Grammar Interpretations and Learning TSL Online

Dakotah Lambert DAKOTAHLAMBERT@ACM.ORG
Department of Linguistics
Institute for Advanced Computational Science
Stony Brook University

Editors: Jane Chandlee, Rémi Eyraud, Jeffrey Heinz, Adam Jardine, and Menno van Zaanen

Abstract

The tier-based strictly local (TSL) languages are a class of formal languages that, alongside the strictly piecewise class, effectively model some long-distance generalizations in natural language (Heinz et al., 2011). Two learning algorithms for TSL already exist: one by Jardine and Heinz (2016) and one by Jardine and McMullin (2017). The former is limited in that it cannot learn all TSL patterns. The latter is restricted to a batch-learning environment. We present a general algorithm without these limitations. In particular we show that TSL is efficiently learnable online via reinterpretation of a strictly local grammar, and this mechanism generalizes to the strictly piecewise class as well. However we note that the known TSL learning algorithms are not robust in the face of interaction with other constraints, posing a challenge for the utility of this class for phonotactics.

Keywords: computational linguistics, formal language theory, online learning, tier-based strictly local languages

1. Introduction

The strictly local (SL) class known from McNaughton and Papert (1971) cannot account for long-distance dependencies. Meanwhile the strictly piecewise (SP) class of Rogers et al. (2010, see also Haines, 1969) can account for long-distance dependencies but cannot handle local constraints. In 2011, Heinz et al. presented a new tier-based strictly local (TSL) class of stringsets, generalizing SL to account for both local phenomena and certain types of long-distance phenomena by relativizing adjacency over a subset of the alphabet. Much like the traditional SL and SP classes, TSL is parameterized by the width k of the factors to which a learner or acceptor attends. But there is another parameter. If the subset T of the alphabet over which adjacency is relativized is known *a priori*, then the input data can be preprocessed by deleting unnecessary symbols and the grammar inferred by any of a number of strictly local learning algorithms, such as that of Garcia et al. (1990) or that of Heinz (2010b). But learning T from data alone has proven difficult, especially in a bounded-memory setting.

Since the introduction of the TSL class, it has been characterized model-, language-, and automata-theoretically by Lambert and Rogers (2020), extended to an even richer class by De Santo and Graf (2019), and shown to be batch-learnable in the style of Gold (1967) by Jardine and Heinz (2016) for k of 2 and by Jardine and McMullin (2017) for arbitrary k. But despite all this progress, no general online learning algorithm has yet been produced, leaving the TSL class behind in an area where several other subregular classes flourish.

© 2021 D. Lambert.

That changes now. Here we discuss an online learning algorithm in the style of Heinz (2010b) that essentially combines the approaches taken in the earlier batch-learning algorithms with a novel representation of the TSL grammar to accommodate online learning. This development removes one argument against TSL descriptions of phonological patterns, namely that human learners likely do not operate in batch. (Batch learning requires perfect awareness of all prior input, which is implausible at best for human learners.)

Section 2 discusses background material. Then section 3 summarizes prior literature on discovering the set of salient symbols and provides space- and time-complexity analyses of the algorithms. Section 4 introduces a structure that can be gathered while determining salience, which provides sufficient information to recover the grammar of forbidden factors while avoiding unbounded data storage. Then section 5 demonstrates a pointwise application of the string extension learning algorithm of Heinz (2010b) to this problem, and introduces a simplification that allows for reinterpretation of an SL grammar as a TSL one.

2. Preliminaries

This section contains background material fundamental to this work. Section 2.1 describes notation that will be used throughout. Section 2.2 provides a brief overview of the SL and TSL classes, with examples. Section 2.3 discusses the learning framework in use, and section 2.4 details the family of learning algorithms that includes our result.

2.1. Notation

Throughout this paper we will use some basic concepts from formal language theory. Given some alphabet Σ, we use Σ^* to refer to the set of all possible finite strings over that alphabet.

A function from a domain $\mathcal{D}$ to a codomain $\mathcal{D}'$ is written $f\colon \mathcal{D} \to \mathcal{D}'$. The set of natural numbers is written $\mathbb{N}$. As a special case reflecting traditional notation for sequences, the value of a function $f\colon \mathbb{N} \to \mathcal{D}'$ at n is written f_n.

