Incremental PDFA Learning for Conversational Agents

Masayuki Okamoto
Department of Social Informatics, Kyoto University
Yoshida Hommachi, Sakyo-ku, Kyoto, 606-8501, Japan
okamoto@kuis.kyoto-u.ac.jp

Abstract

When finite-state machines are used for dialogue models of a conversational agent, learning algorithms which learn cyclic probabilistic finite-state automata with the state merging method are useful. However, these algorithms should learn the whole data every time the number of example dialogues increases. Therefore, the learning cost is larger when we construct dialogue models gradually.

We propose a learning method which decreases the recomputation cost with caching the merging information, and evaluated the number of checking compatibilities for merging states and the perplexities of learned models. From the comparison among the dialogue models, the method which caches only the renewed states reduced the total number of checking compatibility by 13%. We also applied the algorithm to an actual conversational agent.

1. Introduction

So far, many conversational agents with finite state machines (FSMs) as their dialogue models have been developed [1][7]. Usually, each system is used for a limited domain, therefore its FSM is designed only for this domain. We should make up new FSMs whenever we change the scenarios or tasks.

Therefore, learning algorithms in which FSMs can be learned from examples are helpful for solving this problem. In general, it takes much time to collect example dialogues and people who collect dialogues should input many utterances or sentences. We proposed a framework to construct dialogue models incrementally through collecting example dialogues with the Wizard of Oz (WOZ) method [4] in which a human who pretends of a system (called wizard) talks with a user and learning dialogue models from collected dialogues [6].

Deterministic finite-state automata (DFA) learning algorithms are applicable for learning FSMs from examples.

In developing practical systems, there are some problems in example dialogues.

(1) Negative examples cannot be collected, though positive examples can be collected.

(2) Examples may include noises.

(3) The number of examples gradually increases.

If (1) and (2) should be solved, probabilistic deterministic finite-state automata (PDFA) with probabilistic parameters which can treat with noises are used. The first algorithm which learns PDFA is ALERGIA [3]. ALERGIA learns cyclic automata with the state-merging method. MDI [8] is a better version of ALERGIA. MDI uses the Kullback-Leibler divergence for decision of merging states.

However, these algorithm should recompute the whole steps when the number of example data increases. When the condition includes (3), these algorithms take much time. There is no algorithm which satisfies the three conditions, and algorithms which reduce the cost of recomputation are needed.

In this paper, we propose an algorithm which reduces recomputation costs against increasing examples by caching what states are merged in the previous learning session. We also examine how much the recomputation cost is reduced while the quality of learned models is not worse.

2. Preliminaries

A Probabilistic DFA (PDFA) is defined as a five-tuple \((Q, \Sigma, \delta, q_0, \gamma)\) where \(Q\) means a finite set of states, \(\Sigma\) means a finite set of alphabet symbols, \(\delta\) means a transition function \(Q \times \Sigma \rightarrow Q, q_0\) means the initial state, and \(\gamma\) means the next symbol probability function \(Q \times \Sigma \cup \# \rightarrow [0, 1]\) (\# means the end of string symbol).

In each state \(q\), the following equation is satisfied:

\[
\sum_{a \in \Sigma \cup \{\#\}} \delta(q, a) = 1
\]
A tree-formed PDFA in which the initial state is considered as a root is called a prefix tree acceptor (PTA). For example, if there are five inputs \{aa, baa, baa, bba, bbb\}, the PTA is shown as Figure 1. Each number \( s \) in a circle means state \( s \). In each tuple \([m, n]\) below a circle, \( m \) means the number how many times input strings passed the state (we define \( c(s) \) as this count), and \( n \) means the number how many times inputs finished at the state (we define \( c(s, \#) \) as this count). A double circle means that the state is an accepted state. Each symbol \( a[l] \) above a transition means that an alphabet \( a \) passed the transition \( l \) times (we define \( c(s, a) \) as this count).

![Figure 1. Example of PTA.](image)

In learning a PDFA with the state merging method, a PTA which can accept example inputs is made first, then a general PDFA is constructed by merging pairs of two states. The Kullback-Leibler divergence (or relative entropy) \( D(A||A') \) between two PDFA \( A = (Q, \Sigma, \delta, q_0, \gamma) \) and \( A' = (Q', \Sigma', \delta', q'_0, \gamma') \) is defined as the following expression [2]:

\[
D(A||A') = \sum_{q_i \in Q} \sum_{a \in \Sigma \cup \{\#\}} c_{ij} \gamma(q_i, a) \log \frac{\gamma(q_i, a)}{\gamma(q'_j, a)}
\]

where \( c_{ij} \) means the probability mass in \( A \) of the example inputs of prefixes common to state \( q_i \) in \( A \) and \( q'_j \) in \( A' \).

