$, and construct a new graph $f_{\omega,i}$. This procedure generates a sequence of graphs $f_{\omega,1}, f_{\omega,2}, ..., f_{\omega,k}$ such that $f_{\omega,i} = \delta_{\omega,v_i}$.

Thus, it is not necessary to process all edges of $f_{\omega,i} = \delta_{\omega,v_i}$ for each $i = 1, ..., k$. We need to process edges in $E^-_{\omega,i}$ and $E^+_{\omega,i}$ at each step $i = 1, ..., k$, where the process for edges in $E^-_{\omega,i}$'s is the reversal computation of that for $E^+_{\omega,i}$'s. Then, the sum of the number of edges in $E^-_{\omega,i}$'s and $E^+_{\omega,i}$'s for all $\omega \in W(Q)$ is bounded by 2 times the number of edges in Δ, since every edge in Δ exists in at most one of the $E^-_{\omega,i}$'s and in at most one of the $E^+_{\omega,i}$'s. In other words, each edge in Δ is processed twice when adding and deleting it. Therefore, the time complexity can be improved to $O(kn \log n)$.

In order to implement the above idea, we need to keep temporary blocks B'_ω's during the process of the sequence $\delta_{\omega,v_1}, ..., \delta_{\omega,v_k}$ of graphs. More precisely speaking, for representing each subblock B_ω of a block B, we prepare two records $B_{\omega,org}$ and $B_{\omega,tmp}$ such that $B_{\omega,org}$ and $B_{\omega,tmp}$ are disjoint, the union of $B_{\omega,org}$ and $B_{\omega,tmp}$ corresponds to B_ω and $B_{\omega,tmp}$ keeps the states $q \in B_\omega$ satisfying $q \in f^{-1}(B_1)$, where B_1 is the selected block in the $split_1$ operation. In this way, we can always keep the information of $f^{-1}(B_1) \cap B_\omega$ for each B_ω during the process of a sequence of graphs $\delta_{\omega,v_1}, \delta_{\omega,v_2}, ..., \delta_{\omega,v_k}$. The computation of such information is the most essential part in the process of $split_1(B_1, f, \pi)$ operation. Then, the update when adding an edge $x\,f\,y$ in $E^+_{\omega,i}$ is moving its initial vertex x in the case of $y \in B_1$ from $B_{\omega,org}$ to $B_{\omega,tmp}$ as described in step (4) in the previous section. On the other hand, the update when deleting an edge $x\,f\,y$ in $E^-_{\omega,i}$ is just moving its initial vertex x in the case of $y \in B_1$ from $B_{\omega,tmp}$ to $B_{\omega,org}$, which corresponds to the reversal of the procedure described in step 4. For each $i = 1, ..., k$, after finishing the process for $E^-_{\omega,i}$ and $E^+_{\omega,i}$, we should check the emptiness of $B_{\omega,org}$'s and $B_{\omega,tmp}$'s. If both of $B_{\omega,org}$ and $B_{\omega,tmp}$ are not empty, we delete $B_{\omega,tmp}$ from B_ω and create a new block B' containing only the elements of $B_{\omega,tmp}$. But, this new block B' should be represented by a pair of $B'_{\omega,org}$ and $B'_{\omega,tmp}$ such that $B'_{\omega,org}$ is empty and $B'_{\omega,tmp} = B_{\omega,tmp}$. In this way, we can keep the information of $f^{-1}(B_1) \cap B'_\omega$ for this new block B'. Furthermore, we should note that in order to efficiently execute the update processes, at each iteration of i, we need to maintain updated temporary blocks linked together for these splitting processes.

6. Example Run of Algorithm 2

We will show a brief sketch of an example run of Algorithm 2 for the input LSA M_1 in Fig. 1. When representing a block of a partition of Q, we classify its states into subblocks based on their weight values, where each subblock is surrounded by a square bracket with its weight being a label.