2.2. (Tier-Based) Strict Locality

In general, a **factor** is some substructure of a word that is connected in some sense. For the SL languages as defined by McNaughton and Papert (1971), these substructures are simply adjacent sequences of symbols. For example, there are seven factors in the string "abc": "", "a", "b", "c", "ab", "bc", and "abc" itself. Notably, "ac" is not a factor under this interpretation. The size of a factor is the length of the sequence. An SL language is characterized by a set of forbidden factors, containing all and only those strings in which no forbidden factor occurs. If the largest factor in the set of forbidden factors is of size k, then the language is k-SL.

One example of an SL language over the alphabet $\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}\}$ is that in which the forbidden factors are "aa" and "bb". This requires that "a" and "b" alternate within blocks of these two symbols, so "abaca" and "abada" are valid words but "**aa**ca" is not.

One can prove that a language is not k-SL by using what is known as Suffix Substitution Closure: finding two valid words $w = w_p x w_s$ and $v = v_p x v_s$ that share a common factor x of length $k-1$ where $w_p x v_s$ is not a valid word (Rogers and Pullum, 2011, see also De Luca and Restivo, 1980). If such w and v can be found for any k, then the language is not SL.

Consider a slight modification to this example language: not only are “aa” and “bb” forbidden, but also “ada”, “bdb”, “adda”, “bddb”, etc. Essentially, “d” is invisible to the constraint. Now $w = \mathrm{ad}^{k-1}\mathrm{b}$ and $v = \mathrm{bd}^{k-1}\mathrm{a}$ are both valid words, but after suffix-substitution we have “$\mathrm{ad}^{k-1}\mathrm{a}$” which is not valid. Thus this pattern is not SL. The TSL class seeks to capture exactly the constraints that would be SL if only some category of symbols could be ignored. It is defined by applying an SL grammar not to the words themselves but to their projection to a **tier alphabet** T (Heinz et al., 2011). In this case, $\mathrm{T} = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ and the grammar is the same as before. See Lambert and Rogers (2020) for further information on the TSL class, including how to decide membership. Note that while the previous discussion involved forbidden factors, the finiteness of both the factor width and alphabet size allows an equivalent description in terms of permitted factors.

2.3. Our Learning Problem

Gold (1967) introduces a number of learning paradigms, but here we will focus on the case when a language is learned in the limit from distribution-free positive data. We further restrict ourselves to consider only online learning, where the induction function takes as arguments not an entire input set, but only a single input item along with a previously proposed grammar. This section formalizes these notions.

Let $L \subseteq \Sigma^*$ be a stringset and let $L_\circledcirc$ be L with an adjoined element $\circledcirc$ that represents the lack of any string. A text for L is a total, surjective function $t\colon \mathbb{N} \to L_\circledcirc$, an infinite sequence of strings drawn from L that contains each string at least once, and may at some points present no data. Note that for any non-empty stringset, there are infinitely many possible distinct texts. The addition of $\circledcirc$ is a deviation from the original work by Gold but without it no text exists for the empty stringset (Osherson et al., 1986). For a given text t, let $\overrightarrow{t_n}$ represent the sequence $\langle t_0, t_1, \ldots, t_n\rangle$, i.e. the initial segment of t of length $n+1$. We denote the class of texts for $L_\circledcirc$ by $\mathbb{T}$ and the class of initial segments thereof by $\overrightarrow{\mathbb{T}}$.

A grammar is some representation of a mechanism by which the membership of a string in a stringset may be decided. Let $\mathbb{G}$ be a set of possible grammars, and let $\mathcal{L}\colon \mathbb{G} \to \mathcal{P}(\Sigma^*)$ be a function that transforms a grammar into its extension, i.e. the set of all strings that it accepts. Two grammars G_1 and G_2 are equivalent (written $G_1 \equiv G_2$) iff they are extensionally equal, i.e. $\mathcal{L}(G_1) = \mathcal{L}(G_2)$. A batch learner is then a total function $\varphi\colon \overrightarrow{\mathbb{T}} \to \mathbb{G}$. In words, a batch learner is an algorithm that takes as input an initial segment of a text and outputs a guess at the correct grammar.

