# More efficiently identifying the tiers of strictly 2-local tier-based functions

**Phillip Burness**
University of Ottawa
pburn036@uottawa.ca

**Kevin McMullin**
University of Ottawa
kevin.mcmullin@uottawa.ca

## Abstract

String-to-string functions that consider purely local information have proven useful for modelling local phonological processes, and similar modelling of long-distance processes is possible when the assessment of locality is relativized to subsets of the segment inventory (usually called *tiers*). Such tier-based functions can be learned in quadratic time and data when the tier(s) are known in advance, but existing methods for inducing the tier(s) run in quintic time. Current algorithms tailored specifically to learn tier-based functions are thus much slower overall than the cubic upper bound established for learning the superclass of subsequential functions. We show that the bottlenecks responsible for this comparatively inefficient runtime can be circumvented by judiciously using a Prefix Tree Transducer when inducing the tier(s). Doing so brings us down to a quadratic upper bound on overall runtime.

## 1 Introduction

It is generally accepted that the *regular* region of the Chomsky hierarchy (Chomsky, 1956) is sufficiently expressive to describe the attested range of human phonological patterns (Johnson, 1972; Kaplan and Kay, 1994). Equally well-established, though, is that regular languages are not learnable in the limit from positive data alone (Gold, 1967). Accordingly, various subsets of the regular languages and functions (i.e., *subregular* classes) have been explored as alternatives.

Local phonotactics and processes enjoy particularly strong learning results in this regard, as they can respectively be modelled using Strictly Local languages (see for example Rogers and Pullum, 2011; Rogers et al., 2013) and Strictly Local functions (see for example Chandlee and Heinz, 2018). Strictly Local languages are those that ban particular contiguous sequences from appearing in raw strings, and the runtime of their associated learning algorithm is linearly proportional to the size of the sample (Garcia et al., 1990). For their part, Strictly Local functions are those where the transformation of an input segment depends only on its immediately surrounding material in the raw string, and the runtime of their associated learning algorithms is quadratically proportional to the size of the sample (Chandlee et al., 2014, 2015).

Non-local phonotactics also enjoy strong learning results, since they can be modelled as Tier-based Strictly Local languages (see for example McMullin and Hansson, 2016). These are essentially relativized versions of Strictly Local languages that prohibit contiguous sequences after the raw strings have been projected onto a tier (Heinz et al., 2011; Lambert and Rogers, 2020). The runtime of their associated learning algorithm is linear in the size of the sample when the tier is known beforehand, but is quadratic when the tier must be identified (Jardine and McMullin, 2017). As for non-local processes, they have commonly been modelled as subsequential functions (Heinz and Lai, 2013; Luo, 2017; Payne, 2017) which can be learned in cubic time (Oncina et al., 1993).

A separate line of recent work, however, shows that non-local processes can equally be modelled using the weaker class of Tier-based Strictly Local functions, which extend Tier-based Strictly Local languages to functions in the same way that Strictly Local functions extend Strictly Local languages (see for example Andersson et al., 2020; Burness et al., 2021). Previous work on learning tier-based functions found that, while a transducer computing the function could be constructed in quadratic time when the tier is known in advance, learning the tier itself (where this is possible) took quintic time (Burness and McMullin, 2019). This had the odd consequence of making it less efficient to learn a Tier-based Strictly Local function than a subse-



*Proceedings of the 17th Meeting on the Mathematics of Language*, pages 38–49, Montpellier (online),
December 13, 2021. 

quential function, even though the former class is strictly less expressive than the latter. In this paper, we amend this discrepancy, showing that inefficiencies in the existing tier induction methods can be eliminated by manipulating a Prefix Tree Transducer, a data structure commonly used in grammatical inference. Doing so reduces time complexity by three polynomial degrees, making it possible to identify a tier in quadratic time.

The rest of the paper is organized as follows. Section 2 outlines notation and definitions to be used throughout; it also provides formal background on tier-based functions and important properties thereof. Next, Section 3 discusses the first portion of Burness and McMullin's (2019) learning algorithm which extracts particular information from the training sample necessary for subsequent steps; we identify the bottleneck responsible for the relative inefficiency of this procedure and show that it can be avoided using a Prefix Tree Transducer. Then, Section 4 discusses the portion of Burness and McMullin's (2019) algorithm that identifies the target function's tier; this procedure faces a similar bottleneck to the preceding one which is also avoidable by using a Prefix Tree Transducer. Finally, Section 5 concludes and discusses directions for future research.

## 2 Preliminaries

### 2.1 Notation

Given a string $w$ made of symbols from some alphabet $\Sigma$, we write $|w|$ to denote the length of that string. Below, strings will frequently be flanked by the special non-alphabet symbols $\rtimes$ and $\ltimes$, which denote the start and end of a string, respectively. Given an alphabet $\Sigma$, we write $\Sigma^*$ to denote all possible strings made from that alphabet. The unique string of length 0 (i.e. the empty string) is written as $\lambda$. Given two strings $u$ and $v$, we write $u \cdot v$ to denote their concatenation, though when context allows, we will save space by simply writing $uv$.

A suffix of some string $w$ is any string $s$ such that $w = x \cdot s$ and $x, s \in \Sigma^*$. Similarly, a prefix of some string $w$ is any string $p$ such that $w = p \cdot x$ and $p, x \in \Sigma^*$. Note that any string is a prefix and a suffix of itself, and that the empty string $\lambda$ is a prefix and a suffix of every string. When $|w| \geq n$, $\texttt{suff}^n(w)$ denotes the unique suffix of $w$ with a length of $n$; when $|w| < n$, it simply denotes $w$ itself. Similarly, when $|w| \geq n$, $\texttt{pref}^n(w)$ denotes the unique prefix of $w$ with a length of $n$; when $|w| < n$, it simply denotes $w$ itself. We also write $\texttt{pref}^*(w)$ to denote the set of all prefixes of any length in $w$.

A string-to-string function pairs every $w \in \Sigma^*$ with one $y \in \Delta^*$, where $\Sigma$ and $\Delta$ are the input alphabet and output alphabet respectively. Given a set of input strings $I \subseteq \Sigma^*$, $f(I) = \bigcup_{i \in I}\{f(i)\}$ is the set of all outputs associated to at least one of the inputs. Given a set of strings $S$, we write $\texttt{lcp}(S)$ to denote the *longest common prefix* of $S$, which is the string $u$ such that $u$ is a prefix of every $w \in S$, and there exists no other string $v$ such that $|v| \geq |u|$ and $v$ is also a prefix of every $w \in S$.

