In *Proc. of the Eleventh International Conference on Implementation and Application of Automata (CIAA)*, Lecture Notes in Computer Science, ©Springer Verlag, 2006.

Tiburon: A Weighted Tree Automata Toolkit*

Jonathan May and Kevin Knight

Information Sciences Institute
University of Southern California
Marina Del Rey, CA 90292
{jonmay, knight}@isi.edu

Abstract. The availability of weighted finite-state string automata toolkits made possible great advances in natural language processing. However, recent advances in syntax-based NLP model design are unsuitable for these toolkits. To combat this problem, we introduce a weighted finite-state *tree* automata toolkit, which incorporates recent developments in weighted tree automata theory and is useful for natural language applications such as machine translation, sentence compression, question answering, and many more.

1 Introduction

The development of well-founded models of natural language processing applications has been greatly accelerated by the availability of toolkits for finite-state automata. The influential observation of Kaplan & Kay, that cascades of phonological rewrite rules could be expressed as regular relations (equivalent to finite-state transducers) [1], was exploited by Koskenniemi in his development of the two-level morphology and accompanying system for its representation [2]. This system, which was a general program for analysis and generation of languages, pioneered the field of finite-state toolkits [3].

Successive versions of the two-level compiler, such as that written by Karttunen and others at Xerox [4], were used for large-scale analysis applications in many languages [3]. Continued advances, such as work by Karttunen in intersecting composition [5] and replacement [6, 7], eventually led to the development of the Xerox finite-state toolkit, which superseded the functionality and use of the two-level tools [3].

Meanwhile, interest in adding uncertainty to finite-state models grew alongside increased availability of large datasets and increased computational power. Ad-hoc methods and individual implementations were developed for integrating uncertainty into finite-state representations [8, 9], but the need for a general-purpose weighted finite-state toolkit was clear [10]. Researchers at AT&T led the way with their FSM Library [11] which represented weighted finite-state automata by incorporating the theory of semirings over rational power series cleanly into the existing automata theory. Other toolkits, such as van Noord's FSA utilities [12], the RWTH toolkit [13], and the USC/ISI Carmel toolkit [14], provided additional interfaces and utilities for working with weighted finite-state automata. As in the unweighted case, the availability of

* The authors wish to thank Steve DeNeefe, Jonathan Graehl, Mark Hopkins, Liang Huang, Daniel Marcu, and Magnus Steinby for their advice and comments. This work was partially supported by NSF grant IIS-0428020 and by GALE-DARPA Contract HR0011-06-C-0022.

this software led to many research projects that took advantage of pre-existing implementations [15–17] and the development of the software led to the invention of new algorithms and theory [18, 19].

While these toolkits are very robust and capable of development of a wide array of useful applications in NLP and beyond, they all suffer from the limitation that they can only operate on string-based regular languages. In the 1990s, this was begrudgingly accepted as sufficient — the power of computers and the relatively limited availability of data prevented any serious consideration of weighted automata of greater complexity, even though more complex automata models that better captured the syntactic nature of language had long been proposed [20]. As NLP research progressed and computing power and available data increased, researchers started creating serious probabilistic tree-based models for such natural language tasks as translation [21–23], summarization [24], paraphrasing [25], language modeling [26], and others. And once again, software implementations of these models were individual, one-off efforts that took entire PhD theses' worth of work to create [27]. GRM, an extension of the AT&T toolkit that uses approximation theory to represent higher-complexity structure such as context-free grammars in the weighted finite-state string automata framework, was useful for handling certain representations [28], but a tree automata framework is required to truly capture tree models.

Knight and Graehl [29] put forward the case for the top-down tree automata theory of Rounds [20] and Thatcher [30] as a logical sequel to weighted string automata for NLP. All of the previously mentioned tree-based models fit nicely into this theory. Additionally, as Knight and Graehl mention [29], most of the desired general operations in a general weighted finite-state toolkit are applicable to top-down tree automata.

