Learning Tiers for Long-Distance Phonotactics

Adam Jardine

1. Introduction

Long-distance phonotactics, in which the surface sounds of a language are subject to coocurrence constraints referring to nonadjacent segments, present a difficult learning problem. To acquire such patterns, a learner must find dependencies among distant segments. This paper approaches an idealized version of this problem: how can these patterns even be learned at all? By introducing a learning algorithm for a *class* of patterns that fits the typology of long-distance phonotactics, this paper shows that it is possible to induce a tier over which long-distance generalizations can be made. *A priori* knowledge of a specific tier is shown to be unnecessary; the phonological notions of a *tier* and *locality* are enough to discover a tier and long-distance dependencies. The mechanisms of the algorithm thus represent a theoretical step towards a model of how children might acquire phonotactics from raw language data.

Long-distance dependencies have challenged previously proposed learning algorithms. For example, the learning algorithm presented in Hayes & Wilson (2008) cannot find dependencies among vowels in Shona [±ATR] harmony until it is told *a priori* to ignore the consonants. However, they admit that this fix is not sufficient for patterns requiring complex tiers, such as consonantal harmony, disharmony, or vowel harmony with neutral vowels. Given the variety of long-distance phonotactic dependencies there are in natural language phonology, it is entirely possible that humans are endowed with a mechanism for filtering through 'ignorable' material in order to find phonotactic generalizations. The current paper introduces a provably correct algorithm for such a mechanism, based in formal language theory and grammatical inference (de la Higuera, 2010).

The algorithm is designed for a particular class of formal languages known as the Tier-based Strictly 2-Local (TSL_2) languages (Heinz et al., 2011), which are governed by grammars that check the adjacency of symbols on a 'tier' ignoring all symbols not on that tier. This algorithm, the TSL_2 Learning Algorithm, can induce a tier from positive data with no *a priori* knowledge of what that tier is. In brief, it does this by starting with the tier equal to the full inventory of phonemes, and then removing symbols irrelevant to the pattern one-by-one, each time making new generalizations based on what it has previously removed from the tier. This aspect of the algorithm makes it an interesting model of learning.

The formal details of this algorithm, including the proof of its correctness, can be found in Jardine & Heinz (in prep.). The purpose of this paper is to informally discuss the core insight of the algorithm, present results applying it to natural language data, and discuss its relation to the general problem of phonological acquisition. Two simple case studies of Latin liquid dissimilation and of Finnish vowel harmony, in which the learner successfully induces the correct generalization (given particular conditions), are presented.

The paper is structured as follows. §2 gives the relevant background on long-distance phonotactics, issues in learning long-distance phonotactics, and the TSL_2 formal languages. §3 presents the algorithm, and §4 illustrates the algorithm in use with the two case studies. Issues with the learner and how it connects with acquisition are presented in §5, and §6 concludes.

* Preprint of paper to appear in *Proceedings of the 6th Generative Approaches to Language Association North America (GALANA 2015)*. ©Adam Jardine (ajardine@udel.edu).

This research is indebted to the ideas and advice of Jeffrey Heinz and Rémi Eyraud, as well as discussion with the members of Irene Vogel's acquisition of phonology seminar at UD (Nicole Demers, Hyun Jin Hwangbo, Kalyn Matocha, and Taylor Lampton Miller), Kevin McMullin, and an inquisitive audience at GALANA 6. Thanks also to Jason Riggle for providing the Finnish corpus. All errors are my own.

2. Background

This section introduces the kinds of patterns that are, at least theoretically, in the range of the learning algorithm proposed in §3. Examples of long-distance natural language phonotactic patterns are discussed in §2.1, and why these patterns are problematic for existing phonotactic learning algorithms is briefly discussed in §2.2. The *Tier-based Strictly 2-Local* class of formal languages, which can model the phonotactic patterns in §2.1 and which the present learner is designed to learn, are introduced in §2.3.

2.1. Long-distance phonotactic patterns

'Long-distance phonotactic' (LDP) refers to any generalization which holds on surface forms in a language for which the description somehow involves ignoring intervening material. Examples include vowel harmony (Nevins, 2010), consonantal harmony (Rose & Walker, 2004; Hansson, 2001, 2010), and long-distance consonant dissimilation (Suzuki, 1998; Bennett, 2013). Tone also has many long-distance patterns (Yip, 2002; Hyman, 2011), however this paper will focus on segmental phenomena.

To give a concrete example, a (not exceptionless) generalization regarding surface forms in Finnish is that all vowels in a word—except for the 'transparent' [i] and [e]—must agree in the feature [±back]. In (1) below, words only choose from the set of front vowels {ö,ü,ä} or the set of back vowels {o,u,a}; they never mix. Vowels participating in this harmony are underlined in the example.

(1) Finnish (Nevins, 2010; Odden, 1994)

a. pöütä-nä 'table-ESS'	c. ulko-ta 'outside-ABL'
b. väkkärä-nä 'pinwheel-ESS'	d. pappi-na 'priest-ESS'

Note that this generalization 'ignores' both intervening consonants and the transparent vowels {i,e}. Thus in (1d) [pappi-na] 'priest-ESS', the two [a] vowels agree in backness even though they are separated by the 4-segment sequence [ppin]. This gives the generalization its long-distance character.

An interesting example of a long-distance process involving consonants can be found in Latin. Latin is commonly described as having a liquid dissimilation process in which an [l] cannot follow another [l] in a word, ignoring intervening consonants and vowels, unless an [r] intervenes (Jensen, 1974; Odden, 1994; Heinz, 2010a). This can be seen in allomorphs of the /-alis/ adjectival suffix, which surfaces as [-aris] if its root contains an /l/, except in cases where there is an intervening /r/, in which it surfaces as [alis]. Participating segments are again underlined for emphasis (data from Heinz (2010a:(15–6))).

