Efficient Inference Through Cascades of Weighted Tree Transducers

Jonathan May and Kevin Knight
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292
{jonmay,knight}@isi.edu

Heiko Vogler
Technische Universität Dresden
Institut für Theoretische Informatik
01062 Dresden, Germany
heiko.vogler@tu-dresden.de

Abstract

Weighted tree transducers have been proposed as useful formal models for representing syntactic natural language processing applications, but there has been little description of inference algorithms for these automata beyond formal foundations. We give a detailed description of algorithms for application of cascades of weighted tree transducers to weighted tree acceptors, connecting formal theory with actual practice. Additionally, we present novel on-the-fly variants of these algorithms, and compare their performance on a syntax machine translation cascade based on (Yamada and Knight, 2001).

1 Motivation

Weighted finite-state transducers have found recent favor as models of natural language (Mohri, 1997). In order to make actual use of systems built with these formalisms we must first calculate the set of possible weighted outputs allowed by the transducer given some input, which we call forward application, or the set of possible weighted inputs given some output, which we call backward application. After application we can do some inference on this result, such as determining its \( k \) highest weighted elements.

We may also want to divide up our problems into manageable chunks, each represented by a transducer. As noted by Woods (1980), it is easier for designers to write several small transducers where each performs a simple transformation, rather than painstakingly construct a single complicated device. We would like to know, then, the result of transformation of input or output by a cascade of transducers, one operating after the other. As we will see, there are various strategies for approaching this problem. We will consider offline composition, bucket brigade application, and on-the-fly application.

Application of cascades of weighted string transducers (WSTs) has been well-studied (Mohri, 1997). Less well-studied but of more recent interest is application of cascades of weighted tree transducers (WTTs). We tackle application of WTT cascades in this work, presenting:

- explicit algorithms for application of WTT cascades
- novel algorithms for on-the-fly application of WTT cascades, and
- experiments comparing the performance of these algorithms.

2 Strategies for the string case

Before we discuss application of WTTs, it is helpful to recall the solution to this problem in the WST domain. We recall previous formal presentations of WSTs (Mohri, 1997) and note informally that they may be represented as directed graphs with designated start and end states and edges labeled with input symbols, output symbols, and weights.\(^1\) Fortunately, the solution for WSTs is practically trivial—we achieve application through a series of embedding, composition, and projection operations. Embedding is simply the act of representing a string or regular string language as an identity WST. Composition of WSTs, that is, generating a single WST that captures the transformations of two input WSTs used in sequence, is not at all trivial, but has been well covered in, e.g., (Mohri, 2009), where directly implementable algorithms can be found. Finally, projection is another trivial operation—the domain or range language can be obtained from a WST by ignoring the output or input symbols, respectively, on its arcs, and summing weights on otherwise identical arcs. By embedding an input, composing the result with the given WST, and projecting the result, forward application is accomplished.\(^2\) We are then left with a weighted string acceptor (WSA), essentially a weighted, labeled graph, which can be traversed

\(^1\)We assume throughout this paper that weights are in \( \mathbb{R}_+ \cup \{+\infty\} \), that the weight of a path is calculated as the product of the weights of its edges, and that the weight of a (not necessarily finite) set \( T \) of paths is calculated as the sum of the weights of the paths of \( T \).

\(^2\)For backward applications, the roles of input and output are simply exchanged.
3 Application of tree transducers

Now let us revisit these strategies in the setting of trees and tree transducers. Imagine we have a tree or set of trees as input that can be represented as a weighted regular tree grammar\(^3\) (WRTG) and a WTT that can transform that input with some weight. We would like to know the k-best trees the WTT can produce as output for that input, along with their weights. We already know of several methods for acquiring k-best trees from a WRTG (Huang and Chiang, 2005; Pauls and Klein, 2009), so we then must ask if, analogously to the string case, WTTs preserve recognizability\(^4\) and we can form an application WRTG. Before we begin, however, we must define WTTS and WRTGs.

3.1 Preliminaries\(^5\)

A ranked alphabet is a finite set \(\Sigma\) such that every member \(\sigma \in \Sigma\) has a rank \(rk(\sigma) \in \mathbb{N}\). We call \(\Sigma^{(k)} \subset \Sigma, k \in \mathbb{N}\) the set of those \(\sigma \in \Sigma\) such that \(rk(\sigma) = k\). The set of variables is denoted \(X = \{x_1, x_2, \ldots\}\) and is assumed to be disjoint from any ranked alphabet used in this paper. We use \(\bot\) to denote a symbol of rank 0 that is not in any ranked alphabet used in this paper. A tree \(t \in T_{\Sigma}\) is denoted \(\sigma(t_1, \ldots, t_k)\) where \(k \geq 0, \sigma \in \Sigma^{(k)}\), and \(t_1, \ldots, t_k \in T_{\Sigma}\). For \(\sigma \in \Sigma^{(0)}\) we

\(^3\)This generates the same class of weighted tree languages as weighted tree automata, the direct analogue of WSSAs, and is more useful for our purposes.

\(^4\)A weighted tree language is recognizable iff it can be represented by a wrtg.

\(^5\)The following formal definitions and notations are needed for understanding and reimplementation of the presented algorithms, but can be safely skipped on first reading and consulted when encountering an unfamiliar term.
write $\sigma \in T_\Sigma$ as shorthand for $\sigma()$. For every set $S$ disjoint from $\Sigma$, let $T_\Sigma(S) = T_{\Sigma \cup S}$, where, for all $s \in S$, $rk(s) = 0$.