We thus propose and present a toolkit designed in the spirit of its predecessors but with the tree, not the string, as its basic data structure. Tiburon is a toolkit for manipulation of weighted top-down tree automata. It is designed to be easy to construct automata and work with them — after reading this article a linguist with no computer science background or a computer scientist with only the vaguest notions of tree automata should be able to write basic acceptors and transducers. To achieve these goals we have maintained simplicity in data format design, such that acceptors and transducers are very close to the way they appear in tree automata literature. We also provide a small set of generic but powerful operations that allow robust manipulation of data structures with simple commands. In subsequent sections we present an introduction to the formats and operations in the Tiburon toolkit and demonstrate the powerful applications that can be easily built.

2 Related Work

The rich history of finite-state string automata toolkits was described in the previous section. Tree automata theory is extensively covered in [31, 32]. Timbuk [33] is a toolkit for unweighted finite state tree automata that has been used for cryptographic analysis. It is based on ELAN [34], a term rewriting computational system. MONA [35] is an unweighted tree automata tool aimed at the logic community. Probabilistic tree automata were first proposed by Magidor and Moran [36]. Weighted tree transducers were first described by Fülöp and Vogler [37] as an operational representation of tree series transducers, first introduced by Kuich [38]

3 Trees

Tree automata represent sets of trees and tree relations. Formally, a tree is constructed from a ranked alphabet Σ. Each member of the alphabet is assigned one or more non-negative integers, called a *rank*, and Σ_m refers to all $x \in \Sigma$ with rank m. A tree over Σ is thus defined as:

- x, where $x \in \Sigma_0$, or
- $x(t_1, ...t_m)$, where $x \in \Sigma_m$ and $t_1, ..., t_m$ are trees over Σ.

Figure 1 shows a typical tree and its representation in Tiburon. In this example, `NP` has rank 2, `DT` and `NN` have rank 1, and "`the`" and "`boy`" have rank 0.

```
      NP
     /  \
   DT    NN
   |      \
"the"  "boy"
```

(a)

```
NP(DT("the") NN("boy"))
```

(b)

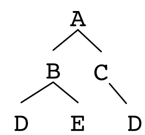
Fig. 1. (a) A typical syntax tree, and (b), its Tiburon representation

4 Regular Tree Grammars

As finite-state string acceptors recognize the same family of string languages as regular string grammars, so do finite-state tree acceptors recognize the same family of tree languages as regular tree grammars (RTG) [39]. For simplicity we favor the grammar representation, as tree acceptors must be written as hypergraphs, and this can be confusing. RTGs look very similar to context-free grammars (CFG) (in fact, a CFG is a special case of an RTG) and thus tend to be a very comfortable formalism. Analogous to their string counterpart, a weighted regular tree grammar (wRTG) recognizes a weighted, possibly infinite set of trees. Formally, a wRTG over Σ and under semiring $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ consists of a finite set N of *nonterminal symbols* disjoint from Σ, a start symbol $s \in N$, and a set P of *productions* of the form $a \to r, \delta$, where $a \in N$, r is a tree over $\Sigma \cup N$, and $\delta \in \mathbb{K}$ is an associated weight. Informally, a wRTG "works" similar to a CFG in that from the start symbol, a sequence of rewrites is performed, replacing nonterminals with trees as specified by the productions, until the generated/recognized tree has no nonterminals remaining. To calculate the weight w of the resulting tree, start with $w = \bar{1}$ and for each production $p = a \to r, \delta$ used, let $w = w \otimes \delta$. Figure 2(a) shows a typical wRTG in Tiburon format. When weights are omitted on productions, a weight of $\bar{1}$ is assumed. Figure 2(b) shows one of the trees that this grammar recognizes. If we use the probability semiring, the tree has an associated weight of 0.16

```
q -> A(q r) # 0.8
q -> B(r E) # 0.2
r -> C(r)
r -> D
```

(a)



(b)

Fig. 2. (a) A regular tree grammar, and (b), a tree in the grammar's language

4.1 Generation

One fundamental operation on a wRTG is the generation of trees that are in the grammar's language. Naturally, one might want to know the tree of highest weight in a grammar. Knuth's extension [40] of Dijkstra's classic best-path algorithm [41] to the hypergraph case efficiently finds the best path in the tree recognizer equivalent of a wRTG. However, in many cases it is desirable to obtain a *list* of trees, ordered by weight. A machine translation application may output a wRTG encoding billions of partial translations, and we may want to list the top scoring 25,000 trees for a subsequent re-ranking operation. The `-k` operation in Tiburon adapts the k-best paths algorithm of Huang and Chiang [42] to wRTGs. For example, given the grammar `even.rtg` depicted in Fig. 3(a), we issue this command:

```
java -jar tiburon.jar -k 5 even.rtg
```