(2)	a. naval<u>i</u>s	'naval'	(3)	d. sol<u>a</u>ris	'solar'	(4)	g. fl<u>o</u>ral<u>i</u>s	'floral'
	b. episcopal<u>i</u>s	'infinitalis'		e. l<u>u</u>nar<u>i</u>s	'lunar'		h. sepul<u>k</u>ral<u>i</u>s	'funeareal'
	c. infinital<u>i</u>s	'negative'		f. mil<u>i</u>tar<u>i</u>s	'military'		i. l<u>i</u>toral<u>i</u>s	'of the shore'

As seen in (3), an underlying /l/ dissimilates to [r] following another /l/. However, an /r/ between two /l/s 'blocks' the dissimilation from occurring, as in (4). Such blocking segments are important because, in order to main that distance generalizations are local on some tier, these blockers must also appear on the tier. Cser (2010) shows that intervening non-coronal consonants also consistently block dissimilation, e.g. *legalis* 'legal' (**legaris*). This complication will be returned to in §5; for the meantime, the examples here will follow the 'simple' Latin pattern noted by the sources above.

2.2. Learning

The goal of any unsupervised human or machine learner of phonotactic patterns is to notice dependencies between segments in the input data. For local phonotactics, this can be achieved by paying attention only to 'windows' of a fixed length when searching for dependencies. This is exactly the idea behind the idea of n-gram models in natural language processing (Jurafsky & Martin, 2009), which keep track of statistical information about contiguous pieces, or *substrings*, of size n of the strings found in a given corpus. An equivalent concept in formal language theory (to be described in more detail below) is the Strictly Local languages (McNaughton & Papert, 1971), which can be learned by remembering all substrings of a certain size (García et al., 1990; Heinz, 2010b). For example, a learning algorithm which

a priori knows to ignore consonants and /i,e/ can learn the Finnish pattern above as a Strictly 2-Local pattern by remembering all of the substrings of length 2 of the strings of harmonizing vowels. The chart in Figure 1 below shows the substrings such a learner would remember given the data in (1) sequentially:

Step	1 (1b)	2 (1b)	3 (1c)	4 (1d)
Datum	pöütä-nä	väkkärä-nä	ulko-ta	pappi-na
	↓↓↓ ↓	↓ ↓↓ ↓	↓ ↓ ↓	↓ ↓
Tier	öü ä ä	ä ä ä ä	u o a	a a
Substrings	{öü, üä, ää}	{ää}	{uo, oa}	{aa}
Learned substrings	{öü, üä, ää}	{öü, üä, ää}	{öü, üä, ää, uo, oa}	{öü, üä, ää, uo, oa, aa,}

Figure 1: Strictly 2-Local learning given a tier *a priori*

As seen in the final row labeled 'Learned substrings' above, this learner simply remembers the union of all substrings of length 2 it has seen on the vowel tier (shown projected from each data point in the row labeled 'Tier'). Such a learner would learn the harmony pattern if it is given data consistent with the Finnish vowel harmony generalization; for example, it will never see disharmonic substrings such as *oö*.

This idea of 'local learing', in its broad sense, also applies to learners like the MaxEnt phonotactic learner of Hayes & Wilson (2008). This algorithm learns constraints built out of strings of feature matrices of some bounded length. Keeping this bound low reduces number of possible constraints the learner needs to consider, and evaluating constraints on input forms is more efficient. However, their work shows how, without prior knowledge of a tier, local learning runs into trouble when faced with data exhibiting long-distance patterns. They discuss how the MaxEnt learner does not learn a Shona vowel harmony pattern, even with constraints of length 5. Their solution is as described above for Finnish: give the learner *a priori* knowledge of a vowel tier, and then learn local constraints on that tier.

However, it is unclear how to impart enough prior knowledge to learn the full range of long-distance phonotactics. Languages have been analyzed using a number of tiers: the entire vowel inventory vowels (Turkish; Clements & Sezer, 1982); vowels except /i,e/ (Finnish; Ringen, 1975); liquids (Sundanese; Cohn, 1992); liquids and non-coronals (Latin; Cser, 2010); sibilants (Samala; Rose & Walker, 2004); and sibilants and round vowels, /r/, and some voiced obstruents (Koorete; McMullin & Hansson, 2014). This range could be covered by a learner which keeps track of local generalizations over all possible tiers simultaneously. This is computationally implausible, as it amounts to learning over each subset of the inventory, which for an inventory of n phonemes yields 2^n tiers. This explodes very quickly; for a language with 13 phonemes, a learner would need to simultaneously learn over $8,192$ tiers.

A reasonable research question, then, is how to *discover* a tier. Goldsmith & Riggle (2012) demonstrate how to induce a tier for Finnish vowel harmony through information-theoretic notion of *mutual information*. Mutual information over adjacent segments measures the probability one kind of segment will follow another. Their mutual information model induces consonant and vowel categories from a Finnish corpus, as consonants tend to be followed by vowels in Finnish. A second round of such analysis on the vowels then finds a harmony pattern among the vowels. However, they do not provide an algorithm with a well-known *range* of generalizations it can learn; it is unclear if a similar paradigm could, for example, learn Latin liquid dissimilation. Also, their methodology does not make clear *what kind of data* their learner requires to successfully learn a long-distance pattern.

An approach which can *guarantee* the range of learnable patterns and the success of an algorithm given a well-defined set of data is the one based on formal language theory and grammatical inference, as mentioned above. One relevant example is the precedence learner of Heinz (2010a), which learns long-distance phonotactics by keeping track of *precedence relations* between segments in the input forms. Thus, the learner notices when a segment precedes (or follows) another segment. Heinz's learner is guaranteed to learn a *class* of patterns, the Strictly 2-Piecewise (SP_2) formal languages (Rogers et al., 2010). This means that once the learner has seen all of the relevant precedence relations for a target SP_2 language, it can be guaranteed to learn it. This is significant because the phonotactic patterns derived from consonant harmony patterns discussed in Hansson (2001) and Rose & Walker (2004) are shown by Heinz (2010a) to be SP_2. Patterns with blocking, such as in dissimilation, are not Strictly k-Piecewise

for any k. For example, in Latin (4a) *floralis* 'flower' *l* precedes *l*, and so Heinz's learner would learn a grammar that allows unattested Latin forms like **militalis* (the attested form is (3f) *militaris* 'military').

The algorithm to be introduced in the following section takes the same approach as Heinz (2010a), but solves the problem of blocking segments. It does this by discovering subsets of the alphabet over which local dependencies are defined. In this sense, while the details of the algorithm are quite different from that of Goldsmith and Riggle, the core idea is the same. The particular class of patterns to be learned is the class of formal languages called the *Tier-based Strictly* 2*-Local languages*.