Given a text t and a learner φ, we say that φ converges on t iff there is some point after which its guess never changes. Formally, that means there exists some $i \in \mathbb{N}$ and some grammar G such that for all $j \geq i$, it holds that $\varphi(\overrightarrow{t_j}) \equiv G$. If given any text t for a stringset L, φ converges on t to a grammar G such that $\mathcal{L}(G) = L$, then we say that φ identifies L in the limit. For any class of stringsets $\mathbb{L} \subseteq \mathcal{P}(\Sigma^*)$, we say that φ identifies $\mathbb{L}$ in the limit iff it identifies L in the limit for all $L \in \mathbb{L}$.

An **online learner** differs from a batch learner only in how it assumes the data is presented. For a batch learner, the function is of type $\varphi\colon \overrightarrow{\mathbb{T}} \to \mathbb{G}$ as discussed. On the other hand, an online learner, sometimes called an **incremental learner**, is of type $\varphi\colon \mathbb{G} \times L_\circledcirc \to \mathbb{G}$, taking as input a previous guess and a single data point (Jain et al., 2007).

Ideally the online learner can take more input over time in an efficient way while maintaining a bounded information store.

2.4. String Extension Learning

Heinz (2010b) described a general algorithm for learning (among others) stringsets that can be described by a set of permitted factors of length bounded above by k. For this case $\mathbb{G} = \mathcal{P}(\Sigma^k)$. In general, given a function $f\colon \Sigma^* \to \mathcal{P}(\Sigma^k)$ that maps a string to the set of factors it contains, one can define a batch learner φ_f as follows

$$\varphi_f(\overrightarrow{t_i}) \triangleq \begin{cases} \varnothing & \text{if } i = 0,\ t_i = \circledcirc \\ f(t_i) & \text{if } i = 0,\ t_i \neq \circledcirc \\ \varphi_f(\overrightarrow{t}_{i-1}) & \text{if } i \neq 0,\ t_i = \circledcirc \\ \varphi_f(\overrightarrow{t}_{i-1}) \cup f(t_i) & \text{otherwise.} \end{cases}$$

Effectively one begins with a guess of the empty grammar, and for each string provided, this guess is updated to include all factors encountered in that string. A factor is **attested** iff it appears in some string in the input. The online variant of this same function is identical except that strings are provided one at a time.

$$\varphi_f(G, w) \triangleq \begin{cases} G & \text{if } w = \circledcirc \\ G \cup f(w) & \text{otherwise.} \end{cases}$$

If the parameter k is fixed and known, this approach identifies the k-SL class in the limit. If this holds and further the parameter T is fixed and known, this approach also identifies k-TSL$^{\mathrm{T}}$ in the limit. But in general when learning TSL generalizations, we want to be able to account for the case where T is unknown.

3. Deciding Salience

When learning a k-TSL$^{\mathrm{T}}$ stringset, if T is not provided then a learner must discover not only the underlying SL constraints, but also the class of symbols that are salient for these constraints. The model-theoretic view of k-TSL$^{\mathrm{T}}$ given by Lambert and Rogers (2020) makes this explicit in that the ordering relation never connects to a position containing a symbol outside of T. There is no way to even talk about the other symbols. It follows then that there is no way to restrict their occurrence, meaning they are freely insertable and deletable in all strings. This property is what allows for the salience-finding algorithm of Jardine and McMullin (2017). We discuss here a simplification of that original work.

Given the set of all factors under adjacency of width up to $k+1$ in the stringset, a symbol x is freely insertable iff for each attested factor of width less than or equal to k, inserting x at each possible point in turn results in an attested factor one symbol wider. Similarly, x is freely deletable iff for each attested factor of width $k+1$ that contains x, the removal of each instance of x in turn results in an attested factor one symbol narrower. This is a deviation from the original work in that Jardine and McMullin use only factors of widths in the range $k \pm 1$, but the formulation here avoids making a special case of shorter words. The symbols that are not both freely insertable and deletable are the salient ones.