The five derivations with highest weight in the grammar are returned, as depicted in Fig. 3(b).

```
qe
qe -> A(qe qo) # .1
qe -> A(qo qe) # .8
qe -> B(qo) # .1
qo -> A(qo qo) # .6
qo -> A(qe qe) # .2
qo -> B(qe) # .1
qo -> C # .1
```

(a)

```
B(C): 0.0100
A(C B(C)): 0.0008
B(A(C C)): 0.0006
A(B(C) C): 0.0001
B(B(B(C))): 0.0001
```

(b)

Fig. 3. (a) `even.rtg`, and (b) its top 5 derivations

Another operation, `-g`, stochastically generates trees from a grammar, probabilistically choosing states to expand until a tree is obtained or a threshold of expansion is reached. This operation is useful for diagnosis — designers of wRTGs may wish to verify that their wRTGs generate trees according to the distribution they have in mind. Given the grammar `vic.rtg`, depicted in Fig. 4(a), we issue the following command, obtaining five random derivations, as seen in Fig. 4(b):

```
java -jar tiburon.jar -g 5 vic.rtg
```

```
q
q -> S(np vp)
np -> NP(dt nn)
np -> NP(dt jj nn)
dt -> the
dt -> a
jj -> funny
jj -> blue
jj -> strange
nn -> fish
nn -> carrot
vp -> VP(v np)
v -> ate
v -> created
```

(a)

```
S(NP(the carrot) VP(ate NP(a fish))): 1.0000
S(NP(the fish) VP(created NP(a carrot))): 1.0000
S(NP(a carrot) VP(created NP(the funny fish))): 1.0000
S(NP(the fish) VP(created NP(a carrot))): 1.0000
S(NP(a fish) VP(created NP(a fish))): 1.0000
```

(b)

Fig. 4. (a) `vic.rtg`, and (b) five random derivations

4.2 Intersection

Weighted intersection of wRTGs is useful for subdividing large problems into smaller ones. As noted in [31], RTGs (and by extension wRTGs) are closed under intersection. Thus, a wRTG representing machine translation candidate sentences can be intersected with another wRTG representing an English syntax language model to produce re-weighted translations. As a simpler example, consider the grammar `even.rtg` depicted in Fig. 3(a), which produces trees with an even number of labels. The grammar `three.rtg` depicted in Fig. 5(a) produces trees with a number of labels divisible by three. We obtain a grammar which produces trees with a number of labels divisible by six. by using the following command. The grammar is partly shown in Fig. 5(b).

```
java -jar tiburon.jar even.rtg three.rtg
```

```
q3
q3 -> A(q1 q1) # .25
q3 -> A(q3 q2) # .25
q3 -> A(q2 q3) # .25
q3 -> B(q2)    # .25
q2 -> A(q2 q2) # .25
q2 -> A(q1 q3) # .25
q2 -> A(q3 q1) # .25
q2 -> B(q1)    # .25
q1 -> A(q3 q3) # .025
q1 -> A(q1 q2) # .025
q1 -> A(q2 q1) # .025
q1 -> B(q3)    # .025
q1 -> C        # .9
```

(a)

```
qe_q3
qe_q3 -> A(qo_q1 qe_q1) # 0.2000
qe_q3 -> A(qo_q3 qe_q2) # 0.2000
qe_q3 -> A(qo_q2 qe_q3) # 0.2000
qe_q3 -> B(qo_q2) # 0.0250
qe_q3 -> A(qe_q1 qo_q1) # 0.0250
qe_q3 -> A(qe_q3 qo_q2) # 0.0250
qe_q3 -> A(qe_q2 qo_q3) # 0.0250
qe_q2 -> A(qe_q3 qo_q1) # 0.0250
qe_q2 -> A(qe_q1 qo_q3) # 0.0250
qe_q2 -> A(qe_q2 qo_q2) # 0.0250
qe_q2 -> A(qo_q3 qe_q1) # 0.2000
qe_q2 -> A(qo_q1 qe_q3) # 0.2000
qe_q2 -> A(qo_q2 qe_q2) # 0.2000
qe_q2 -> B(qo_q1) # 0.0250
```

(b)

Fig. 5. (a) `three.rtg`, and (b) a portion of the intersection of `even.rtg` (see Fig. 3) and `three.rtg`. The complete grammar has 43 productions.