2.3. Tier-based Strictly Local languages

The learning algorithm proposed in this paper is designed to learn a particular class of *formal language*. The term 'formal language' refers to a (possibly infinite) set of strings built on some *alphabet*, a finite set of symbols. This alphabet is usually referred to as Σ, and the set of all possible strings over that alphabet Σ^*. While formal languages are mathematical objects, they can be used to model natural language phonotactics. The formal languages that are interesting to us are those which are associated with some *grammar* which, for any possible string, will tell you whether or not that string is in the language. Formal languages can be categorized into classes by the type of grammar they have.

The class of formal languages of central interest to this paper is the Tier-based Strictly Local languages, first studied by Heinz et al. (2011) specifically to model the phonological concept of locality on a tier. Like the Strictly Local and Strictly Piecewise languages discussed above, the Tier-based Strictly Local languages are usually qualified with a k-value which indicates that the grammar checks pieces of a string of length k. Heinz et al. (2011) note that it is likely sufficient for natural language for $k = 2$, so this paper (and Jardine & Heinz, in prep.) focuses on the Tier-based Strictly 2-Local, or TSL_2, languages. The grammar of a TSL_2 language comprises two parts: a subset of the alphabet called the *tier*, and a set of allowable *substrings* of length 2 which specify which members of the tier are allowed to be 'adjacent' to one another. This adjacency is relative to the tier, i.e., it ignores the existence of any segments not on the tier. Formally, we say a grammar G is a two-tuple, $G = (T, substrings)$ where T is the tier and $substrings$ is the set of substrings.

In order to check the substrings on a tier, we need a function $proj(w)$ which 'projects' the tier of a string w by ignoring all of the non-tier symbols. To give an example, let us assume a simple alphabet $\{i, u, æ, a, t, d, s, z\}$. Let $T = \{i, u, æ, a\}$ be a tier of vowels. The consonants $\{t, d, s, z\}$ are completely irrelevant to G, and would be erased by $proj(w)$. If we have a string $tistæz$ in this alphabet, $proj(tistæz) = iæ$, or the string of vowels in the word.

Now say the allowable substrings are $substrings = \{ii, iæ, æi, ææ, uu, ua, au, aa\}$. The grammar $G = (T, substrings)$ thus allows only vowels of the same backness to be adjacent on the tier. Thus the language accepted by G, which we write $L(G)$, includes the string $tistæz$ because, ignoring the consonants, its vowels i and æ are adjacent, and $iæ$ are an allowed substring. In contrast, $^*tustæz$ is not in $L(G)$, because $proj(tustæz) = uæ$, and $uæ$ is not an allowed substring. Some more examples are given in Table 1.

$$G = (\{i, u, æ, a\}, \{ii, iæ, æi, ææ, uu, ua, au, aa\})$$

In L(G)			*Not in L(G)*		
$sazutu$	$\xrightarrow{proj}$	auu	$sizuti$	$\xrightarrow{proj}$	iui
$tisitæzi$	$\xrightarrow{proj}$	$iiæi$	$tusitazi$	$\xrightarrow{proj}$	$uiai$
$ætttttti$	$\xrightarrow{proj}$	$æi$	$ætttttu$	$\xrightarrow{proj}$	$æu$
...			...		

Table 1: Example of a TSL_2 language modelling vowel harmony

Importantly, note that the grammar G says nothing about the segments *not* on the tier. They are thus free to occur in any position. This thus results in forms like $ætttttti$, which does not seem like a plausible word in any language, is part of $L(G)$. This is because $L(G)$, like a single phonological rewrite rule (a la Chomsky & Halle, 1968) or partial ranking of Optimality Theory (Prince & Smolensky, 1993) constraints, $L(G)$ models a single phonological generalization. It can be combined with other

generalizations (like one that excludes a sequence *ttttt*) to create a full grammar through intersection of formal languages (Heinz, 2010a).

Given this, $L(G)$ from Table 1 models a vowel harmony generalization like the one for Finnish in §2.1. This is because a TSL_2 grammar captures exactly the idea of 'ignoring' certain kinds of intervening material (i.e., whatever is not on the tier). As another example, the reader is invited to check that the grammar in Table 2 models a pattern like Latin liquid dissimilation over the alphabet $\{a, t, l, r\}$:

$$G = (\{l, r\}, \{lr, rl\})$$

In L(G)			*Not in L(G)*		
$tatat$	$\xrightarrow{proj}$	$(empty)$	$alla$	$\xrightarrow{proj}$	ll
$talat$	$\xrightarrow{proj}$	l	$ralal$	$\xrightarrow{proj}$	rll
$ratal$	$\xrightarrow{proj}$	rl	rar	$\xrightarrow{proj}$	rr
$larala$	$\xrightarrow{proj}$	lrl	$latatatatal$	$\xrightarrow{proj}$	ll
$\ldots$			$\ldots$		

Table 2: Example of a TSL_2 language modelling Latin dissimilation

While a full typological study is beyond the scope of this paper, most common long-distance phonotactic patterns fall into the TSL_2 class (Heinz et al., 2011; McMullin & Hansson, 2014). Not only do they provide a good approximation of the typology, but there is also psycholinguistic evidence that TSL_2 grammars accurately model how human beings reason about long-distance generalizations (McMullin & Hansson, 2015). As such, a learning algorithm for TSL_2 formal languages can provide insight into how such patterns can be learned. Furthermore, as TSL_2 grammars represent the phonological concepts of a tier and locality, this theoretical result shows that these concepts alone are sufficient for a learner to induce a long-distance pattern from positive data. The next section illustrates how this can be done.

3. The TSL_2 Learning Algorithm

This section introduces a new algorithm, the TSL_2 Learning Algorithm (henceforth TLA), which can provably learn any TSL_2 grammar. For expositional clarity a simplified version of the algorithm is presented first. This simplified version is still able to learn most TSL_2 patterns relevant for phonology. The patterns it cannot learn, those with 'free' blocking segments on the tier which do not participate in any tier-local cooccurrence restrictions, are discussed later in §5. What follows is an informal description of the algorithm; for the formal details, including proofs of completeness and time complexity, see Jardine & Heinz (in prep.). This section is broken into subsections which first introduce the concept of a *path*, then describe the algorithm in detail.

