Efficient Inference of a Subclass of Even Linear Languages

J.A.Laxminarayana and G.Nagaraja
CSE Department, Indian Institute of Technology Bombay, INDIA
Email: {jalana,gn}@cse.iitb.ac.in

Abstract
Importance of grammatical inference field is growing as it is finding wider acceptance in many practical fields. Since it is established that the family of even-linear languages is not inferable from positive data alone, researchers have proposed several approximation algorithms to infer by the way of defining new subfamilies of even-linear languages. In this work a polynomial time method is proposed to infer a subclass of even linear languages, Terminal Distinguishable Even Linear Languages (TDELL), using positive samples only. The inference method for TDELL is based on UNION-FIND algorithm. We make some considerations about the performance of the proposed method in identifying TDELL in the limit and we provide the complexity analysis of the proposed method. Suitability of the proposed algorithm to the practical applications is also discussed.

1 Introduction

Grammatical Inference is a learning process to find the structural model (grammar) which best accounts for all given patterns. This is a central issue in syntactic pattern recognition because of its machine-learning implication. Potential engineering applications of grammatical inference include areas of pattern recognition, information retrieval, programming language design, translation and compiling, graphics languages, man-machine communication, and artificial intelligence [7].

A learning algorithm can receive two types of string patterns: positive examples and negative examples. A positive example is an example of something that is correct in the language that has to be acquired and a negative example is an example of something that is incorrect.

In 1967, E.M.Gold presented a landmark paper [6] to show that it is not possible to build a perfect model for a general language that contains an infinite number of strings by only looking at positive samples of the language. The reason for this is that for any set of positive sample strings there will be an infinite number of models that can produce these sample strings. Without negative samples it is not possible to decide which of these models is the correct one.

In order to make the language learning problem mathematically tractable, Gold introduced the concept of identification in the limit. In this, a teacher is used to provide the training string patterns to a learner who will guess a grammar. The teacher then analyses the learning behaviour of the learner and accordingly take the decision to continue or stop the learning process. Within this learning model Gold proved some important results. The most important result for the grammar learning community was that the classes of languages defined by context free grammars and regular grammars cannot be learnt from positive samples alone.

From Gold’s results, it is concluded that any class of recursively enumerable languages can be identified in the limit from complete presentation (both positive and negative samples) and no superfinite class of languages (class of all the finite cardinality languages with at least one infinite cardinality language) can be identified in the limit from only positive presentation.

But it is known that negative samples are not always available and also their use can lead to probably intractable problems. The problem of inferring grammars from positive data, though important and having many practical applications, attracted little theoretical attention till the work of Angluin [2] in 1980. The main reason for this is the strong negative result of Gold.

The main result of Angluin is stated as follows. If the language denoted by the guessed grammar from the set of positive samples is the smallest in the class of target languages, then that grammar is the inferred grammar from the set of positive samples. This limits the main inference problem such that only a subclass of the target languages are learnt but not entire class.
Due to Angluin’s result, we can use only positive samples and infer non-superfinite classes of languages using characterizable methods. It is observed that undecidability results strictly apply to superfinite classes. But many non-superfinite classes may exist that are identifiable with just positive information; i.e. undecidability does not apply to classes that do not contain all finite languages. One potential problem is over-generalization. But, by allowing only restricted generalizations over-generalization can be avoided for non-trivial classes of languages.

The class of Even Linear Languages (ELL) was initially proposed by Amar and Putzolu [1] as a subclass of more generic Linear Languages class. There are some works in the literature that have focused on the inference problem of even linear languages. Radhakrishnan and Nagaraja [10] have presented a method to infer a subclass of regular languages, Terminal Distinguishable Regular Languages (TDRLL), from the structural information induced by the positive strings. They have proposed a similar method [10, 11] to infer a subclass of even linear languages, Terminal Distinguishable Even Linear Languages (TDELL), from positive strings. Sempere and Fos [12] proposed a heuristic technique, based on the previous methods by Radhakrishnan and Nagaraja, to work with positive linear skeletons. In [14], Sempere and Nagaraja presented a characteristic method to infer a subclass of linear languages, Terminal and Structural Distinguishable Linear Languages (TSDL), from positive structural information (linear skeletons). Further contributions toward the theoretical aspects of Terminal Distinguishable Languages are given by Fernau [4] and [5]. Takada [15] established that every even linear language can be generated by a universal grammar provided with certain control sets that regulate the application of its rules. A similar result is obtained by Sempere and Garcia [13], in which they propose to learn even linear languages by a reduction to regular languages.