We define the positions of a tree $t = (t_1, \ldots, t_k)$, for $k \geq 0$, $\sigma \in \Sigma(k)$, $t_1, \ldots, t_k \in T_\Sigma$, as a set $pos(t) \subseteq \mathbb{N}^*$ such that $pos(t) = \{i\} \cup \{iv | 1 \leq i \leq k, v \in pos(t_i)\}$. The set of leaf positions $lv(t) \subseteq pos(t)$ are those positions $v \in pos(t)$ such that for no $i \in \mathbb{N}$, $vi \in pos(t)$. We presume standard lexicographic orderings $\leq$ and $\leq pos$.

Let $t, s \in T_\Sigma$ and $v \in pos(t)$. The label of $t$ at position $v$, denoted by $t(v)$, the subtree of $t$ at $v$, denoted by $t|_v$, and the replacement at $v$ by $s$, denoted by $t[s|_v]$, are defined as follows:

1. For every $\sigma \in \Sigma(0)$, $\sigma(e) = \sigma$, $\sigma|_e = \sigma$, and $\sigma[s|_e] = s$.
2. For every $t = (t_1, \ldots, t_k)$ such that $k = rk(\sigma)$ and $k \geq 1$, $t(\epsilon) = \sigma$, $t|_e = t$, and $t[s|_e] = s$. For every $1 \leq i \leq k$ and $v \in pos(t_i)$, $t(iv) = t_i(v)$, $t[i|_v] = t|_i[v]$, and $t[s|_v] = \sigma(t_1, \ldots, t_{i-1}, t_i|_v, t_{i+1}, \ldots, t_k)$.

The size of a tree $t$, size($t$) = $|pos(t)|$, the cardinality of its position set. The yield set of a tree is the set of labels of its leaves: for a tree $t$, $yd(t) = \{t(v) | v \in lv(t)\}$.

Let $A$ and $B$ be sets. Let $\varphi : A \rightarrow T_\Sigma(B)$ be a mapping. We extend $\varphi$ to the mapping $\overline{\varphi} : T_\Sigma(A) \rightarrow T_\Sigma(B)$ such that for $a \in A$, $\overline{\varphi}(a) = \varphi(a)$ and for $k \geq 0$, $\sigma \in \Sigma(k)$, and $t_1, \ldots, t_k \in T_\Sigma(A)$, $\overline{\varphi}(\sigma(t_1, \ldots, t_k)) = \sigma(\overline{\varphi}(t_1), \ldots, \overline{\varphi}(t_k))$. We indicate such extensions by describing $\varphi$ as a substitution mapping and then using $\overline{\varphi}$ without further comment.

We use $\mathbb{R}_+$ to denote the set $\{w \in \mathbb{R} | w \geq 0\}$ and $\mathbb{R}_+^\infty$ to denote $\mathbb{R}_+ \cup \{+\infty\}$.

**Definition 3.1** (cf. [Alexandrakis and Bozapalidis, 1987]) A weighted regular tree grammar (WRTG) is a 4-tuple $G = (N, \Sigma, P, n_0)$ where:

1. $N$ is a finite set of nonterminals, with $n_0 \in N$ the start nonterminal.
2. $\Sigma$ is a ranked alphabet of input symbols, where $\Sigma \cap N = \emptyset$.
3. $P$ is a tuple $(P', \pi)$, where $P'$ is a finite set of productions, each production $p$ of the form $n \rightarrow u$, $n \in N$, $u \in T_\Sigma(N)$, and $\pi : P' \rightarrow \mathbb{R}_+$ is a weight function of the productions. We will refer to $P$ as a finite set of weighted productions, each production $p$ of the form $n \rightarrow u$.

A production $p$ is a chain production if it is of the form $n_i \rightarrow n_j$, where $n_i, n_j \in N$. A WRTG $G$ is in normal form if each production is either a chain production or is of the form $n \xrightarrow{\pi} \sigma(n_1, \ldots, n_k)$ where $\sigma \in \Sigma(k)$ and $n_1, \ldots, n_k \in N$.

For WRTG $G = (N, \Sigma, P, n_0)$, $s, t, u \in T_\Sigma(N)$, $n \in N$, and $p \in P$ of the form $n \xrightarrow{\pi} u$, we obtain a derivation step from $s$ to $t$ by replacing some leaf nonterminal in $s$ labeled with $n$ with $u$. Formally, $s \Rightarrow_p t$ if there exists some $v \in lv(s)$ such that $s(v) = n$ and $\pi[|_v] = t$. We say this derivation step is leftmost if, for all $v' \in lv(s)$ where $v' < v, s(v') \in \Sigma$. We henceforth assume all derivation steps are leftmost. If, for some $m \in \mathbb{N}$, $p_1 \in P$, and $t_i \in T_\Sigma(N)$ for all $1 \leq i \leq m$, $n_0 \Rightarrow p_1 t_1 \Rightarrow p_m t_m$, we say the sequence $d = (p_1, \ldots, p_m)$ is a derivation of $t_m$ in $G$ and that $n_0 \Rightarrow^* t_m$; the weight of $d$ is $\pi(d) = \pi(p_1) \cdots \pi(p_m)$. The weighted tree language recognized by $G$ is the mapping $L_G : T_\Sigma \rightarrow \mathbb{R}_+^\infty$ such that for every $t \in T_\Sigma$, $L_G(t)$ is the sum of the weights of all (possibly infinitely many) derivations of $t$ in $G$. A weighted tree language $f : T_\Sigma \rightarrow \mathbb{R}_+^\infty$ is recognizable if there is a WRTG $G$ such that $f = L_G$.