3.1. Paths

The crux of the algorithm is the idea of a *path*. A path is like the precedence relations of Heinz (2010a), however a path keeps track of both precedence between two symbols and the set of symbols seen between them. For example, take the string $tustæz$ from the examples in §2.3. The path $\langle t, \{u, s, t, æ\}, z\rangle$ is in the paths of $tustæz$, because t precedes z in the word and $\{u, s, t, æ\}$ is the set of symbols that come between them. Incidentally, $\langle t, \{æ\}, z\rangle$ is also in the paths of $tustæz$, because the second t also precedes z, but only æ lies between them. Table 3 gives the full list of paths for $tustæz$.

String	Paths				
$tustæz$	$\langle t, \{\}, u\rangle$,	$\langle t, \{u\}, s\rangle$,	$\langle t, \{u, s\}, t\rangle$,	$\langle t, \{u, s, t\}, æ\rangle$,	$\langle t, \{u, s, t, æ\}, z\rangle$,
	$\langle t, \{æ\}, z\rangle$	$\langle u, \{\}, s\rangle$,	$\langle u, \{s\}, t\rangle$,	$\langle u, \{s, t\}, æ\rangle$,	$\langle u, \{s, t, æ\}, z\rangle$,
	$\langle s, \{\}, t\rangle$,	$\langle s, \{t\}, æ\rangle$,	$\langle s, \{t, æ\}, z\rangle$,	$\langle æ, \{\}, z\rangle$	

Table 3: Paths for $tustæz$

The paths of a string can tell us what is or isn't *tier-adjacent* depending on the tier. For example,

take the path $\langle u, \{s, t\}, æ\rangle$ from the fourth column in Table 3. If the tier for our grammar is $\{i, u, æ, a\}$, as in the first example in §2.3, then this path tells us that u and æ are tier-adjacent, because the set $\{s, t\}$ of symbols in between them is composed entirely of symbols not on the tier (in set-theoretic notation, $\{s, t\} \subseteq \Sigma - T$). In this way, keeping track of paths in the input allows the learner to reason about what tier-local dependencies there are in a word (and thus set of words) given different hypotheses for the tier.

3.2. Algorithm

With the concept of a path established, the TLA itself is quite simple. The main goal is to begin by guessing that the tier for the grammar is equal to the full alphabet, and then to recursively remove a symbol from that tier whenever there are no grammatical dependencies for that symbol given the current guess for the tier. By doing this recursively, the algorithm can make new generalizations based on what it has learned about the tier. This subsection will outline how the algorithm works, and the following section, §4, walks through concrete examples of running the algorithm on natural language data.

The TLA is given in Algorithm 1. It takes as its input a full alphabet Σ and a finite set D of input strings augmented with a word boundary $\#$ at the beginning and end, e.g. $\#tustæz\#$. It then sets its initial hypothesis for T equal to Σ, and generates a set P of paths which is the union of each set of paths from every string in D. Based on these paths, and given that D contains the right kind of data, it will return a TSL_2 grammar $(T, substrings)$ that generated the input data D.

Data: An alphabet Σ; finite set D of strings in Σ^*
Result: A TSL_2 grammar $(T, substrings)$
Initialize $P = paths(D)$;
Initialize $T = \Sigma$;
Initialize $substrings = \emptyset$;

function $get\_tier(T, P)$
  Set P_T to the set of all paths $p = \langle \sigma_1, X, \sigma_2\rangle$ where σ_1, σ_2 are either on T or are $\#$;
  for *each* σ *in* T **do**
    if *for each* σ' *in* T *and* $\#$, *there exists a path* $\langle \sigma, X, \sigma'\rangle$ *in* P_T *and a path* $\langle \sigma', X', \sigma\rangle$ *in* P_T, *where* X *and* X' *are subsets of* $\Sigma - T$ *(i.e., they only contain symbols not on* T*)* **then**
      Remove σ from T;
      Return $get\_tier(T, P_T)$;
  Return T;

Set $T, P_T = get\_tier(T, P)$;
for *each* $p = \langle \sigma_1, X, \sigma_2\rangle$ *in* P_T **do**
  if σ_1 *and* σ_2 *are in* T *and* X *is a subset of* $\Sigma - T$ **then**
    Add $\sigma_1\sigma_2$ to $substrings$;
Return $(T, substrings)$;

Algorithm 1: The TSL_2 Learning Algorithm (TLA)

Basically, the algorithm implements the following step-by-step process:

(5)
a. Calculate all paths for all words in D.
b. Start with $T = \Sigma$ as the initial guess for tier.
c. Look at set of paths $\langle \sigma_1, X, \sigma_2\rangle$ where σ_1, σ_2 are on the tier.
d. Cycle through the word boundary $\#$ and each member of T. Is there any member σ of T which is tier-adjacent to every other member (by looking at paths in step (c) whose set of intervening elements is not on the tier)?
e. If so, remove that member from T, and repeat from step (c) with the new T
f. If not, return T and $substrings$.

The core of the algorithm—steps (c) through (e) from (5)—is in the internal function get_tier, which recursively edits the tier hypothesis T in the following manner. It starts cycling through each

symbol σ in T, checking to see if there are any grammatical restrictions on σ, given the guess T. The 'grammatical restrictions' are checked by the if statement in the definition for get_tier:

(6) **If** *for each σ' in T and $\#$, there exists a path $\langle \sigma, X, \sigma' \rangle$ in P_T and a path $\langle \sigma', X', \sigma \rangle$ in P_T, where X and X' are subsets of $\Sigma - T$ (i.e., they only contain symbols not on T)*

This simply states that for all σ' in the set T, or for the word boundary $\#$ (which can be used to keep track of generalizations such as "the tier cannot start with σ"), is σ' tier-adjacent with σ? This is accomplished by checking if both $\sigma'\sigma$ and $\sigma\sigma'$ exist in the sample separated only by symbols which are not in T—which is exactly what the paths in (6) tell us. If this is true, than this means that σ is free to be adjacent, in either order, with any other symbol on T. This means that σ is no longer restricted by the grammar (recall that TSL_2 grammars say nothing about elements not on the tier), and thus can be safely be taken out of T without changing the set of strings the grammar accepts. If the if statement in (6) is true, then σ is removed, and get_tier is called again recursively with the new tier passed as an argument. This corresponds to step (e) in (5).