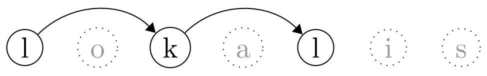

Figure 1: The tier-successor relation preserves linear order, but ignores certain symbols. Importantly, if a symbol is included then it is not also ignored.

In summary, let $\mathcal{F}_{k+1}$ represent all attested factors of width $k+1$ or less and $\mathcal{F}_k$ represent all those of width k or less, and define for each symbol $\sigma \in \Sigma$ two sets: $\sigma_\oplus$ containing all possible factors obtained by adding a single instance of σ to $\mathcal{F}_k$, and $\sigma_\ominus$ containing all possible factors obtained by deleting a single instance of σ from $\mathcal{F}_{k+1}$. Then the set of salient symbols is

$$\mathrm{T} = \{\sigma : \sigma_\oplus \not\subseteq \mathcal{F}_{k+1} \text{ or } \sigma_\ominus \not\subseteq \mathcal{F}_k\}.$$

Thus to decide salience we can use exactly the SL string extension learner described in section 2.4 and then post-process the resulting grammar by this algorithm.

In terms of time and space complexity, this portion of the algorithm is relatively efficient. There are $|\Sigma|^k$ possible factors of width k, and thus by summation there are $\frac{|\Sigma|^{k+2}-|\Sigma|}{|\Sigma|-1}$ possible factors of width up to $k+1$. Supposing we store one bit per possible factor that represents whether or not it is attested, we require $\mathcal{O}(|\Sigma|^{k+1})$ bits. For each word, its factors can be found in linear time, and each factor can be marked as attested in this set in time logarithmic in the set's size. That is, for an input of size n, the time complexity of gathering factors to determine salience is $\mathcal{O}(nk \log |\Sigma|)$.

4. The Substructures

Because Jardine and McMullin (2017) assume a batch-learning setting, they can simply learn the set of salient symbols, then erase all other symbols from the input and (in a second pass) analyze the result with any SL learner. Performing a second pass over the input requires this input to be retained. This, of course, results in unbounded space requirements and is therefore unsuitable for an online setting.

Fortunately, we can avoid this memorization of the input. A factor over relativized adjacency is made up of a sequence of symbols that appear in order, but not necessarily adjacently. These structures are, in general, referred to as **subsequences** (Heinz, 2010a; Rogers et al., 2010). Figure 1 shows one possible factor of width 3 ("lkl") in the string "lokalis". Crucially, when gathering subsequences, if a symbol is included in the subsequence, it can never later be excluded in that same subsequence. The factors "lki" and "lks" then would be invalid in this example and excluded from consideration because they both contain an "l" but go on to skip the second "l".

There are $\binom{n}{k}$ subsequences of width k in a word of length n, where this notation represents a binomial coefficient. It follows that the time complexity to find them is $\mathcal{O}(n^k/k!)$, and the same holds when including smaller factors as well. If for each factor we store the set of symbols that intervened, much like the paths of Jardine and Heinz (2016), then we need to account for the time it takes to find the set associated with the particular factor

Table 1: In gathering augmented subsequences for “cabacba”, many possibilities can be ignored. The intervener-sets are shown simply as sorted strings to avoid nested braces. Here, only the undecorated sets are maintained; those struck through were invalid from the start, while those in light gray are subsumed.

Factor	Intervener-Sets
aa	{b, ~~abc~~, bc}
ab	{∅, ~~abc~~, c}
ac	{~~ab~~, ∅}
ba	{∅, ~~abc~~}
bb	{ac}
bc	{a}
ca	{∅, ~~ab~~, ~~abc~~, b}
cb	{a, ~~abc~~, ∅}
cc	{ab}

and to mark the intervener-set as attested. These are additional multipliers of $k \log |\Sigma|$ and $|\Sigma|$, respectively. So in total the time complexity is $\mathcal{O}\big(n^k/(k-1)! \cdot |\Sigma| \log |\Sigma|\big)$.