An important concept is that of the *tails* of an input string $w$ with respect to a function $f$. In words, $\texttt{tails}_f(w)$ pairs every possible string $y \in \Sigma^*$ with the portion of $f(wy)$ that is directly attributable to $y$. Stated differently, $\texttt{tails}_f(w)$ describes the effect that $w$ has on the output of any subsequent string of input symbols. When $\texttt{tails}_f(w_1) = \texttt{tails}_f(w_2)$ we say that $w_1$ and $w_2$ are *tail-equivalent* with respect to $f$.

**Definition 1.** Tails (Oncina and Garcia 1991)
*Given a function $f$ and an input $w \in \Sigma^*$:*

$$\texttt{tails}_f(w) = \{(y, v) \mid f(wy) = uv \wedge u = \texttt{lcp}(f(w\Sigma^*))\}$$

Throughout the rest of this paper, we will need to be able to pick out the portion of the output that corresponds to actual input material. Given a transducer representation of the relevant function, this boils down to a distinction between the writing that occurs while reading segments from sigma $\Sigma$ and any writing that occurs when the end of the word is reached (i.e., when $\ltimes$ is read). To make this distinction, Chandlee et al. (2015) defined the prefix function $f^p$ associated with a subsequential function $f$ as below. An example where $f(w)$ and $f^p(w)$ differ would be a function that appends $a$ to the end of every input string. In this case, $f^p$ is simply the identity map, so $f^p(abc) = abc$ whereas $f(abc) = abca$.

**Definition 2.** Prefix function (Chandlee et al. 2015)
*Given a function $f$, its associated prefix function $f^p$ is such that:*

$$f^p(w) = \texttt{lcp}(f(w\Sigma^*))$$

Finally, a useful concept related to tails, tail-equivalency, and prefix functions is the *contribution* of a symbol $\sigma \in \Sigma$ relative to a string $w \in \Sigma^*$ with respect to a function $f$. In words, for an input

string $x$ that has the prefix $w\sigma$, the contribution of the $\sigma$ in $w\sigma$ is the portion of $f(x)$ that is uniquely and directly attributable to that instance of $\sigma$. We also define a special case for the word-end symbol $\ltimes$ that is not part of $\Sigma$. The notation $x^{-1} \cdot w$ represents the string $w$ with $x$ removed from its front, so $a^{-1} \cdot aba = ba$ for example.

**Definition 3.** Contribution
*Given a function $f$ and some $w \in \Sigma^*$:*

- *For $\sigma \in \Sigma$:*
  $\texttt{cont}_f(\sigma, w) = f^p(w)^{-1} \cdot f^p(w\sigma) = \texttt{lcp}(f(w\Sigma^*))^{-1} \cdot \texttt{lcp}(f(w\sigma\Sigma^*))$
- *For $\ltimes \notin \Sigma$:*
  $\texttt{cont}_f(\ltimes, w) = f^p(w)^{-1} \cdot f(w) = \texttt{lcp}(f(w\Sigma^*))^{-1} \cdot f(w)$

### 2.2 Single-tiered functions

Where a Strictly Local (SL) function divides $\Sigma^*$ into tail-equivalence classes based on suffixes of raw strings (Chandlee, 2014; Chandlee et al., 2014, 2015), a Tier-based Strictly Local (TSL) function's tail-equivalence classes are based on suffixes of strings *after masking irrelevant elements* (Burness and McMullin, 2019; Hao and Andersson, 2019; Hao and Bowers, 2019). Relevant elements are those that belong to the specified *tier* (a subset of the alphabet) and the masking is accomplished with an *erasure* function, sometimes also called a *tier projection*.

**Definition 4.** Erasure function
*Given a tier $T \subseteq \Sigma$, the* erasure function *applied by $T$ on $\Sigma^*$ is such that:*

$$\begin{aligned} \texttt{erase}_T(\lambda) &= \lambda \\ \texttt{erase}_T(w) &= \texttt{erase}_T(u) \cdot \sigma \text{ if } w = u \cdot \sigma \wedge \sigma \in T \\ \texttt{erase}_T(w) &= \texttt{erase}_T(u) \text{ if } w = u \cdot \sigma \wedge \sigma \notin T \end{aligned}$$

SL and TSL functions are really divided into two types. On the one hand are the the Input (Tier-based) Strictly Local or I(T)SL functions which care about suffixes of the input string. On the other hand are the Output (Tier-based) Strictly Local or O(T)SL functions which care about suffixes of the output string. For reasons of space, we focus on the output-oriented OTSL functions in this paper, but the results herein are easily extended to the input-oriented ITSL case. As indicated in the following formal definition, the OTSL functions are further subdivided based on the length of suffix that is being tracked, although the learning results below apply only for $k = 2$. Note that we write $\texttt{suff}_T^n(w)$ as shorthand for $\texttt{suff}^n(\texttt{erase}_T(w))$.

**Definition 5.** Output Tier-based Strictly $k$-Local Functions (Burness and McMullin, 2019)
*A function $f$ is OTSL$_k$ if there is a tier $T \subseteq \Delta$ such that for all $w_1, w_2$ in $\Sigma^*$:*

$$\texttt{suff}_T^{k-1}(f^p(w_1)) = \texttt{suff}_T^{k-1}(f^p(w_2)) \implies \texttt{tails}_f(w_1) = \texttt{tails}_f(w_2)$$

The tier-induction strategy of Burness and McMullin (2019), which we optimize in this paper, relies on some important properties of OTSL$_2$ functions. First, many OTSL$_2$ functions can be described using a variety of tiers (e.g., the identity map can be described using any subset of the output alphabet), but taking the union of two potential tiers will always result in another potential tier (i.e., potential tiers can be freely combined).

**Lemma 1.** Free combination of tiers
*Given an OTSL$_2$ function $f$, if $A \subseteq \Delta$ and $B \subseteq \Delta$ are both tiers for $f$, then $\Omega = A \cup B$ is also a tier for $f$.*

*Proof.* See the proof of Lemma 4 in Burness and McMullin (2019). □

The above Lemma implies the existence of a unique largest tier for any OTSL$_2$ function that is a superset of its other possible tiers (if any others exist). Following Burness and McMullin (2019), we call this the *canonical* tier for $f$.

**Definition 6.** Canonical tier
*Given an OTSL$_2$ function $f$, the tier $T \subseteq \Delta$ is the* canonical *tier for $f$ if and only if there is no other tier $\Omega \subseteq \Delta$ for $f$ such that $|\Omega| \geq |T|$.*

Burness and McMullin (2019) go on to show that, if one attempts to describe an OTSL$_2$ function using a superset of its canonical tier, then there will always be at least one input-output pair which acts as evidence that one of the superfluous tier elements cannot be a member of *any* tier for $f$.