Initially, two partitions π and π' are given as $\pi = \{B_1^{(0)}, B_2^{(0)}\}$ and $\pi' = \{Q\}$, where

$$B_1^{(0)} = \{w_1 : [q_2, q_4]\}, \quad B_2^{(0)} = \{w_1 : [q_1, q_3, q_5, q_7, q_{10}], w_2 : [q_6, q_8, q_9, q_{11}, q_{12}]\}.$$

A compound block Q contains $B_1^{(0)}$ and $B_2^{(0)}$, and therefore, the smaller block $B_1^{(0)}$ is selected. We first process the graph Δ. After scanning edges $x\Delta y$ with $y \in B_1^{(0)}$, the counters are given as follows.

$cnt(q_1) = 0,\ cnt(q_2) = 1,\ cnt(q_3) = 0,$
$cnt(q_4) = 1,\ cnt(q_5) = 2,\ cnt(q_6) = 2,$
$cnt(q_7) = 2,\ cnt(q_8) = 2,\ cnt(q_9) = 0,$
$cnt(q_{10}) = 0,\ cnt(q_{11}) = 0,\ cnt(q_{12}) = 0.$

Then, the partition π is obtained as $\pi = \{B_1^{(1)}, B_2^{(1)}, B_3^{(1)}\}$, where

$$B_1^{(1)} = \{w_1 : [q_2, q_4]\}, \quad B_2^{(1)} = \{w_1 : [q_5, q_7], \ \ w_2 : [q_6, q_8]\},$$
$$B_3^{(1)} = \{w_1 : [q_1, q_3, q_{10}], \ \ w_2 : [q_9, q_{11}, q_{12}]\}.$$

The partition π' is updated as $\pi' = \{B_1^{(0)}, B_2^{(0)}\}$. The partition π is not changed during the process for the graphs in $\overline{\Delta}$.

Next, we choose a compound block $B_2^{(0)}$ in π', which contains $B_2^{(1)}$ and $B_3^{(1)}$. The smaller block $B_2^{(1)}$ is selected.

We first process the graph Δ. After scanning edges $x\Delta y$ with $y \in B_2^{(1)}$, the counters are given as follows.

$cnt(q_1) = 2,\ \ cnt(q_2) = 0,\ \ cnt(q_3) = 2,$
$cnt(q_4) = 0,\ \ cnt(q_5) = 0,\ \ cnt(q_6) = 0,$
$cnt(q_7) = 0,\ \ cnt(q_8) = 0,\ \ cnt(q_9) = 0,$
$cnt(q_{10}) = 0,\ \ cnt(q_{11}) = 0,\ \ cnt(q_{12}) = 0.$

Then, the partition π is obtained as $\pi = \{B_1^{(2)}, B_2^{(2)}, B_3^{(2)}, B_4^{(2)}\}$, where

$$\begin{aligned} &B_1^{(2)} = \{w_1 : [q_2, q_4]\},\ \ B_2^{(2)} = \{w_1 : [q_5, q_7],\ \ w_2 : [q_6, q_8]\},\\ &B_3^{(2)} = \{w_1 : [q_1, q_3]\},\ \ B_4^{(2)} = \{w_1 : [q_{10}],\ \ w_2 : [q_9, q_{11}, q_{12}]\}.\end{aligned}$$

The partition π' is updated as $\pi' = \{B_1^{(1)}, B_2^{(1)}, B_3^{(1)}\}$. The partition π is not changed during the process for the graphs in $\overline{\Delta}$.

Next, we choose a compound block $B_3^{(1)}$ in π', which contains $B_3^{(2)}$ and $B_4^{(2)}$. The smaller block $B_3^{(2)}$ is selected.

We first process the graph Δ. After scanning edges $x\Delta y$ with $y \in B_3^{(2)}$, the counters are given as follows.

