Efficient Learning of Tier-based Strictly k-Local languages *

Adam Jardine[1] and Kevin McMullin[2]

[1] Department of Linguistics, Rutgers University, New Brunswick, New Jersey, USA
adam.jardine@rutgers.edu
[2] Department of Linguistics, University of Ottawa, Ottawa, Canada
kevin.mcmullin@uottawa.ca

Abstract. We introduce the Tier-based k-Strictly Local Inference Algorithm (kTSLIA), an algorithm that learns the class of Tier-based Strictly k-Local (TSL$_k$) formal languages in polynomial time on a sample of positive data whose size is bounded by a constant. The TSL$_k$ languages are useful in modeling the cognition of sound patterns in natural language [6, 11], and it is known that they can be efficiently learned from positive data in the case that $k = 2$ [9]. The kTSLIA extends this result to any k and improves on its time efficiency. We also refine the definition of a canonical TSL$_k$ grammar and prove several properties about these grammars that aid in their learning.

1 Introduction

This paper introduces an algorithm that provably and efficiently learns a grammar for any Tier-based Strictly k-Local (TSL) formal language [6]. In brief, this subclass of the regular languages encompasses those languages that prohibit certain sequences of adjacent symbols, where adjacency is assessed with respect to a subset of the alphabet (the tier) while ignoring all intervening non-tier symbols. A TSL$_k$ grammar therefore comprises a tier, labeled T, and a set of k-factors (length-k sequences) that are forbidden on the tier, labeled R.

The TSL languages have been motivated from linguistic and cognitive perspectives [6, 8, 11, 12], drawing from patterns in natural language phonology in which co-occurrence restrictions apply to a subset of the sounds, ignoring all sounds not in that subset. For example, Finnish words only contain vowels from the set $\{\ddot{u}, \ddot{o}, \ddot{a}\}$ or the set $\{u, o, a\}$, and not both, regardless of any intervening consonants or vowels $\{i, e\}$. For example, words *pöütänä* 'the table (essive)' and *pappina* 'priest (essive)' are attested in Finnish, but words like *poütäna* and *päppina*, which contain vowels from both sets, are not attested [14, 15]. This can be modeled by a TSL$_2$ grammar which bans, e.g., $\ddot{a}a$ and $a\ddot{a}$ sequences once all sounds not in the set $\{\ddot{u}, \ddot{o}, \ddot{a}, u, o, a\}$ (i.e, all consonants and the vowels $\{i, e\}$) have been deleted [6]. Such patterns are common in natural language, ranging over a variety of tiers [15, 11].

* We thank Jane Chandlee, Gunnar Hansson, and Jeff Heinz for their many insights.

As such, it is of interest to understand how such grammars can be learned from positive evidence. While the learning problem is trivial when the contents of the tier are provided *a priori* [6], it has not yet been established that an efficient, non-enumerative algorithm exists which can induce both the tier and the k-factor components of the grammar for TSL_k languages of any k. For $k = 2$, the Tier-based Strictly 2-Local Inference Algorithm (2TSLIA) of [9] exactly identifies a TSL_2 tier and permitted 2-factors given positive data in quartic time. The present work generalizes and improves on this result, introducing an algorithm we call the Tier-based Strictly k-Local Inference Algorithm (kTSLIA), that provably learns the class of TSL_k languages for any value of k in quadratic time with a data set bounded by a constant. A secondary result is that we refine the notion of canonical TSL_k grammars, distinct from the notion introduced for TSL_k grammars in [9], and prove properties of these grammars that the kTSLIA can use to induce a TSL grammar exactly from positive examples.

The paper is structured as follows. §2 summarizes the relevant notation, concepts, and definitions used throughout this paper. §3 defines the TSL class of formal languages and proves some properties of this class that can be exploited in learning. §4 presents the kTSLIA and §5 proves that the algorithm learns the TSL_k class of languages in polynomial time and data for any value of k. §6 summarizes the contributions of the paper, discusses future work, and concludes.

2 Preliminaries

We use standard set notation. For a set S, $|S|$ denotes its cardinality and $\mathcal{P}_{\text{fin}}(S)$ the set of finite subsets of S. For sets S and R, $S - R$ denotes $\{s \in S | s \notin R\}$.

An *alphabet* Σ is a finite set of symbols. A *string* $w = \sigma_1\sigma_2...\sigma_n$ over Σ is a finite sequence of n symbols $\sigma_i \in \Sigma$; let $|w|$ denote its length (i.e., n). Let λ denote the *empty string*, the unique string of length 0. We write wv for the concatenation of strings w and v; note that $w\lambda = \lambda w = w$. By Σ^* we denote the set of all strings over Σ, including λ. A set L of strings or *language* is a subset of Σ^*. The *size* $||L||$ of a language L is the sum of the lengths of its composite strings; i.e. $||L|| = \sum_{w \in L} |w|$. A *grammar* G is some finite representation of a language; we write $L(G)$ for the language represented by G.

We often make use of special boundary symbols $\rtimes$ and $\ltimes$, assumed not to be members of Σ, to mark the beginning and end of a strings, respectively. We abuse the concatenation notation somewhat and write wL (Lw) for the concatenation of w to the beginning (and/or end) of each string in L. Thus, $\rtimes\Sigma^*\ltimes$ denotes the set of strings over Σ marked with beginning and end boundaries.