We define a partial ordering $\preceq$ on WRTGs such that for WRTGs $G_1 = (N_1, \Sigma, P_1, n_0)$ and $G_2 = (N_2, \Sigma, P_2, n_0)$, we say $G_1 \preceq G_2$ iff $N_1 \subseteq N_2$ and $P_1 \subseteq P_2$, where the weights are preserved.

**Definition 3.2** (cf. Def. 1 of [Maletti, 2008]) A weighted extended top-down tree transducer (wXTT) is a 5-tuple $M = (Q, \Sigma, \Delta, R, q_0)$ where:

1. $Q$ is a finite set of states.
2. $\Sigma$ and $\Delta$ are the ranked alphabets of input and output symbols, respectively, where $(\Sigma \cup \Delta) \cap Q = \emptyset$.
3. $R$ is a tuple $(R', \pi)$, where $R'$ is a finite set of rules, each rule $r$ of the form $q.y \rightarrow u$ for $q \in Q$, $y \in T_\Sigma(X)$, and $u \in T_\Delta(Q \times X)$. We further require that no variable $x \in X$ appears more than once in $y$, and that each variable appearing in $u$ is also in $y$. Moreover, $\pi : R' \rightarrow \mathbb{R}_+^\infty$ is a weight function of the rules. As for WRTGs, we refer to $R$ as a finite set of weighted rules, each rule $r$ of the form $q.y \xrightarrow{\pi(r)} u$.

A wXTT is linear (respectively, nondeleting) if, for each rule $r$ of the form $q.y \xrightarrow{\pi} u$, each $x \in yd(y) \cap X$ appears at most once (respectively, at least once) in $u$. We denote the class of all wXTT's as wXT and add the letters L and N to signify the subclasses of linear and nondeleting WTT, respectively. Additionally, if $y$ is of the form $\sigma(x_1, \ldots, x_k)$, we remove the letter “x” to signify.

---

6In [Alexandrakis and Bozapalidis, 1987], chain productions are forbidden in order to avoid infinite summations. We explicitly allow such summations.
the transducer is not extended (i.e., it is a “traditional” WTT [Fülöp and Vogler, 2009]).

For WXTT \( M = (Q, \Sigma, \Delta, R, q_0) \), \( s, t \in T_\Delta(\Sigma \times T_\Delta) \), and \( r \in R \) of the form \( q.y \xrightarrow{w} u \), we obtain a derivation step from \( s \) to \( t \) by replacing some leaf of \( s \) labeled with \( q \) and a tree matching \( y \) by a transformation of \( u \), where each instance of a variable has been replaced by a corresponding subtree of the \( y \)-matching tree. Formally, \( s \xrightarrow{M} t \) if there is a position \( v \in pos(s) \), a substitution mapping \( \varphi : X \to T_\Sigma \), and a rule \( q.y \xrightarrow{w} u \in R \) such that \( s(v) = (q, \varphi(y)) \) and \( t = s[\varphi(u)] \), where \( \varphi' \) is a substitution mapping \( Q \times X \to T_\Delta(Q \times T_\Sigma) \) defined such that \( \varphi'(q', x) = (q', \varphi(x)) \) for all \( q' \in Q \) and \( x \in X \). We say this derivation step is leftmost if, for all \( v' \in \hat{v}(s) \) where \( v' < v \), \( s(v') \in \Delta \). We henceforth assume all derivation steps are leftmost. If, for some \( s \in T_\Sigma, m \in \mathbb{N}, r_i \in R, \) and \( t_i \in T_\Delta(\Sigma \times T_\Sigma) \) for all \( 1 \leq i \leq m \), \( (q_0, s) \xrightarrow{r_1} t_1 \cdots \xrightarrow{r_m} t_m \), we say the sequence \( d = (r_1, \ldots, r_m) \) is a derivation of \((s, t_m)\) in \( M \); the weight of \( d \) is \( w(d) = \pi(r_1) \cdot \cdots \cdot \pi(r_m) \).

The weighted tree transformation recognized by \( M \) is the mapping \( \tau_M : T_\Sigma \times T_\Delta \to \mathbb{R}_+^\infty \), such that for every \( s \in T_\Sigma \) and \( t \in T_\Delta \), \( \tau_M(s, t) \) is the sum of the weights of all (possibly infinitely many) derivations of \((s, t)\) in \( M \). The composition of two weighted tree transformations \( \tau : T_\Sigma \times T_\Delta \to \mathbb{R}_+^\infty \) and \( \mu : T_\Delta \times T_\Gamma \to \mathbb{R}_+^\infty \) is the weighted tree transformation \( (\tau; \mu) : T_\Sigma \times T_\Gamma \to \mathbb{R}_+^\infty \), where for every \( s \in T_\Sigma \) and \( u \in T_\Gamma \), \( (\tau; \mu)(s, u) = \sum_{t \in T_\Delta} \tau(s, t) \cdot \mu(t, u) \).

### 3.2 Applicable classes

We now consider transducer classes where recognizability is preserved under application. Table 1 presents known results for the top-down tree transducer classes described in Section 3.1. Unlike the string case, preservation of recognizability is not universal or symmetric. This is important for us, because we can only construct an application WRTG, i.e., a WRTG representing the result of application, if we can ensure that the language generated by application is in fact recognizable. Of the types under consideration, only wxLNT and WLNT preserve forward recognizability. The two classes marked as open questions and the other classes, which are superclasses of wNT, do not or are presumed not to. All subclasses of wxLT preserve backward recognizability.\(^7\) We do not consider cases where recognizability is not preserved in the remainder of this paper. If a transducer \( M \) of a class that preserves forward recognizability is applied to a WRTG \( G \), we can call the forward application \( wrtG M(G)^6 \) and if \( M \) preserves backward recognizability, we can call the backward application \( wrtG M(G)^6 \).

