JMLR: Workshop and Conference Proceedings 21:19–33, 2012 The 11th ICGI

Actively Learning Probabilistic Subsequential Transducers

Hasan Ibne Akram HASAN.AKRAM@SEC.IN.TUM.DE
Department of Computer Science, Technische Universität München
Boltzmannstraße 3, 85748 Garching, Germany

Colin de la Higuera CDLH@UNIV-NANTES.FR
LINA, Nantes University, Nantes, France

Claudia Eckert CLAUDIA.ECKERT@SEC.IN.TUM.DE
Department of Computer Science, Technische Universität München
Boltzmannstraße 3, 85748 Garching, Germany

Editors: Jeffrey Heinz, Colin de la Higuera and Tim Oates

Abstract

In this paper we investigate learning of probabilistic subsequential transducers in an active learning environment. In our learning algorithm the learner interacts with an oracle by asking probabilistic queries on the observed data. We prove our algorithm in an identification in the limit model. We also provide experimental evidence to show the correctness and to analyze the learnability of the proposed algorithm.

Keywords: active learning, probabilistic transducer learning

1. Introduction

In the active learning paradigm used for language learning tasks, the learner has access to an *oracle* or a *minimally adequate teacher* (MAT) (which can in practice be a corpus, a human expert or the Web) and the learner is able to interact with this oracle. The learner generates strings and queries the oracle about the strings (Angluin and Smith, 1983; Angluin, 1987a). In the field of grammatical inference, traditionally, the learner generates the data it needs and makes a membership query, asking if the string is or isn't in the language: the learner asks queries about data which has not been observed. In practice, querying about unseen data or data artificially generated by the learner can sometimes lead to problems: the oracle may find it difficult to classify this *nonsense* data. This has been described and analyzed in (Lang and Baum, 1992); a recent survey in active learning and discussion can be found in (Settles, 2011).

To mitigate such practical issues, as conversed to the L^* approach (Angluin, 1987a) where the learner can ask questions about any data, in this paper we work on a scenario where positive data is given to the learner and the learner is only allowed to ask queries about the observed data. Furthermore, we make use of the extra information that exists when the data has been randomly drawn following an unknown distribution, distribution itself generated by a finite state machine. We present a novel learning algorithm for learning probabilistic subsequential transducers (PSTS), which are deterministic transducers with probabilities. The algorithm learns from positive examples by making probabilistic queries only regarding the data present in the training sample.

The problem of learning subsequential transducers from a given set of positive examples in the limit was first addressed by Oncina *et al.* in their seminal paper (Oncina and García, 1991; Oncina et al., 1993) where they presented their well known Ostia algorithm, which is based on state merging strategies similar to those used in Dfa learning (Oncina and García, 1992). Afterwards, there has been work done with regards to transducer learning in an active learning setting (Vilar, 1996; Oncina, 2008) and also from positive presentation (Oncina and Varó, 1996; Wakatsuki and Tomita, 2010). Moreover, heuristics have been applied to adopt Fst learning to machine translation (Vilar, 2000; Vidal and Casacuberta, 2004) and bioinformatics (Peris and López, 2010). In a recent work Clark presented an algorithm for learning inversion transduction grammar in an identification in the limit model (2011).

The new proposed algorithm is essentially the hybridization of the state merging and active learning paradigms. In our proposed algorithm we build a tree transducer from the observed positive data, which is an exact representation of the training data and ask probabilistic queries only regarding the observed data. In our algorithm, instead of asking queries about the data that are not present in the training set, we utilize the lack of information to make state merging decisions. This brings an improvement over Ostia: while Ostia can only learn transduction schemes which are *total functions*, the proposed algorithm is also capable of learning transduction schemes which are *partial functions*. We prove the correctness of our algorithm in an identification in the limit model. Moreover, we report experimental evidence that shows that our algorithm converges with relatively few training examples and produces an acceptable translation accuracy.

2. Definitions and Notations

Let $[n]$ denote the set $\{1, \ldots, n\}$ for each $n \in \mathbb{N}$. An *alphabet* Σ is a non-empty set of symbols and the symbols are called *letters*. Σ^* is a *free-monoid* over Σ. Subsets of Σ^* are known as (*formal*) *languages* over Σ. A *string* w over Σ is a finite sequence $w = a_1 \ldots a_n$ of letters. Let $|w|$ denote the length of the string w. In this case we have $|w| = |a_1 \ldots a_n| = n$. The empty string is denoted by ϵ. For every $w_1, w_2 \in \Sigma^*$, $w_1 \cdot w_2$ is the concatenation of w_1 and w_2. The concatenation of ϵ and a string w is given by $\epsilon \cdot w = w$ and $w \cdot \epsilon = w$. When decomposing a string into substrings, we will write $w = w_1, \ldots, w_n$ where $\forall i \in [n], w_i \in \Sigma^*$. If $w = w_1 w_2$ is a string, then w_1 is a *prefix* and w_2 is a *suffix* of the string w. Given a language $\mathcal{L} \subseteq \Sigma^*$, the *prefix set* of $\mathcal{L}$ is defined as $\mathit{Pref}(\mathcal{L}) = \{u \in \Sigma^* : \exists v \in \Sigma^*, uv \in \mathcal{L}\}$ and the *suffix set* of $\mathcal{L}$ is defined as $\mathit{Suff}(\mathcal{L}) = \{v \in \Sigma^* : \exists u \in \Sigma^*, uv \in \mathcal{L}\}$. $\mathit{Pref}(w)$ and $\mathit{Suff}(w)$ are defined as the set of all the substrings of w that are prefixes and suffixes of w correspondingly. The *longest common prefix* of $\mathcal{L}$ is denoted as $\mathit{lcp}(\mathcal{L})$, where $\mathit{lcp}(\mathcal{L}) = w \iff w \in \bigcap_{x \in \mathcal{L}} \mathit{Pref}(x) \wedge \forall w' \in \bigcap_{x \in \mathcal{L}} \mathit{Pref}(x) \Rightarrow |w'| \leq |w|$. Less formally, *lcp* is a function that returns the longest possible string which is the prefix of all the strings in a given set of strings. For example, for $\mathcal{L} = \{aabb, aab, aababa, aaa\}$ the $\mathit{lcp}(\mathcal{L})$ is aa.