A string u is a *substring* of w if there are two strings v_1, v_2 such that $w = v_1uv_2$. Let $|w|$ be the length of w. We say u is a k-*factor* of w if u is a substring of $\rtimes w\ltimes$ and $|u| = k$. Note that λ is the single 0-factor for any string. Define a function $\texttt{fac}_k$ which returns the k-factors of $\rtimes w\ltimes$ (or $\rtimes w\ltimes$ if $| \rtimes w \ltimes | \leq k$):

$$\texttt{fac}_k(w) = \begin{array}{ll} \{u \mid u \text{ is a } k\text{-factor of } w\} & \text{if } | \rtimes w \ltimes | > k \\ \{\rtimes w \ltimes\} & \text{otherwise} \end{array}$$

Finally, we extend $\texttt{fac}_k$ to languages such that $\texttt{fac}_k(L) = \bigcup_{w \in L} \texttt{fac}_k(w)$.

2.1 Learning Paradigm

The learning paradigm considered here is exact identification in the limit [2] in polynomial time and data [7]. Under this paradigm, a learning algorithm is assumed to be given sufficient examples to identify the learning target. However, the amount of information necessary to distinguish a learning target must be polynomial in the size of the representation (grammar) for this target, and the algorithm must run in time polynomial in the size of the input data.

A class $\mathcal{C}$ of languages is said to be describable by a class $\mathcal{G}$ of grammars if each $G \in \mathcal{G}$ is finite and there is a total, surjective naming function $f : \mathcal{G} \to \mathcal{C}$ from any grammar to a language in $\mathcal{C}$. The goal is an algorithm A which, given a finite set of examples from any $L \in \mathcal{C}$, returns a grammar $G \in \mathcal{G}$ for L.

Definition 1 ($(\mathcal{C}, \mathcal{G})$-learning algorithm). *Let $\mathcal{C}$ be a class of languages over an alphabet Σ describable by a class $\mathcal{G}$ of grammars. An input* sample *I of some $L \in \mathcal{C}$ is a finite set of strings in L; that is, $I \subseteq L$. A* $(\mathcal{C}, \mathcal{G})$-learning algorithm *is a function $A : \mathcal{P}_{\text{fin}}(\Sigma^*) \to \mathcal{G}$ that takes a finite sample of strings as an input and outputs a grammar in $\mathcal{G}$.*

Given a particular $(\mathcal{C}, \mathcal{G})$-learning algorithm, a *characteristic sample* of a language in $\mathcal{C}$ is a sample of that language such that the algorithm is guaranteed to identify a grammar which exactly describes the language.

Definition 2 (Characteristic sample). *A* characteristic sample *I_C for a language L for a $(\mathcal{C}, \mathcal{G})$-learning algorithm A is a sample of L such that for all samples $I \supseteq I_C$ of L, $L = L(A(I))$.*

We also define a further goal which is that an algorithm can induce any member of a class of languages in time polynomial in the size of the data, and that the size of the characteristic sample for any language is polynomial in the size of its grammar the size of a grammar G (which is denoted $|G|$).

Definition 3 (Identification in polynomial time and data).
A $(\mathcal{C}, \mathcal{G})$-learning algorithm A identifies *$\mathcal{C}$ in polynomial time and data when there are two polynomial functions f and g such that, for any language $L \in \mathcal{C}$, given a sample I containing a characteristic sample of L, $L(A(I)) = L$, and*

1. *the size I_C of the characteristic sample of L is, for any grammar $G \in \mathcal{G}$ that represents L, at most $f(n)$, where $n = |G|$.*
2. *A returns a grammar for L in $\mathcal{O}(g(||I||))$ time.*

3 Tier-based Strictly Local Languages

3.1 Strictly Local Languages

The Tier-based Strictly Local (TSL) class of formal languages [6] is a generalization of the Strictly Local (SL) (otherwise known as Locally Testable in the Strict

Sense) class of languages first studied in [13]. It will be useful to understand some concepts with respect to the SL languages, so we review these languages first.

Given an alphabet Σ, define a *SL$_k$ grammar* as a set $R \subseteq \texttt{fac}_k(\Sigma^*)$. The language of this grammar is thus $L(R) = \{w \in \Sigma^* \mid \texttt{fac}_k(w) \cap R = \emptyset\}$. In other words, R defines a set of *forbidden k-factors* that do not appear as substrings of any string in $L(R)$. For example, for the alphabet $\Sigma = \{a, b\}$, $R = \{\rtimes a, a \ltimes\}$ describes the set of strings $L(R) = \{\lambda, b, bb, bab, bbb, ...\}$; i.e., exactly the strings in Σ^* that do not begin or end with an a. This $L(R)$ is a Strictly 2-Local (SL_2) language, as R consists of 2-factors of Σ^*.

The SL languages lie at the bottom of the *Sub-Regular Hierarchy* of subclasses of the Regular languages [13, 17, 16]. Though simple in computational complexity, SL languages are formally similar to n-gram models [10], and have been shown to model local co-occurrence restrictions in natural language phonology [4, 16]. Furthermore, given k, any SL_k class is efficiently identifiable in the limit from positive data using a very simple procedure [1, 3]. For a sample I of strings from a target SL_k language L, we can return a grammar R where $R = \texttt{fac}_k(\Sigma^*) - \bigcup_{w \in I} \texttt{fac}_k(w)$ such that $L(R) = L$. Remark 1 notes the efficiency of this procedure (which bears on the results of §5).

Remark 1. Calculating $\texttt{fac}_k(I)$ is linear in $||I||$.