4.3 Weighted Determinization

wRTGs produced by automated systems such as those used to perform machine translation [43] or parsing [44] frequently contain multiple derivations for the same tree with different weight. This is due to the systems' representation of their result space in terms of weighted partial results of various sizes that may be assembled in multiple ways. This property is undesirable if we wish to know the total probability of a particular tree in a language. It is also frequently undesirable to have repeated results in a *k*-best list. The `-d` operation invokes May and Knight's weighted determinization algorithm for tree automata [45]. As an example, consider the grammar `undet.rtg`, depicted in Fig. 6(a). This grammar has two differently-weighted derivations of the tree `D(A B)`, as we see when the top three derivations are obtained, depicted in Fig. 6(b). The following command determinizes the grammar and attempts to return the top three derivations in the resulting grammar:

```
java -jar tiburon.jar -d 5 -k 3 undet.rtg
```

Of course, since there are now only two derivations, only two trees are returned, as seen in Fig. 6(c). The `-d 5` argument signifies the maximum time allowed for determinization, in minutes. Determinization is, in the worst case, an exponential-time operation, so it is helpful in practical matters to prevent overly lengthy operations.

```
t
t -> D(q r) # 0.2
t -> D(q s) # 0.3
q -> A # 0.3
r -> B # 0.2
s -> B # 0.6
s -> C # 0.4
```

(a)

```
D(A B): 0.0540
D(A C): 0.0360
D(A B): 0.0120
```

(b)

```
Warning: returning fewer
trees than requested
D(A B): 0.0660
D(A C): 0.0360
```

(c)

Fig. 6. (a) Undeterminized grammar `undet.rtg`, (b) *k*-best list without determinization, and (c) *k*-best list with determinization

4.4 Pruning

In real systems using large grammars to represent complex tree languages, memory and cpu time are very real issues. Even as computers increase in power, the added complexity of tree automata forces practitioners to combat computationally intensive processes. One way of avoiding long running times is to prune weighted automata before operating on them. One technique for pruning finite-state (string) automata is to use the forward-backward algorithm to calculate the highest-scoring path each arc in the automaton is involved in, and then prune the arcs that are only in relatively low-scoring paths [46].

We apply this technique for tree automata by using an adaptation [47] of the inside-outside algorithm [48]. The `-p` option with argument x removes productions from a tree grammar that are involved in paths x times or more worse than the best path. The `-c` option provides an overview of a grammar, and we can use this to demonstrate the effects of pruning. The file `c1s4.determ.rtg` represents a language of possible translations of a particular Chinese sentence. We inspect the grammar as follows:

```
java -jar tiburon.jar -m tropical -c c1s4.determ.rtg
Check info:
        113 states
        168 rules
        28 unique terminal symbols
        2340 derivations
```

Note that the `-m tropical` flag is used because this grammar is weighted in the tropical semiring. We prune the grammar and then inspect it as follows:

```
java -jar tiburon.jar -m tropical -p 8 -c c1s4.determ.rtg
Check info:
        111 states
        158 rules
        28 unique terminal symbols
        780 derivations
```

Since we are in the tropical semiring, this command means "Prune all productions that are involved in derivations scoring worse than the best derivation plus 8". This roughly corresponds to derivations with probability 2980 times worse than the best derivation. Note that the pruned grammar has fewer than half the derivations of the unpruned grammar. A quick check of the top derivations after the pruning (using `-k`) shows that the pruned and unpruned grammars do not differ in their sorted derivation lists until the 455th-highest derivation.

5 Tree Transducers

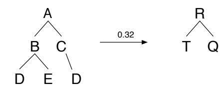
Top-down tree transducers in Tiburon come in two varieties: *tree-to-tree* [20, 30] and *tree-to-string* [49]. They represent the weighted transformation of a tree language into either a tree language or a string language, respectively. One can also think of transducers as representing a language of weighted tree/tree or tree/string pairs [50, 51]. We omit a formal definition of top-down tree transducers here; we refer the reader to [31] for a thorough treatment.