Now that we have explained the application problem in the context of weighted tree transducers and determined the classes for which application is possible, let us consider how to build forward and backward application WRTGs. Our basic approach mimics that taken for WSTs by using an embed-compose-project strategy. As in string world, if we can embed the input in a transducer, compose with the given transducer, and project the result, we can obtain the application WRTG. Embedding a WRTG in a WLNT is a trivial operation—if the WRTG is in normal form and chain production-free,\(^8\) for every production of the form \( n \xrightarrow{w} \sigma(n_1, \ldots, n_k) \), create a rule of the form \( n.\sigma(x_1, \ldots, x_k) \xrightarrow{w} \sigma(n_1.x_1, \ldots, n_k.x_k) \). Range projection of a wxLNT is also trivial—for every \( q \in Q \) and \( u \in T_\Delta(\Sigma \times X) \), create a production of the form \( q.y \xrightarrow{w} u \) by replacing all leaves of the form \( q.x \) with the leaf \( q \), i.e., removing references to variables, and \( w \) is the sum of the weights of all rules of the form \( q.y \xrightarrow{w} u \in R \).\(^9\) Domain projection for wxLT is best explained by way of example. The left side of a rule is preserved, with variables leaves replaced by their associated states from the right side. So, the rule \( q_1.\sigma(\gamma(x_1), x_2) \xrightarrow{w} \delta(q_2.x_2, \beta(\alpha, q_3,x_1)) \) would yield the production \( q_1 \xrightarrow{w} \sigma(\gamma(q_3), q_2) \) in the domain projection. However, a deleting rule such as \( q_1.\sigma(x_1, x_2) \xrightarrow{w} \gamma(q_2,x_2) \) necessitates the introduction of a new nonterminal \( \bot \) that can generate all of \( T_\Sigma \) with weight 1.

The only missing piece in our embed-compose-project strategy is composition. Algorithm 1, which is based on the declarative construction of Maletti (2006), generates the syntactic composition of a wxLT and a wLNT, a generalization of the basic composition construction of Baker (1979). It calls Algorithm 2, which determines the sequences of rules in the second transducer that match the right side of a single rule in the first transducer. Since the embedded WRTG is of type wNT, it may be either the first or second argument provided to Algorithm 1, depending on whether the application is forward or backward. We can thus use the embed-compose-project strategy for forward application of wLNT and backward application of wxLT and wxLNT. Note that we cannot use this strategy for forward application.

---

\(^6\)Without loss of generality we assume this is so, since standard algorithms exist to remove chain productions [Kuich, 1998; Ésik and Kuich, 2003; Mohri, 2009] and convert into normal form [Alexandrakis and Bzapalidis, 1987].

\(^7\)Finitely many such productions may be formed.
tion of wxLNT, even though that class preserves recognizability.

Algorithm 1 COMPOSE

1: inputs
2: wxLT $M_1 = (Q_1, \Sigma, \Delta, R_1, q_{10})$
3: wLNT $M_2 = (Q_2, \Delta, \Gamma, R_2, q_{20})$
4: outputs
5: wxLT $M_3 = ((Q_1 \times Q_2), \Sigma, \Delta, R_3, (q_{10}, q_{20}))$ such that $M_3 = (\tau M_1, \tau M_2)$.
6: complexity
7: $O(|R_1| \max(|R_2|, |Q_2|))$, where $\bar{u}$ is the largest right side tree in any rule in $R_i$

8: Let $R_3$ be of the form $(R_3, \pi)$
9: $R_1 \leftarrow \emptyset$
10: $\Xi \leftarrow \{(q_{10}, q_{20})\}$ \{seen states\}
11: $\Psi \leftarrow \{(q_{10}, q_{20})\}$ \{pending states\}
12: while $\Psi \neq \emptyset$ do
13: \{any element of $\Psi$\}
14: for all $(q_1, q_2) \in R_3$ do
16: for all $(z, w_2) \in COVER(u, M_2, q_2)$ do
17: for all $(q, x) \in \gamma(q, z)$ \{$(Q_1 \times Q_2) \times X$\} do
18: if $q_1 \not\in \Xi$ then
20: $\Psi \leftarrow \Psi \cup \{q\}$
21: $r \leftarrow (q_1, q_2) y \rightarrow z$
22: $R_3' \leftarrow R_3' \cup (r)$
23: $\pi(r) \leftarrow \pi(r) + (w_1 \cdot w_2)$
24: return $M_3$

4 Application of tree transducer cascades

What about the case of an input wRTG and a cascade of tree transducers? We will revisit the three strategies for accomplishing application discussed above for the string case.

In order for offline composition to be a viable strategy, the transducers in the cascade must be closed under composition. Unfortunately, of the classes that preserve recognizability, only wLNT is closed under composition (Gécseg and Steibny, 1984; Baker, 1979; Malletti et al., 2009; Fülöp and Vogler, 2009).