2.1. Distributions

A *stochastic language* $\mathcal{D}$ is a probability distribution over Σ^*. The probability of a string $x \in \Sigma^*$ under the distribution $\mathcal{D}$ is denoted as $Pr_{\mathcal{D}}(x)$ and must satisfy $\sum_{x \in \Sigma^*} Pr_{\mathcal{D}}(x) = 1$. If the distribution is modelled by some syntactic machine M, the probability of x according to the probability distribution defined by M is denoted as $Pr_M(x)$. The distribution modelled by a machine M will be denoted as $\mathcal{D}_M$ and simplified to $\mathcal{D}$ if the context is not ambiguous.

2.2. Stochastic Transductions

In order to represent *transductions* we now use two alphabets, not necessarily distinct, Σ and Ω. We use Σ to denote the input alphabet and Ω to denote the output alphabet. For technical reasons, to denote the end of an input string we use a special symbol $\sharp$ as an end marker.

A *stochastic transduction* $\mathcal{R}$ is given by a function $Pr_{\mathcal{R}}\colon \Sigma^*\sharp \times \Omega^* \to \mathbb{R}_+$, such that :

$$\sum_{u \in \Sigma^*\sharp} \sum_{\mathbf{v} \in \Omega^*} Pr_{\mathcal{R}}(u, \mathbf{v}) = 1,$$

where $Pr_{\mathcal{R}}(u, \mathbf{v})$ is the joint probability of u and $\mathbf{v}$. Otherwise stated, a stochastic transduction $\mathcal{R}$ is the joint distribution over $\Sigma^*\sharp \times \Omega^*$. Let $L \subset \Sigma^*\sharp$ and $L' \subset \Omega^*$;

$$Pr_{\mathcal{R}}(L, L') = \sum_{u \in L} \sum_{\mathbf{v} \in L'} Pr_{\mathcal{R}}(u, \mathbf{v}).$$

Example 1 *The transduction $\mathcal{R}\colon \Sigma^*\sharp \times \Omega^* \to \mathbb{R}_+$ where $Pr_{\mathcal{R}}(a^n\sharp, \mathbf{1}^n) = \frac{1}{2^n}, \forall n > 0$, and $Pr_{\mathcal{R}}(u, \mathbf{v}) = 0$ for every other pair.*

In the sequel, we will use $\mathcal{R}$ to denote a stochastic transduction and T to denote a transducer. Note that the end marker $\sharp$ is needed for technical reasons only. The probability of generating a $\sharp$ symbol is equivalent to the stopping probability of an input string.

2.3. Probabilistic Subsequential Transducers

A transduction scheme can be modelled by transducers or probabilistic transducers. In this section, we will define *probabilistic subsequential transducers* (Pst) that can be used to model a specific subclass of stochastic transductions. The definitions presented in this section are inspired by a number of work in machine learning, pattern recognition, language processing, and automata theory including (Salomaa and Soittola, 1978; Reutenauer and Schützenberger, 1991, 1995; Oncina et al., 1993; Allauzen and Mohri, 2003; Vidal et al., 2005a,b; Mohri, 2009).

Definition 2.1 *A **probabilistic subsequential transducer** (Pst) defined over the probability semiring $\mathbb{R}_+$ is a 5-tuple $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E\rangle$ where:*

- *Q is a non-empty finite set of states,*
- *$q_0 \in Q$ is the unique initial state,*
- *Σ and Ω are the sets of input and output alphabets,*
- *$E \subseteq Q \times \Sigma \cup \{\sharp\} \times \Omega^* \times \mathbb{R}_+ \times Q$, and given $e = (q, a, \mathbf{v}, \alpha, q')$ we denote: $prev\,[e] = q$, $next\,[e] = q'$, $i\,[e] = a$, $o\,[e] = \mathbf{v}$, and $prob\,[e] = \alpha$,*
- *the following conditions hold:*
  - $\forall q \in Q$,
    $\forall (q, a, \mathbf{v}, \alpha, q'), (q, a', \mathbf{v}', \beta, q'') \in E$, $a = a' \Rightarrow \mathbf{v} = \mathbf{v}',\ \alpha = \beta,\ q' = q''$,

- $\forall q \in Q, \sum_{a \in \Sigma \cup \{\sharp\}, q' \in Q} Pr(q, a, q') = 1,$
- $\forall (q, a, \mathbf{v}, \alpha, q') \in E, a = \sharp \Rightarrow q' = q_0.$

Note that in some related work, the definition of subsequential transducers admits state outputs (*e.g.*, (Mohri, 2009; Mohri et al., 2002)). For technical convenience and without loss of generality we have substituted the state outputs and the final state probabilities by edges with input symbol $\sharp$.

A probability distribution $\mathcal{R}$ is a *stochastic deterministic regular transduction* (SDRT) if it is produced by a PST.

The quotient $(u, \mathbf{v})^{-1}\mathcal{R}$ where $u \in \Sigma^*$ and $\mathbf{v} \in \Omega^*$ is the stochastic transduction that obeys the following property:

$$Pr_{(u,\mathbf{v})^{-1}\mathcal{R}}(w\sharp, \mathbf{w}') = \frac{Pr_{\mathcal{R}}(uw\sharp, \mathbf{v}\mathbf{w}')}{Pr_{\mathcal{R}}(u\Sigma^*\sharp, \mathbf{v}\Omega^*)}$$

If $Pr(u\Sigma^*, \mathbf{v}\Omega^*) = 0$, then by convention $(u, \mathbf{v})^{-1}\mathcal{R} = \emptyset$ and $Pr_{(u,\mathbf{v})^{-1}\mathcal{R}}(w, \mathbf{w}') = 0$. If $\mathcal{R}$ is SDRT, the number of different stochastic sets of $(u, \mathbf{v})^{-1}\mathcal{R}$ is finite.

At this point we introduce the concept of *onward* (Oncina et al., 1993) PSTs, which is required to define the minimal canonic form of PST that our learning algorithm infers.

Definition 2.2 *A* PST $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E\rangle$ *is said to be* in onward form *if the following property holds:*

$$\forall q \in Q \backslash \{q_0\}, lcp\left(\bigcup_{e \in E[q]} \{o\,[e]\} \right) = \epsilon.$$

The onward form makes sure that a translation is given by the PST as early as possible.