Figure 7(a) shows a sample tree-to-tree transducer in Tiburon format. Like a tree grammar, it has a start symbol and a set of optionally weighted productions. A transduction operation walks down an input tree, transforming it and recursively processing its branches. For example, the first production in Fig. 7(a) means: "When in state q, facing an input subtree with root symbol A and two children about which we know nothing, replace it with an output subtree rooted at R with two children. To compute the output subtree's left child, recursively process the input subtree's right child beginning in state r. To compute the output subtree's right child, recursively process the input subtree's left child beginning in state r." Figure 7(b) shows one transduction licensed by this transducer. The format for tree-to-string transducers in Tiburon is similar to that for tree-to-tree transducers; the sole differences are the right side of productions are strings, not trees, and the special symbol `*e*` representing the empty string may be used.

Knight and Graehl [29] describe a wide hierarchy of transducer classes. Transducers in Tiburon are specifically *top-down transducers with extended left-hand sides*, also

```
q
q.A(x0: x1:) -> R(r.x1 r.x0) # 0.4
q.A(x0: x1:) -> R(r.x0 s.x1) # 0.6
q.C(x0:) -> L(q.x0)
r.C(x0:) -> T # 0.8
r.C(x0:) -> S # 0.2
r.C(x0: B(D x1:)) -> R(q.x0 r.x1)
r.B(x0: x1:E) -> Q
s.B -> X
```

(a)



(b)

Fig. 7. (a) A tree-to-tree transducer, and (b), a weighted transduction licensed by the transducer

known as **xR** in Knight and Graehl's hierarchy, where "R" denotes "root-to-frontier" (i.e. top-down) and the "x" denotes the extended left-hand sides. By extended, we mean that the left side of productions can be trees of arbitrary depth. The sixth and seventh productions in Fig. 7(a) show an example of extended left-hand side productions. Strictly speaking, the class xR refers to only tree-to-tree transducers; tree-to-string transducers with the same characteristics are in the class **xRs**. There are no restrictions on copying or deleting of variable children in xR or xRs; the seventh production shows an example of a deleted child. The xR and xRs class of transducers were selected for Tiburon because of their good fit with natural language applications [20, 29].

5.1 Forward and Backward Application

Application is the operation of passing a tree or grammar onto a transducer and obtaining the resultant image. Tiburon supports forward application of a tree onto a tree-to-tree or tree-to-string transducer with the `-l` operation (for "left-side" transduction). If the transducer in Fig. 7(a) is in a file `xr1.trans` the transduction performed in Fig. 7(b) can be accomplished as follows:

```
echo "A(B(D E) C(D))" | java -jar tiburon.jar -l -s xr1.trans
q2
q2 -> R(q0 q1) # 0.4000
q1 -> Q # 1.0000
q0 -> S # 0.2000
q0 -> T # 0.8000
```

The `-s` flag tells Tiburon to expect the input tree from *stdin* instead of a file. As seen above, the image of a tree onto a tree-to-tree transducer is a wRTG. The image of a tree onto a tree-to-string transducer is a wCFG, currently represented in Tiburon as a one-state wRTG. The image of a wRTG onto the transducers supported in Tiburon is not a wRTG [31] and as such is currently not supported. However, limited versions of the transducers supported, such as transducers that do not copy their variables, do produce wRTG images [31]. We will soon release the next version of Tiburon, which will support the `-r` operation for backward application (the inverse image of a tree or wRTG onto a transducer is a wRTG) and forward application of wRTGs onto limited classes of transducers.

5.2 Composition

We often want to build a cascade of several small transducers and then programmatically combine them into one. Unlike string transducers, general top-down tree transducers are not closed under composition, that is, a transduction carried out by a sequence of two transducers may not be possible with a single transducer. Engelfriet showed that top-down tree transducers that do not allow deletion or copying of variables (known as RLN transducers; the L signifies "linear" and the "N" signifies "non-deleting") are closed under composition [52]. Tiburon, however, allows composition of tree-to-tree transducers without checking if the transducers to be composed are composable. For example, consider the transducer below, which is in a file `xr2.trans`:

```
q
q.R(x0: x1:) -> R(R q.x1 R q.x0)
q.T -> B # 0.6
q.T -> D # 0.4
q.Q -> C
q.S -> E
q.X -> A
```