Proof. Returning the k-factors of a string w takes $| \rtimes w \ltimes | - k - 1$ steps, which is effectively $\mathcal{O}(|w|)$. For a set of strings I the complexity is thus $\mathcal{O}(||I||)$. □

3.2 TSL Languages, Grammars, and Known Properties

The Tier-based Strictly Local (TSL) languages generalize the SL languages in that forbidden k-factors are interpreted to only apply to some subset or *tier* T of the alphabet Σ. Following [6], we define a function $\texttt{erase}_T$ that returns a string with all non-members of T removed: $\texttt{erase}_T(\sigma_1\sigma_2...\sigma_n) = u_1u_2...u_n$ where each $u_i = \sigma_i$ if $\sigma_i \in T$; $u_i = \lambda$ otherwise. For example, for $\Sigma = \{a, b\}$ and $T = \{b\}$, then $\texttt{erase}_T(abbaaabab) = bbbb$. We then can define the *tier k-factors* of a string as $\texttt{fac}_k(\texttt{erase}_T(w))$. A TSL grammar is thus a pair $\langle T, R\rangle$ where T is a tier and $R \subseteq \texttt{fac}_k(T^*)$ is a set of *forbidden tier k-factors*. The language of such a grammar is thus $L(\langle T, R\rangle) = \{w \in \Sigma^* \mid \texttt{fac}_k(\texttt{erase}_T(w)) \cap R = \emptyset\}$.

For example, for $\Sigma = \{a, b\}$, $L(\langle \{b\}_T, \{bbb\}_R\rangle)$ is the set of strings such that no 3 bs occur in the string, no matter how many as intervene between the bs (note the above string *abbaaabab* is not in this language; *abaaab*, for example, is). Because they restrict how members on T may appear in a language, we will sometimes refer to members of R as *restrictions* on symbols on the tier.

We say a grammar $G = \langle T, R\rangle$ is *canonical* iff T is as small as it can be without changing the language. The notion of canonical grammar leads to important properties that a learning algorithm can use to induce the correct tier.

Definition 4 (Canonical TSL_k grammar). *A TSL_k grammar $G = \langle T, R\rangle$ is* canonical *iff for any TSL_k grammar $G' = \langle T', R'\rangle$, $L(G) = L(G')$ implies $T \subseteq T'$.*

Example 1. Let $\Sigma = \{a, b, c\}$. Let $G = \langle \{b\}_T, \{bb\}_R \rangle$; $L(G)$ is the set of strings of any number of as and cs but at most one b. G is canonical as no TSL_2 grammar for $L(G)$ a smaller tier. For example, $L(\langle \{a, b\}_{T'}, \{bb\}_{R'} \rangle) = L(G')$ but $T \subset T'$.

Note that, under this definition, the canonical grammar for Σ^* is $\langle \emptyset, \emptyset \rangle$, which does not specify restrictions on any $\sigma \in \Sigma$, as the empty set is the smallest tier needed to describe Σ^* (note that this is the unique case in which R can be empty in a canonical grammar). Likewise, the canonical grammar for the empty language is $\langle \emptyset, \{\rtimes, \ltimes\} \rangle$ for $k = 1$, and $\langle \emptyset, \{\rtimes \ltimes\} \rangle$ for $k > 1$.

Several existing learning results pertain to the TSL languages. First, given T, learning R is identical to SL learning [6] procedure outlined in §3.1 [6]. An interesting question, however, is whether or not T can also be learned. As Σ is finite, the set of possible Ts is finite, and so the class of possible TSL languages for a given Σ and k is finite, and thus can be learned via an enumerative method [2]. Thus, in principle, there is a method for learning T. However, as TSL grammars appear to be relevant to linguistic cognition and learning [5, 11], it is of interest to pursue a non-enumerative method that meets the efficiency criteria in §2.1.

For TSL_2 languages, such a method does exist. The Tier-based Strictly 2-Local Inference Algorithm (2TSLIA) of [9] can learn TSL_2 languages in quartic time with a data sample bounded by a constant. Essentially, the 2TSLIA does this by recursively removing symbols in Σ from its hypothesis for T and checking the tier 2-factors present in the data based on this new guess for the tier.

The algorithm described in the following section improves on this result in two ways. One, it can learn a TSL_k language given any k. Two, it accomplishes this in quadratic time by removing the recursive step.

3.3 Some Useful Properties of Canonical TSL Grammars

Given a canonical grammar $\langle T, R \rangle$, Lemma 1 gives the following important property of members of T which belong to some restriction in R. We henceforth assume factors are of strings in $\rtimes \Sigma^* \ltimes$ (e.g., ignoring all $u_1 \rtimes u_2$ where $u_1 \neq \lambda$).

Lemma 1. *If $G = \langle T, R \rangle$ is a canonical TSL_k grammar, then $\forall \sigma \in T$ for which there exists some $u_1 \sigma u_2 \in R$, $\exists v_1 \sigma v_2 \in R$ such that $v_1 v_2 \in \mathtt{fac}_{k-1}(L(G))$.*

Proof. Consider a grammar G and $L = L(G)$ for which there exists a $\sigma \in T$ such that a $u_1 \sigma u_2 \in R$ and for *all* $v_1 \sigma v_2 \in R$, $v_1 v_2 \notin \mathtt{fac}_{k-1}(L)$. We show that there is a $G' = \langle T', R' \rangle$ such that $\sigma \notin T'$ but $L(G') = L$, and thus G is not canonical.