**Lemma 2.** Absolute non-tier status
*Let $f$ be an OTSL$_2$ function where $T \subseteq \Delta$ is the canonical tier. For every $\Omega$ such that $T \subset \Omega$ there will exist $a \in (\Omega - T)$, $w_1, w_2 \in \Sigma^*$, and $x \in \Sigma \cup \{\ltimes\}$ such that $\texttt{suff}_\Omega^1(f^p(w_1)) = \texttt{suff}_\Omega^1(f^p(w_2)) = a$ and $\texttt{cont}_f(x, w_1) \neq \texttt{cont}_f(x, w_2)$.*

*Proof.* See the proof of Lemma 5 in Burness and McMullin (2019). □

Taking advantage of Lemma 1 and Lemma 2 together, we can begin by hypothesizing that the canonical tier is equal to the entire output alphabet $\Delta$ and whittle this hypothesis down as needed until we converge on the canonical tier. To do so, we look through our sample for evidence that some element cannot be on the tier, and if such an element is found, we remove it from the hypothesized tier. When no elements can be flagged for removal, we will have found the canonical tier.

### 2.3 Multi-tiered functions

With a TSL function, we are limited to a single tier. This is not necessarily an issue when we are considering isolated long-distance processes, but it severely limits our capacity to describe fuller phonological systems. Burness and McMullin (2021) address this issue at least partially by defining a class of Multi-Tiered Strictly Local (MTSL) functions that tracks multiple independent tier projections in parallel. The class they develop imposes a particular relationship between the tiers and the input alphabet. Namely, the contribution (see Definition 3) of a given input element can always be linked back to the effects of a set tier, although different input elements can be affected by different tiers. Viewed another way, each input element specifies a tier to which it pays exclusive attention.

In light of space limitations, and in order to cut down on redundancy in the proofs below, we henceforth stick to single-tiered functions, noting that the results herein are straightforwardly extended to Burness and McMullin's (2021) *strongly target-specified* MTSL functions just described. The major motivation behind this particular type of MTSL function was that Burness and McMullin's (2019) method for learning the single tier of a TSL function readily generalizes to the "one tier per input element" case. Our changes to the single-tier learner below do not affect any of the properties that allowed for Burness and McMullin's (2021) generalization to such multiple independent tiers.

## 3 Estimating the prefix function

Identifying the tier of a target function is done by comparing contributions and checking for any that do not match when the current hypothesis says they should. Calculating contributions requires knowledge of the target function's associated prefix function $f^p$, so the first step in the tier-learning pipeline is to extract as much knowledge as possible about $f^p$ from the given sample. Burness and McMullin (2019) devised a method of doing so whose worst-case run time is in $\mathcal{O}(|S|^4)$, where $|S|$ is the size of the training sample. The relative inefficiency of this method comes from the fact that it must read through the sample once for each prefix in the sample, and must calculate the longest common prefix of the set returned by each of these nested reads. We present an alternative method in this section whose worst-case run time is in $\mathcal{O}(|S|^2)$ and which also allows us to greatly improve the efficiency of later learning steps. By creating and manipulating an auxiliary data structure, we completely eliminate the need for the problematic nested reads.

### 3.1 Building an onward PTT

We start with what is known as a Prefix Tree Transducer (PTT), defined in Definition 7 which is adapted from Chandlee et al. (2014). The PTT corresponding to a sample of input-output pairs effectively generates all and only the pairs in the sample, writing the entire output in one fell swoop after reading the entire input. For example, the PTT corresponding to {(s, s), (ss, ss), (sʃ, ss), (so, so), (sos, sos), (soʃo, soso), (sooʃ, soos)} is shown in Figure 1. To avoid visual clutter, all transitions landing in the designated final state ($q_f$) are incorporated into the label of their origin state.

**Definition 7.** Prefix Tree Transducer (adapted from Chandlee et al. 2014)
*A* Prefix Tree Transducer (PTT) for the finite set $D$ of pairs $(w, w')$ from some function $f$ is $PTT(D) = (Q, q_0, q_f, \Sigma, \Delta, \delta)$ where:

- $Q = \bigcup_{(w,w')\in D}\{\texttt{pref}^*(w)\}$
- $(\forall u \in \Sigma^*)(\forall a \in \Sigma)$ $[u, ua \in Q \iff (u, a, \lambda, ua) \in \delta]$
- $(w, w') \in D \iff (w, \ltimes, w', q_f) \in \delta$
- $(q_0, \rtimes, \lambda, \lambda) \in \delta$

In this initial form, a PTT is maximally lazy, waiting as late as possible before writing any output. For tier learning, we need to modify the PTT so that it is *minimally* lazy, producing as much output as it can as early as it can. To perform such a conversion, we can perform a depth-first parse of the sample working backwards from the leaves of the prefix tree towards the root. For each state along the way, we calculate the longest common prefix of the output edges on its outgoing transitions, pushing that string onto the output edge of its

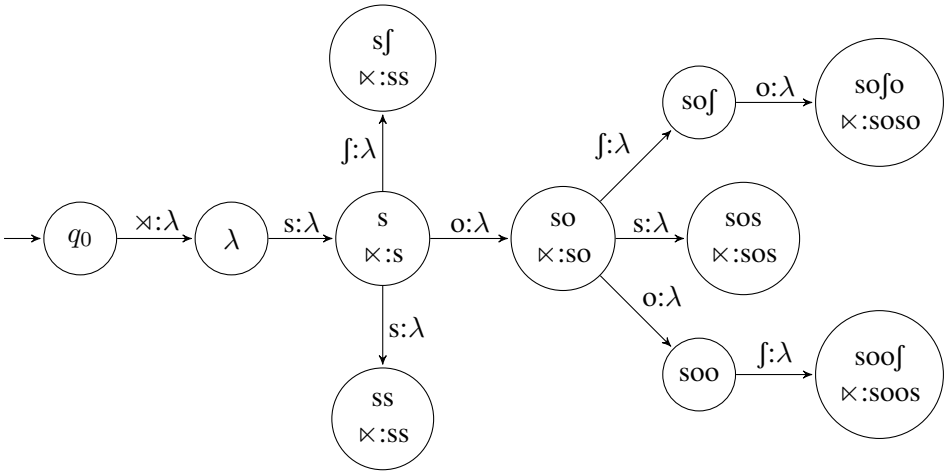


Figure 1: The PTT for the sample {(s, s), (ss, ss), (sʃ, ss), (sa, sa), (sas, sas), (saʃa, sasa), (saaS, saas)}

lone incoming transition (de la Higuera, 2010, pp. 377-379). We use the term onward PTT to refer to such a converted PTT and write $\texttt{onward}(M)$ to denote the process being applied to $M$. For the full details of how to build a PTT and make it onward, see chapter 18 of de la Higuera (2010). Figure 2 shows the result of onwarding the PTT in Figure 1.