The following command composes `xr1.trans` with `xr2.trans` and passes a tree through them, returning the top three output derivations:

```
echo "A(B(D E) C(D))" | java -jar tiburon.jar -ls -k 3 \
    xr1.trans xr2.trans
R(R C R B): 0.1920
R(R C R D): 0.1280
R(R C R E): 0.0800
```

`xr1.trans` is not RLN, it is xRL (i.e. it has an extended left side and deletes variables but does not copy), but in this case the two transducers are composable. We believe that many of the xR transducers used in natural language applications will not suffer from the general noncomposability of their class.

5.3 Training

A common task in building tree transducer models is the assignment of appropriate weights to productions. We can use Expectation-Maximization training [53] to set the weights of an unweighted tree transducer such that they maximize the likelihood of a training corpus of tree/tree or tree/string pairs. Tiburon provides the `-t` operation, which implements the technique described by Graehl and Knight for training tree transducers using EM [54].

As an example, consider training a machine translation model using bilingual input/output pairs. Given the 261-production unweighted tree-to-string transducer depicted in Fig. 8(a) in file `y1.ts`, and the 15 tree/string pairs in Fig. 8(b) in file `y1.train`, we run this command to produce the transducer in Fig. 8(c):

```
java -jar tiburon.jar -t 20 y1.train y1.trans
```

```
q
q.X(x0: x1:) -> q.x0 q.x1
q.X(x0: x1:) -> q.x1 q.x0
q.a -> *e*
q.a -> empresa
q.a -> Garcia
q.also -> asociados
q.also -> *e*
q.also -> empresa
q.also -> enfadados
q.also -> estan
q.also -> Garcia
q.also -> los
q.also -> tambien
q.also -> tiene
q.also -> una
q.and -> asociados
q.and -> clientes
q.and -> *e*
q.and -> enemigos
...
```

(a)

```
X(Garcia X(and associates))
Garcia y asociados
X(X(Carlos Garcia) X(has X(three associates)))
Carlos Garcia tiene tres asociados
X(X(his associates) X(X(are not) strong))
sus asociados no son fuertes
X(Garcia X(X(has X(a company)) also))
Garcia tambien tiene una empresa
X(X(its clients) X(are angry))
sus clientes estan enfadados
X(X(the associates) X(X(are also) angry))
los asociados tambien estan enfadados
X(X(X(the clients) X(and X(the associates))) X(are enemies))
los clientes y los asociados son enemigos
X(X(the company) X(has X(three groups)))
la empresa tiene tres grupos
X(X(its groups) X(are X(in Europe)))
sus grupos estan en Europa
X(X(the X(modern groups)) X(sell X(strong pharmaceuticals)))
los grupos modernos venden medicinas fuertes
X(X(the groups) X(X(do not) X(sell zanzanine)))
los grupos no venden zanzanina
X(X(the X(small groups)) X(X(are not) modern))
los grupos pequenos no son modernos
```

(b)

```
q
q.X(x0: x1:) -> q.x0 q.x1 # 0.8571
q.X(x0: x1:) -> q.x1 q.x0 # 0.1429
q.are -> son # 0.5
q.are -> estan # 0.5
q.the -> los # 0.8571428571428571
q.the -> la # 0.14285714285714288
q.not -> no # 1.0
q.do -> *e* # 1.0
q.Garcia -> Garcia # 1.0
q.enemies -> enemigos # 1.0
q.angry -> enfadados # 1.0
q.has -> tiene # 1.0
q.zanzanine -> zanzanina # 1.0
```

(c)

Fig. 8. (a) Portion of a 261-production unweighted tree-to-string transducer. (b) 15 (tree, string) training pairs. (c) Portion of the 31-production weighted tree-to-string transducer produced after 20 iterations of EM training (all other productions had probability 0).

6 Applications Using Tiburon

The translation model of Yamada and Knight [22] is a specialized model for predicting a Japanese string given an English tree. The custom implementation of this model, built by Yamada as part of his PhD thesis [27], took more than one year to complete. Graehl and Knight [54] showed how this model could be represented as a four-state tree-to-string transducer. We built an untrained transducer from Yamada's model and trained it on the same data used by Yamada and Knight to produce their alignment sentence pairs [22]. The complete process took only 2 days. For details we refer the reader to [55].