In this paper, a method is proposed to infer one of the characterizable subclass of even-linear languages, TDELL, in an efficient way. The basic algorithm used here is the one proposed by V. Radhakrishnan and G. Nagaraja [11]. An alternative method based on UNION-FIND algorithm [16].

The paper is organized as follows. Necessary background from language theory are provided in sections 2. A method to infer a subclass of linear languages (TDELL) based on UNION-FIND algorithm and its properties are discussed in section 3. Section 4 shows some experimental results whereas the usage of the proposed algorithm in one of the practical application is discussed in section 5. Section 6 concludes the paper.

2 Preliminaries and Definitions

Some basic concepts about formal language theory and formal grammars are introduced. Most of them are found in any introductory book on formal languages and automata eg. [8].

In what follows Σ denotes an alphabet and Σ* is the set of words over the alphabet Σ. Given an alphabet Σ and x ∈ Σ*, |x| is used to denote the length of x. λ denotes the empty string with |λ| = 0. Φ denotes an empty set. Given any string x ∈ Σ*, |x| denotes the set of terminals in the string x. A grammar is a quadruple G = (N, Σ, P, S), where N is the set of non-terminal symbols, Σ (with Σ ∩ N = Φ) the set of terminal symbols, P the rule set and S ∈ N the start symbol. A grammar is called context free if rules are of the form A → β where A ∈ N and β ∈ {N ∪ Σ}* . Given a grammar G = (N, Σ, P, S), and two strings x, y ∈ {N ∪ Σ}*, one can say that
1) x directly derives to y (using rule α → β ) if y is obtained from x by replacing one occurrence of the subword α in x by β and we write x ⇒ y;
2) x derives to y, denoted by x ⇒* y, if y can be obtained by applying to x a finite set of productions of P.

The language generated by the grammar G is denoted as L(G) and is defined as the set L(G) = {x ∈ Σ* | S ⇒* x}. Generally, we will work with reduced grammars, that is, grammars without useless symbols, unit productions(with the exception of the productions with start symbol) or empty productions. L(G, A) denotes the language obtained by the grammar GA = (N, Σ, P, A). It is written as L(A) instead of L(G, A) whenever G is understood.

**Definition 2.1** Let G = (N, Σ, P, S) be a grammar. G is said to be linear if every production in P takes one of the following forms. 1) A → uBv, where A, B ∈ N and u, v ∈ Σ*. 2) A → u where A ∈ N and u ∈ Σ*.

An even linear grammar generating an even linear language, is a linear grammar in which all elements of P are of the form A → uBv or A → w where A, B ∈ N and u, v, w ∈ Σ* such that |u| = |v|. Every even linear grammar G = (N, Σ, P, S) admits the following normal form in its productions A → aBb | a | λ where A, B ∈ N and a, b ∈ Σ.

**Definition 2.2** Let G = (N, Σ, P, S) be an even-linear grammar. An even-linear skeleton in G is a derivation tree in which the internal nodes are not labeled.
Let \( ND(S^+) \) denote the set of nodes found in all even linear skeletons of \( S^+ \), where \( S^+ \) is a set of strings. Formally, \( ND(S^+) = \bigcup_{w \in S^+} \{ N_{ij} \mid 1 \leq j \leq \lfloor |w|/2 \rfloor \} \) where \( 1 \leq i \leq |S^+| \). A partition of a set \( ND(S^+) \) is a collection of pairwise disjoint nonempty subsets of \( ND(S^+) \) whose union is \( ND(S^+) \).

If \( \pi \) is a partition of \( ND(S^+) \), then for any element \( p \in ND(S^+) \) there is a unique element of \( \pi \) containing \( p \), which is denoted as \( B(p, \pi) \) and call the block of \( p \) containing \( p \). A partition \( \pi \) is said to refine another partition \( \pi' \) if and only if every block of \( \pi' \) is a union of blocks of \( \pi \).