Formally, if $w = \sigma_1 \dots \sigma_n$ ($\sigma_i \in \Sigma$) is a string and $X = \langle i_1, \dots, i_k \rangle$ ($k \leq n$) is an increasing sequence of indices, then the subsequence indicated by X is $Q = \langle \sigma_{i_1}, \dots, \sigma_{i_k} \rangle$. The intervener-set is $\mathcal{I} = \{\sigma_j : i_1 < j < i_k \text{ and } j \notin X\}$. The pair $\langle Q, \mathcal{I} \rangle$ is the **augmented subsequence** indicated by X. A **valid subsequence** is one where no symbol appears in both Q and $\mathcal{I}$. Henceforth, any mention of subsequences is restricted to the valid ones.

Unfortunately it appears that to store all possible augmented subsequences, we would need space significantly beyond exponential in the size of the alphabet and factor width, $\mathcal{O}(|\Sigma|^k \cdot 2^{|\Sigma|})$. But it turns out that we can exploit some structure in order to store significantly less. If a subsequence is in fact contiguous, that is it skips nothing, then no matter how adjacency is relativized that subsequence will still be an attested factor as long as it is valid. In fact a generalization holds: if a factor is attested with intervener-set $\mathcal{I}$, then it can also be assumed to be attestable for any superset of $\mathcal{I}$ for which it remains valid. So one needs only maintain the smallest observed intervener-sets (partially-ordered by subset). This means that the size of the set stored by any particular factor will never exceed $\mathcal{O}\big(\binom{|\Sigma|}{|\Sigma|/2}\big)$, which is still exponential in $|\Sigma|$, but many factors will store just a single set: ∅. Given these interactions, we conjecture that the space complexity will often be subexponential in the size of $|\Sigma|$. Table 1 shows the possible augmented subsequences for $k = 2$ in the string “cabacba” and indicates which subset of those actually need to be maintained. However, we show in section 5 that we can avoid this source of space complexity entirely.

Once the set of salient symbols T is known, we can derive a standard k-TSL$^{\mathrm{T}}$ grammar from this set of augmented subsequences. A subsequence is a permitted factor iff all of the symbols that comprise it are salient and it is attested for an intervener-set that is disjoint with T. Otherwise it is a forbidden factor.

5. Pointwise String Extension Learning

In sections 3 and 4, we discussed two different kinds of substructure that can be gathered when learning k-TSL: the substrings of length bounded above by $k+1$, which allow us to determine which symbols are salient, and the augmented subsequences of length bounded above by k, which allow us to select a set of permitted factors once salience has been determined. If we let the hypothesis space $\mathbb{G} = \mathcal{P}(\Sigma^{\leq k+1}) \times \mathcal{P}(\Sigma^{\leq k} \times \mathcal{P}(\Sigma))$, then we can define a learner

$$\varphi(\langle G_\ell, G_s\rangle, w) \triangleq \begin{cases} \langle G_\ell, G_s\rangle & \text{if } w = \circledcirc \\ \bigl\langle G_\ell \cup f(w), r\bigl(G_s \cup x(w)\bigr)\bigr\rangle & \text{otherwise,} \end{cases}$$

where $f\colon \Sigma^* \to \mathcal{P}(\Sigma^{\leq k+1})$ gathers all and only those substrings of w whose width is bounded above by $k+1$, $x\colon \Sigma^* \to \mathcal{P}(\Sigma^{\leq k} \times \mathcal{P}(\Sigma))$ extracts the valid augmented subsequences of w of width bounded above by k, and $r\colon \mathcal{P}(\Sigma^{\leq k} \times \mathcal{P}(\Sigma)) \to \mathcal{P}(\Sigma^{\leq k} \times \mathcal{P}(\Sigma))$ restricts the set of augmented subsequences to exclude any that are entailed by any other. This is effectively two distinct string extension learners run in parallel, pointwise on the composite grammar.

The composite grammar can immediately be used as an acceptor without further processing. We replace the cost of deciding salience by that of finding augmented subsequences.

$$\mathcal{L}(\langle G_\ell, G_s\rangle) \triangleq \bigl\{w : f(w) \subseteq G_\ell \text{ and } r\bigl(G_s \cup x(w)\bigr) \subseteq G_s\bigr\}.$$

In words, a string is accepted iff it has only permitted substrings and each of its valid augmented subsequences is attested or entailed by something that is attested.