We construct the minimal canonical PST $M = \left\langle Q^M, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}^M, E^M \right\rangle$ in onward form as the following:

$$\begin{array}{ll}
Q^M & = \{(u, \mathbf{v})^{-1}\mathcal{R} \neq \emptyset, u \in \Sigma^*, \mathbf{v} \in \Omega^*\} \\
\{q_0\}^M & = \{(\epsilon, \mathbf{v})^{-1}\mathcal{R}\} \\
E^M & = \{(q, a, \mathbf{w}, \alpha, q')| \\
& \quad q = (u, \mathbf{v})^{-1}\mathcal{R} \in Q^M, \\
& \quad q' = (ua, \mathbf{v}\mathbf{v}')^{-1}\mathcal{R} \in Q^M \text{ where,} \\
& \qquad a \in \Sigma \cup \{\sharp\}, \mathbf{v}' \in \Omega^*, \\
& \quad \mathbf{w} = lcp(\{\mathbf{v}|(u\Sigma^*, \mathbf{v}\Omega^*) \in \mathcal{R}\})^{-1} \\
& \qquad lcp(\{\mathbf{v}\mathbf{v}'|(ua\Sigma^*, \mathbf{v}\mathbf{v}'\Omega^*) \in \mathcal{R}\}), \\
& \quad \alpha = \frac{Pr_{\mathcal{R}}(ua\Sigma^*, \mathbf{v}\mathbf{v}'\Omega^*)}{Pr_{\mathcal{R}}(u\Sigma^*, \mathbf{v}\Omega^*)}\}
\end{array}$$

The canonical PST generates transductions that are in $\mathcal{R}$ and have non-zero probabilities.

When learning, the algorithm will be given a randomly drawn sample: the pairs of strings will be drawn following the joint distribution defined by the target PST. Therefore, such a sample is a multiset, since more frequent translation pairs may occur more than once.

3. The Inference Algorithm

In the domain of grammatical inference, queries (Angluin, 1981, 1987a,b) have been used to learn different types of automata including transducers (Vilar, 1996). There are various types of queries that have been used such as membership queries (Angluin, 1987a,b), equivalence queries (Angluin, 1987a,b), extended membership queries (Bergadano and Varricchio, 1996), and translation queries (Vilar, 1996). In our proposed algorithm we will use *extended prefix language queries*. Extended prefix language queries were introduced by de la Higuera and Oncina in (2004) where such queries have been used for identification of *probabilistic finite state automata* (Pfas) in the limit.

Definition 3.1 ***Extended prefix language queries*** *(*EXPQ*) are made by submitting a string w to an oracle. Then the oracle returns the probability $Pr_{\mathcal{D}}(w\Sigma^*)$, i.e., the probability of w being a prefix of the stochastic language $\mathcal{D}$.*

EXPQs are used to obtain probabilities regarding the training data. In our learning environment the learner is not allowed to ask queries on any data outside the training sample. However, any string $w' \notin \mathcal{D}$ has a probability $Pr_{\mathcal{D}}(w'\Sigma^*) = 0$. In order to incorporate such information in the learning algorithm, we introduce another term called a *phantom.* A phantom φ *w.r.t.* a Ptst $T\left\langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E\right\rangle$ is a 3-tuple $\varphi = \langle q_u, e, q'_u\rangle$ where, $q_u \in Q$, $e \notin E$ and $q'_u \notin Q$. The construction of each of these edges e is the following:

- $prev\left[e\right] = q_u$,
- $i\left[e\right] = a$, such that $a \in \{\{\Sigma \cup \{\sharp\}\}\backslash\{b | \forall e' \in E\left[q_u\right], i\left[e'\right] = b\}\}$,
- $o\left[e\right] = \perp$,
- $prob\left[e\right] = 0$,
- $next\left[e\right] = q'_u = q_{ua}$.

Figure 1 depicts examples of phantoms with dotted lines. Phantoms are edges and states in the Ptst that do not exist in the target Pst.

The proposed algorithm Apti (Algorithm for Probabilistic Transducer Inference) (Algorithm 1) consists of four phases: 1. building a tree transducer (Definition 3.2) in onward form, which is the exact representation of the training data, 2. populating the probabilities of the edges of the tree transducer by means of EXPQ on the observed data, 3. adding phantoms to the tree transducer with zero probability for those states $q \in Q$, where:

$$\sum_{e \in E[q]} prob\left[e\right] = 1 \ \wedge \ \left|E\left[q\right]\right| < \left|\Sigma \cup \{\sharp\}\right| \tag{1}$$

and 4. the state merging phase, where we iteratively keep on merging states keeping the hypothesis transducer consistent with the training sample till the algorithm terminates. In this section we describe the details of each phase of Apti.

During the run of the learning algorithm, we may have transitions for which at a given time the outputs are still unknown. In order to denote the outputs of the transitions that are still unknown, we introduce a new symbol $\perp$ such that, $\forall a \in \ \Omega^*, lcp(\{a\} \cup \{\perp\}) = a$ and $\forall a \in \Omega^*, \perp \cdot u = u \cdot \perp = \perp$.

Definition 3.2 *A* ***probabilistic tree subsequential transducer*** *(*Ptst*) is a 5-tuple* $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E\rangle$ *where* $\psi(T) = \langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E\rangle$ *is a* Pst *and* T *is built from a training sample* S_n *such that:*

- $Q = \bigcup_{(u,\mathbf{v})\in S_n} \{q_x \colon x \in \mathit{Pref}(u)\}$,
- $E = \{e \mid \mathit{prev}\,[e] = q_u, \mathit{next}\,[e] = q_v \Rightarrow v = ua, a \in \Sigma, i\,[e] = a, o\,[e] = \epsilon\}$,
- $\forall q_u \in Q, \forall e \in E\,[q]\,,\ i\,[e] = \sharp, o\,[e] = \mathbf{v}$ *if* $(u, \mathbf{v}) \in S_n, \perp$ *otherwise,*
- $\forall e \in E\,[q_0]\,,\ \mathit{prob}\,[e] = \mathrm{EXPQ}(i\,[e]\,\Sigma^*)$,
- $\forall q_u \in Q \setminus \{q_0\}, \forall e \in E\,[q_u]\,,\ \mathit{prob}\,[e] = \frac{\mathrm{EXPQ}(ui[e]\Sigma^*)}{\mathrm{EXPQ}(u\Sigma^*)}$.