First, if $v_1 v_2 \notin \mathtt{fac}_{k-1}(L)$, then all tier k-factors containing it must be banned. This means that either $v_1 v_2 = \rtimes v_3 \ltimes$ and $v_1 v_2 \in R$, or for all $\tau \in (T \cup \{\rtimes, \ltimes\})$, both $\tau v_1 v_2 \in R$ and $v_1 v_2 \tau \in R$ (allowing that $\rtimes$ can only be initial and $\ltimes$ final). Note that this applies recursively, e.g. that if $v_1 v_2 \tau = v_1' \sigma v_2'$, then for all $\tau' \in (T \cup \{\rtimes, \ltimes\}), v_1' v_2' \tau \in R$ and $\tau' v_1' v_2' \in R$ (again modulo the restrictions on $\rtimes$ and $\ltimes$). Thus for any string $v = t_0 \sigma t_1 \sigma ... t_{n-1} \sigma t_n \in R$ consisting of n σs interpolated with $n + 1$ strings $t_i \in ((T \cup \{\rtimes, \ltimes\}) - \{\sigma\})^*$, then $u t_0 t_1 ... t_n u' \in R$ for all strings $u, u' \in ((T \cup \{\rtimes, \ltimes\}) - \{\sigma\})^*$ such that

$ut_0t_1...t_nu' \in \mathtt{fac}_k((T - \{\sigma\})^*)$. In other words, for all $ut_0...t_nu' \in \mathtt{fac}_k(\rtimes(T - \{\sigma\})^*\ltimes)$, $ut_0...t_nu' \in R$. Now consider $G' = \langle T', R'\rangle$ where $T' = T - \{\sigma\}$ and $R' = R - \{w | w = v_1\sigma v_2 \text{ for some } v_1, v_2 \in \mathtt{fac}_k(T^*)\}$. That is, G' is identical to G except that σ has been completely removed from T' and R'.

Let $L' = L(G')$. We show that $L' = L$. For $L' \subseteq L$, note that $R' \subset R$ and, because there is no $v_1\sigma v_2 \in R'$, for all $v \in R'$, $\mathtt{erase}_{T'}(v) = \mathtt{erase}_T(v)$. So if there is some $w \in \Sigma^*$ s.t. there is a $v \in R'$ and $v \in \mathtt{fac}_k(\mathtt{erase}_{T'}(w))$, $v \in R$ and $u \in \mathtt{fac}_k(\mathtt{erase}_T(w))$. Thus $w \notin L'$ implies $w \notin L$.

For $L \subseteq L'$, consider any $v = t_0\sigma t_1\sigma...t_{n-1}\sigma t_n \in R$ (where each t_i is a string in $((T \cup \{\rtimes, \ltimes\}) - \{\sigma\})^*$ and some $w \in \Sigma^*$ such that $v \in \mathtt{fac}_k(\mathtt{erase}_T(w))$. As shown above, for all $ut_0...t_nu' \in \mathtt{fac}_k((T - \{\sigma\})^*)$, $ut_0...t_nu' \in R$, and so also $ut_0...t_nu' \in R'$. Thus there is some $ut_0t_1...t_nu' \in R'$ such that $ut_0t_1...t_nu' \in \mathtt{fac}_k(\mathtt{erase}_{T'}(w))$, and so $w \notin L$ implies $w \notin L'$.

Thus, $L(G) = L(G')$ but $T \not\subseteq T'$, so G is not canonical. □

Example 2. Consider $G = \langle \{b, a\}_T, \{bab\}_R\rangle$. This is canonical for $k = 3$, and satisfies Lemma 1. For example, for a, bab is a banned 3-factor but bb is an allowed 2-factor in $L(G)$ (witnessed by the string $abba \in L(G_2)$) and for b, ab is an allowed 2-factor (as $aab \in L(G_2)$). Consider then G' which is identical except bb is not a 2-factor in $L(G')$. This means that $L(G') = \langle \{a, b\}_{T'}, \{bab, \rtimes bb, bb\ltimes, bba, abb\}_{R'}\rangle$, where R' bans all tier 3-factors that contain bb. However, $L(G')$ is not canonical, as $L(G') = L(\langle \{b\}, \{\rtimes bb, bbb, bb\ltimes\}\rangle)$, and $\{b\} \subset T'$.

Finally, if G is a canonical grammar then the following holds for all $\sigma \notin T$.

Lemma 2. *The following are both true if and only if $\sigma \notin T$:*

1. $\forall v_1v_2 \in \mathtt{fac}_{k-1}(L), v_1\sigma v_2 \in \mathtt{fac}_k(L)$.
2. $\forall v_1\sigma v_2 \in \mathtt{fac}_{k+1}(L), v_1v_2 \in \mathtt{fac}_k(L)$.

Proof. ($\rightarrow$) This follows straightforwardly from the definition of a TSL grammar. For (1), if $\sigma \notin T$, then for any $w = w_1v_1v_2w_2 \in L$, there is also a $w' = w_1v_1\sigma v_2w_2 \in L$. This is because when $\sigma \notin T$, $\mathtt{erase}_T(w) = \mathtt{erase}_T(w')$, and thus the two strings are equivalent with respect to G. The same applies to (2): if $\sigma \notin T$, then for any $w' = w_1v_1\sigma v_2w_2 \in L$, there is also a $w = w_1v_1v_2w_2 \in L$.

($\leftarrow$) We show that for any σ that is a member of T, either (1) or (2) is false. There are two important cases for $\sigma \in T$. Either there is some $v_1\sigma v_2 \in R$, or there is no such member of R and σ is thus unrestricted. For the case in which there *does* exist a $v_1\sigma v_2 \in R$, it follows directly from Lemma 1 that (1) is false, because we know that there is some $u_1u_2 \in \mathtt{fac}_{k-1}(L(G))$ for which $u_1\sigma u_2 \in R$, which implies that $u_1\sigma u_2 \notin \mathtt{fac}_k(L(G))$.

For any $\sigma \in T$ for which there is *not* some $v_1\sigma v_2 \in R$, (2) will always be false as long as R is nonempty. Consider any $v_1'v_2' \in R$. It must be true that $v_1'\sigma v_2' \in \mathtt{fac}_{k+1}(L)$, because any $u \in \mathtt{fac}_k(v_1'\sigma v_2')$ necessarily contains σ, and thus $u \notin R$. However, because $v_1'v_2' \in R$, $v_1'v_2' \notin \mathtt{fac}_k(L)$ by the definition of a TSL grammar, and so (2) is false. □

Example 3. For $\Sigma = \{a, b, c\}$, consider $G = \langle \{a, b, c\}_T, \{bab\}_R\rangle$. For b, (1) is false because bb is a 2-factor in $L(G)$, but bab is not a 3-factor in $L(G)$. For c, (2) is false because $bcab$ is a 4-factor in $L(G)$ but bab is not a 3-factor.

