In *Proc. of the Human Language Technology conference - North American chapter of the Association for Computational Linguistics annual meeting (HLT-NAACL)*, 2006

A Better N-Best List: Practical Determinization of Weighted Finite Tree Automata

Jonathan May
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292
jonmay@isi.edu

Kevin Knight
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292
knight@isi.edu

Abstract

Ranked lists of output trees from syntactic statistical NLP applications frequently contain multiple repeated entries. This redundancy leads to misrepresentation of tree weight and reduced information for debugging and tuning purposes. It is chiefly due to nondeterminism in the weighted automata that produce the results. We introduce an algorithm that determinizes such automata while preserving proper weights, returning the sum of the weight of all multiply derived trees. We also demonstrate our algorithm's effectiveness on two large-scale tasks.

1 Introduction

A useful tool in natural language processing tasks such as translation, speech recognition, parsing, etc., is the ranked list of results. Modern systems typically produce competing partial results internally and return only the top-scoring complete result to the user. They are, however, also capable of producing lists of runners-up, and such lists have many practical uses:

- The lists may be inspected to determine the quality of runners-up and motivate model changes.
- The lists may be re-ranked with extra knowledge sources that are difficult to apply during the main search.
- The lists may be used with annotation and a tuning process, such as in (Collins and Roark, 2004), to iteratively alter feature weights and improve results.

Figure 1 shows the best 10 English translation parse trees obtained from a syntax-based translation system based on (Galley, et. al., 2004). Notice that the same tree occurs multiple times in this list. This repetition is quite characteristic of the output of ranked lists. It occurs because many systems, such as the ones proposed by (Bod, 1992), (Galley, et. al., 2004), and (Langkilde and Knight, 1998) represent their result space in terms of weighted partial results of various sizes that may be assembled in multiple ways. There is in general more than one way to assemble the partial results to derive the same complete result. Thus, the n-best list of results is really an n-best list of *derivations*.

When list-based tasks, such as the ones mentioned above, take as input the top n results for some constant n, the effect of repetition on these tasks is deleterious. A list with many repetitions suffers from a lack of useful information, hampering diagnostics. Repeated results prevent alternatives that would be highly ranked in a secondary reranking system from even being considered. And a list of fewer unique trees than expected can cause overfitting when this list is used to tune. Furthermore, the actual weight of obtaining any particular tree is split among its repetitions, distorting the actual relative weights between trees.

(Mohri, 1997) encountered this problem in speech recognition, and presented a solution to the problem of repetition in n-best lists of strings that are derived from finite-state automata. That work described a way to use a powerset construction along

```
 34.73:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBD(had) VP-C(VBN(caused) NP-C(NPB(DT(the) JJ(american) NNS(protests))))) .(.))
•34.74:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBD(had) VP-C(VBN(aroused) NP-C(NPB(DT(the) JJ(american) NNS(protests))))) .(.))
 34.83:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBD(had) VP-C(VBN(caused) NP-C(NPB(DT(the) JJ(american) NNS(protests))))) .(.))
•34.83:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBD(had) VP-C(VBN(aroused) NP-C(NPB(DT(the) JJ(american) NNS(protests))))) .(.))
 34.84:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBD(had) VP-C(VBN(caused) NP-C(NPB(DT(the) JJ(american) NNS(protests))))) .(.))
 34.85:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBD(had) VP-C(VBN(caused) NP-C(NPB(DT(the) JJ(american) NNS(protests))))) .(.))
•34.85:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBD(had) VP-C(VBN(aroused) NP-C(NPB(DT(the) JJ(american) NNS(protests))))) .(.))
•34.85:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBD(had) VP-C(VBN(aroused) NP-C(NPB(DT(the) JJ(american) NNS(protests))))) .(.))
 34.87:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBD(had) VP-C(VB(arouse) NP-C(NPB(DT(the) JJ(american) NNS(protests))))) .(.))
•34.92:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBD(had) VP-C(VBN(aroused) NP-C(NPB(DT(the) JJ(american) NNS(protests))))) .(.))
```