Algorithm 1: Apti

```
Input: a sample $S_n$
Output: a Pst $T$
$T \leftarrow$ OnwardPtst($S_n$);
$T \leftarrow$ PopulateByQuery($T, S_n$);
$T \leftarrow$ AddPhantoms($T$);
Red $\leftarrow \{q_\epsilon\}$;
Blue $\leftarrow \{q_a \colon a \in \Sigma \cap \mathit{Pref}(S_{\Sigma^*})\}$;
while Blue $\neq \emptyset$ do
    $q =$ Choose$_{<_{lex}}$(Blue);
    for $p \in$ Red in lex-length order do
        ($T'$, IsAccept) $\leftarrow$ Merge($T, p, q$);
        if IsAccept then
            $T \leftarrow T'$;
        end
        else
            Red $\leftarrow$ Red $\cup \{q\}$;
        end
        Blue $\leftarrow$ $\{q \colon \forall p \in$ Red$, \forall e \in E\,[p]\,,$
                 $q = \mathit{next}\,[e] \wedge q \notin$ Red$\}$;
    end
end
$T \leftarrow$ RemovePhantoms($T$);
return $T$;
```

The algorithm Apti (Algorithm 1) starts with building a Ptst from the training data and asking queries (EXPQ) to a Mat according to the Definition 3.2. The two hypothetical functions OnwardPtst and PopulateByQuery used in Algorithm 1 perform these tasks. The formal construction of the Ptst from S_n and by EXPQ is shown in Definition 3.2. It is important to note that the states in the Ptst are numbered by the *lex-length* order of the input strings. We will denote the number of states if of a Ptst T as $|T|$.

After having built the onward Ptst, in the third phase of the algorithm, phantoms are added to the Ptst by the hypothetical routine AddPhantoms. We add such edges for every state in the Ptst where the conditions in (1) hold.

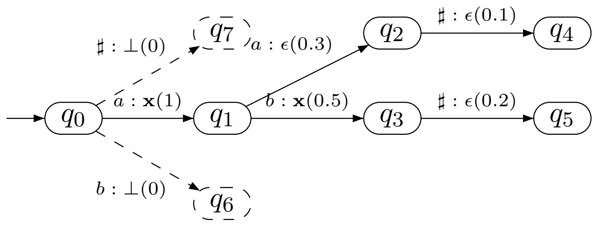

Figure 1: An onward Ptst built from $S_2 = \{(aa\sharp, \mathbf{x}), (ab\sharp, \mathbf{xx})\}$ and by asking EXPQ. Then the phantoms are added (dotted lines). The labels of the edges should be interpreted as *input*:**output**(probability).

Figure 1 shows an example of a Ptst where phantoms have been added. Notice that in this particular example, it is only possible to add phantoms at the state q_0. After populating the tree with probabilistic queries (EXPQ) we know that the probabilities of the edges from q_0 with input symbols b and $\sharp$ are zero, and hence why we add the two phantoms with probability 0. Syntactically these edges are of no use. However, this extra bit of information in the Ptst improves the learning capacities of the algorithm.

Finally, the state merging phase of Apti is similar to the Ostia algorithm, with modified state merging strategy. The details of Ostia can be found in (Oncina et al., 1993; Castellanos et al., 1998; de la Higuera, 2010). Here we follow the recursive formalism given in (de la Higuera, 2010).

The proposed algorithm Apti (Algorithm 1) selects a candidate pair of Red and Blue states in lex-length order using the Choose$_{<_{lex}}$ function. The Merge function merges the two selected states and recursively performs a cascade of folding of a number of edges (see (de la Higuera, 2010) for details). As a result of the onwarding process, some of the output strings may come too close to the initial state. During the run of the algorithm these strings or the suffixes of these strings are pushed back in order to make state merging possible by deferring the translations and this is done in the standard way as in Ostia. During the recursive fold operation, it is actually decided whether a merge is accepted or not. A merge is rejected if any one of the following holds: 1) if there is a conflict *w.r.t.* the outputs of any two edges having the input symbol $\sharp$, 2) if there is a conflict *w.r.t.* the probabilities of any two edges.

4. Analysis of the Algorithm

We define the *prefix set* ($\mathbb{PR}$) and the *short prefix set* ($\mathbb{SP}$) with respect to a stochastic transduction $\mathcal{R}$ as the following:

$$\begin{aligned} \mathbb{PR}(\mathcal{R}) = \quad & \{u \in \Sigma^* | (u, \mathbf{v})^{-1}\mathcal{R} \neq \emptyset, \mathbf{v} \in \Omega^*\} \\ \mathbb{SP}(\mathcal{R}) = \quad & \{u \in \mathbb{PR}(\mathcal{R}) | (u, \mathbf{v})^{-1}\mathcal{R} = (w, \mathbf{x})^{-1} \\ & \mathcal{R} \Rightarrow |u| \leq |w|\} \end{aligned}$$

The *kernel set* ($\mathbb{K}$) of $\mathcal{R}$ is defined as follows:

$$\mathbb{K}(\mathcal{R}) = \{\epsilon\} \cup \{ua \in \mathbb{PR}(\mathcal{R}) | u \in \mathbb{SP}(\mathcal{R}) \wedge a \in \Sigma\}$$

Note that $\mathbb{SP}$ is included in $\mathbb{K}$.

Identification with probability one will be achieved if given any possible target transduction $\mathcal{R}$ defined by a Pst, given any infinite presentation of translation pairs drawn according to the joint distribution, and, denoting by S_n the multiset consisting of the n first pairs, with probability one, Apti returns a correct hypothesis from S_n for all but a finite number of values of n.