4 The Tier-based Strictly k-Local Inference Algorithm

We can now define an efficient learning algorithm for learning a TSL_k language, for any k, by taking advantage of the properties of canonical TSL_k grammars proven in Lemmas 1 and 2 in §3.3. The algorithm is given below in Algorithm 1. Proofs of its correctness and efficiency are given in §5.

Data: A finite input sample $I \subset \Sigma^*$, a positive integer k
Result: A k-TSL grammar $G = \langle T, R \rangle$
Initialize $T = \Sigma$;
foreach $\sigma \in T$ **do**
  if a) $\forall v_1 v_2 \in \mathtt{fac}_{k-1}(I), v_1 \sigma v_2 \in \mathtt{fac}_k(I)$ and
    b) $\forall v_1 \sigma v_2 \in \mathtt{fac}_{k+1}(I), v_1 v_2 \in \mathtt{fac}_k(I)$ **then**
    remove σ from T
  end
end
Initialize R to $\mathtt{fac}_k(T^*)$;
foreach $u \in \mathtt{fac}_k(T^*), \mathtt{fac}_k(I)$ **do** remove u from R;

Algorithm 1: The Tier-based Strictly k-Local Inference Algorithm

Recall from Lemma 2 that, for a language L whose canonical grammar is $G = \langle T, R \rangle$, for any $\sigma \in \Sigma$, $\sigma \notin T$ if and only if (1) $\forall v_1 v_2 \in \mathtt{fac}_{k-1}(L), v_1 \sigma v_2 \in \mathtt{fac}_k(L)$ and (2) $\forall v_1 \sigma v_2 \in \mathtt{fac}_{k+1}(L), v_1 v_2 \in \mathtt{fac}_k(L)$. Given a set of input data I, the algorithm searches for exactly these pairs for each $\sigma \in \Sigma$. If all such pairs are present in I, then σ is removed from the algorithm's hypothesis for the tier. In this way, the algorithm can identify members of Σ that have the properties established in Lemma 2 and correctly remove them from the tier.

To describe the algorithm in more detail, it first sets its hypothesis for the tier to Σ. Then, in the first **foreach** loop, it cycles through each $\sigma \in \Sigma$, searching for the above information. Specifically, for every $v_1 v_2$ in the $k-1$-factors of I, it searches for $v_1 \sigma v_2$ in the k-factors of I, in order to determine whether σ is permitted to co-occur with every $k-1$-factor that is otherwise unrestricted. Likewise, for every $v_1 \sigma v_2$ in the $k+1$-factors of I, the algorithm searches for $v_1 v_2$ in the k-factors of I, in order to determine that there is no k-factor whose occurrence in the language is *dependent* on the intervening σ. If it finds all such pairs, then it removes σ from its hypothesis for the tier. Having calculated a hypothesis for the tier, it then calculates a hypothesis for R by initializing its hypothesis to the full set of possible k-factors of T^* and then removing from this hypothesis any tier k-factor it finds in the k-factors of I. In other words, all tier k-factors not seen in I will appear in the algorithm's hypothesis for R.

Example 4. Consider a case in which $k = 3$, $\Sigma = \{a, b\}$, and the target language is $L(G_*)$ where $G_* = \langle \{b\}_{T_*}, \{bbb\}_{R_*} \rangle$ and the algorithm is given as an input a sample I from L where $I = \{\lambda, a, b, bb, aaa, bab, abba, aabaabaa\}$. Given I, the algorithm will return a grammar $G = \langle T, R \rangle$ where $T = T_*$ and $R = R_*$. To see how, Table 1 lists the $k-1$-factors, k-factors, and $k+1$-factors of I.

First, the algorithm initializes its hypothesis T to $\{a, b\}$, then in the first **foreach** loop checks if either a or b meets the conditions in the **if** statement for removal from T. Say the algorithm first considers a. In condition (a) of the **if** statement, the algorithm checks that for every $v_1v_2 \in \mathtt{fac}_{k-1}(I)$, $v_1av_2 \in \mathtt{fac}_k(I)$. This is true in all cases: for example, $\rtimes a \in \mathtt{fac}_{k-1}(I)$ (row (b) in Table 1), and $\rtimes aa \in \mathtt{fac}_k(I)$ (row (e)); also, $bb \in \mathtt{fac}_{k-1}(I)$ (row (d)), and $abb, bab, bba \in \mathtt{fac}_k(I)$ (rows (g), (f), and (g), respectively). Thus, a satisfies condition (a) of the **if** statement. Next, condition (b) in the **if** statement checks that, for every $v_1av_2 \in \mathtt{fac}_{k+1}(I)$, $v_1v_2 \in \mathtt{fac}_k(I)$. This is also true: for example, $\rtimes aaa \in \mathtt{fac}_{k+1}(I)$ (row (e)) and $\rtimes aa \in \mathtt{fac}_k(I)$ (also row (e); also, $abba \in \mathtt{fac}_{k+1}(I)$ (row (g)) and $bba, abb \in \mathtt{fac}_k(I)$ (also row (g)). So a also satisfies condition (b) of the **if** statement. It is thus removed from T.

This, however, does not occur with b. In condition (a) of the **if** statement, the algorithm will check to see if the $k-1$-factor bb (which appears in row (c)) has a corresponding k-factor bbb. This does not appear in I. In fact, no sample of $L(G_*)$ will contain bbb as a 3-factor, as $bbb \in R_*$. Thus b will remain on T.