Onward PTTs are used by the Onward Subsequential Transducer Inference Algorithm (OSTIA) of Oncina et al. (1993) and are used by the learning algorithm for ISL functions (Chandlee et al., 2014) but not the one for OSL functions (Chandlee et al., 2015). Interestingly, both OSTIA and the ISL function learning algorithm obtain a transducer representation of the target function by applying a process of state merging to an onward PTT. In contrast, the learning algorithm in this paper merely uses an onward PTT as a sort of oracle, consulting it for essential pieces of information without modifying it in any way.

### 3.2 Extracting useful information

A PTT that has been made onward exhibits some important properties that will be exploited below. First, given a state $q \in Q$ that has an outgoing transition for all $x \in \Sigma \cup \{\ltimes\}$, we will have produced exactly $f^p(q)$ so far when we enter the state $q$. We call such a state a *supported* state. Assuming that $\Sigma = \{\text{s, o, ʃ}\}$, the supported states in Figure 2 are 's' and 'so'.

**Definition 8.** Supported state
*Given an onward PTT* $P = (Q, q_0, q_f, \Sigma, \Delta, \delta)$, *the state* $q \in Q$ *is* supported *if and only if:*

$$(\forall x \in \Sigma \cup \{\ltimes\})[\exists (q, x, y, q') \in \delta]$$

**Lemma 3.** *Let* $P$ *be the onward PTT constructed according to sample* $S$ *drawn from function* $f$. *Given a supported state* $q \in Q$, *it is the case that we will have written exactly* $f^p(q)$ *upon entering* $q$ *after starting in* $q_0$.

*Proof.* Since $q$ is supported, it is the case that $(\forall x \in \Sigma \cup \{\ltimes\})[\exists (q, x, y, q') \in \delta]$. This in turn means that we have $(q, f(q)) \in S$ and for each $a \in \Sigma$ we have $(qab, f(qab)) \in S$ for some $b \in \Sigma^*$. An onward PTT is deterministic and acyclic, so inputs will only pass through $q$ if they have $q$ as a prefix, and all such inputs in $S$ are guaranteed to do so. Let the set $M_q$ (for "matching $q$") be this subset of the inputs in $S$. Because all and only the inputs in $S$ that are also in $M_q$ will pass through $q$, the process of making the $PTT$ onward will push $\texttt{lcp}(f(M_q))$ past $q$ towards the root such that exactly this $\texttt{lcp}$ will have been written when $q$ is entered after starting in $q_0$. Now recall that $f^p(w) = \texttt{lcp}(\{u \mid u = f(wy) \wedge y \in \Sigma^*\})$. It is sufficient to use a set containing $f(w)$ and at least one $f(wv) = f(wab)$ for each $a \in \Sigma$ (where $b \in \Sigma^*$) because every $v \in \Sigma^*$ is either $\lambda$ or begins with some $a \in \Sigma$. The set $M_q$ fulfills this criterion and so $\texttt{lcp}(f(M_q)) = f^p(q)$. □

The other crucial property of an onward PTT follows from the first: given a transition $(q, x, y, q') \in \delta$ such that $q$ is a supported state and $q'$ is either another supported state or $q_f$, the string $y$ is guaranteed to be equal to $\texttt{cont}_f(x, q)$ provided that $f$ is a subsequential function. The transitions meeting these criteria in Figure 2 are (s, ⋉, λ, $q_f$), (s, o, o, so) and (so, ⋉, λ, $q_f$).

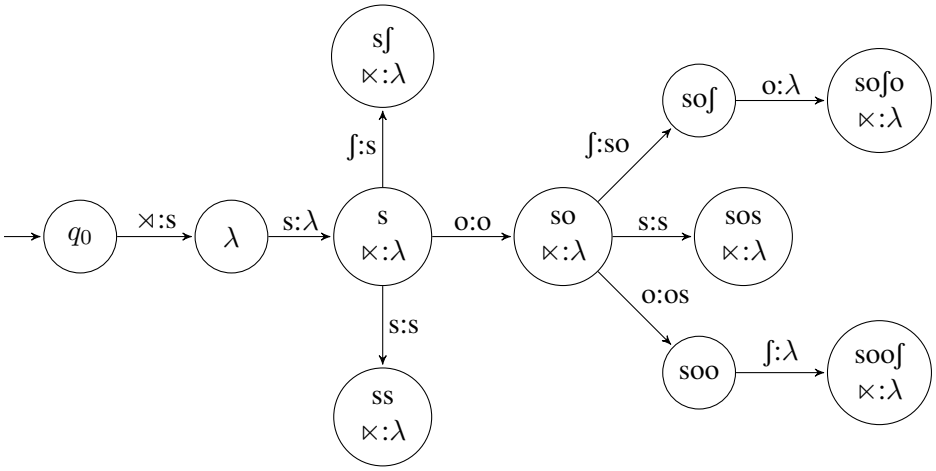


Figure 2: The onward version of Figure 1

**Remark 1.** *Let $P$ be the onward PTT constructed according to a sample drawn from function $f$. A corollary of Lemma 3 is that, given a supported $q$:*

- $(q, \ltimes, u, q_f)$ *is such that* $u = \texttt{cont}_f(\ltimes, q) = f^p(q)^{-1} \cdot f(q)$
- $(q, \sigma, v, r)$ *for* $\sigma \in \Sigma$ *is such that* $v = \texttt{cont}_f(\sigma, q) = f^p(q)^{-1} \cdot f^p(q\sigma)$ *if $r$ is also supported.*

The procedure `estimate_fp` shown in Algorithm 1 sends a sample $S$ once through its onward PTT and returns all pairs $(q, f^p(q))$ such that $q$ is a supported state. This set will be exploited along with the PTT when inducing the tier; the tier-learning algorithm below essentially treats this PTT and the constructed set as a sort of oracle that it can consult for information about $f^p$. We close this section by showing that extracting information about $f^p$ from a sample using Algorithm 1 takes quadratic time in the worst case.

**Lemma 4.** Quadratic time (`estimate_fp`)
*For any sample $S$ of input-output pairs, Algorithm 1 runs in $\mathcal{O}(|S|^2 \cdot |\Sigma|)$ time.*

*Proof.* Let $l = \sum_{(w,u)\in S} |w|$ be the summed lengths of all inputs in the sample, let $o = max\{|u| : (w, u) \in S\}$ be the longest output length in the sample, and let $s$ be the number of pairs in the sample. These magnitudes are all linear in the size of the sample.