The characteristic sample for the given algorithm is required to meet three conditions: firstly, it should include all the states and edges. Secondly, some properties should hold that disallow wrong merges to take place. Finally, it should have enough examples that ensure the alignments of the input and output during the learning phase. For the first condition we only need the kernel of the stochastic transduction or strings that contain the elements of the kernel as prefixes to be included in the sample. For the second condition, for states to be declared non mergeable we need any one of the following: 1) output mismatch 2) probability mismatch. Since we are adding zero probability edges for the non existing edges in the Ptst, at least one of these conditions is guaranteed to be eventually met by the stochastic sample. Formally, we can define the characteristic sample as:

Definition 4.1 *A stochastic sample S_n is said to be characteristic w.r.t. a* Sdrt *$\mathcal{R}$ if it satisfies the following conditions:*

1. $\forall u \in \mathbb{K}, \exists (uw, \mathbf{vx}) \in S_n$ *such that* $w \in \Sigma^*, \mathbf{vx} \in \Omega^*, (w, \mathbf{x}) \in (u, \mathbf{v})^{-1}\mathcal{R}$,
2. $\forall u \in \mathbb{SP}, u' \in \mathbb{K}$, *if* $(u, \mathbf{v})^{-1}\mathcal{R} \neq (u', \mathbf{v}')^{-1}\mathcal{R}$ *where* $\mathbf{v}, \mathbf{v}' \in \Omega^*$, *then any one of the following holds:*
   - $\exists (uw, \mathbf{vx}), (u'w, \mathbf{v'x'}), (ur, \mathbf{vy})$, $(u'r, \mathbf{v'y'}) \in S_n$ *such that:* $\frac{Pr_{\mathcal{R}}(uw,\mathbf{vx})}{Pr_{\mathcal{R}}(ur,\mathbf{vy})} \neq \frac{Pr_{\mathcal{R}}(u'w,\mathbf{v'x'})}{Pr_{\mathcal{R}}(u'r,\mathbf{v'y'})}$
   - $\exists (uw, \mathbf{vx}), (u'w, \mathbf{v'x'}) \in S_n$ *such that:* $(w, \mathbf{x}) \in (u, \mathbf{v})^{-1}\mathcal{R}, (w, \mathbf{x}') \in (u', \mathbf{v}')^{-1}\mathcal{R}$, $x \neq x'$
3. $\forall u \in \mathbb{K}, \exists (uw, \mathbf{vx})(uw', \mathbf{vx}') \in S_n$ *such that* $lcp(\mathbf{x}, \mathbf{x}') = \epsilon \wedge w \neq w'$.

Lemma 1 *For any $i^{th}(i \geq 1)$ call of the subroutine* Merge*, it recursively computes the i^{th} hypothesis* Pst *H_i such that if the merge is successful, $|H_i| < |H_{i-1}|$ and H_i is consistent with the training sample S_n.*

Proof It is easy to see that $|H_i| < |H_{i-1}|$ after each merge is accepted. H_i is consistent with the training sample S_n because of the following: initially the hypothesis H_0 is the Ptst which is the exact representation of S_n. A merge between two states q and q' is accepted only when the recursive fold operation is successful. The recursive fold operation returns a negative result if at any time during the fold there are two candidate edges e and e' such that $i\,[e] = i\,[e']$ are incompatible *w.r.t.* the probabilities of e and e' or if there is incompatibility between the output strings when $i\,[e] = i\,[e'] = \sharp$. Therefore, at anytime during the algorithm H_i will be consistent with the training sample. ■

Lemma 2 *The algorithm* Apti *terminates after a finite number of steps.*

Proof The initial Ptst consists of a finite number of states. Therefore, there can be only finite number of Red and Blue states. At each iteration of the outer most loop of the algorithm (Algorithm 1) either a Blue state will disappear (if the merge is accepted) or will be promoted to a Red state and at the same time the offspring of the new Red states will become Blue states. Therefore, it can be seen that eventually the set Blue will be empty, which is precisely the terminating condition for the outermost loop. ■

Theorem 3 *The algorithm* Apti *identifies* Sdrt *in the limit with probability one from positive presentation and* EXPQ*.*

Proof The condition 1 of the $\mathbb{CS}$ ensures that there will be at least as many states in the initial hypothesis H_0 (the Ptst) as the target T. The condition 2 of the $\mathbb{CS}$ prevents merges of the non-equivalent states during the run. The weights of the edges are populated using EXPQ *w.r.t.* the input language. Even if the transduction scheme is not a total function, the phantoms added to the Ptst will prevent merges of non-equivalent states. The condition 3 of $\mathbb{CS}$ ensures factorization of the output strings and correct alignment *w.r.t.* the input symbols. Moreover, any positive presentation of the target T will eventually include $\mathbb{CS}$. Thus, from lemmata 1 and 2, Apti converges to a Pst which represents the same Sdrt as the target machine T. Therefore, Apti satisfies the conditions of identification in the limit with probability one. ■

5. Complexity Analysis

In this section we present a complexity analysis of the proposed algorithm. We are primarily interested in the runtime complexity and the query complexity.

Let $\|S_n\| = \sum_{(u,\mathbf{v})\in S_n} |u|$ and $m = max\{|u| : (u, \mathbf{v}) \in S_n\}$.

Theorem 4 *The worst case run time complexity of the algorithm* Apti *is polynomially bounded by* $\mathcal{O}((\|S_n\|\,|\Sigma \cup \{\sharp\}|)^3(m+n) + \|S_n\|\,mn)$*.*

Proof The worst case run time complexity of Apti is similar to Ostia with some additional computation costs. The run time complexity of Ostia has been shown to be in $\mathcal{O}(\|S_n\|^3\,(m+n) + \|S_n\|\,mn)$ in (Oncina et al., 1993). In Apti we are populating the probabilities of the Ptst by asking queries and adding phantoms. The computation cost for populating the probabilities of the Ptst is linear *w.r.t.* $\|S_n\|$. The computation cost for adding the phantoms is in the worst case $\|S_n\|\,mn$. Due to the phantoms, in the worst case the total number of states in the Ptst will be $\|S_n\|\,|\Sigma \cup \{\sharp\}|$. Therefore, the worst case run time complexity of Apti is $\mathcal{O}((\|S_n\|\,|\Sigma \cup \{\sharp\}|)^3(m+n) + \|S_n\|\,mn)$. ■

The worst case analysis shown is pessimistic and hardly occurs in practice. Therefore, although the worst case run time complexity is shown to be cubic, practically the runtime is much lower. We report some empirical evidence of Apti's run time in Section 6.