Importantly, because T has changed, the set of paths get_tier checks when called again also changes. The condition for removing a symbol from the tier in (6) only considers paths whose sets of intervening symbols (X and X') consist entirely of symbols not on the tier. This relaxes the definition of tier-adjacency, as now $\sigma'\sigma$ and $\sigma\sigma'$ may be separated by the symbol just removed from T and still be considered tier-adjacent. This opens up the learner to make new generalizations based on its new hypothesis T. Concrete examples of this will be explained in the next section.

The internal function get_tier thus continually edits T until it finds that no symbol in T meets condition (6). When this final version of T has been found (i.e., there are no more symbols which can be removed from the tier), get_tier returns T and P_T, the set of paths whose σ_1 and σ_2 are on the tier. The last for loop of Algorithm 1 thus calculates the set $substrings$ of allowed tier substrings by searching by searching through P_T and finding all σ_1, σ_2 on the tier that are tier-adjacent. Finally, the algorithm returns the TSL_2 grammar T and $substrings$ (this is step (f) in (5)).

The TLA is *provably correct* Jardine & Heinz (in prep.). This means that, for any TSL_2 pattern with the grammar (T, S), given a set of strings exhibiting this pattern containing the right information, the TLA is guaranteed to return (T, S). The 'right information', at least for the version presented in Algorithm 1, simply is that there are strings in the data such that, given the paths of all the set of strings, there will be sufficient paths to remove each non-tier element from T, and that the last for loop in the algorithm will find each substring in S. The full version of the algorithm has one more condition on removing a symbol for the tier, and so it requires more information to converge. This will be touched on again in §5. The following section uses examples from natural language to show how the algorithm works; for a full proof, see Jardine & Heinz (in prep.).

It should be emphasized that probability plays no role in the algorithm. This makes it 'brittle' in the sense that any exceptional strings which do not exhibit the target pattern, no matter how rare, are treated equally with strings which do. The following demonstrations of the TLA thus assume exceptionless input to the algorithm. This is a standard factorization of the learning problem—the question asked is, can we learn the TSL_2 languages, even with perfect data? The following sections illustrate how we can. The harder problem of learning from imperfect, noisy data is not a primary concern of this paper, although integrating probability into the framework used here is touched on in §5.

4. Demonstrations of the TLA

While this algorithm is not complex, it would be illustrative to go through some examples. This section discusses application of the TLA to two natural-language examples, the Latin liquid dissimilation and Finnish vowel harmony patterns discussed in §2.1. First, a step-by-step walkthrough of how the algorithm works on a simplified Latin corpus is given in §4.1. Results of running the TLA on a corpus of Finnish data are discussed in §4.2.

4.1. Latin liquid dissimilation

The Latin data from Heinz (2010a) originally given in this paper in examples (2) through (4) are repeated below in (7).

(7) Latin data (from (Heinz, 2010a:(15-6))

a. navalis	'naval'	d. solaris	'solar'	g. floralis	'floral'
b. episcopalis	'infinitalis'	e. lunaris	'lunar'	h. sepulkralis	'funeareal'
c. infinitalis	'negative'	f. militaris	'military'	i. litoralis	'of the shore'

Recall that the Latin dissimilation generalization can be modeled with a TSL_2 grammar where $T = \{l, r\}$ and $subs = \{lr, rl\}$, excluding tier-adjacent subsstrings ll and rr. To get a working example from a smaller set of data, let us simplify the alphabet Σ to $\{l, r, a, t\}$, where [a] and [t] represent any vowel and any consonant, respectively. This yields the following data corresponding to the items in (7):

(8) Latin data simplified

a.	tatalat	d.	talarat	g.	tlaralat
b.	atattatalat	e.	latarat	h.	tataltralat
c.	attatatalat	f.	talatarat	i.	lataralat

The learner cannot succeed on this alone; in addition to the data in (8), let us add the following forms from the University of Notre Dame's online Latin dictionary (Cawley, 2014):

(9) a. tatrat (*migrus*, 'small, puny')
b. tarta (*certe*, 'certainly')

With the above data, the learner can sucessfully learn the grammar for Latin. The process is as follows. First, with $\text{T} = \Sigma$, the algorithm runs an initial call of get_tier and checks each symbol in $\{l, r, t, a\}$ to see if it is a free element; i.e., that it satisfies the if statement condition in (6) for removal. Because $\text{T} = \Sigma$, there are no symbols not on the tier, and so get_tier can only check paths where the set of intervening symbols is empty. In other words, at this stage, the only symbols that count as tier-adjacent are those which are strictly adjacent. The relevant set of paths for t is given in Table 4.

Path	Datum	Path	Datum
$\langle \#, \{\}, t \rangle$	navalis 'naval'	$\langle r, \{\}, t \rangle$	certe 'certainly'
$\langle t, \{\}, \# \rangle$	navalis 'naval'	$\langle t, \{\}, r \rangle$	migrus 'puny'
$\langle a, \{\}, t \rangle$	navalis 'naval'	$\langle l, \{\}, t \rangle$	sepulkralis 'funereal'
$\langle t, \{\}, a \rangle$	navalis 'naval'	$\langle t, \{\}, l \rangle$	floralis 'floral'
$\langle t, \{\}, t \rangle$	episcopalis 'episcopal'		

Table 4: Relevant paths for t when $T = \{t, r, l, a\}$

Note that for 't', every substring—that is, $\#t$, tt, ta, etc.—is present in the data. Thus, t is removed from T, and get_tier is called with this new T.

With $\text{T} = \{l, r, a\}$, now every substring for a is present, as can be seen in Table 5. This is because the learner can consider paths where the intervening set is $\{\}$ and $\{t\}$. So while there are no paths $\langle a, \{\}, a \rangle$ in the data (i.e., there were no strictly adjacent aa sequences), the path $\langle a, \{t\}, a \rangle$ is, for example in (7b) navalis 'naval'. Because the algorithm can now ignore t, the substring aa counts as tier-adjacent.