### 3. PDFA Learning Algorithm

This section describes polynomial-time learning algorithms which learn PDFA with the state merging method.

#### 3.1. State Merging

Merging of state \( s_1 \) and state \( s_2 \) means folding two subtrees of a PTA in which \( s_1 \) and \( s_2 \) are the roots of these subtrees (Figure 2).

Procedure \( \text{merge}(A, s_1, s_2) \) describes the process (Figure 3).

![Figure 2. Intuitive expression of merging two states.](image)

![Figure 3. Procedure merge.](image)

The quality of each algorithm changes according to the compatibility criterion (b) in Figure 4. If needed, preprocess (a) is used.

In MDI algorithm, the preprocess

\[
A' \leftarrow \text{merge}(A, s_j, s_i)
\]

is used for (a), and the compatibility measure

\[
\frac{D(PTA(I+)||A') - D(PTA(I+)||A)}{|A| - |A'|} < \alpha_M
\]

is used for (b).

\( A' \) is a PDFA which is derived by the merging of state \( s_i \) and state \( s_j \). \(|A| \) and \(|A'| \) are the numbers of \( A \) and \( A' \), respectively. \( \alpha_M \) is a parameter.

### 4. Incremental Learning

The algorithms shown in Section 3 assume that there is the whole learning set. When the number of data increases gradually, these algorithms should confirm all compatibilities again.

In this section, we try to reduce the cost of checking compatibilities by storing pairs of merged states and a set of states which need to be recomputed.

#### 4.1. Recomputing all possible changes

To extend a PDFA-learning algorithm to an incremental one, we need to manage the difference between the current
input: $I_+$ // a positive example set
$\alpha$ // a precision parameter
output: PDFA // probabilistic DFA

begin
$n$ means the number of states in $PTA(I_+)$
sorted in the lexicographic breadth-first order of $PTA(I_+)$
for $i = 1$ to $n - 1$
for $j = 0$ to $i - 1$ (needed preprocess)
if the compatibility is confirmed then (b)
$A \leftarrow \text{merge}(A, s_j, s_i)$
break
end if
end for
end for
return $A$
end

Figure 4. General form of quadratic algorithm by lexicographic breadth-first order.

PTA and the previous PTA, and states with potential that the compatibility changes.
When the phenomenon that a state which was merged in the previous process changes not to be merged or that a state which was not merged in the previous process changes to be merged occurs (we call the state compatibility-changed state), the following states have such kind of possibility:

1. states in a subtree of PTA in which the compatibility-changed state is the root
2. states on a path from the root of PTA to the compatibility-changed state (all predecessors of the compatibility-changed state in PTA)
3. states acquired by applying (1) and (2) to the states which are (or were) merged with the changed state

Figure 5 shows the range of above processes. The function $\text{effect}(s)$ returns these states when $s$ is changed.

Figure 5. Range of states which needs recomputation

In general, the changes to apply a PDFA-learning algorithm to an incremental situation are described below.
There are two additional inputs and the initial values, which is the set of pairs of merged states in the previous learning

$$M = \{(s_{01}, s_{02}), (s_{11}, s_{12}), \ldots, (s_{m1}, s_{m2})\}$$

and the set of states which need recomputation

$$R = \{s_{r1}, s_{r2}, \ldots, s_{rm}\}$$

where each $(s_{k1}, s_{k2})$ corresponds to $\text{merge}(A, s_{k1}, s_{k2})$ called directly from the algorithm in the previous learning, and each $s_k$ is the state corresponds to a prefix of added examples.

The changes about the process to confirm compatibilities and the postprocess are as follows:

- When $R$ does not include the target two states, that is, there is no difference from the previous learning, they are merged if $M$ includes the pair. If $M$ does not include the pair, this step is skipped.
- If at least one state belongs to $R$, the compatibility check is examined.
  - If the target two states are compatible, two states are merged. Then, $\text{effect}(s)$ for these two states and the merged state are added to $R$.
  - If the target two states are not compatible and the pair is in $M$, that is, the two states become to be not compatible, two states and $\text{effect}(s)$ for them are added to $R$.
  - If the target two states are not compatible and the pair is not in $M$, that is, there is no change, this step is skipped.

This means that the decision of compatibility involving a state which needs recomputation follows

- the previous result when the decision of compatibility is not changed, or
- the result after the renewal of $R$ and recomputation when the decision of compatibility is changed.

Figure 6 is a procedure that above renewing process is added to the merge procedure described in Figure 3, and Figure 7 is an algorithm that above renewing process is added to the algorithm described in Figure 4.