Knight and Graehl [56] describe a cascade of finite-state string transducers that perform English-Japanese transliteration. Of course, a weighted string transducer toolkit such as Carmel is well suited for this task, but Tiburon is suited for the job as well. By converting string transducers into monadic (non-branching) tree transducers, we obtain equivalent results. We used simple scripts to transform the string transliteration transducers into these monadic trees, and reproduced the transliteration operations. Thus, we see how Tiburon may be used for string-based as well as tree-based applications.

7 Conclusion

We have described Tiburon, a general weighted tree automata toolkit, and described some of its functions and their use in constructing natural language applications. Tiburon can be downloaded at http://www.isi.edu/licensed-sw/tiburon/

References

1. Kaplan, R.M., Kay, M.: Phonological rules and finite-state transducers. In: Linguistic Society of America Meeting Handbook, Fifty-Sixth Annual Meeting. (1981) Abstract.
2. Koskenniemi, K.: Two-level morphology: A general computational model for word-form recognition and production. Publication 11, University of Helsinki, Department of General Linguistics, Helsinki (1983)
3. Karttunen, L., Beesley, K.R.: A short history of two-level morphology. Presented at the ESSLLI-2001 Special Event titled "Twenty Years of Finite-State Morphology" (2001) Helsinki, Finland.
4. Karttunen, L., Beesley, K.R.: Two-level rule compiler. Technical Report ISTL-92-2, Xerox Palo Alto Research Center, Palo Alto, CA (1992)
5. Karttunen, L., Kaplan, R.M., Zaenen, A.: Two-level morphology with composition. In: COLING Proceedings. (1992)
6. Karttunen, L.: The replace operator. In: ACL Proceedings. (1995)
7. Karttunen, L.: Directed replacement. In: ACL Proceedings. (1996)
8. Riccardi, G., Pieraccini, R., Bocchieri, E.: Stochastic automata for language modeling. Computer Speech & Language **10**(4) (1996)
9. Ljolje, A., Riley, M.D.: Optimal speech recognition using phone recognition and lexical access. In: ICSLP Proceedings. (1992)
10. Mohri, M., Pereira, F.C.N., Riley, M.: The design principles of a weighted finite-state transducer library. Theoretical Computer Science **231** (2000)
11. Mohri, M., Pereira, F.C.N., Riley, M.: A rational design for a weighted finite-state transducer library. In: Proceedings of the 7th Annual AT&T Software Symposium. (1997)
12. van Noord, G., Gerdemann, D.: An extendible regular expression compiler for finite-state approaches in natural language processing. In: 4th International Workshop on Implementing Automata. (2000)
13. Kanthak, S., Ney, H.: Fsa: An efficient and flexible c++ toolkit for finite state automata using on-demand computation. In: ACL Proceedings. (2004)
14. Graehl, J.: Carmel finite-state toolkit. http://www.isi.edu/licensed-sw/carmel (1997)
15. Kaiser, E., Schalkwyk, J.: Building a robust, skipping parser within the AT&T FSM toolkit. Technical report, Center for Human Computer Communication, Oregon Graduate Institute of Science and Technology (2001)
16. van Noord, G.: Treatment of epsilon moves in subset construction. Comput. Linguist. **26**(1) (2000)
17. Koehn, P., Knight, K.: Feature-rich statistical translation of noun phrases. In: ACL Proceedings. (2003)
18. Pereira, F., Riley, M.: Speech recognition by composition of weighted finite automata. In Roche, E., Schabes, Y., eds.: Finite-State Language Processing. MIT Press, Cambridge, MA (1997)
19. Mohri, M.: Finite-state transducers in language and speech processing. Comput. Linguist. **23**(2) (1997)
20. Rounds, W.C.: Mappings and grammars on trees. Mathematical Systems Theory **4** (1970)
21. Och, F.J., Tillmann, C., Ney, H.: Improved alignment models for statistical machine translation. In: EMNLP/VLC Proceedings. (1999)
22. Yamada, K., Knight, K.: A syntax-based statistical translation model. In: ACL Proceedings. (2001)
23. Eisner, J.: Learning non-isomorphic tree mappings for machine translation. In: ACL Proceedings (companion volume). (2003)
24. Knight, K., Marcu, D.: Summarization beyond sentence extraction: A probabilistic approach to sentence compression. Artificial Intelligence **139** (2002)
25. Pang, B., Knight, K., Marcu, D.: Syntax-based alignment of multiple translations extracting paraphrases and generating new sentences. In: NAACL Proceedings. (2003)