Path	Datum	Path	Datum
$\langle \#, \{\}, a \rangle$	episcopalis 'episcopal'	$\langle a, \{\}, r \rangle$	certe 'certainly'
$\langle a, \{\}, \# \rangle$	certe 'certainly'	$\langle r, \{\}, a \rangle$	migrus 'puny'
$\langle l, \{\}, a \rangle$	navalis 'naval'	$\langle a, \{\}, a \rangle$	(none)
$\langle a, \{\}, l \rangle$	navalis 'naval'	$\langle a, \{t\}, a \rangle$	navalis 'naval'

Table 5: Relevant paths for a when $T = \{r, l, a\}$

Thus, by taking advantage of the knowledge that t is not on the tier, the learner has made a generalization that it could not before. With a taken off of T, the tier is now $\{l, r\}$. The reader can confirm that even when $T = \{l, r\}$, the data do not show paths like $\langle l, \{a, t\}, l \rangle$ or $\langle r, \{a, t\}, r \rangle$, and so l and r will not be taken off of the tier. The tier is then correctly returned as $\{l, r\}$. Since no

substrings ll and rr were tier-adjacent, it also correctly returns the set of tier substrings $substrings = \{\rtimes l, l\#, lr, r\#, rl, \#r\}$, which is the Latin TSL$_2$ grammar from Table 2 (with word boundaries added).

In fact, given any set of data which conforms to the Latin liquid dissimilation generalization given in §2.1 will never contain a path like $\langle l, \{a, t\}, l\rangle$ or $\langle r, \{a, t\}, r\rangle$. As the generalization states that ls and rs must alternate regardless of intervening vowels and non-liquids—represented here as a and t—two ls, for example, will never be seen with only as and ts between them. This is of course assuming no exceptions are present, a point which will be returned to later in §5.

We have now seen how the core idea of how the TLA works. It finds material which can be removed from the tier, symbol by symbol, and then generalizes based on changes to the tier. This example involved a small set of data with an extremely simplified alphabet. The next section discusses how the algorithm may be applied to a larger corpus.

4.2. Finnish vowel harmony

The TLA was also tested on Goldsmith & Riggle (2012)'s Finnish data. As Goldsmith & Riggle (2012) note, Finnish is a convenient language to use because its phonemic inventory is relatively small and its orthography is faithful to its pronunciation. The corpus used was 2,000 items randomly chosen from Goldsmith & Riggle's corpus, and then 'sanitized' to remove exceptions.

The full consonant inventory in the data, represented orthographically, was $\{p, t, d, k, g, v, h, s, m, n, r, l, j\}$. The vowels, split into their harmony groups, are given in Table 6.

Transparent	[−back]	[+back]
i	$\ddot{u}$	u
e	$\ddot{o}$	o
	$\ddot{a}$	a

Table 6: Finnish vowels

A TSL$_2$ grammar with this alphabet modeling the Finnish backness vowel harmony generalization discussed in §2.1 would have $T = \ddot{u}, \ddot{o}, \ddot{a}, u, o, a$ and (excluding the beginning and end symbols) $substrings = \{\ddot{u}\ddot{u}, \ddot{u}\ddot{o}, \ddot{u}\ddot{a}, \ddot{o}\ddot{u}, \ddot{o}\ddot{o}, \ddot{o}\ddot{a}, \ddot{a}\ddot{u}, \ddot{a}\ddot{o}, \ddot{a}\ddot{a}, uu, uo, ua, ou, oo, oa, au, ao, aa\}$. This grammar states that only vowels agreeing in backness can be tier-adjacent.

When tested on the data using the full inventory, the learner failed, returning a tier equal to the full alphabet. This is because local dependencies between symbols prevented the learner from reducing the tier. For example, the local sequence dp was never found in the data, and so neither d nor p were taken off of the tier. This is an issue for the TLA: to begin removing symbols from the tier, it needs to find at least one symbol which is freely distributed in the data. This is probably quite rare in natural language corpora, as most phonemes are subject to local phonotactic restrictions.

The solution taken here is one similar to the approach taken by Goldsmith & Riggle (2012), which is to use natural classes to help make generalizations. The difference here is that we shall assume that natural classes have already been found. As with the Latin example in the previous section, natural classes can be built in by simplifying the alphabet. For Finnish, this can be done as follows: stops $\{p, t, d, k, g\}$ are represented as t, fricatives $\{v, h, s\}$ as s, nasals $\{m, n\}$ as n, and other sonorants $\{r, l, j\}$ as j. Vowels were left unsimplified. Example data points, with both the unsimplified orthography and the simplified alphabet are given in Table 7.

	English	Finnish: orthog.	nat. classes
a.	'bear'	karhu	tajsu
b.	'bird'	lintua	jintua
c.	'cat'	kissa	tissa

Table 7: Examples of Finnish data

This time, the learner was successful, discovering the grammar in (10) (ignoring substrings including the word boundaries, which are not relevant for the Finnish generalization).

(10) $T = \{a, \ddot{a}, o, u, \ddot{o}, \ddot{u}\}$
$substrings = \{aa, ao, au, oa, oo, ou, ua, uo, uu, \ddot{a}\ddot{a}, \ddot{a}\ddot{o}, \ddot{a}\ddot{u}, \ddot{o}\ddot{a}, \ddot{o}\ddot{o}, \ddot{o}\ddot{u}, \ddot{u}\ddot{a}, \ddot{u}\ddot{o}, \ddot{u}\ddot{u}\}$

The learner correctly learned that the tier is only the vowels, even excluding the transparent vowels i and e. This means that, given the consonant natural classes $\{t, s, n, j\}$, the algorithm found in the Finnish data enough information to remove each from the tier. The same goes for the transparent vowels $\{i, e\}$. Thus, the information needed for the algorithm to successfully infer the pattern is found in the natural language data, although with the caveat that the data needs to be filtered with natural classes. This point is returned to in more detail in the following section.

5. Discussion

To give a brief summary of what has been presented so far, §3 gave an overview of a simple version of the TSL_2 algorithm. Next, §4 demonstrated this algorithm using examples from natural language. This section will review the issues raised in §3 and §4.