Having checked through each symbol in Σ for removal from T, the algorithm then calculates its hypothesis for R in the second **foreach** loop. First, it initializes R to all possible tier k-factors; in this case, this is equal to $\{\rtimes\ltimes, \rtimes b\ltimes, \rtimes bb, bb\ltimes, bbb\}$. Note that all but bbb are attested in Table 1, so the **foreach** loop will remove all but bbb from R. Thus, the algorithm returns a grammar $\langle T, R\rangle = \langle T_*, R_*\rangle$.

Input string	k-1-**factors**	k-**factors**	k+1-**factors**
a. λ	$\rtimes\ltimes$		
b. a	$\rtimes a, a\ltimes$	$\rtimes a\ltimes$	
c. b	$\rtimes b, b\ltimes$	$\rtimes b\ltimes$	
d. bb	bb	$\rtimes bb, bb\ltimes$	$\rtimes bb\ltimes$
e. aaa	aa	$\rtimes aa, aaa, aa\ltimes$	$\rtimes aaa, aaa\ltimes$
f. bab	ba, ab	$\rtimes ba, bab, ab\ltimes$	$\rtimes bab, bab\ltimes$
g. $abba$	ab, ba	$\rtimes ab, abb, bba, ba\ltimes$	$\rtimes abb, abba, bba\ltimes$
h. $aabaabaa$	—	aab, aba, baa	$\rtimes aab, aaba, abaa, baa\ltimes$

Table 1. Breakdown of the factors of I from Example 4. Each row gives the new 2-, 3-, and 4-factors (i.e., the $k-1$-, $k-$, and $k+1$-factors) generated from each data point.

Of course, the algorithm can only accurately determine if a symbol σ is on the tier if the data in I is somehow representative of the target language L. Definition 5 below defines a set D of data that I must contain in order for it to be representative of L. The following section will then prove that any such *representative sample* is a *characteristic* sample, in the sense defined in §2.1.

Definition 5 (Representative sample). *For a TSL_k language L whose canonical grammar is $G = \langle T, R\rangle$, $S = \mathtt{fac}_k(T^*) - R$, $P = \{\sigma \mid v_1\sigma v_2 \in R\}$, a set D is a* representative sample *of L if it meets the following conditions:*

1. ***Non-tier element condition.*** *For all non-tier elements* $\sigma \notin T$,
   (a) $\forall v_1 v_2 \in \mathtt{fac}_{k-1}(L), \exists v_1 \sigma v_2 \in \mathtt{fac}_k(D)$.
   (b) $\forall v_1 \sigma v_2 \in \mathtt{fac}_{k+1}(L), \exists v_1 v_2 \in \mathtt{fac}_k(D)$.
2. ***The tier element condition.*** *For each* $\sigma \in T$,
   (a) *if* $\sigma \in P$, *then* $\exists v_1 v_2 \in \mathtt{fac}_{k-1}(D)$ *s.t.* $v_1 \sigma v_2 \in R$
   (b) *if* $\sigma \notin P$, *then* $\exists v_1 \sigma v_2 \in \mathtt{fac}_{k+1}(D)$, *where* $v_1 v_2 \in R$
3. ***Allowed tier*** k***-factor condition.*** *For all* $v \in S, \exists w \in D$ *s.t.* $v \in \mathtt{fac}_k(w)$.

Essentially, Definition 5 states that a representative sample is a set that contains the relevant information for each $\sigma \in \Sigma$ to distinguish it as a non-member or member of the tier. The *non-tier element condition* ensures that for each $\sigma \notin T$, a representative sample includes in its k-factors at least one $v_1 \sigma v_2$ for every $v_1 v_2$ in the $k-1$-factors of L and at least one $v_1 v_2$ for every $v_1 \sigma v_1$ in the $k+1$-factors of L. Again, the presence of these k-factors was shown by Lemma 2 to be definitional for non-members of T.

For any $\sigma \in T$, the *tier element condition* ensures that the representative sample includes sufficient information to identify σ as a member of the tier. This is done by ensuring the inverse of Lemma 2 is true: that for σ for which such there is some $v_1 \sigma v_2 \in R$ (and thus will never appear as a k-factor in a sample of L), there is some $v_1 v_2 \in \mathtt{fac}_{k-1}(D)$. Note that Lemma 1 guarantees such a $v_1 v_2$ to exist as a $k-1$-factor of L. For $\sigma \in T$ for which no such k-factor exists in R, part (b) of the tier element condition states that a representative sample must include some $v_1 \sigma v_2$ as a $k+1$-factor, where $v_1 v_2 \in R$. Thus, for any $\sigma \in T$, a representative sample includes information showing that σ does not obey the properties Lemma 2 establishes for non-tier elements.

Finally, the *allowed tier k-factor condition*, ensures that the representative sample includes all tier k-factors allowed by G.

5 Identification in polynomial time and data

We now prove that the kTSLIA can correctly identify any TSL language in polynomial time and data. First, we establish its time complexity.

Lemma 3. *Given an input sample I of size n, kTSLIA runs in time polynomial in the size of n.*

Proof. The algorithm needs to calculate the $k-1$-, k-, and $k+1$-factors of I, each of which (per Remark 1) takes time linear in n. Note also that the cardinality of each of these sets of factors is bound by n.