In the even linear skeleton of a string, the \textit{frontier string} of a node \( N_{ij} \) denoted by \( FS(N_{ij}) \) is defined as the string formed by the sub-skeleton rooted at node \( N_{ij} \). Also the \textit{head string} and \textit{tail string} of a node \( N_{ij} \), denoted \( \text{Head}(N_{ij}) \) and \( \text{Tail}(N_{ij}) \) are defined as \( \text{Head}(N_{ij}) = u \) and \( \text{Tail}(N_{ij}) = v \) where \( uFS(N_{ij})v = FS(N_{ij}) \). A \textit{skeletal structure node function} of a node \( N_{ij} \) in an even linear skeleton, denoted by \( SSNF(N_{ij}) \), is defined as an ordered triple; \( N_{ij} \) is mapped to a value in the range \( \Sigma^* \times 2^S \times \Sigma^* \) as given by \( SSNF(N_{ij}) = \langle \text{Head}(N_{ij}), \text{Ter}(FS(N_{ij})), \text{Tail}(N_{ij}) \rangle \).

Figure 1 exhibits an example of even linear skeleton and its associated function values. Here each internal node is represented by an arbitrary node symbol. This is only to understand the computations of function values. They should not be misunderstood for the non-terminal labels which are to be assigned by the inference algorithm later.

A TDELG, which denotes a language in the class of a TDELL, is defined as follows.

**Definition 2.3** An even linear grammar \( G \) is called a Terminal Distinguishable Even Linear Grammar(TDELG) iff it satisfies the following conditions.

1. Let \( B, C \in N \) and \( w \in \Sigma \{ N \cup \Lambda \}, \Sigma \cup \Sigma \) then the production rules \( B \rightarrow w \) and \( C \rightarrow w \) implies \( B = C \).

2. For all \( A \in N - \{ S \} \) and for all \( x, y \in L(A), \text{Ter}(x) = \text{Ter}(y) \)

3. Let \( A, B, C \in N, a, b \in \Sigma \) and i) the productions \( S \rightarrow B \) and \( S \rightarrow C \) where \( S \) is the start symbol ii) the productions \( A \rightarrow aBb \) and \( A \rightarrow aCb \) appears in the set of productions, then for all \( x \in L(B) \) and for all \( y \in L(C), \text{Ter}(x) \neq \text{Ter}(y) \)

The equivalent definition for the class of TDELLs is given as follows.

**Definition 2.4** An even linear language \( L \) is TDELL iff for all \( u_1, u_2, v_1, v_2, w, z \in \Sigma^* \) such that \( |u_1| = |v_1|, |u_2| = |v_2| \) and \( \text{Ter}(w) = \text{Ter}(z) \), if \( u_1wv_1, u_2wv_2 \in L \) then \( u_1zv_1 \in L \) iff \( u_2zv_2 \in L \).

### 3 Proposed Algorithm to Infer TDELL

We consider that the set of strings are from a language from TDELL class. Hence in an even-linear skeleton for a given string two internal nodes \( p \) and \( q \) are \( \infty \) (i.e. TDELG-equivalent) iff at least one of the following three conditions holds.
1. \(SSNF(p) = SSNF(q)\)
2. \(FS(p) = FS(q)\) or
3. \(PTF(p) = PTF(q)\)

It should be noted that in order to obtain a set of equivalence classes on the nodes of skeletons of input strings, a transitive closure of \(\bowtie\) has to be obtained.

The proposed algorithm, \textit{TDELG-Inference} to infer a TDELG from a set of positive samples is described and analyzed in this section. The input to the \textit{TDELG-Inference} is a finite nonempty set of strings \(S^+\). The output is TDELG \(G\) such that \(S^+ \subseteq L(G)\). It is shown that \(L(G)\) is a smallest TDELL that contains \(S^+\). This algorithms is influenced by the works of [3] and [5] and shown as Algorithm 1.

\textbf{Algorithm 1: TDELG-Inference}

\textbf{Input:} A nonempty set of positive sample \(S^+\).
\textbf{Output:} Inferred TDELG.

- **Initialization**
  1. Construct the skeletons for each \(x_i\), where \(x_i\) is the \(i^{th}\) sample of \(S^+\).
  2. Compute \(ND(S^+)\)
  3. Compute the initial partition \(\pi_0\) of the \(ND(S^+)\) such that two nodes \(p\) and \(q\) are in the same block \(B\) if \(SSNF(p) = SSNF(q)\)
  4. Let \(LIST\) contain all unordered pairs \((p, q)\) of nodes of \(ND(S^+)\) such that \(p \in B_1, q \in B_2, B_1, B_2 \in \pi_0, B_1 \neq B_2\) and \(FS(p) = FS(q)\).
  5. set \(i\) to 0.