26. Charniak, E.: Immediate-head parsing for language models. In: ACL Proceedings. (2001)
27. Yamada, K.: A Syntax-Based Translation Model. PhD thesis, University of Southern California (2002)
28. Allauzen, C., Mohri, M., Roark, B.: A general weighted grammar library. In: CIAA Proceedings. (2004)
29. Knight, K., Graehl, J.: An overview of probabilistic tree transducers for natural language processing. In: CICLing Proceedings. (2005)
30. Thatcher, J.W.: Generalized^2 sequential machines. J. Comput. System Sci. **4** (1970)
31. Gécseg, F., Steinby, M.: Tree Automata. Akadémiai Kiadó, Budapest (1984)
32. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata` (1997) release October 1 2002.
33. Genet, T., Tong, V.V.T.: Reachability analysis of term rewriting systems with timbuk. In: LPAR Proceedings. (2001)
34. Borovansky, P., Kirchner, C., Kirchner, H., Moreau, P., Vittek, M.: Elan: A logical framework based on computational systems. In: Proceedings of the first international workshop on rewriting logic. (1996)
35. Henriksen, J., Jensen, J., Jørgensen, M., Klarlund, N., Paige, B., Rauhe, T., Sandholm, A.: Mona: Monadic second-order logic in practice. In: TACAS Proceedings. (1995)
36. Magidor, M., Moran, G.: Probabilistic tree automata. Israel Journal of Mathematics **8** (1969)
37. Fülöp, Z., Vogler, H.: Weighted tree transducers. J. Autom. Lang. Comb. **9**(1) (2004)
38. Kuich, W.: Tree transducers and formal tree series. Acta Cybernet. **14** (1999)
39. Brainerd, W.S.: Tree generating regular systems. Inform. and Control **14** (1969)
40. Knuth, D.: A generalization of Dijkstra's algorithm. Inform. Process. Lett. **6**(1) (1977)
41. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1** (1959)
42. Huang, L., Chiang, D.: Better k-best parsing. In: IWPT Proceedings. (2005)
43. Galley, M., Hopkins, M., Knight, K., Marcu, D.: What's in a translation rule? In: HLT-NAACL Proceedings. (2004)
44. Bod, R.: An efficient implementation of a new DOP model. In: EACL Proceedings. (2003)
45. May, J., Knight, K.: A better n-best list: Practical determinization of weighted finite tree automata. In: NAACL Proceedings. (2006)
46. Siztus, A., Ortmanns, S.: High quality word graphs using forward-backward pruning. In: Proceedings of the IEEE Conference on Acoustic, Speech and Signal Processing. (1999)
47. Graehl, J.: Context-free algorithms. Unpublished handout (2005)
48. Lari, K., Young, S.J.: The estimation of stochastic context-free grammars using the inside-outside algorithm. Computer Speech and Language **4** (1990)
49. Aho, A.V., Ullman, J.D.: Translations of a context-free grammar. Inform. and Control **19** (1971)
50. Shieber, S.M.: Synchronous grammars as tree transducers. In: TAG+7 Proceedings. (2004)
51. Schabes, Y.: Mathematical and Computational Aspects of Lexicalized Grammars. PhD thesis, Univ. of Pennsylvania, Phila., PA (1990)
52. Engelfriet, J.: Bottom-up and top-down tree transformations. a comparison. Mathematical Systems Theory **9** (1976)
53. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society, Series B **39**(1) (1977)
54. Graehl, J., Knight, K.: Training tree transducers. In: HLT-NAACL Proceedings. (2004)
55. Graehl, J., Knight, K., May, J.: Training tree transducers. Comput. Linguist. (Submitted)
56. Knight, K., Graehl, J.: Machine transliteration. Comput. Linguist. **24**(4) (1998)