$cnt(q_1) = 1,\ \ cnt(q_2) = 0,\ \ cnt(q_3) = 1,$
$cnt(q_4) = 0,\ \ cnt(q_5) = 0,\ \ cnt(q_6) = 0,$
$cnt(q_7) = 0,\ \ cnt(q_8) = 0,\ \ cnt(q_9) = 2,$
$cnt(q_{10}) = 2,\ \ cnt(q_{11}) = 1,\ \ cnt(q_{12}) = 1.$

Then, the partition π is obtained as $\pi = \{B_1^{(3)}, B_2^{(3)}, B_3^{(3)}, B_4^{(3)}, B_5^{(3)}\}$, where

$$\begin{aligned} &B_1^{(3)} = \{w_1 : [q_2, q_4]\},\ \ B_2^{(3)} = \{w_1 : [q_5, q_7],\ \ w_2 : [q_6, q_8]\},\\ &B_3^{(3)} = \{w_1 : [q_1, q_3]\},\ \ B_4^{(3)} = \{w_1 : [q_{10}],\ \ w_2 : [q_9]\},\\ &B_5^{(3)} = \{w_2 : [q_{11}, q_{12}]\}.\end{aligned}$$

The partition π' is updated as $\pi' = \{B_1^{(2)}, B_2^{(2)}, B_3^{(2)}, B_4^{(2)}\}$. The partition π is changed only during the process for the graph $\Delta_{w_2,20}$ as follows.

$$\begin{aligned} &B_1^{(4)} = \{w_1 : [q_2, q_4]\},\ \ B_2^{(4)} = \{w_1 : [q_5, q_7],\ \ w_2 : [q_6, q_8]\},\\ &B_3^{(4)} = \{w_1 : [q_1, q_3]\},\ \ B_4^{(4)} = \{w_1 : [q_{10}],\ \ w_2 : [q_9]\},\\ &B_5^{(4)} = \{w_2 : [q_{11}]\},\ \ B_6^{(4)} = \{w_2 : [q_{12}]\}.\end{aligned}$$

No change on π occurs after this step, and thus, this is the final partition. Then, a minimum state LSA of M_1 is given in Fig. 2.

7. Conclusions

In this paper, we proposed an efficient algorithm which minimizes the number of states of a given LSA. The time complexity of the proposed algorithm is $O(kn \log n)$, where k is the maximum number of edges going out from a state of a given LSA, and n is the number of its states.

Future works include the application of LSAs to some classification problems containing real-valued time series data, and the development of theory and algorithms for learning LSAs from given sample data.

References

1) Aho, A.V., Hopcroft, J.E. and Ullman, J.D.: *Design and Analysis of Computer Algorithms*, Addison-Wesley (1974).
2) Hopcroft, J.E.: An $n \log n$ algorithm for minimizing states in a finite automaton, *Theory of Machines and Computations*, pp.189–196 (1971).
3) Matsunaga, T. and Oshita, M.: Recognition of Walking Motion Using Support Vector Machine, *ISICE2007*, pp.337–342 (2007).
4) Matsunaga, T. and Oshita, M.: Automatic estimation of motion state for motion recognition using SVM, *IPSJ SIG Technical Report*, Vol. 2008-CG-133, pp. 31–36 (2008).
5) Mohri, T. and Tanaka, H.: Weather Prediction by Memory-Based Reasoning, *Journal of Japanese Society for Artificial Intelligence*, Vol.10, No.5, pp.798–805 (1995).
6) Numai, Y., Udagawa, Y. and Kobayashi, S.: Minimization Algorithm of Linear Separation Automata, *IPSJ SIG Technical Report*, Vol.2010-MPS-77, pp.1–8 (2010).
7) Numai, Y., Udagawa, Y. and Kobayashi, S.: Theory of Minimizing Linear Separation Automata, *IPSJ Transactions on Mathematical Modeling and Its Applications*, Vol.3, No.2, pp.83–91 (2010).
8) R.Paige, R.E.T.: Three Partition Refinement Algorithms, *SIAM Journal on Computing*, Vol.16, No.6, pp.973–989 (1987).
9) Yamato, J., Ohya, J. and Ishii, K.: Recognizing human action in time-sequential images using hidden Markov model, *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp.379–385 (1992).