- **Merging**
  1. While \(LIST \neq \emptyset\) do
     - Remove some element \((p, q)\) from \(LIST\)
     - Find two blocks \(B_1\) and \(B_2\) from \(\pi_i\) such that \(\exists p \in B_1\) and \(\exists q \in B_2\).
     - If \(B_1 \neq B_2\) then
       * Construct \(\pi_{i+1}\) which is same as \(\pi_i\) by \(B_1\) and \(B_2\) merged.
       * For every pair of disjoint nodes \((r, s)\) whose parent nodes are in \(B_1 \cup B_2\) and \(PTF(r) = PTF(s)\), place \(\{r, s\}\) on \(LIST\)
       * Increment \(i\) by one
  2. Construct the production rules \(P\), using the skeletons, which have been modified to reflect the equivalence classes in \(\pi_i\)

\subsection{3.1 Description of TDELG-Inference}

The proposed algorithm \textit{TDELG-Inference} first constructs even linear skeletons of all the input strings \(S^+\) and form a node set \(ND(S^+)\) representing all the nodes found in such even linear skeletons. The main aim of the algorithm to establish an equivalence relation on the set of \(ND(S^+)\) and to construct a TDELG using final partition obtained.

In order to compute the final partition of \(ND(S^+)\), algorithm starts with an initial partition \(\pi_0\) and repeatedly merge any two distinct blocks \(B_1\) and \(B_2\) such that \(B_1\) and \(B_2\) both contain nodes having either common frontier string or PTFs. When no such blocks exists, the resulting partition is the final partition.

A set of SSNF’s \(H(S^+)\) for all the nodes in \(ND(S^+)\) is formed which is used to evolve an initial partition \(\pi_0\) on \(ND(S^+)\). Two nodes \(p\) and \(q\) are in a block \(B\) of \(\pi_0\) if \(SSNF(p) = SSNF(q)\). To facilitate this merging process in an efficient way, algorithm uses a structure called \textit{LIST} to keep track of the possible future merges. The structure \(LIST\) contains a list of pairs of nodes whose corresponding blocks are to be merged. The algorithm initializes the structure \(LIST\) with all pairs \((p, q)\) such that \(FS(p) = FS(q)\) which ensures all blocks having nodes with common frontier strings be merged.
After the initialization part of the algorithm completed, \textit{TDELG-Inference} enters into merging part of the algorithm. The algorithm removes the first pair of nodes \((n_1, n_2)\) as long as the \textit{LIST} is not empty. If \(n_1\) and \(n_2\) are not already in the same block of the current partition, the blocks containing \(n_1\) and \(n_2\) are merged to form a new block and new pairs of nodes \((r, s)\) are added to \textit{LIST} for all \(r\) and \(s\) in new merged block with \(PTF(r) = PTF(s)\). Otherwise the algorithm goes to the next pair of nodes from \textit{LIST}.

When the \textit{LIST} becomes empty, the current partition is \(\pi\), for some arbitrary \(i\). \textit{TDELG-Inference} computes TDELG using skeletons of input samples that are modified to reflect recent merging according to \(\pi\). Following example illustrates the working of proposed algorithm.

**Example 3.1** Let the sample set be \(S^+ = \{ab, aabb, aaabbb\}\).

The even-linear skeletons are constructed for the sample set \(S^+\) and using these even-linear skeletons a node set \(ND(S^+)\) is constructed: \(ND(S^+) = \{N_{11}, N_{21}, N_{22}, N_{31}, N_{32}, N_{33}\}\). Head strings, tail strings and frontier strings for each of the nodes in the \(ND(S^+)\) are computed and are shown in the Table 1. Using information in the Table 1, an initial partition \(\pi_0\) is computed as \(\pi_0 = \{\{N_{11}, N_{21}, N_{31}\}, \{N_{22}, N_{32}\}, \{N_{33}\}\}\) such that nodes in each block of the partition have common SSNF. A list of pair of nodes, \textit{LIST} is computed as follows. Since both the nodes \(N_{11}\) and \(N_{22}\) have same frontier string \(a\), a pair of nodes \(N_{11}\) and \(N_{22}\) is formed and added to \textit{LIST}. Similarly other pairs of nodes are computed and hence, \(\textit{LIST} = \{(N_{11}, N_{22}), (N_{11}, N_{33}), (N_{22}, N_{33}), (N_{21}, N_{32})\}\) is removed.