The first **foreach** loop is called exactly $|\Sigma|$ times. Checking condition (a) in this loop requires that for every member of $\mathtt{fac}_{k-1}(I)$, the algorithm passes through $\mathtt{fac}_k(I)$ $k+1$ times (for each $\sigma_1 ... \sigma_{k-1} \in \mathtt{fac}_{k-1}(I)$, it must check for $v_1 \sigma v_2 \in \mathtt{fac}_k(I)$ where $v_1 = \lambda, v_2 = \sigma_1 ... \sigma_{k-1}$, $v_1 = \sigma_1, v_2 = \sigma_2 ... \sigma_{k-1}$, $v_1 = \sigma_1 \sigma_2, v_2 = \sigma_3 ... \sigma_{k-1}$, etc.). Thus condition (a) requires $n \cdot (k+1) \cdot n = (k+1) \cdot n^2$ steps. Checking condition (b) requires that, in the worst case, for each member of $\mathtt{fac}_{k+1}(I)$, the algorithm passes through $\mathtt{fac}_k(I)$ $k+1$ times, taking $(k+1) \cdot n^2$

steps. Since $\texttt{fac}$ is calculated in linear time, the time complexity for this loop is therefore $\mathcal{O}(|\Sigma|2(k+1)n^2) = \mathcal{O}(n^2)$ since $|\Sigma|$ and k are constants.

The final step of the algorithm, identifying the permissible tier-based k-factors, requires one final run through the k-factors of I, which again takes n steps. In total, the algorithm runs in $\mathcal{O}(3n + n^2 + n) = \mathcal{O}(n^2)$ time. □

Lemma 4. *For a language L, the size of the representative sample D for L is polynomial in the size of G for any grammar G of L.*

Proof. The non-tier condition requires that for each $\sigma \notin T$ and for each $v_1v_2 \in \texttt{fac}_{k-1}(L)$, the sample minimally contains a $v_1\sigma v_2$ factor. The length of these strings equals $|\Sigma - T| \cdot |\Sigma|^{k-1} \cdot k$. Additionally, for every $\sigma \notin T$ and for each $v_1\sigma v_2 \in \texttt{fac}_{k+1}(L)$, the sample includes minimally a v_1v_2 substring. The combined length of these strings is $|\Sigma - T| \cdot |\Sigma|^k \cdot (k+1)$. The tier condition requires that for each $\sigma \in T$ there is minimally either one string v_1v_2 such that $v_1\sigma v_2 \in R$ or one string $v_1\sigma v_2$ such that $v_1v_2 \in R$. The total length of these strings is at most $|T| \cdot (k+1)$.The allowed tier k-factor condition requires for each $u \in S$ that minimally u is contained in the sample. Hence the length of these words equals $|S|$. Altogether, there is therefore a sample D such that $||D|| = |\Sigma - T| \cdot |\Sigma|^{k-1} \cdot k + |\Sigma - T| \cdot |\Sigma|^k \cdot (k+1) + |T| \cdot (k+1) + |S|$. Since $|T|$ is bounded by $|\Sigma|$ and $|S|$ and $|R|$ bounded by $|\Sigma|^k$, $||D||$ is bounded by $\mathcal{O}(|\Sigma|^k)$. As $|\Sigma|$ and k are constant, $||D||$ is thus bounded by a constant. □

Lemma 5. *Given any superset I of a representative sample D for a TSL_k language L whose canonical grammar is $G = \langle T, R\rangle$, the kTSLIA will return a grammar $G' = \langle T', R'\rangle$ such that $T' = T$.*

Proof. This is essentially due to the fact that a representative sample is defined as one which contains the information established in Lemmas 1 and 2 as distinguishing symbols on the tier from non-tier symbols in a canonical TSL grammar. Let $L = L(G)$. We show that $\sigma \notin T$ implies $\sigma \notin T'$ and that $\sigma \in T$ implies $\sigma \in T'$, and thus that $T = T'$.

First, for any non-tier element $\sigma \notin T$, $\sigma \notin T'$ if and only if $\forall v_1v_2 \in \texttt{fac}_{k-1}(I), v_1\sigma v_2 \in \texttt{fac}_k(I)$ (condition (a) of the algorithm) and $\forall v_1\sigma v_2 \in \texttt{fac}_{k+1}(I), v_1v_2 \in \texttt{fac}_k(I)$ (condition (b) of the algorithm). Part (1a) of the non-tier element condition for the representative sample (Def 5-1a) guarantees that for any possible $k-1$-factor v_1v_2 of L that $v_1\sigma v_2 \in \texttt{fac}_k(D)$. (Such a $k+1$ factor is guaranteed to exist in L by Lemma 2-2. Recall that if $\sigma \in T$, then $\texttt{erase}_T(v_1v_2) = \texttt{erase}_T(v_1\sigma v_2)$, and so G cannot distinguish between them; i.e. v_1v_2 appears in L iff $v_1\sigma v_2$ does as well). Thus for any superset I of D consistent with L, if $v_1v_2 \in \texttt{fac}_{k-1}(I)$ then $v_1\sigma v_2 \in \texttt{fac}_k(I)$. So σ satisfies condition (a).

Similarly, part (1b) of the non-tier element condition for the representative sample requires that for any $v_1v_2 \in \texttt{fac}_k(L)$, then $v_1\sigma v_2 \in \texttt{fac}_{k+1}(D)$. (Again, this is possible because G will not distinguish between v_1v_2 and $v_1\sigma v_2$, and so if $v_1v_2 \in \texttt{fac}_k(L)$, then $v_1\sigma v_2 \in \texttt{fac}_{k+1}(L)$). Thus, for any $v_1v_2 \in \texttt{fac}_k(I)$ that the algorithm encounters, it will also $v_1v_2 \in \texttt{fac}_k(I)$. Thus σ will also satisfy condition (b) and be taken off of the tier. Thus $\sigma \notin T$ implies $\sigma \notin T'$.