However, the general lack of composability of tree transducers does not preclude us from conducting forward application of a cascade. We revisit the bucket brigade approach, which in Section 2 appeared to be little more than a choice of composition order. As discussed previously, application of a single transducer involves an embedding, a composition, and a projection. The embedded wRTG is in the class wLNT, and the projection forms another wRTG. As long as every transducer in the cascade can be composed with a wLNT to its left or right, depending on the application type, application of a cascade is possible. Note that this embed-compose-project process is somewhat more burdensome than in the string case. For strings, application is obtained by a single embedding, a series of compositions, and a single projec-

Algorithm 2 COVER

1: inputs
2: $u \in T_\Delta(Q_1 \times X)$
3: wT $M_2 = (Q_2, \Delta, \Gamma, R_2, q_{20})$
4: outputs
5: state $q_{21} \in Q_2$
6: set of pairs $(z, w)$ with $z \in T^*_\Gamma((Q_1 \times Q_2) \times X)$ formed by one or more successful runs on $u$ by rules in $R_2$, starting from $q_{21}$, and $w \in \mathbb{R}_+^*$ the sum of the weights of all such runs.
7: complexity
8: $O(|R_2|^*lex(u))$

9: if $u(z)$ is of the form $(q_1, x) \in Q_1 \times X$ then
10: $\pi_{init} \leftarrow (q_{11}, q_{21})$
11: else
12: \{definition of $\pi_{init}$\}
13: $\Pi_{last} \leftarrow \{\pi_{init}, \{(z, e), q_{21}\}, 1\}$
14: for all $v \in pos(u)$ such that $u(v) \in \Delta^{(k)}$ for some $k \geq 0$ in prefix order do
15: $\Pi_v \leftarrow \emptyset$
16: for all $(z, \theta, w) \in \Pi_{last}$ do
17: for all $v' \in \gamma(z)$ such that $z(v') = \bot$ do
18: for all $(\theta, v'), u(v(x_1, \ldots, x_k) w' \rightarrow h) \in R_2$
19: $\theta' \leftarrow \theta$
20: $\text{Form substitution mapping } \varphi: (Q_2 \times X)\rightarrow T^*_\Gamma((Q_1 \times Q_2 \times X) \cup \{\bot\})$
21: for $i = 1$ to $k$ do
22: for all $v'' \in \gamma(z)$ such that $h(v'') = (q_2', x_i)$ and $q_2' \in Q_2$ do
23: $\theta'(w_i, v'' \rightarrow q_2')$
24: if $u(w) = \bot$ then
25: $\varphi(q_2', x_i) \leftarrow (q_{11}, q_{21})$
26: else
27: $\varphi(q_2', x_i) \leftarrow z$
28: $\Pi_v \leftarrow \Pi_v \cup \{\{(z, [\theta']^w), \theta', w \cdot w'\}\}$
29: $z \leftarrow \{z \mid (z, \theta, w) \in \Pi_{last}\}$
30: return $\{z \mid (z, \theta, w) \in \Pi_{last}\}$

10 In which case the definition is functionally unchanged from before.
Algorithm 3 PRODUCE

1: inputs
2: w
3: \text{wrtg} G_{in} = (N_{in}, \Delta, P_{in}, n_0, \overline{M}, \overline{G}) \text{ such that } \overline{M} = (Q, \Sigma, \Delta, R, \emptyset_0) \text{ is a wLNT and } \overline{G} = (N, \Sigma, P, n_0', M', G') \text{ is a wrtg in normal form with no chain productions}
3: \text{return } G_{in}

4: outputs
5: w
6: \text{wrtg } G_{out} = (N_{out}, \Delta, P_{out}, n_0, \overline{M}, \overline{G}), \text{ such that } G_{in} \preceq G_{out} \text{ and } (n_1 \xrightarrow{w} u) \in P_{out} \Leftrightarrow (n_1 \xrightarrow{w} u) \in \overline{M}(\overline{G})^w
7: \text{complexity}
8: O(|\overline{R}| |\overline{P}|_{\text{size}}(\overline{\theta})) \text{, where } \overline{\theta} = \text{the largest left side tree in any rule in } \overline{R}
9: if \(P_{in}\) contains productions of the form \(n_1 \xrightarrow{w} u\) then
10: \text{return } G_{in}
11: end if
12: \text{return } G_{in}

13: for all \((\theta, w_2)\) do
14: for all \((\theta, w_2)\) do
15: Form substitution mapping \(\varphi : Q \times X \rightarrow T_{\Delta}(N \times Q)\) such that, for all \(v \in yd(y)\) and \(q' \in Q, \text{if there exist } n' \in N \text{ and } x \in X \text{ such that } \theta(v) = n' \text{ and } y(v) = x, \text{then } \varphi(q', x) = (n', q'). \)
16: \(p' \leftarrow ((n, q_w \xrightarrow{w} u) \xrightarrow{w} (u'))\)
17: for all \(p \in \text{Norm}(\overline{P}, \overline{N}_{out})\) do
18: Let \(p'\) be of the form \(n_{1}, n_{2}, \ldots, n_{k}\) for \(\delta \in \Delta^{(k)}\)
19: \(N_{out} \leftarrow N_{out} \cup \{n_0, \ldots, n_k\}\)
20: \(P_{out} \leftarrow P_{out} \cup \{p\}\)
21: \text{return } \text{CHAIN-REM}(G_{out})

\(G \preceq \overline{M}(\overline{G})^w\). In the latter case, \(G\) is a stand-in for \(\overline{M}(\overline{G})^w\), analogous to the stand-ins for WSAs and wSTs described in Section 2.