Once the \textit{LIST} is constructed, an element is removed from \textit{LIST} and processed for merging nodes. Consider the element \((N_{11}, N_{22})\) from \textit{LIST}. It is found that the node \(N_{11}\) is from block \(B_1 = \{N_{11}, N_{21}, N_{31}\}\) and node \(N_{22}\) is from the block \(B_2 = \{N_{22}, N_{32}\}\). Hence \(B_1 \cup B_2\) is computed and a new partition \(\pi_1\) is formed by updating \(\pi_0\): remove the blocks \(B_1\) and \(B_2\) and insert \(B_1 \cup B_2\). The new partition \(\pi_1 = \{\{N_{11}, N_{21}, N_{22}, N_{31}, N_{32}\}\}\) is merged until \textit{LIST} becomes empty. The final partition is \(\pi_{\text{final}} = \{\{N_{11}, N_{21}, N_{22}, N_{31}, N_{32}\}\}\). A unique label \(S_1\) is assigned to the block in the final partition. Now each single node in the skeletons of the sample strings is replaced by its equivalent label, i.e., the label of the block of the partition in which the node appears. Using the updated skeletons, which are the derivation trees, the TDELG is constructed. \(TDELG = \{\{S, S_1\}, \{a, b\}, \{S \rightarrow S_1, S_1 \rightarrow aS_1b, S_1 \rightarrow ab\}, S\}\) where \(S\) is the start symbol.

The convergence issue of the \textit{TDELG-Inference} to the target grammar if enough information can be shown by using the method similar to [9].

### 3.2 Complexity Analysis

The time complexity of \textit{TDELG-Inference} is presented as follows. Let \(S^+ = \{w_1, w_2, \ldots, w_n\}\) be the input sample set without any repetition. Let \(k\) be the size of the input i.e., sum of lengths of all input strings belonging to \(S^+\).

The formation of \(ND(S^+)\) is done in time \(O(k)\) using the skeletons of sample strings. The construction of initial partition of \(ND(S^+)\) is also done in time \(O(k)\) using the similar notion expressed in [3] for the construction of prefix tree accepter. The partition may be queried and updated using collapsing FIND and weighted UNION operations as described by [3]. Using the similar arguments given in [3] for the running time of \(\Delta R\) algorithm, we can conclude that the running time of \textit{TDELG-Inference} is also \(O(k\alpha(k))\) where \(\alpha\) is a very slowly growing function.
4 Experimental Results

The proposed $TDELG$-Inference algorithm is implemented and tested. The implemented algorithm shows the desired behaviour. This fact is shown in the Table 2.

<table>
<thead>
<tr>
<th>Sample Set</th>
<th>Inferred Grammar</th>
<th>Inferred Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 {aababa, aab^iab^iaab^*ab^ia}</td>
<td>$S \rightarrow S_1, S_1 \rightarrow aS_2a, S_2 \rightarrow aS_3b, S_3 \rightarrow ba</td>
<td>bS_3b$</td>
</tr>
<tr>
<td>2 {abaca, ab^iac^*aa ab^iac^*aa}</td>
<td>$S \rightarrow S_1 S_1 \rightarrow aS_2a, S_2 \rightarrow bS_3a</td>
<td>bS_4a$</td>
</tr>
<tr>
<td>3 {cab, caab, caaad}</td>
<td>$S \rightarrow S_1 S_1 \rightarrow cS_3b, S_2 \rightarrow cS_3d, S_3 \rightarrow a</td>
<td>aa</td>
</tr>
</tbody>
</table>

Table 2: Computation of $TDELG$ for different sample sets

5 Conclusion

The proposed method for the class of TDELL infers correctly for many languages in the class. As the UNION-FIND algorithm is used to obtain a partition of the set of non-terminals of grammar, it becomes efficient to implement and simple to understand. Further, it can be shown that, this method will not over-generalize when the samples are from a higher class than even linear.

This polynomial time algorithm can be used in the applications like RNA sequence analysis and XML document analysis. Further, this work can also be extended to infer higher class of languages using error correcting approach.

References