First, the full version of the algorithm should be briefly addressed. Patterns in the TSL_2 class include those without blocking, as in Finnish, and those with blocking, as in Latin: an intervening r prevents two ls from dissimilating. However, there is a special subset of blocking patterns which are outside the reach of the algorithm given in §3. These are the set of patterns in which blockers exist, but there are no constraints on their distribution with regards to the tier. This is exactly the case that Cser (2010) argues for Latin: it is not just /r/ that blocks /l/ dissimilation, but any noncoronal consonant. Some examples are given below.

(11) Noncoronals blocking dissimilation in Latin (Cser, 2010:(13))

a. plectilis ‘pliable’ (*plectiris)
b. labilis ‘slippery’ (*labiris)
c. fluviatilis ‘river–’ (*fluviatiris)

To model for this behavior with a TSL_2 grammar, the alphabet from Table 2 must be increased to at least $\Sigma = \{l, r, t, a, k\}$, where k represents noncoronals. If we set the tier to $T = \{l, r, k\}$, then k can intervene between liquids. For example, for (11a) plectilis ‘pliable’, would in this alphabet be $tlaktalat$, and $proj(tlaktalat) = lkl$. As ll is not a substring of lkl, then the word is allowed. However, while they appear on the tier, the cooccurrence of noncoronals is not restricted in the way that the liquids are. Recall that in Algorithm 1, any symbol found not to be restricted is immediately taken off of the tier. Thus, if it found k to be freely distributed, this version of the TLA would incorrectly take k off of its guess for the tier.

As detailed in Jardine & Heinz (in prep.), the solution for this is to add a second condition on removal from the tier. Briefly, this condition states that a symbol a can only be removed from the tier if for all paths $\langle b, X, c\rangle$ in the input, where b and c are on the tier and X is made up only of non-tier elements and a, there must also be a path $\langle b, X', c\rangle$ where X' is made up only of non-tier elements. In other words, there can be no b, c whose tier adjacency depends on a. Continuing with the Latin example, k would not satisfy this condition if $\langle l, \{t, a, k\}, l\rangle$ appeared in the data (and it is indeed a path for (11a) plectilis, or $tlaktalat$), because $\langle l, \{t, a\}, l\rangle$ never will (as this can only come from a word with a ll substring on the tier). By checking for these dependencies, the algorithm can correctly identify ‘free’ blockers like k.

This full version of the TLA with the second condition for removal from the tier is also provably correct. However, the second condition increases the sample of data the algorithm needs in order to be guaranteed to give the correct grammar, as now in order to remove a symbol from the tier, it needs to see more data in order to make sure it is not a ‘free’ blocker. This extra information may not be present in natural language data. For instance, the full version of the TLA fails to remove any symbols from the tier given on the Finnish corpus, even with the alphabet of natural classes used in §4.2.

This leads to the issue of using the TLA on natural language data. The algorithm in §3 is idealized in the sense that it is only guaranteed to learn the target pattern if there are no exceptions to the target pattern in the data and if it can extract from the data all of the paths necessary to take each non-tier symbol off

of its guess for the tier. This represents a factorization of the learning problem—can we learn long-distance phonotactics even when unhindered by data exhibiting exceptions and other generalizations? The algorithm in §3 is a positive answer to this heretofore unanswered question, and thus is one step towards understanding learning on natural language data.

The next steps entail dealing with exceptions and factoring out other generalizations in the data. There are clear paths to solving both of these problems. Exceptions in the data can be handled by considering a probabilistic version of the learner. The formal languages discussed here were categorical, but formal language grammars admit stochastic counterparts, and such probabilistic formal languages can be learned (Heinz & Rogers, 2010). Thus, it is likely that a stochastic version of the TLA could take advantage of the insights into learning TSL_2 languages used by the algorithm presented in this paper. A stochastic TLA might, for example, take into account *frequency* of certain paths before removing a symbol from the tier.

As for factoring out other generalizations, the results in (4) suggest one possible solution. As shown explicitly for Finnish in §4.2, local phonotactic dependencies can prevent the algorithm from changing its guess for the tier, as it relies on finding 'free' symbols. However, it was also shown that by learning over natural classes, instead of individual phoneme symbols, the algorithm was able to overcome this difficulty. This suggests that coupled with a principled way of inducing natural classes, long-distance dependencies can be learned from raw natural data.

There remain a few more challenges. One is that the TSL_2 languages as defined in §2.3 only admit *one* tier, but languages have been analyzed as having multiple, independent tiers. One example is Yaka, which has both vowel harmony and long-distance harmony of liquids with nasals (Kidima, 1991; Hyman, 1998), but the two generalizations are independent of each other. As the TLA can only learn single TSL_2 languages, it would not be able to learn both generalizations. A potential solution to this problem is positing multiple TLA learners focused on different natural classes. One could focus on vowels but collapse the distinctions between consonants (as in the Finnish case here), and the other could focus on the distinctions between consonants but collapse the vowels.

One last avenue for further work is to learn these long-distance generalizations not as phonotactics, but as phonological transformations. This is particularly relevant for patterns like Latin which are clearly related to allomorphy and perhaps better characterized as a transformation from underlying form to surface form. While the TSL_2 formal languages can only model phonotactics, there is precedent for work on formal languages informing work on learning phonological transformations. Chandlee (2014)'s Strictly Local functions, which can model many phonological transformations, are based on the Strictly Local formal languages (McNaughton & Papert, 1971), and it is this notion of locality that makes this class of functions learnable (Chandlee et al., 2014; Jardine et al., 2014).

Finally, what does this algorithm teach us about acquisition? As mentioned in §2.3, while TSL_2 grammars are in some sense abstract, there is both typological and psycholinguistic evidence that they capture something about how human beings reason about dependencies between non-adjacent segments. The TLA takes advantage of the structure of these grammars to learn; it is entirely possible that human beings do as well. To test this, the learning mechanisms of the TLA can be taken as psycholinguistic predictions about how people learn. For example, the notion of paths predicts that people are not just sensitive to precedence relations between sounds, but the sets of sounds which come between them. Additionally, it predicts that people first consider a large tier, then consider increasingly smaller tiers as learning progresses.