Algorithm 3, PRODUCE, takes as input a wrtg \(G_{in} = (N_{in}, \Delta, P_{in}, n_0, \overline{M}, \overline{G})\) and a desired nonterminal \(n_1\) and returns another wrtg, \(G_{out}\) that is different from \(G_{in}\) in that it has more productions, specifically those beginning with \(n_1\) that are in \(\overline{M}(\overline{G})^w\). Algorithms using stand-ins should call PRODUCE to ensure the stand-in they are using has the desired productions beginning with the specific nonterminal. Note, then, that PRODUCE obtains the effect of forward applica-

Algorithm 4 REPLACE

1: inputs
2: \(y \in T_{\Delta}(X)\)
3: \(w\text{rtg } G = (N, \Sigma, P, n_0, \overline{M}, \overline{G})\) in normal form, with no chain productions
4: \(n \in N\)
5: outputs
6: set \(\Pi\) of pairs \((\theta, w)\) where \(\theta\) is a mapping \(\text{pos}(y) \rightarrow N\) and \(w \in \mathbb{R}_+^n\), each pair indicating a successful run on \(y\) by productions in \(G\), starting from \(n\), and \(w\) is the weight of the run.
7: complexity
8: \(O(|P|_{\text{size}}(\overline{\theta}))\)
9: \(\Pi_{last} \leftarrow \{\{(c, n), \emptyset]\}\}
10: for all \(v \in \text{pos}(y)\) such that \(y(v) \notin X\) in prefix order do
11: \(\Pi_{v} \leftarrow \emptyset\)
12: end if
13: if \(\overline{M} = \emptyset\) and \(\overline{G} = \emptyset\) then
14: \(G \leftarrow \text{PRODUCE}(\overline{G}(\theta(v)))\)
15: for all \((\theta(v) \xrightarrow{w} y(v)(n_1, \ldots, n_k)) \in P\) do
16: \(\Pi_{\varphi} \leftarrow \Pi_{v} \cup \{(\theta, \{\psi(n_i), 1 \leq i \leq k\}, \sigma, w'\})\)
17: \(\Pi_{last} \leftarrow \Pi_{v}\)
18: return \(\Pi_{last}\)

Algorithm 5 MAKE-EXPLICIT

1: inputs
2: \(w\text{rtg } G = (N, \Sigma, P, n_0, \overline{M}, \overline{G})\) in normal form
3: outputs
4: \(w\text{rtg } G' = (N', \Sigma, P', n_0, \overline{M}, \overline{G})\) in normal form, such that \(\overline{M} = \emptyset\) and \(\overline{G} = \emptyset\), \(L_{G'} = L_{\overline{M}(\overline{G})}\), and otherwise \(G' = G\).
5: complexity
6: \(O(|P|)\)
7: \(G' \leftarrow G\)
8: \(\Xi \leftarrow \{n_0\}\) \{seen nonterminals\}
9: \(\Psi \leftarrow \{n_0\}\) \{pending nonterminals\}
10: while \(\Psi = \emptyset\) do
11: \(n \leftarrow \text{any element of } \Psi\)
12: \(\Psi \leftarrow \Psi \setminus \{n\}\)
13: if \(\overline{M} = \emptyset\) and \(\overline{G} = \emptyset\) then
14: \(G' \leftarrow \text{PRODUCE}(G', n)\)
15: for all \((n \xrightarrow{w} \sigma(n_1, \ldots, n_k)) \in P'\) do
16: for \(i = 1 \text{ to } k\) do
17: if \(n_i \notin \Xi\) then
18: \(\Xi \leftarrow \Xi \cup \{n_i\}\)
19: \(\Psi \leftarrow \Psi \cup \{n_i\}\)
20: return \(G'\)
\begin{align*}
g_0 & \xrightarrow{w_1} \sigma(g_0, g_1) \\
g_0 & \xrightarrow{w_2} \alpha \\
g_1 & \xrightarrow{w_3} \alpha
\end{align*}

(a) Input WRTG G

\begin{align*}
a_0 \xrightarrow{\sigma(x_1, x_2)} & \, w_4 \\
a_0 \xrightarrow{\sigma(x_1, x_2)} & \, \psi(x_2, x_1, a_1) \\
a_0 & \xrightarrow{\alpha} \, \alpha, \alpha \xrightarrow{w_7} \alpha \\
a_0 & \xrightarrow{\alpha} \, a_2, \alpha \xrightarrow{w_8} \rho
\end{align*}

(b) First transducer \(M_A\) in the cascade

\begin{align*}
b_0 & \xrightarrow{\sigma(x_1, x_2)} \sigma(b_0, x_1, b_0, x_2) \\
b_0 & \xrightarrow{\alpha} \alpha
\end{align*}

(c) Second transducer \(M_B\) in the cascade

\begin{align*}
g_0 & \xrightarrow{w_1, w_2} \sigma(g_0 a_0, g_1 a_1) \\
g_0 & \xrightarrow{w_2, w_3} \psi(g_0 a_2, g_1 a_1) \\
g_0 & \xrightarrow{w_6} \alpha, a_1 \xrightarrow{w_7} \alpha
\end{align*}

(d) Productions of \(M_A(G)^r\) built as a consequence of building the complete \(M_B(M_A(G)^r)^r\)

\begin{align*}
g_0 & \xrightarrow{w_1, w_2, w_9} \sigma(g_0 a_0 b_0, g_1 a_1 b_1) \\
g_0 & \xrightarrow{w_2, w_6, w_10} \alpha, g_1 a_1 b_1 \xrightarrow{w_7, w_10} \alpha
\end{align*}

(e) Complete \(M_B(M_A(G)^r)^r\)

Figure 2: Forward application through a cascade of tree transducers using an on-the-fly method.

tion in an on-the-fly manner.\(^{11}\) It makes calls to REPLACE, which is presented in Algorithm 4, as well as to a NORM algorithm that ensures normal form by replacing a single production not in normal form with several normal-form productions that can be combined together (Alexandrkis and Bozapalis, 1987) and a CHAIN-REM algorithm that replaces a WRTG containing chain productions with an equivalent WRTG that does not (Mohri, 2009).