Depending on the parameters and the size of the input words, this strategy might be a good one. In other situations, it might be better to actually determine which symbols are salient. Recall that a text contains every valid word at least once, and that non-salient symbols are freely deletable in all strings. Free deletability of non-salient symbols implies that any subsequence that includes only salient symbols will, in some word of the text, have only its salient interveners as interveners. Those subsequences that do not violate constraints on the tier then must appear with an empty intervener set. In other words, such subsequences will appear as factors in terms of adjacency and will be accounted for by that component of the composite grammar.

Thus upon convergence the left component of this composite grammar, G_ℓ, is sufficient on its own to decide salience and to extract the grammar itself. G_s is unnecessary. Let $s\colon \mathbb{G} \to \mathcal{P}(\Sigma)$ be the function that decides salience in the manner described in section 3, and let $\pi_\mathrm{T}(w)$ represent the projection to T of w as described by erasing symbols that are not in T. Then we might have the following as an equivalent alternative definition

$$\mathcal{L}(G) \triangleq \bigl\{w : f\bigl(\pi_{s(G)}(w)\bigr) \subseteq G\bigr\}.$$

As an aside, the deletion-closure of the strictly piecewise class of stringsets (Rogers et al., 2010, see also Haines, 1969) enables this same sort of learning of long-distance patterns using only adjacent substrings.

Revisiting the time and space complexities mentioned previously, this optimized version of the grammar can be learned in $\mathcal{O}(nk \log|\Sigma|)$ time and $\mathcal{O}(|\Sigma|^k)$ space, exactly those values that were assumed for only the salience-decision component of learning. The time complexity is deferred to later, in interpreting the grammar as k-TSL rather than as $(k+1)$-SL.

Table 2: Some words of varying degrees of phonological plausibility. Each set is in the order produced by a sliding 3-window. Factors already accounted for are in light gray.

akkalkak	{akk, kka, kal, alk, lka, kak}
klark	{kla, lar, ark}
kralk	{kra, ral, alk}
karlakalra	{kar, arl, rla, lak, aka, kal, alr, lra}
akrala	{akr, kra, ral, ala}
aklara	{akl, kla, lar, ara}
rakklarkka	{rak, akk, kkl, kla, ark, rkk, kka}
arkralkla	{ark, rkr, kra, ral, alk, lkl, kla}
laarlraalr	{laa, aar, arl, rlr, lra, raa, aal, alr}
kaaakkrka	{kaa, aaa, aak, akk, kkr, krk, rka}
klkkklrk	{klk, lkk, kkk, kkl, klr, lrk}
krlkrkl	{krl, rlk, lkr, krk, rkl}
alrla	{alr, lrl, rla}

6. A Worked Example of the Final Simplified Approach

Consider a blocked-assimilation constraint such as the liquid dissimilation of Latin (Cser, 2010), where identical liquids may not occur in sequence unless a non-coronal intervenes. This is equivalent to the example language described in section 2.2. We will assume for now that no other constraint interacts with this. Assuming an alphabet consisting of two distinct liquids ("l" and "r"), a non-coronal consonant ("k"), and a vowel ("a"), we discuss a worked example that learns this constraint.

As this can be described by the 2-TSL$^{\{l,k,r\}}$ constraint whose forbidden factors are $\{ll, rr\}$, the text must contain all substrings of width 3 or less whose projection to $\{l, k, r\}$ do not contain these prohibited bigrams. The words shown in Table 2 would constitute a representative sample for this constraint, though one may notice that some of the 3-factors that need to appear are phonologically implausible.

If we assume that domain boundaries are explicit, then we would also need to encounter any permitted factors that include these boundary symbols. For the sake of brevity such an account has been omitted, but one could easily construct similarly implausible words to account for this change.

7. Non-Strict Locality

These methods can be extended beyond just TSL. Lambert (2021) demonstrates that the locally testable (McNaughton and Papert, 1971) and locally threshold testable (Beauquier and Pin, 1989) stringsets admit T-relativized variants with the same properties as TSL^{T}:

- A string appears iff its projection to T appears, and
- All symbols that are not in T are freely insertable and deletable.