Proposition 5 *The algorithm* Apti *requires at most* $\|S_n\| \cdot |\Sigma \cup \{\sharp\}|$ *extended prefix language queries for a given training sample* S_n*.*

Proof The total number of states in the Ptst will be $\|S_n\| \, |\Sigma \cup \{\sharp\}|$ in the worst case. Apti only asks queries to populate the Ptst and therefore, will ask at most $\|S_n\| \, |\Sigma \cup \{\sharp\}|$ number of EXPQ. ■

6. Experiments

6.1. Data Sets

We conduct our experiments with two types of data sets: 1) artificial data sets generated from random transducers 2) data generated from the so-called Miniature Language Acquisition (Mla) task (Feldman et al., 1990) adapted to English-French translations. Details of the data generation protocol follow.

6.1.1. Data Generation Protocol

For the artificial data sets, we first generate a random Pst with m states. The states are numbered from q_0 to q_{m-1} where state q_0 is the initial state. The states are connected randomly; labels on transitions preserve the deterministic property. Then the unreachable states are removed. The outputs are assigned as random strings drawn from a uniform distribution over $\Omega^{\leq k}$, for an arbitrary value of k. The probabilities of the edges are randomly assigned making sure the following condition holds:

$$\forall q_i \in Q, \sum_{e \in E[q_i]} prob\,[e] = 1 \tag{2}$$

Using the random Pst T, we generate a stochastic training sample S_n where each translation pair is drawn i.i.d. from the joint distribution $\mathcal{R}_T$. The test set S'_p is also drawn i.i.d. from the joint distribution $\mathcal{R}_T$, restricted to $\Sigma^*\sharp \times \Omega^* \backslash S_n$. Therefore $S'_p \cap S_n = \emptyset$.

The second type of data set which is adapted from the Mla task, consists of pseudo-natural sentences in English and French describing visual scenes within a restricted conceptual domain. The target transducer built under the Mla task is depicted in Figure 2. Random probabilities are assigned to the edges of the target Pst (Figure 2) satisfying condition (2). The generation procedure of S_n and S'_p for the Mla task is as above. Similar kind of data sets have been used in the context of transducer learning in (Angluin and Becerra-Bonache, 2009; Castellanos et al., 1998).

6.1.2. Evaluation

In our experiments we measure the word error rate (Wer) as a metric of correctness. Wer is the percentage of erroneous symbols in the hypothesis translation *w.r.t.* the reference translation. Other type of evaluation, such as Bleu (Papineni et al., 2002), is not reported because we have small alphabet size for artificially generated data with a lot of repetitions and Bleu score can be rather misleading. Our objective is to analyze the correctness and learnability of Apti for which Wer is more appropriate.

The results of the experiment conducted with the artificial data sets is shown in Figures 3, 4, and 5. We have generated a random Pst of size 10 and $|\Sigma| = |\Omega| = 5$. Apti is executed with different sizes of input, starting from 500 training pairs to 20,000 training pairs, each time incrementing the training set size by 500. Test sets of 1,000 pairs are also being generated for each run. For the sake of statistical significance, the procedure is repeated 10 times for each data point.

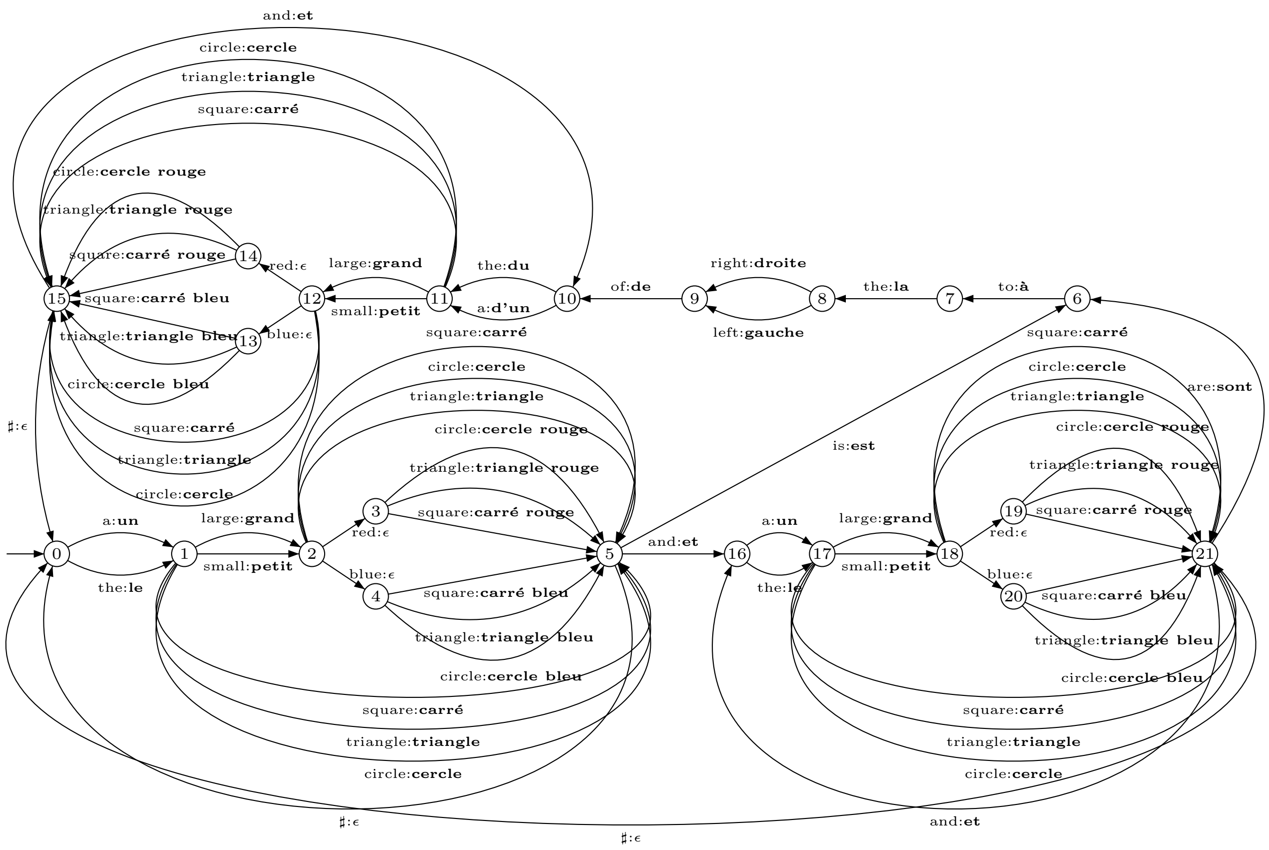

Figure 2: The target Pst for English-French translation under the Mla framework. The probabilities of the edges are not shown for clarity.