Constructing $P = PTT(S)$ requires a single read through $S$, taking $l$ steps. Making this $P$ onward takes at most $ol$ steps (for details, see chapter 18 of de la Higuera, 2010). There are at most $l$ non-initial/non-final states in $P$ and `estimate_fp` starts by checking each of these once. For each, it verifies whether all possible $|\Sigma| + 1$ outgoing transitions exist. There are at most $l + s$ transitions in $P$, meaning that checking whether a transition exists takes at most $l + s$ steps. The first portion of `estimate_fp` thus takes at most $l((l+s)(|\Sigma|+1))$ steps. The second portion sends the sample through $P$ one input letter at a time, checking on each step whether the landing state is in $A$. Checking whether a state is in $A$ takes at most $l$ steps, so the second portion of `estimate_fp` takes at most $l^2$ steps. Taken together, the run time is in $\mathcal{O}(l+ol+l(l+s)|\Sigma|+l^2)$, which is quadratic in the size of $S$ and linear in the size of $\Sigma$. □

## 4 Identifying tier(s)

### 4.1 Overview of the process

The full tier induction process is shown in Algorithm 2. This algorithm adapts the overall strategy from Burness and McMullin (2019) so that it can be used with the PTT objects described above. In their original implementation of the strategy, the learner had to (1) sift through the sample for pairs meeting a certain criterion and (2) calculate contributions relative to each member of the collected subset. The latter step requires an additional scan through the sample for each pair acting as the basis of comparison, and like above, this nested reading of the sample creates a significant bottleneck which ultimately makes the process run in $\mathcal{O}(|S|^5)$ time.

The key insight in this paper is that certain transitions in an onwarded PTT will be equal to their corresponding contribution. Remark 1 tells us that these are easily identified by checking them against the set of pairs $(q, f^p(q))$ produced by the revised

**Data:** A sample $S$
**Result:** An onward PTT $P$ and the set $A$ containing the pair $(q, f^p(q))$ for each supported $q \in Q$
**Function** `estimate_fp(`$S$`)`**:**
  $P \leftarrow PTT(S) = (Q, q_0, q_f, \Sigma, \Delta, \delta)$;
  $P \leftarrow \texttt{onward}(P)$;
  $A \leftarrow \emptyset$;
  $B \leftarrow \emptyset$;
  **for** $q \in Q$ **do**
    **if** $\forall x \in \Sigma \cup \{\rtimes\}$, $\exists (q, x, y, q') \in \delta$ **then**
      $B \leftarrow B \cup \{q\}$
  **for** *each* $(x, y) \in S$ **do**
    $a \leftarrow \lambda$;
    $\mathtt{b} \leftarrow z$ such that $(q_0, \rtimes, z, \lambda) \in \delta$;
    **if** $\mathtt{a} \in B$ **then**
      $A \leftarrow A \cup \{(\mathtt{a}, \mathtt{b})\}$;
    **for** $n$ *from* 1 *to* $|x|$ **do**
      $\mathtt{r} \leftarrow$ the $n$-th letter of $x$;
      $\mathtt{w} \leftarrow u$ such that $(\mathtt{a}, \mathtt{r}, u, q) \in \delta$;
      $\mathtt{a} \leftarrow q$ such that $(\mathtt{a}, \mathtt{r}, \mathtt{w}, q) \in \delta$;
      $\mathtt{b} \leftarrow \mathtt{b} \cdot \mathtt{w}$;
      **if** $\mathtt{a} \in B$ **then**
        $A \leftarrow A \cup \{(\mathtt{a}, \mathtt{b})\}$;
  **return** $A, P$
**Algorithm 1:** Prefix function estimation

`estimate_fp`. Instead of calculating and recalculating contributions to find mismatches, then, we can simply cycle through the list of transitions in the onwarded PTT, sorting them into bins based on the current tier hypothesis. Doing so eliminates the problematic nesting and accordingly reduces the upper bound on runtime by three polynomial degrees to $\mathcal{O}(|S|^2)$.

This reduction in the degree of nesting is similar to how the ISL learning algorithm's quadratic time complexity relates to OSTIA's cubic time complexity. Both learning algorithms take an onward PTT and merge pairs of states until they terminate. Where $n$ is the number of states in the provided PTT, the number of possible merges performed by OSTIA is in $\mathcal{O}(n^2)$ since it can reject and undo merges (Oncina et al., 1993); in contrast, the number of possible merges performed by the ISL learning algorithm is in $\mathcal{O}(n)$ since it cannot reject and undo merges (Chandlee et al., 2014). For each state merging, both algorithms apply operations that run in $\mathcal{O}(|S|)$, and as a result, the overall complexity of OSTIA and the ISL learning algorithm are cubic and quadratic in the size of the sample, respectively.[1]

We start by hypothesizing that $T = \Delta$ (i.e., that all members of the output alphabet are on the tier). Then, for each transition $(q, x, y, r)$ in the onward PTT, the algorithm checks whether $q$ is a supported state (i.e., whether there is a pair in $A$ associated to it) and whether $r$ is a supported state or $q_f$. If both of these conditions hold, then the output edge $y$ of the transition is equal to $\texttt{cont}_f(x, q)$. Accordingly, the algorithm takes $(q, f^p(q)) \in A$, calculates $t = \texttt{suff}^1_T(f^p(q))$, and adds $y$ to the bin $C_{x,t}$ (the set of contributions for $x$ when the tier suffix is $t$) if it is not already there. If $t$ is on the function's canonical tier, the cardinality of this bin should always be equal to or less than 1 since the target function is $\text{OTSL}_2$ and so the contribution should be the same whenever the output tier suffix is equal to $t$.

After scanning through all transitions in the onward PTT, the algorithm looks for any constructed bins of contributions $C_{x,t}$ with cardinality greater than 1. If none of the bins associated with a specific $t \in T$ has cardinality greater than 1, the element $t$ will get added to the auxiliary set $K$ (for "keep") containing elements that are safe to keep on the tier for now. If any of the bins associated to $t \in T$ have cardinality greater than 1, the element $t$ is removed from the tier hypothesis since it cannot possibly be a member of the function's canonical tier. If at any point some symbol gets removed from $T$, the set $K$ is immediately emptied. The algorithm repeatedly alternates between scanning the PTT transitions and checking the cardinality of contribution sets until every $t$ in the current hypothesis for $T$ gets added to the set $K$, in which case it has found the canonical tier of the target function.

### 4.2 Proofs of correctness/efficiency

In this subsection, we establish that our revision of Burness and McMullin's (2019) algorithm achieves the same result in much less time.