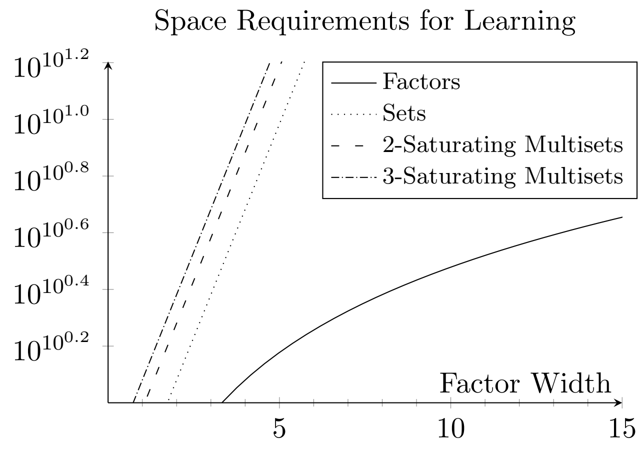

Figure 2: While gathering factors requires space exponential in terms of factor width, the requirements are doubly exponential for any of the larger structures we might employ. Here the space requirements are shown for just a binary alphabet.

Locally threshold testable stringsets are characterized by not just the factors in each word but the saturating multisets of factors in the words. A **saturating multiset** is a variant of the multiset in which the counts associated with elements are capped to some maximum value, t. Multisets that saturate at a count of $t = 1$ are simply sets, and these are the characterization of locally testable.

Due to the two properties of relativization, if we collect the saturating multisets of substrings of width bounded above by k along with the individual factors of width $k+1$, we can still use the algorithm described in section 3 to determine salience and again treat the non-relativized grammar in a relativized way. The issue is not finding a learning algorithm; instead it is the space complexity. Figure 2 shows the amount of space required for factors, $\mathcal{O}(|\Sigma|^k)$, compared to that of saturating multisets, $\mathcal{O}\big((t+1)^{|\Sigma|^k}\big)$. Of course, Σ and k are fixed parameters for any given run of the algorithm, and thus constant, but this complexity should not be ignored.

8. Conclusions

We proposed an online learning algorithm for the tier-based strictly k-local class of stringsets that operates in linear time, $\mathcal{O}(nk \log |\Sigma|)$, and constant space, $\mathcal{O}(|\Sigma|^{k+1})$, in terms of the size of the input. This space complexity is exponential in the factor width. We demonstrated that the grammar representation given by a strictly k-local learner can also be interpreted as a strictly k-piecewise or tier-based strictly $(k-1)$-local grammar. The difference comes later, in the interpretation of the grammar. The algorithms presented here can be incorporated into any sufficiently general implementation of string extension learners (Heinz, 2010b).

Efficient learning of interacting constraints remains an open question. Generally the set of symbols salient to a pattern as a whole will be some superset of those sets for its constraints. If a stringset L consists of a TSL component with an additional constraint imposed that restricts the set of substrings that may occur, then L will not in general be learnable by the known TSL-learning algorithms including those presented here. If multiple TSL constraints over different tier alphabets interact, the learned stringset will consider the set of salient symbols to be the union of all such alphabets, but other sorts of interactions have yet to be explored. Notably, SL is equivalent to TSL^{Σ} by definition, and any SP constraint imposes a tier of salience including the symbols that it mentions, so many cases of interaction will result in a TSL^{Σ} (that is, SL) approximation of the target stringset.

This lack of robustness in the face of constraint interaction poses a challenge for the learnability of TSL constraints within a more complex structure in a natural setting. If the solution to this problem is learning a new grammar for each possible tier alphabet, rather than trying to determine which such alphabet to consider, then we must bear in mind the additional space requirements, exponential in the size of the alphabet. Such an approach yields the multiple-tier-based strictly local languages of De Santo and Graf (2019).

References

Danièle Beauquier and Jean-Éric Pin. Factors of words. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simonetta Ronchi Della Rocca, editors, *Automata, Languages and Programming: 16th International Colloquium*, volume 372 of *Lecture Notes in Computer Science*, pages 63–79. Springer Berlin / Heidelberg, 1989. doi: 10.1007/BFb0035752.