6. Conclusion

This paper has presented an algorithm which learns a class of patterns that includes many common long-distance phonotactics. While abstract, the algorithm provides some insights into how long-distance phonotactics can be learned. One, the concepts of a *tier* and *locality* are sufficient for learning a particular tier and tier-local dependencies over that tier. Two, by initially guessing that the tier is equal to the full inventory and then incrementally removing elements from that guess, it is possible to infer new generalizations. Thus, the TLA offers theoretical, yet concrete, understanding of how long-distance phonotactics may be learned.

References

Bennett, William (2013). *Dissimilation, Consonant Harmony, and Surface Correspondence*. Ph.D. thesis, Rutgers, the State University of New Jersey.

Cawley, Kevin (2014). Latin dictionary and grammar aid (website). http://archives.nd.edu/latgramm.htm.

Chandlee, Jane (2014). *Strictly Local Phonological Processes*. Ph.D. thesis, University of Delaware.

Chandlee, Jane, Rémi Eyraud & Jeffrey Heinz (2014). Learning Strictly Local subsequential functions. *Transactions of the Association for Computational Linguistics* 2, 491–503.

Chomsky, Noam & Morris Halle (1968). *The Sound Pattern of English*. Harper & Row.

Clements, George N. & Engin Sezer (1982). Vowel and consonant disharmony in Turkish. van der Hulst, Harry & Norval Smith (eds.), *The Structure of Phonological Representations (Part II)*, Foris, Dordrecht.

Cohn, Abigail (1992). The consequences of dissimilation in Sundanese. *Phonology* 9, 199–220.

Cser, András (2010). The -alis/aris- allomorphy revisited. Kastovsky, D., F. Rainer, W. U. Dressler & H. C. Luschützky (eds.), *Variation and change in morphology: selected papers from the 13th international morphology meeting*, Philadelphia: John Benjamins, 33–51.

García, Pedro, Enrique Vidal & José Oncina (1990). Learning locally testable languages in the strict sense. *Proceedings of the Workshop on Algorithmic Learning Theory*, 325–338.

Goldsmith, John & Jason Riggle (2012). Information theoretic approaches to phonological structure: the case of finnish vowel harmony. *Natural Language & Linguistic Theory* 30:3, 859–896, URL `http://dx.doi.org/10.1007/s11049-012-9169-1`.

Hansson, Gunnar Ólafur (2001). *Theoretical and Typological Issues in Consonant Harmony*. Ph.D. thesis, University of California, Berkeley.

Hansson, Gunnar Ólafur (2010). *Consonant harmony: Long-distance interaction in phonology*. Berkeley: University of California Press.

Hayes, Bruce & Colin Wilson (2008). A maximum entropy model of phonotactics and phonotactic learning. *Linguistic Inquiry* 39, 379–440.

Heinz, Jeffrey (2010a). Learning long-distance phonotactics. *Linguistic Inquiry* 41, 623–661.

Heinz, Jeffrey (2010b). String extension learning. *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, 897–906.

Heinz, Jeffrey & James Rogers (2010). Estimating strictly piecewise distributions. *Proceedings of the 48th Annual Meeting of the ACL*, Association for Computational Linguistics.

Heinz, Jeffrey, Chetan Rawal & Herbert G. Tanner (2011). Tier-based strictly local constraints for phonology. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, Portland, Oregon, USA, 58–64.

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

Hyman, Larry (1998). Positional prominence and the 'prosodic trough' in Yaka. *Phonology* 15, 14–75.

Hyman, Larry (2011). Tone: Is it different? Goldsmith, John A., Jason Riggle & Alan C. L. Yu (eds.), *The Blackwell Handbook of Phonological Theory*, Wiley-Blackwell, 197–238.

Jardine, Adam & Jeffrey Heinz (in prep.). Learning tier-based strictly local languages. Paper to be submitted to *Transactions of the Association of Computational Linguistics*, URL `udel.edu/ ajardine/files/jardineheinz15tsl.pdf`.

Jardine, Adam, Jane Chandlee, Rémi Eryaud & Jeffrey Heinz (2014). Very efficient learning of structured classes of subsequential functions from positive data. *Proceedings of the 12th International Conference on Grammatical Inference (ICGI 2014)*, JMLR Workshop Proceedings, 94–108.

Jensen, John (1974). Variables in phonology. *Language* 50, 675–686.

Jurafsky, Daniel & James H. Martin (2009). *Speech and Language Processing*. Pearson Prentice Hall, 2nd edition edn.

Kidima, Lukowa (1991). *Tone and Accent in KiYaka*. Ph.D. thesis, University of California, Los Angeles.

McMullin, Kevin & Gunnar Ólafur Hansson (2014). Long-distance phonotactics as Tier-Based Strictly 2-Local languages. *Proceedings of the Annual Meeting on Phonology 2015*. Talk; manuscript to be submitted.

McMullin, Kevin & Gunnar Ólafur Hansson (2015). Locality in long-distance phonotactics: evidence for modular learning. *NELS 44, Amherst, MA*. In press.

McNaughton, Robert & Seymour Papert (1971). *Counter-Free Automata*. MIT Press.

Nevins, Andrew (2010). *Locality in Vowel Harmony*. No. 55 in Linguistic Inquiry Monographs, MIT Press.

Odden, David (1994). Adjacency parameters in phonology. *Language* 70:2, 289–330.

Prince, Alan & Paul Smolensky (1993). Optimality Theory: Constraint interaction in generative grammar. *Rutgers University Center for Cognitive Science Technical Report* 2.

Ringen, Catherine (1975). *Vowel Harmony: Theoretical Implications*. Ph.D. thesis, Indiana University.

Rogers, James, Jeffrey Heinz, Gil Bailey, Matt Edlefsen, Molly Visscher, David Wellcome & Sean Wibel (2010). On languages piecewise testable in the strict sense. Ebert, Christian, Gerhard Jäger & Jens Michaelis (eds.), *The Mathematics of Language*, Springer, vol. 6149 of *Lecture Notes in Artifical Intelligence*, 255–265.

Rose, Sharon & Rachel Walker (2004). A typology of consonant agreement as correspondence. *Language* 80, 475–531.

Suzuki, Keiichiro (1998). *A typological investigation of dissimilation*. Ph.D. thesis, University of Arizona, Tucson.

Yip, Moira (2002). *Tone*. Cambridge University Press.