4.2. Recomputing only changed states

In above algorithm, when mergings which involve states near the initial state (or the root of the PTA) occur, almost all states are included in the recomputation set. Therefore, the cost of checking compatibilities may not reduce.
input: PDFA $A$
states $s_1, s_2 (s_1 \neq s_2)$
// a set including pairs of merged states
$M \leftarrow \{(s_{01}, s_{02}), (s_{11}, s_{12}), \ldots, (s_{m1}, s_{m2})\}$
// a set including states which need recomputation
$R \leftarrow \{s_{r1}, s_{r2}, \ldots, s_{rm}\}$
output: merged PDFA

begin
$c(s_1) \leftarrow c(s_1) + c(s_2)$
$c(s_1, a) \leftarrow c(s_1, a) + c(s_2, a), \forall a \in \Sigma \cup \{\#\}$
if $s_2 \in R$ then
  // because state $s_2$ which will be merged is in $R$,
  // add state $s_1$ which includes state $s_2$ to $R$
  $R \leftarrow R \cup \{s_1\}$
else if
  remove $(s_h, s_2)$ except for $(s_1, s_2)$ from $M$
if both $\delta(s_1, a)$ and $\delta(s_2, a)$ exist for $a \in \Sigma$ then
  merge $(A, \delta(s_1, a), \delta(s_2, a), M, R)$ // merging recursively
end if
$A \leftarrow A - \{s_2\}$
return $A$
end

Figure 6. Procedure incremental_merge with renewal of recomputation set

On the other hand, the recomputation occurs in various states of the PDFA in the condition that learning data increase gradually. It means that we can acquire the result almost same in quality even if we do not consider all possible changes.
The algorithm which merges only changed states is what the two lines $(*)$ and $(**)$ are removed from Figure 7.

5. Evaluation

In this section, we examined the following two items:
1. the quality of the learned model
2. the number of comparisons for confirming compatibilities

We used dialogues about Kyoto tour guide for the evaluation. This dialogues consist of speakers of university students in Kyoto and an interface agent by the Wizard of Oz method, and each utterance has an utterance tag which means the type of the utterance such as a greeting, a request, an answer, and so on. A PDFA constructed from the strings of utterance tags is a dialogue model which can be used in dialogue systems.

We used 90 dialogues of 99 dialogues for the learning set, and the other 9 dialogues for the test set. We examined the number of comparisons for compatibility and the quality of learned dialogue models for a normal PDFA-learning algorithm, an incremental algorithm with effect$(s)$, and an incremental algorithm without effect$(s)$ which recomputes only compatibility-changed states.

We used the MDI algorithm as the target PDFA-learning algorithm.

5.1. The number of comparisons for compatibility

Figure 8 shows the number of decisions for the merging method. In this evaluation, we used $\alpha_M = 0.1$ and $\beta = 0.5$ for the parameters.

From Figure 8, the algorithm with effect$(s)$ does not reduce the recomputation cost. It is because when a merge involves a state near the initial state, almost all states are included in effect$(s)$, therefore the algorithm works almost same as the usual MDI.

On the other hand, the algorithm without effect$(s)$ reduced the number of comparison 5% in average and 13% in
The number of learned data

The number of comparisons for compatibility (total)

Figure 8. Comparison of the count for decision of merging.

5.2. Perplexity

We used the test set perplexity $PP$ to examine the quality of learned models. It is given by $PP = 2^{LL}$.

When $n$ strings $S = \{s_1, \ldots, s_n\}$ (each $s_k$ consists of series of $m_k$ symbols $x_{1i}^{k} \cdots x_{mj}^{k}$) are input into the PDFA, the per symbol log-likelihood of strings is denoted by:

$$LL = -\frac{1}{||S||} \sum_{i=1}^{n} \sum_{j=1}^{m_i} \log P(x_{ij}^{k} | q_{ij}^{k})$$

where $P(x_{ij}^{k} | q_{ij}^{k})$ denotes the probability (weight) that $i$-th symbol $x_{ij}^{k}$ in the string $s_i$ will be input in state $q_{ij}^{k}$. $||S||$ denotes the number of symbols in all strings.

When the transition probability $\delta(q_{ij}^{k}, x_{ij}^{k})$ is used as $P(x_{ij}^{k} | q_{ij}^{k})$, the perplexity cannot be measured if there is an input which is not in the transition. Therefore, we also use the simple occurrence probability (unigram) of each symbol in the test data.

When $P(x)$ denotes the occurrence probability of symbol $x$ in all inputs,

$$P(x_{ij}^{k} | q_{ij}^{k}) = \beta \delta(q_{ij}^{k}, x_{ij}^{k}) + (1 - \beta)P(x_{ij}^{k})$$

where $\beta (0 \leq \beta \leq 1)$ is a parameter.

Figure 9 shows the comparison of the perplexities between the MDI and the incremental algorithm which recomputes only renewed states.