András Cser. The -alis/-aris allomorphy revisited. In Franz Rainer, Wolfgang Dressler, Dieter Kastovsky, and Hans Christian Luschützky, editors, *Variation and Change in Morphology: Selected Papers from the 13th International Morphology Meeting*, pages 33–52. John Benjamins Publishing Company, Vienna, Austria, 2010. doi: 10.1075/cilt.310.02cse.

Aldo De Luca and Antonio Restivo. A characterization of strictly locally testable languages and its application to subsemigroups of a free semigroup. *Information and Control*, 44(3):300–319, March 1980. doi: 10.1016/S0019-9958(80)90180-1.

Aniello De Santo and Thomas Graf. Structure sensitive tier projection: Applications and formal properties. In Raffaella Bernardi, Greg Kobele, and Sylvain Pogodalla, editors, *Formal Grammar 2019*, volume 11668 of *Lecture Notes in Computer Science*, pages 35–50. Springer Verlag, 2019. doi: 10.1007/978-3-662-59648-7_3.

Pedro Garcia, Enrique Vidal, and José Oncina. Learning locally testable languages in the strict sense. In *Proceedings of the 1st International Workshop on Algorithmic Learning Theory*, pages 325–338, Tokyo, Japan, 1990. URL https://grfia.dlsi.ua.es/repositori/grfia/pubs/111/alt1990.pdf.

Edward Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, May 1967. doi: 10.1016/S0019-9958(67)91165-5.

Leonard H. Haines. On free monoids partially ordered by embedding. *Journal of Combinatorial Theory*, 6(1):94–98, 1969. doi: 10.1016/s0021-9800(69)80111-0.

Jeffrey Heinz. Learning long-distance phonotactics. *Linguistic Inquiry*, 41(4):623–661, October 2010a. doi: 10.1162/ling_a_00015.

Jeffrey Heinz. String extension learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 897–906, Uppsala, Sweden, July 2010b. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P10-1092.

Jeffrey Heinz, Chetan Rawal, and Herbert G. Tanner. Tier-based strictly local constraints for phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Short Papers*, volume 2, pages 58–64, Portland, Oregon, 2011. Association for Computational Linguistics. URL https://aclweb.org/anthology/P11-2011.

Sanjay Jain, Steffen Lange, and Sandra Zilles. Some natural conditions on incremental learning. *Information and Computation*, 205(11):1671–1684, November 2007. doi: 10.1016/j.ic.2007.06.002.

Adam Jardine and Jeffrey Heinz. Learning tier-based strictly 2-local languages. *Transactions of the Association for Computation in Linguistics*, 4:87–98, 2016. doi: 10.1162/tacl_a_00085.

Adam Jardine and Kevin McMullin. Efficient learning of tier-based strictly k-local languages. In Frank Drewes, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications: 11th International Conference*, volume 10168 of *Lecture Notes in Computer Science*, pages 64–76. Springer, Cham, 2017. doi: 10.1007/978-3-319-53733-7_4.

Dakotah Lambert. Relativized adjacency, 2021. In review.

Dakotah Lambert and James Rogers. Tier-based strictly local stringsets: Perspectives from model and automata theory. In *Proceedings of the Society for Computation in Linguistics*, volume 3, pages 330–337, New Orleans, Louisiana, 2020. doi: 10.7275/2n1j-pj39.

Robert McNaughton and Seymour Aubrey Papert. *Counter-Free Automata*. MIT Press, 1971.

Daniel Nathan Osherson, Michael Stob, and Scott Weinstein. *Systems That Learn*. MIT Press, 1986.

James Rogers and Geoffrey K. Pullum. Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information*, 20(3):329–342, June 2011. doi: 10.1007/s10849-011-9140-2.

James Rogers, Jeffrey Heinz, Gil Bailey, Matt Edlefsen, Molly Visscher, David Wellcome, and Sean Wibel. On languages piecewise testable in the strict sense. In Christian Ebert, Gerhard Jäger, and Jens Michaelis, editors, *The Mathematics of Language: Revised Selected Papers from the 10th and 11th Biennial Conference on the Mathematics of Language*, volume 6149 of *LNCS/LNAI*, pages 255–265. FoLLI/Springer, 2010. doi: 10.1007/978-3-642-14322-9_19.