**Lemma 5.** Quadratic time (`get_tier`)
*For any input sample $S$, $\texttt{get\_tier}(S)$ produces a tier $T$ in $\mathcal{O}(|S|^2 \cdot |\Sigma| \cdot |\Delta|^2)$ time.*

[1] The quadratic time complexity of the OSL learning algorithm, for its part, comes from repeatedly calculating the longest common prefix of stringsets lifted from the sample (Chandlee et al., 2015).

**Data:** A sample $S$
**Result:** A tier $T \subseteq \Delta$
**Function** `get_tier(`$S$`)`**:**
  $A, P \leftarrow$ `estimate_fp`$(S)$;
  $T \leftarrow \Delta$;
  $K \leftarrow \emptyset$;
  **while** $K \neq T$ **do**
    **for** *each* $t \in T$ **do**
      **for** *each* $x \in \Sigma \cup \{\ltimes\}$ **do**
        $C_{x,t} = \emptyset$;
    **for** *each* $(q, x, y, r) \in \delta$ *from* $P$ **do**
      **if** $[\exists(q, a) \in A] \wedge [[r = q_f] \vee [\exists(r, b) \in A]]$ **then**
        $t \leftarrow \mathtt{suff}^1_T(a)$;
        $C_{x,t} \leftarrow C_{x,t} \cup \{y\}$;
    **for** *each* $t \in T$ **do**
      **for** *each* $x \in \Sigma \cup \{\ltimes\}$ **do**
        **if** $|C_{x,t}| > 1$ **then**
          $T \leftarrow T - \{t\}$;
          $K \leftarrow \emptyset$;
      **if** $t \in T$ **then**
        $K \leftarrow K \cup \{t\}$
  **return** $T$

**Algorithm 2:** Single tier induction

*Proof.* Let $l = \sum_{(w,u)\in S} |w|$ be the summed lengths of all inputs in the sample, let $o = max\{|u| : (w, u) \in S\}$ be the longest output length in the sample, let $i = max\{|w| : (w, u) \in S\}$ be the longest input length in the sample, and let $s$ be the number of pairs in the sample. These are all linear in the size of the sample.

The first step is to run `estimate_fp` on the sample which Lemma 1 already established as running in $\mathcal{O}(|S|^2)$. Following that, the *while* loop can run up to $|\Delta|$ times. The first *for* loop initializes the contribution sets that will be constructed, of which there are at most $|\Delta| \cdot |\Sigma|$. Then, for each of the up to $l + s$ transitions in $P$, the second *for* loop it searches $A$ up to two times to check whether the origin is supported and whether the destination is supported or final. A single search of $A$ take at most $l$ steps, and if both conditions are met we calculate the relevant output tier suffix, taking at most $o$ steps. Finally, the third *for* loop inspects all the transitions in $P$, we check the cardinality of each contribution set, which takes at most $|\Delta| \cdot |\Sigma|$ steps. The overall run time of `get_tier`$(S)$ is thus in $\mathcal{O}(|\Delta|^2|\Sigma| + |\Delta|(l + s)(l + o))$, which is quadratic in the size of $S$, linear in the size of $\Sigma$ and quadratic in the size of $\Delta$. □

The remaining lemmata of this section will show that for each total $\text{OTSL}_2$ function $f$, there is a finite kernel of data consistent with $f$ that is a characteristic set for the algorithm (i.e., if the training set subsumes this kernel, the algorithm is guaranteed to succeed). The $\text{OTSL}_k$ functions divide $\Sigma^*$ into a finite number of equivalence classes according to sets of tails, meaning that the $\text{OTSL}_k$ functions are also subsequential functions. Oncina and Garcia (1991) show how the finite partition of $\Sigma^*$ lets us build the smallest finite-state transducer that computes a given subsequential function. Given a state $q$ in this canonical transducer $\mathcal{F}$, we write $w_q$ to denote the length-lexicographically earliest input string that reaches the state $q$, and define the characteristic set as follows. Note that this same characteristic set is used by Burness and McMullin (2021) for their multi-tier learner; they showed that its size is in $\mathcal{O}(|\mathcal{F}|^2)$.

**Definition 9.** Characteristic set
*A sample $S$ contains a characteristic set iff it contains the following for each state $q$ in $\mathcal{F}$:*

1. *The input-output pair* $(w_q, f(w_q))$.
2. *For all triples* $a, b, c \in \Sigma$*:*
   i. *some pair* $(w_q a, f(w_q a))$,
   ii. *some pair* $(w_q ab, f(w_q ab))$, *and*
   iii. *some pair* $(w_q abcv, f(w_q abcv))$, *where* $v \in \Sigma^*$

**Lemma 6.** Quadratic data
*There exists a characteristic set whose size is in* $\mathcal{O}(|\mathcal{F}|^2)$.

*Proof.* See the proof of Lemma 17 in Burness and McMullin (2021). □

**Lemma 7.** Evidence availability
*If a learning sample $S$ contains a characteristic set and $P$ is the onward PTT for $S$ then for all $w \in \Sigma^*$ and all pairs $x, y \in \Sigma$ there is at least one transition in $P$ corresponding to each of the following that (1) leaves a supported state and (2) ends in $q_f$ or a supported state:*

- $\mathtt{cont}_f(x, w)$
- $\mathtt{cont}_f(\ltimes, w)$
- $\mathtt{cont}_f(y, wx)$
- $\mathtt{cont}_f(\ltimes, wx)$

*Proof.* For any input string $w \in \Sigma^*$, reading $w$ will lead to some non-initial and non-final state $q$ in $\mathcal{F}$. The target function is subsequential which means that that either $w = w_q$ or else can be replaced thereby since subsequentiality implies that $\texttt{cont}_f(i, w) = \texttt{cont}_f(i, w_q)$ for any $i \in \Sigma \cup \{\ltimes\}$. By the definition of the seed, for every state $q$ in $\mathcal{F}$ and for every triple $x, y, z \in \Sigma$, the learner will see $w_q$, $w_q x$, $w_q xy$, and $w_q xyzv$ where $v \in \Sigma^*$. This means that the states $w_q$, $w_q x$, and $w_q xy$ in $P$ are all supported. As Remark 1 notes, Lemma 3 then implies that:

- $(w_q, x, a_1, w_q x)$ in $P$ is such that $a_1 = f^p(w_q)^{-1} \cdot f^p(w_q x) = \texttt{cont}_f(x, w_q)$
- $(w_q, \ltimes, a_2, q_f)$ in $P$ is such that $a_2 = f^p(w_q)^{-1} \cdot f(w_q) = \texttt{cont}_f(\ltimes, w_q)$
- $(w_q x, y, a_3, w_q xy)$ in $P$ is such that $a_3 = f^p(w_q x)^{-1} \cdot f^p(w_q xy) = \texttt{cont}_f(y, w_q x)$
- $(w_q x, \ltimes, a_4, q_f)$ in $P$ is such that $a_2 = f^p(w_q x)^{-1} \cdot f(w_q x) = \texttt{cont}_f(\ltimes, w_q x)$

□

**Lemma 8.** Tier convergence
*Given a learning sample $S$ that contains a characteristic sample, $\texttt{get\_tier}(S)$ will produce the canonical tier of $f$.*

*Proof.* Let $T$ be the canonical tier of $f$, and let $H$ be the tier constructed by the algorithm. The algorithm begins with $H = \Delta$, and so either $H = T$ already, or else $H \supset T$. The algorithm is designed to consider all and only the transitions $(q, x, y, r)$ in $P = \texttt{onward}(PTT(S))$ such that $q$ is a supported state and $r$ is a supported state or $q_f$. As Remark 1 notes, Lemma 3 implies that $y = \texttt{cont}_f(x, q)$ for all such transitions. For each considered transition $(q, x, y, r)$ in $P$, the algorithm sorts $y$ into the bin associated simultaneously with $x$ and with $z = \texttt{suff}^1_H(p)$, where $p$ is equal to the output produced upon reading $q$ in $P$. Note that the algorithm has easy access to the string $p$ because it is paired with $q$ in the auxiliary set $A$. Note also that since $q$ is a supported state, Lemma 3 tells us that $p = f^p(q)$.

Now, we know from Lemma 2 that if $H \supset T$, there will exist a pair of input strings $w_1$ and $w_2$ in the domain of $f$ such that $\texttt{cont}_f(x, w_1) \neq \texttt{cont}_f(x, w_2)$ even though $\texttt{suff}^1_H(f^p(w_1)) = \texttt{suff}^1_H(f^p(w_2)) = a$ for some $a \in (H - T)$ and some $x \in \Sigma \cup \{\ltimes\}$. Furthermore, Lemma 7 tells us that for all non-initial/non-final states $s$ in the minimal FST $\mathcal{F}$ producing $f$, each transition along every possible sequence of two or fewer steps out of $s$ will have at least one equivalent transition in $P$ that is considered by the algorithm. If the first transition along one of these paths produces any elements not in $T$, we stand the chance of incorrectly binning the second transition when $H \supset T$. Since we see all possible paths of two transitions, at least one pair of unequal contributions (which *should* be placed into two different bins linked to two different members of $T$, since $f$ is OTSL$_2$) will be placed together into a bin that should not exist (because that bin is linked to a non-member of $T$) when $H \supset T$.

Accordingly, at least one of the bins associated with some $b \in (H - T)$ will have a cardinality greater than 1 (assuming repeated strings are counted only once) when $H \supset T$. The algorithm will thus flag and remove at least one $b \in (H - T)$ when $H \supset T$. Conversely, there will be no pair of input strings $w_3$ and $w_4$ in the domain of $f$ such that $\texttt{cont}_f(x, w_3) \neq \texttt{cont}_f(x, w_4)$ when $\texttt{suff}^1_H(f^p(w_3)) = \texttt{suff}^1_H(f^p(w_4)) = c$ for any $c \in T$ and any $x \in \Sigma \cup \{\ltimes\}$. Consequently, none of the bins associated with any $c \in T$ will ever surpass a cardinality of 1 (assuming repeated strings are counted only once). When $H = T$, then, the algorithm will add all $d \in H$ to $K$, at which point $K = H = T$. □

**Theorem 1.** *$\texttt{get\_tier}$ identifies the canonical tier of any total OTSL$_2$ function in $\mathcal{O}(|S|^2)$ time and $\mathcal{O}(|\mathcal{F}|^2)$ data.*

*Proof.* Immediate from Lemmata 5, 6, and 8. □

The sample and the tier returned by $\texttt{get\_tier}(S)$ can then be fed to the transducer building algorithm from Burness and McMullin (2019), which is a generalization of the transducer building algorithm from Chandlee et al. (2015) and whose worst-case runtime is also in $\mathcal{O}(|S|^2)$. Since all three components of the tier-based function learning pipeline now have a quadratic upper bound on runtime, the overall process from start to finish now also has a quadratic upper bound.

## 5 Discussion and conclusion

SL functions, aside from closely approximating the typology of local processes (Chandlee and Heinz,

2018), are highly useful from a learnability standpoint. With their quadratic upper bounds on runtime, it can be preferable to use the SL function learning algorithms (Chandlee et al., 2014, 2015) over the Onward Subsequential Transducer Inference Algorithm (OSTIA) of Oncina et al. (1993). While OSTIA can learn all the same functions as the SL function learners and more (since the subsequential functions properly contain the SL functions), its run time is cubic in the worst case. Sacrificing expressiveness, in this case, is offset by a gain in efficiency, making it worthwhile in appropriate circumstances.

Prior to this paper, the same tradeoff was only true for TSL and MTSL functions when the learner already knew the necessary tier(s). Such advance knowledge permits basic generalizations of the SL learning algorithms that preserve their complexity bounds (Burness and McMullin, 2019). Of course, it is not realistic for a learner to come equipped with foreknowledge of the relevant tier(s), so a focus of research on tier-based functions has been whether and how tiers can be identified from positive examples drawn from the target function. Initial methods from this enterprise were limited in that they (i) only work for functions with a window length (the parameter $k$) of 2 and (ii) are less efficient than OSTIA by two polynomial degrees, with a quintic worst-case runtime. Accordingly, the fact that some tiers could be learned from positive data was effectively a technical curiosity from the perspective of learning performance.

Practically speaking, a learner was better off attempting to build a subsequential function than a TSL or MTSL function when an SL function was not sufficient. Our contribution here was to show that the inefficiencies of tier learning could be overcome by manipulating a Prefix Tree Transducer (a data structure also used by OTSIA and the ISL learning algorithm) rather than just manipulating the sample. Doing so circumvents the need for nested reading of the sample, which we identified as the major bottleneck of previous methods. Our revision of the methods from Burness and McMullin (2019, 2021) reduces their upper bound on runtime by three polynomial degrees. As was the case for the SL functions, then, the sacrifice in expressiveness from eschewing a subsequential function in favour of a $\text{TSL}_2$ function (or a *strongly-target specified* $\text{MTSL}_2$ function; Burness and McMullin, 2021) is offset by an appreciable gain in efficiency.