From Figure 9, we found the algorithm without $effect(s)$ does not make perplexities worse so much.

As a result, the method which manages only states in which the decision of merging changed reduced the number of comparisons without increase of perplexity, while the algorithm which includes $effect(s)$ in the set of states which need recomputation does not perform more than the usual algorithm.

5.3. Actual Use in a Conversational Agent

We also applied the incremental algorithm without $effect(s)$ to our WOZ-based conversational agent [6]. In this system, there are a client for a user, a client for the wizard, and the inference engine.

The agent system works as follows:

1. When a user inputs an utterance, the system searches pairs of a user’s utterance and the agent’s response from the collected example dialogues. Each pair should satisfy that state transitions are possible from the current state with the utterance tags. For such utterances, the products of the keyword vectors between the user’s input and utterances in the history are calculated, and the utterances with the highest scores are shown to the wizard.

2. When the wizard selects a candidate, the inferred utterance and its utterance tag are sent to the system. If there are not any proper candidates, the wizard selects an utterance from the preset utterances or inputs a sentence, and annotates both the user’s utterance and the agent’s utterance (this work will be the wizard’s load). Finally, the state transitions are executed according to the utterance tags.

3. In any cases, the utterance is spoken by the interface agent (the user thinks that the agent talks always).

We examined how much the quality of the agent’s inference improved as the number of example dialogues increased.

We used another 41 dialogues (1147 utterances including 409 user utterances) in Kyoto tour guide task. We used 31
dialogues for the learning data, and 10 dialogues for the test
data.

Figure 10 shows the change of the ratio of modifications
by the wizard. The horizontal axis means the number of
learned dialogues (1, 10, 20, and 31 dialogues). The ver-
tical axis means the ratio of the wizard’s modifications to
candidate answers from the system (0 means the system in-
fers all utterances correctly, and 1 means the wizard inputs
all utterances). In Figure 10, there are two graphs. One is
the ratio when the system infers only one candidate. The
other is the ratio when the system infers three candidates,
and the inference is considered as correct if one of these
candidates is appropriate.

![Graph showing the change of the wizard’s load](image)

**Figure 10. The change of the wizard’s load**

From Figure 10, we found the wizard’s load decreased
as the learning process made progress. We also found that
our algorithm totally reduced 25% checking compatibilities
from the original learning algorithm.

On condition that the system infers three candidates, the
quality is continuously increasing, while the quality of di-
ologue models does not improve from 20-example condi-
tion when the system infers only one utterance. It implies
that we can improve the system if we consider the scoring
method of utterance recognition and the process of the nar-
rowing candidates.

As the result, we reduced the two kinds of costs to con-
struct conversational agent. That is, we reduced the cost to
input example dialogues and the cost to learn the dialogue
models.

6. Conclusion

We proposed two methods for reducing the recomputation
cost of PDFA-learning algorithms in the situation that
collecting examples takes much cost and the number of ex-
amples increases gradually. In these methods, the pairs
of merged states and states which need recomputation are
cached. We also evaluated the number of comparisons for
confirming compatibilities and the quality of learned model.

From the experiment with tour guide dialogues, we found that

- the method which manages all states which need re-
computation did not performed well, and
- the method which manages only states in which the
decision of merging changed performed 13% redu-
tion of the number of comparisons without increase of
perplexity.

We also applied the algorithm to an actual conversational
agent, and we confirmed that the cost to input example di-
alogues and to learn the dialogue models were reduced. We
think the tour-guide agent will be applicable in Digital
City[5].

This kind of methods which reduces the recomputation
cost are useful for domains in which it is hard to construct
FSMs in advance and they are constructed gradually from
actual examples.

Acknowledgements

We would like to thank professor Toru Ishida at Depart-
ment of Social Informatics, Kyoto University for his ad-
vice. The work is partially supported by the Japan Science
and Technology CREST Universal Design of Digital City
Project. We used example dialogues collected by the New
Energy and Industrial Technology Development Organiza-
tion for Senior Support System Project.

References

[1] J. Alexandersson, E. Maier, and N. Reithinger. A robust and
efficient three-layered dialogue component for a speech-to-
speech translation system. In *Proc. EACL-95*, pages 188–193,
1995.

[2] R. Carrasco. Accurate computation of the relative entropy be-
tween stochastic regular grammars. *Theoretical Informatics

ICGI-94*, volume 862 of *Lecture Notes in Artificial Intelli-


[5] T. Ishida. Digital city kyoto: Social information infras-

structing conversational web agents. *Trans. JSAI*, 17(3):293–

[7] A. Stent, J. Dowding, J. M. Gawron, E. O. Bratt, and

inference using kullback-leibler divergence and minimality.