For $\sigma \in T$, if there is a $v_1\sigma v_2 \in R$, then part (2a) of the tier element condition for the representative sample (Def 2-1a requires that $v_1v_2 \in \mathtt{fac}_{k-1}(D)$. Thus I is guaranteed to contain some $k-1$-factor v_1v_2 for which there will be no $v_1\sigma v_2 \in \mathtt{fac}_k(I)$, as long as I is consistent with L. Thus σ will fail condition (a) of the algorithm for removal from the tier.

If there is no $v_1\sigma v_2 \in R$, then part (2b) of the tier element condition for the representative sample (Def 2-2b) requires that there must be some $v_1v_2 \in R$ such that $v_1\sigma v_2 \in \mathtt{fac}_{k+1}(D)$. (Such a $k+1$ factor is guaranteed to exist because, again, there are no restrictions on σ.) As any sample consistent with L will never contain v_1v_2, σ is guaranteed to fail condition (b) for removal from the tier.

Thus, in either situation σ fails one of the algorithm's conditions (a) and (b) for removal from the tier, and so $\sigma \in T$ implies $\sigma \in T'$. Thus $T = T'$. □

Lemma 6. *Given any superset I of a representative sample D for a TSL_k language L whose canonical grammar is $G = \langle T, R\rangle$, the kTSLIA will return a grammar $G' = \langle T', R'\rangle$ such that $R' = R$.*

Proof. By Lemma 5, $T = T'$. Thus, in the last two lines of the algorithm R' is initialized to $\mathtt{fac}_k(T'^*) = \mathtt{fac}_k(T^*)$, and the final **foreach** loop will correctly locate all $u \in \mathtt{fac}_k(T^*)$ found in D. By Definition 2 of the representative sample, D will contain all allowable tier substrings in $S = \mathtt{fac}_k(T^*) - R$. These will thus all be removed from R' by the algorithm, and so $R' = \mathtt{fac}_k(T^*) - S = R$. □

Lemma 7. *A representative sample D for L is a characteristic sample for L.*

Proof. From Lemmas 5 and 6, given a superset of D the kTSLIA will return a grammar $G' = \langle T', R'\rangle$ such that $T' = T$ and $R' = R$, where $\langle T, R\rangle$ is the canonical grammar for L. Thus $L(G') = L$. □

Theorem 1. *For any TSL_k language L, the kTSLIA identifies L in polynomial time and data.*

Proof. From Lemmas 3, 4, and 7. □

6 Discussion and Conclusion

The main contributions of this paper have been twofold. One, it redefined the notion of a canonical TSL_k grammar, and proved several resulting important properties of members and non-members of the tier. Two, it established an algorithm which uses these properties that is guaranteed to induce a TSL_k grammar in polynomial time and data.

Future work can now examine how results can be applied directly to natural language learning, the context that TSL languages were initially designed to model. While this paper has established, based on the aforementioned properties of members and non-members of the tier, a characteristic sample for the kTSLIA, it remains to be seen whether or not this characteristic sample appears

in natural language data. Future work can examine to what extent this information is present in phonological corpora, and discuss what modifications may be necessary for the algorithm to be successful in these situations (as done in [8] for the 2TSLIA). Furthermore, while the learner presented here is fundamentally categorical and thus brittle in the face of noisy data, future work can build on the results here to create a stochastic version of the kTSLIA.

References

1. García, P., Vidal, E., Oncina, J.: Learning locally testable languages in the strict sense. In: Proceedings of the Workshop on Algorithmic Learning Theory. pp. 325–338 (1990)
2. Gold, M.E.: Language identification in the limit. Information and Control 10, 447–474 (1967)
3. Heinz, J.: The Inductive Learning of Phonotactic Patterns. Ph.D. thesis, University of California, Los Angeles (2007)
4. Heinz, J.: Learning long-distance phonotactics. Linguistic Inquiry 41, 623–661 (2010)
5. Heinz, J.: Computational phonology part I: Foundations. Language and Linguistics Compass 5(4), 140–152 (2011)
6. Heinz, J., Rawal, C., Tanner, H.G.: Tier-based strictly local constraints for phonology. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics. pp. 58–64. Association for Computational Linguistics, Portland, Oregon, USA (June 2011)
7. de la Higuera, C.: Characteristic sets for polynomial grammatical inference. Machine Learning 27(2), 125–138 (1997)
8. Jardine, A.: Learning tiers for long-distance phonotactics. In: Proceedings of the 6th Conference on Generative Approaches to Language Acquisition in North America (GALANA 2015) (2016)
9. Jardine, A., Heinz, J.: Learning tier-based strictly 2-local languages. Transactions of the Association for Computational Linguistics 4 (2016)
10. Jurafsky, D., Martin, J.H.: Speech and Language Processing. Pearson Prentice Hall, 2nd edition edn. (2009)
11. McMullin, K.: Tier-Based Locality in Long-Distance Phonotactics: Learnability and Typology. Ph.D. thesis, University of British Columbia (2016)
12. McMullin, K., Hansson, G.Ó.: Long-distance phonotactics as Tier-based Strictly 2-Local languages. In: Proceedings of the 2014 Annual Meeting on Phonology. Linguistic Society of America, Washington, DC (2016)
13. McNaughton, R., Papert, S.: Counter-Free Automata. MIT Press (1971)
14. Nevins, A.: Locality in Vowel Harmony. No. 55 in Linguistic Inquiry Monographs, MIT Press (2010)
15. Odden, D.: Adjacency parameters in phonology. Language 70(2), 289–330 (1994)
16. Rogers, J., Heinz, J., Fero, M., Hurst, J., Lambert, D., Wibel, S.: Cognitive and sub-regular complexity. In: Formal Grammar, Lecture Notes in Computer Science, vol. 8036, pp. 90–108. Springer (2013)
17. Rogers, J., Pullum, G.: Aural pattern recognition experiments and the subregular hierarchy. Journal of Logic, Language and Information 20, 329–342 (2011)