The results in Figure 3 show that the Wer gets close to 0 only with 5,000 training pairs and with 8,000 training pairs onwards the algorithm has converged. The objective of this experiment is to show the correctness of the algorithm.

It is important to note that the translation pairs of each training sample have been drawn i.i.d. with replacement. In order to analyze the generalization capabilities of the algorithms *w.r.t.* percentage of the stochastic transduction $\mathcal{R}$ we define the following. Let U be the set of unique training pairs in S_n defined as:

$$U = \bigcup_{(u,\mathbf{v})\in S_n} \{(u,\mathbf{v})\}.$$

We define the *sample density* d as:

$$d = \sum_{(u,\mathbf{v})\in U} Pr_{\mathcal{R}}(u,\mathbf{v}).$$

Figure 4 depicts how Wer varies with the sample density of the training set. It shows that even for less than 40% of the stochastic transduction $\mathcal{R}$, the Wer is as low as 25%. With 45% of $\mathcal{R}$, the Wer almost converges to zero.

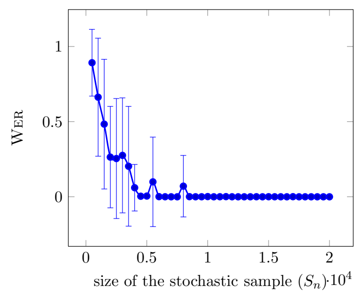

Figure 3

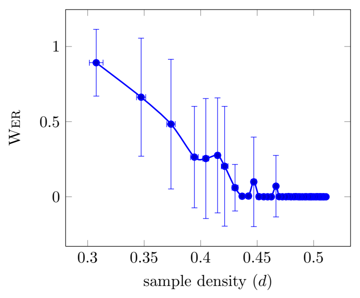

Figure 4

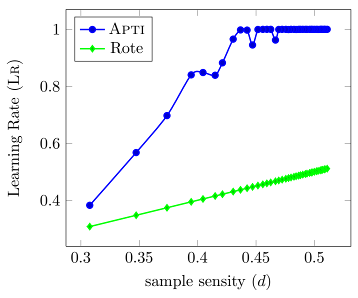

Figure 5

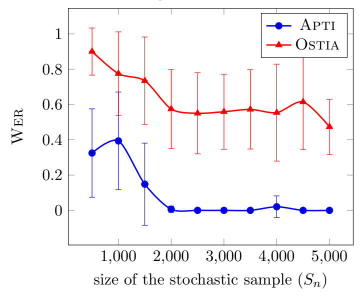

Figure 6

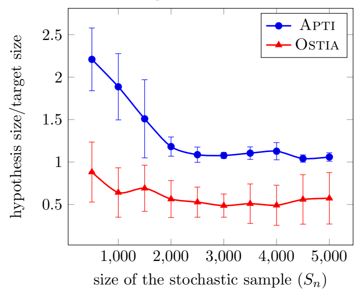

Figure 7

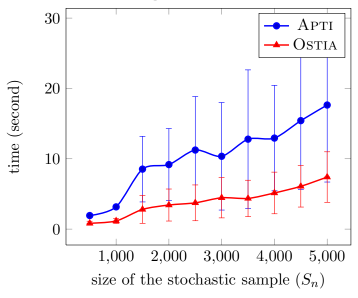

Figure 8

Performance of Apti reported for the artificial data set (Figures 3, 4, and 5) and the data set for the Mla task (Figures 6, 7, and 8).

Figure 5 shows the *learning rate* of Apti in comparison with rote learner. The learning rate (Lr) is defined as: Lr $= d + (1-d) * (1 -$ Wer$)$. Intuitively, Lr tells us the percentage of $\mathcal{R}$ the learner has learnt. In other words, a rote learner can only translate the strings it has seen during training and will therefore have a learning rate of d.

As expected, in Figure 5 Apti performs better than a rote learner. The slope of the learning rate - density curve is high when the training density is low. The slope decreases as the density gets higher and eventually approaches to 0 after a certain transition phase.

The results shown in Figures 6, 7, and 8 are based on data sets for the Mla task. The experiments were conducted on 5000 training pairs and 1000 test pairs. Both the algorithms Apti and Ostia were executed with input size starting from 500 training pairs to 5000 training pairs, incremented by 500 at each step. Similar to the previous experiment, each run is repeated 10 times.

Figure 6 shows the Wer for Apti and Ostia for different sizes of training data. The Wer for Apti almost converges to zero from 2000 training pairs onward, whereas, Ostia continues to result in more than 50% of Wer.

Apti outperforms Ostia for a number of reasons. Firstly, in Apti we are exploiting probabilistic information by asking EXPQ to a Mat. Secondly, theoretically Ostia has got the limitation that it is guaranteed to converge only in case of subsequential transductions that are total functions (Oncina et al., 1993). Our target transducer for the Mla task (Figure 2) represents a transduction which is a partial function. Therefore, Ostia is not capable of rejecting some of the merges between states that should not take place. As a result, Ostia over-generalizes the hypothesis and leads to transducers with fewer number of states than the target Pst. Figure 7 shows the ratio between the hypothesis and the target size for each training size. Here, we see that in the case of Ostia, the hypothesis-target ratio always remains below 1. On the other hand, by introducing the notion of phantoms, we are able to overcome such limitations in Apti. Figure 7 shows that the ratio of the hypothesis and the target size in case of Apti always remains above or equal to 1, *i.e.*, Apti does not over-generalize.