As an example of stand-in construction, consider the invocation PRODUCE\(G_1, g_0 a_0, G\), where \(G_1 = \{g_0 a_0\}, \{\sigma, \psi, \alpha, \rho\}, \emptyset, g_0 a_0, M_A, G\), \(G\) is in Figure 2a.\(^{12}\) and \(M_A\) is in 2b. The stand-in WRTG that is output contains the first three of the four productions in Figure 2d.

To demonstrate the use of on-the-fly application in a cascade, we next show the effect of PRODUCE when used with the cascade \(G \circ M_A \circ M_B\), where \(M_B\) is in Figure 2c. Our driving algorithm in this case is Algorithm 5, MAKE-

\(^{11}\)Note further that it allows forward application of class wxLNT, something the embed-compose-project approach did not allow.

\(^{12}\)By convention the initial nonterminal and state are listed first in graphical depictions of WRTGs and WXTTs.

\(\tau_B, J_3\, J_1(x_1, x_2, x_3) \rightarrow JJ(\tau_B, x_1, \tau_B, x_2, \tau_B, x_3)\)
\(\tau_B, \tau_B, \tau_B, x_1, x_2, x_3) \rightarrow VB(\tau_B, x_1, \tau_B, x_2)\)
\(\tau_B, \tau_B, \tau_B, x_1, x_2, x_3) \rightarrow VB(\tau_B, x_1, \tau_B, x_2)\)

\(\text{"gentle"} \rightarrow \text{"gentle"}\)

(a) Rotation rules

\(i_V B, \text{NN}(x_1, x_2) \rightarrow \text{NN}((\text{INS} \text{NN} \text{NN}(x_1, x_2)))\)
\(i_V B, \text{NN}(x_1, x_2) \rightarrow \text{NN}((\text{INS} \text{NN} \text{NN}(x_1, x_2)))\)
\(i_V B, \text{NN}(x_1, x_2) \rightarrow \text{NN}((\text{INS} \text{NN} \text{NN}(x_1, x_2)))\)

(b) Insertion rules

\(t. \text{VB}(x_1, x_2, x_3) \rightarrow x(t. x_1, t. x_2, t. x_3)\)
\(t. \text{"gentleman"} \rightarrow j_1 \)
\(t. \text{"gentleman"} \rightarrow \text{EPS}\)
\(t. \text{INS} \rightarrow j_1 \)
\(t. \text{INS} \rightarrow j_2 \)

(c) Translation rules

Figure 3: Example rules from transducers used in decoding experiment. \(j_1\) and \(j_2\) are Japanese words.

EXPLICIT, which simply generates the full application WRTG using calls to PRODUCE. The input to MAKE-EXPLICIT is \(G_2 = \{g_0 a_0 b_0\}, \{\sigma, \alpha\}, \emptyset, g_0 a_0 b_0, M_B, G\).\(^{13}\) MAKE-EXPLICIT calls PRODUCE\((G_2, g_0 a_0 b_0)\). PRODUCE then seeks to cover \(g_0 \sigma(x_1, x_2) \xrightarrow{w_9} \sigma(b_0, x_1, b_0, x_2)\) with productions from \(G_1\), which is a stand-in for \(M_A(G)^r\). At line 14 of REPLACE, \(G_1\) is improved so that it has the appropriate productions. The productions of \(M_A(G)^r\) that must be built to form the complete \(M_B(M_A(G)^r)^r\) are shown in Figure 2d. The complete \(M_B(M_A(G)^r)^r\) is shown in Figure 2e. Note that because we used this on-the-fly approach, we were able to avoid building all the productions in \(M_A(G)^r\); in particular we did not build \(g_0 a_2 \xrightarrow{w_2, w_8} \rho\), while a bucket brigade approach would have built this production. We have also designed an analogous on-the-fly PRODUCE algorithm for backward application on linear WTT.

We have now defined several on-the-fly and bucket brigade algorithms, and also discussed the possibility of embed-compose-project and offline composition strategies to application of cascades of tree transducers. Tables 2a and 2b summarize the available methods of forward and backward application of cascades for recognizability-preserving tree transducer classes.

5 Decoding Experiments

The main purpose of this paper has been to present novel algorithms for performing application. However, it is important to demonstrate these algorithms on real data. We thus demonstrate bucket-brigade and on-the-fly backward application on a typical NLP task cast as a cascade of wLNT. We adapt the Japanese-to-English transla-
We evaluate the speed of each approach: bucket brigade and on-the-fly. The algorithm we use to obtain the 1-best path is a modification of the k-best algorithm of Pauls and Klein (2009). Our algorithm finds the 1-best path in a WRTG and thus fits naturally into the experimental framework. In our experiments we try several different language models to demonstrate varying performance of the application algorithms. The most realistic language model is a PCFG. Each rule captures the probability of a particular sequence of child labels given a parent label. This model has 7,765 rules.

To demonstrate more extreme cases of the usefulness of the on-the-fly approach, we build a language model that recognizes exactly the 2,087 trees in the training corpus, each with equal weight. It has 39,455 rules. Finally, to be ultra-specific, we include a form of the “specific” language model just described, but only allow the English counterpart of the particular Japanese sentence being decoded in the language. The goal in our experiments is to apply a single tree t backward through the cascade $L \circ R \circ I \circ T \circ o t$ and find the 1-best path in the application WRTG.