While we have focused mainly on concerns of practicality in this paper, we do note that there are also conceptual grounds for using TSL and MTSL functions over subsequential functions as models of long-distance phonological process. It is well-established that subsequential computation is sufficiently expressive to model non-local vowel harmony (Heinz and Lai, 2013), consonant harmony (Luo, 2017), and consonant dissimilation (Payne, 2017) with a handful of exceptions in the form of *unbounded circumambience* (Jardine, 2016; McCollum et al., 2020). That being said, the relativized locality underpinning tier-based functions has been shown to more intuitively capture attested long-distance behaviours (Andersson et al., 2020; Burness et al., 2021), while excluding some pathological behaviours like modulo counting which are otherwise amenable to a subsequential analysis (Burness et al., 2021). Combining this work with the learnability results in the current paper solidifies the appropriateness of TSL and MTSL functions as models of long-distance phonological processes.

Several hurdles, however, still remain to be overcome in the area of tier-based function learning. First and foremost, the results herein require a window size ($k$) of 2; the properties exploited by the learner do not hold for larger window sizes. This is in stark contrast to tier-based *languages*, whose tiers are efficiently learnable for arbitrary window sizes (Jardine and McMullin, 2017; Lambert, 2021). Second, the learner developed above is a batch learner (as are OSTIA and the SL learners), making it unlikely as a model of real human phonological learning. In this regard as well, the existing work on languages outpaces the work on functions, since an online learner was recently developed for TSL languages (Lambert, 2021). Finally, the way in which we manipulate the PTT during tier learning assumes that the function is total. To learn partial functions, it may be necessary to provide the learner with some additional information, like how Oncina and Varó (1996) and Castellanos et al. (1998) augment OSTIA by giving it access to domain and range information, respectively.

## References

Samuel Andersson, Hossep Dolatian, and Yiding Hao. 2020. Computing vowel harmony: The generative

capacity of search and copy. In *Proceedings of the 2019 Annual Meeting on Phonology*.

Phillip Burness and Kevin McMullin. 2019. Efficient learning of Output Tier-Based Strictly 2-Local functions. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 78–90. Association for Computational Linguistics.

Phillip Burness and Kevin McMullin. 2021. Learning multiple independent tier-based processes. In *Proceedings of the Fifteenth International Conference on Grammatical Inference*, volume 153 of *Proceedings of Machine Learning Research*, pages 66–80. PMLR.

Phillip Burness, Kevin McMullin, and Jane Chandlee. 2021. Long-distance phonological processes as tier-based strictly local functions. *Glossa*, 6.

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

Jane Chandlee. 2014. *Strictly Local Phonological Processes*. Doctoral dissertation, University of Delaware.

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

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2015. Output Strictly Local functions. In *Proceedings of the 14th Meeting on the Mathematics of Language (MOL 2015)*, pages 112–125.

Jane Chandlee and Jeffrey Heinz. 2018. Strict Locality and phonological maps. *Linguistic Inquiry*, 49:23–60.

Noam Chomsky. 1956. Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124.

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

Pedro Garcia, Enrique Vidal, and José Oncina. 1990. Learning Locally Testable languages in the strict sense. In *Proceedings of the Workshop on Algorithmic Learning Theory*, pages 325–338. Japanese Society for Artificial Intelligence.

E. Mark Gold. 1967. Language identification in the limit. *Information and Control*, 10:447–474.

Yiding Hao and Samuel Andersson. 2019. Unbounded stress in subregular phonology. In *Proceedings of the 16th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology and Morphology*, pages 135–143, Florence, Italy. Association for Computational Linguistics.

Yiding Hao and Dustin Bowers. 2019. Action-sensitive phonological dependencies. In *Proceedings of the 16th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology and Morphology*, pages 218–228, Florence, Italy. Association for Computational Linguistics.

Jeffrey Heinz and Regine Lai. 2013. Vowel harmony and subsequentiality. In *Proceedings of the 13th Meeting on the Mathematics of Language (MOL 13)*, pages 52–63, Sofia, Bulgaria. Association for Computational Linguistics.

Jeffrey Heinz, Chetan Rawal, and Herbert G. Tanner. 2011. Tier-based strictly local constraints for phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 58–64, Portland, OR. Association for Computational Linguistics.

Adam Jardine. 2016. Computationally, tone is different. *Phonology*, 33:247–283.

Adam Jardine and Kevin McMullin. 2017. Efficient learning of Tier-Based Strictly k-Local languages. In *International Conference on Language and Automata Theory and Applications (LATA 2017)*, pages 64–76.

C. Douglas Johnson. 1972. *Formal Aspects of Phonological Description*. Mouton, The Hague.

Ronald M. Kaplan and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20:331–378.

Dakotah Lambert. 2021. Grammar interpretations and learning TSL online. In *Proceedings of the Fifteenth International Conference on Grammatical Inference*, volume 153 of *Proceedings of Machine Learning Research*, pages 81–91. PMLR.

Dakotah Lambert and James Rogers. 2020. Tier-Based Strictly Local stringsets: Perspectives from model and automata theory. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2020*, pages 330–337, New Orleans, Louisianna.

Huan Luo. 2017. Long-distance consonant agreement and subsequentiality. *Glossa: A Journal of General Linguistics*, 2:1–25.

Adam G. McCollum, Eric Baković, Anna Mai, and Eric Meinhardt. 2020. Unbounded circumambient patterns in segmental phonology. *Phonology*, 37:215–255.

Kevin McMullin and Gunnar Ólafur Hansson. 2016. Long-distance phonotactics as Tier-Based Strictly 2-Local Languages. In *Proceedings of the 2014 Annual Meeting on Phonology*, Washington, DC. Linguistic Society of America.

José Oncina and Pedro Garcia. 1991. Inductive learning of subsequential functions. Technical Report DSIC II-34, University Politecnia de Valencia.

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

José Oncina and Miguel A. Varó. 1996. Using domain information during the learning of a subsequential transducer. In Laurent Miclet and Colin de la Higuera, editors, *Grammatical Interference: Learning Syntax from Sentences*, number 1147 in Lecture Notes in Artificial Intelligence, pages 301–312. Springer, Berlin.

Amanda Payne. 2017. All dissimilation is computationally subsequential. *Language*, 93:353–371.

James Rogers, Jeffrey Heinz, Margaret Fero, Jeremy Hurst, Dakotah Lambert, and Sean Wibel. 2013. Cognitive and sub-regular complexity. In *Formal Grammar*, number 8036 in Lecture Notes in Artificial Intelligence, pages 90–108. Springer.

James Rogers and Geoffrey K. Pullum. 2011. Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information*, 20:329–342.