For our experiments, we have implemented Ostia and Apti using OpenFST (Allauzen et al., 2007), an open source C++ library for Fsts. Figure 8 shows some timing results of OpenFST implementation of Ostia and Apti. It shows that the execution time of Apti is more than the execution time of Ostia. In case of Apti, additional edges and states are added to the initial tree as phantoms. Therefore, in most of the cases the size of the initial tree for Apti is bigger than the size of the initial tree for Ostia which is the reason for longer execution time for Apti. Nevertheless, the execution time of Apti is reasonably low as shown in Figure 8: this always remained below 30 seconds.

7. Conclusion

We presented an algorithm for learning Psts from positive training sample and by asking probabilistic queries about the observed data. We have shown that our algorithm Apti is capable of learning partial functions, which were not guaranteed to be learnt using Ostia. We have experimentally shown that our algorithm outperforms Ostia. We have used EXPQ *w.r.t.* the given data, and thus, avoided peculiar problems as reported in (Lang and Baum, 1992). This kind of queries can easily be simulated in real life using relative frequencies of the observed data.

The expressive power of our model is limited to the class of Sdrt, which surely does not include a large part of natural translation schemes. How to learn more complex classes of transductions, remains an open question.

References

Cyril Allauzen and Mehryar Mohri. Finitely subsequential transducers. *International Journal of Foundations of Computer Science*, 14(6):983–994, 2003.

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. Openfst: A general and efficient weighted finite-state transducer library. In *Proceedings of CIAA*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer-Verlag, 2007.

Dana Angluin. A note on the number of queries needed to identify regular languages. *Information and Control*, 51(1):76–87, 1981.

Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987a.

Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987b.

Dana Angluin and Leonor Becerra-Bonache. Experiments using ostia for a language production task. In *Proceedings of the EACL*, CLAGI '09, pages 16–23. Association for Computational Linguistics, 2009.

Dana Angluin and Carl H. Smith. Inductive inference: Theory and methods. *ACM Computer Surveys*, 15(3):237–269, 1983.

Francesco Bergadano and Stefano Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. *SIAM Journal on Computing*, 25:1268–1280, 1996.

Antonio Castellanos, Enrique Vidal, Miguel Angel Varó, and José Oncina. Language understanding and subsequential transducer learning. *Computer Speech & Language*, 12(3): 193–228, 1998.

Alexander Clark. Inference of inversion transduction grammars. In *Proceedings of ICML*, pages 210–208. Omnipress, 2011.

Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.

Colin de la Higuera and José Oncina. Learning stochastic finite automata. In *Proceedings of ICGI*, volume 3264 of *Lecture Notes in Computer Science*, pages 175–186. Springer-Verlag, 2004.

Jerome A. Feldman, George Lakoff, Andreas Stolcke, and Susan H. Weber. Miniature language acquisition: A touchstone for cognitive science. Technical Report TR-90-009, International Computer Science Institute, Berkeley CA, 1990.

Kevin J. Lang and Eric B. Baum. Query learning can work poorly when a human oracle is used. In *Proceedings of IJCNN*, pages 335–340. IEEE Press, 1992.

Mehryar Mohri. *Handbook of Weighted Automata*, chapter Weighted automata algorithms, pages 213–254. Monographs in Theoretical Computer Science, an EATCS Series. Springer-Verlag, 2009.

Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

José Oncina. Using multiplicity automata to identify transducer relations from membership and equivalence queries. In *Proceedings of ICGI*, volume 5278 of *Lecture Notes in Computer Science*, pages 154–162. Springer-Verlag, 2008.

José Oncina and Pedro García. Inductive learning of subsequential functions. Technical report, Univ. Polit'ecnica de Valencia, 1991.

José Oncina and Pedro García. Identifying regular languages in polynomial time. In H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition*, volume 5 of *Series in Machine Perception and Artificial Intelligence*, pages 99–108. World Scientific, 1992.

José Oncina and Miguel Angel Varó. Using domain information during the learning of a subsequential transducer. In *Proceedings of ICGI*, volume 1147 of *Lecture Notes in Computer Science*, pages 301–312. Springer-Verlag, 1996.

José Oncina, Pedro García, and Enrique Vidal. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):448–458, 1993.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Procceedings of ACL*, pages 311–318. The Association for Computer Linguistics, 2002.

Piedachu Peris and Damián López. Transducer inference by assembling specific languages. In *Proceedings of ICGI*, volume 6339 of *Lecture Notes in Computer Science*, pages 178–188. Springer-Verlag, 2010.

Christophe Reutenauer and Marcel Paul Schützenberger. Minimization of rational word functions. *SIAM Journal on Computing*, 20(4):669–685, 1991.

Christophe Reutenauer and Marcel Paul Schützenberger. Variétés et fonctions rationnelles. *Theoretical Computer Science*, 145(1&2):229–240, 1995.

Arto Salomaa and Matti Soittola. *Automata: Theoretic Aspects of Formal Power Series*. Springer-Verlag, 1978.

Burr Settles. From theories to queries: Active learning in practice. In *Proceeingds of Active Learning and Experimental Design workshop*, volume 16 of *JMLR: Workshop and Conference Proceedings*, pages 1–18. MIT Press, 2011.

Enrique Vidal and Francisco Casacuberta. Learning finite-state models for machine translation. In *Proceedings of ICGI*, volume 3264 of *Lecture Notes in Computer Science*, pages 3–15. Springer-Verlag, 2004.

Enrique Vidal, Franck Thollard, Colin de la Higuera, Francisco Casacuberta, and Rafael C. Carrasco. Probabilistic finite-state machines-part ii. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1026–1039, 2005a.

Enrique Vidal, Franck Thollard, Colin de la Higuera, Francisco Casacuberta, and Rafael C. Carrasco. Probabilistic finite-state machines-part i. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1013–1025, 2005b.

Juan Miguel Vilar. Query learning of subsequential transducers. In *Proceedings of ICGI*, volume 1147 of *Lecture Notes in Computer Science*, pages 72–83. Springer-Verlag, 1996.

Juan Miguel Vilar. Improve the learning of subsequential transducers by using alignments and dictionaries. In *Proceedings of ICGI*, volume 1891 of *Lecture Notes in Computer Science*, pages 298–311. Springer-Verlag, 2000.

Mitsuo Wakatsuki and Etsuji Tomita. Polynomial time identification of strict prefix deterministic finite state transducers. In *Proceedings of ICGI*, volume 6339 of *Lecture Notes in Computer Science*, pages 313–316. Springer-Verlag, 2010.