The results of the experiments are shown in Table 3. As can be seen, on-the-fly application is generally faster than the bucket brigade, about double the speed per sentence in the traditional experiment that uses an English PCFG language model. The results for the other two language models demonstrate more keenly the potential advantage that an on-the-fly approach provides—the simultaneous incorporation of information from all models allows application to be done more effectively than if each information source is considered in sequence. In the “exact” case, where a very large language model that simply recognizes each of the 2,087 trees in the training corpus is used, the final application is so large that it overwhelms the resources of a 4gb MacBook Pro, while the on-the-fly approach does not suffer from this problem. The “1-sent” case is presented to demonstrate the ripple effect caused by using on-the-fly. In the other two cases, a very large language model generally overwhelms the timing statistics, regardless of the method being used. But a language model that represents exactly one sentence is very small, and thus the effects of simultaneous inference are readily apparent—the time to retrieve the 1-best sentence is reduced by two orders of magnitude in this experiment.

| method | WST | wxLNT | wLNT |
|--------|-----|-------|------|
| oc     | √   | ×     | √    |
| bb     | √   | ×     | √    |
| otf    | √   | √     | √    |

(a) Forward application

| method | WST | wxLNT | wLNT |
|--------|-----|-------|------|
| oc     | √   | ×     | ×    |
| bb     | √   | √     | √    |
| otf    | √   | √     | √    |

(b) Backward application

Table 2: Transducer types and available methods of forward and backward application of a cascade. oc = offline composition, bb = bucket brigade, otf = on the fly.

| LM type | method | time/sentence |
|---------|--------|---------------|
| pcfg    | bucket | 28s           |
| pcfg    | otf    | 17s           |
| exact   | bucket | >1m           |
| exact   | otf    | 24s           |
| 1-sent  | bucket | 2.5s          |
| 1-sent  | otf    | .06s          |

Table 3: Timing results to obtain 1-best from application through a weighted tree transducer cascade, using on-the-fly vs. bucket brigade backward application techniques. pcfg = model recognizes any tree licensed by a pcfg built from observed data, exact = model recognizes each of 2,000+ trees with equal weight, 1-sent = model recognizes exactly one tree.

6 Conclusion

We have presented algorithms for forward and backward application of weighted tree transducer cascades, including on-the-fly variants, and demonstrated the benefit of an on-the-fly approach to application. We note that a more formal approach to application of WTTs is being developed.
independent from these efforts, by Fülöp et al. (2010).

Acknowledgments
We are grateful for extensive discussions with Andreas Maletti. We also appreciate the insights and advice of David Chiang, Steve DeNeefe, and others at ISI in the preparation of this work. Jonathan May and Kevin Knight were supported by NSF grants IIS-0428020 and IIS-0904684. Heiko Vogler was supported by DFG VO 1011/5-1.

References
Athanasios Alexandrakis and Symeon Bozapalidis. 1987. Weighted grammars and Kleene’s theorem. *Information Processing Letters*, 24(1):1–4.

Brenda S. Baker. 1979. Composition of top-down and bottom-up tree transductions. *Information and Control*, 41(2):186–213.

Zoltán Ésik and Werner Kuich. 2003. Formal tree series. *Journal of Automata, Languages and Combinatorics*, 8(2):219–285.

Zoltán Fülöp and Heiko Vogler. 2009. Weighted tree automata and tree transducers. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, chapter 9, pages 313–404. Springer-Verlag.

Zoltán Fülöp, Andreas Maletti, and Heiko Vogler. 2010. Backward and forward application of weighted extended tree transducers. Unpublished manuscript.

Ferenc Gécseg and Magnus Steinby. 1984. *Tree Automata*. Akadémiai Kiadó, Budapest.

Liang Huang and David Chiang. 2005. Better k-best parsing. In Harry Bunt, Robert Malouf, and Alon Lavie, editors, *Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT)*, pages 53–64, Vancouver, October. Association for Computational Linguistics.

Werner Kuich. 1998. Formal power series over trees. In Symeon Bozapalidis, editor, *Proceedings of the 3rd International Conference on Developments in Language Theory (DLT)*, pages 61–101, Thessaloniki, Greece. Aristotle University of Thessaloniki.

Werner Kuich. 1999. Tree transducers and formal tree series. *Acta Cybernetica*, 14:135–149.

Andreas Maletti, Jonathan Graehl, Mark Hopkins, and Kevin Knight. 2009. The power of extended top-down tree transducers. *SIAM Journal on Computing*, 39(2):410–430.

Andreas Maletti. 2006. Compositions of tree series transformations. *Theoretical Computer Science*, 366:248–271.

Andreas Maletti. 2008. Compositions of extended top-down tree transducers. *Information and Computation*, 206(9–10):1187–1196.

Andreas Maletti. 2009. Personal Communication.

Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. 2000. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231:17–32.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–312.

Mehryar Mohri. 2009. Weighted automata algorithms. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, chapter 6, pages 213–254. Springer-Verlag.

Adam Pauls and Dan Klein. 2009. K-best A* parsing. In Keh-Yih Su, Jian Su, Janyce Wiebe, and Haizhou Li, editors, *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 958–966, Suntec, Singapore, August. Association for Computational Linguistics.

Fernando Pereira and Michael Riley. 1997. Speech recognition by composition of weighted finite automata. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*, chapter 15, pages 431–453. MIT Press, Cambridge, MA.

William A. Woods. 1980. Cascaded ATN grammars. *American Journal of Computational Linguistics*, 6(1):1–12.

Kenji Yamada and Kevin Knight. 2001. A syntax-based statistical translation model. In *Proceedings of 39th Annual Meeting of the Association for Computational Linguistics*, pages 523–530, Toulouse, France, July. Association for Computational Linguistics.