Figure 1: Ranked list of machine translation results with repeated trees. Scores shown are negative logs of calculated weights, thus a lower score indicates a higher weight. The bulleted sentences indicate identical trees.

with an innovative bookkeeping system to determinize the automaton, resulting in an automaton that preserves the language but provides a single, properly weighted derivation for each string in it. Put another way, if the input automaton has the ability to generate the same string with different weights, the output automaton generates that string with weight equal to the sum of all of the generations of that string in the input automaton. In (Mohri and Riley, 2002) this technique was combined with a procedure for efficiently obtaining n-best ranked lists, yielding a list of string results with no repetition.

In this paper we extend that work to deal with grammars that produce trees. *Regular tree grammars* (Brainerd, 1969), which subsume the *tree substitution grammars* developed in the NLP community (Schabes, 1990), are of particular interest to those wishing to work with additional levels of structure that string grammars cannot provide. The application to parsing is natural, and in machine translation tree grammars can be used to model syntactic transfer, control of function words, reordering, and target-language well-formedness. In the world of automata these grammars have as a natural dual the *finite tree recognizer* (Doner, 1970). Like tree grammars and packed forests, they are compact ways of representing very large sets of trees. We will present an algorithm for determinizing weighted finite tree recognizers, and use a variant of the procedure found in (Huang and Chiang, 2005) to obtain n-best lists of trees that are weighted correctly and contain no repetition.

Section 2 describes related work. In Section 3, we introduce the formalisms of tree automata, specifically the tree-to-weight transducer. In Section 4, we present the algorithm. Finally, in Section 5 we show the results of applying weighted determinization to recognizers obtained from the packed forest output of two natural language tasks.

2 Previous Work

The formalisms of tree automata are summarized well in (Gecseg and Steinby, 1984). Bottom-up tree recognizers are due to (Thatcher and Wright, 1968), (Doner, 1970), and (Magidor and Moran, 1969). Top-down tree recognizers are due to (Rabin, 1969) and (Magidor and Moran, 1969). (Comon, et. al., 1997) show the determinization of unweighted finite-state tree automata, and prove its correctness. (Borchardt and Vogler, 2003) present determinization of weighted finite-state tree automata with a different method than the one we present here. While our method is applicable to finite tree sets, the previous method claims the ability to determinize some classes of infinite tree sets. However, for the finite case the previous method produces an automaton with size on the order of the number of derivations, so the technique is limited when applied to real world data.

3 Grammars, Recognizers, and Transducers

As described in (Gecseg and Steinby, 1984), tree automata may be broken into two classes, recognizers and transducers. Recognizers read tree input and decide whether the input is in the language represented by the recognizer. Formally, a bottom-up tree recognizer R is defined by $R = (Q, \Sigma, i, F, E)$:[1]

- Q is a finite set of states,

[1] Readers familiar with (Gecseg and Steinby, 1984) will notice that we have introduced a start state, modified the notion of initial assignment, and changed the arity of nullary symbols to unary symbols. This is to make tree automata more palatable to those accustomed to string automata and to allow for a useful graphical interpretation.

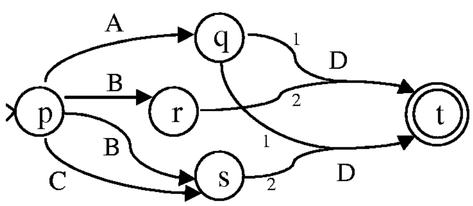

Figure 2: Visualization of a bottom-up tree recognizer

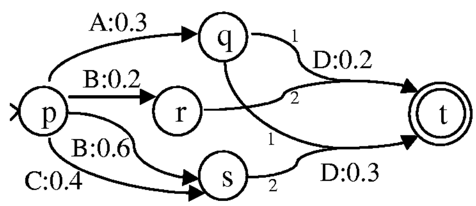

Figure 3: Bottom-up tree-to-weight transducer

- Σ is a ranked alphabet,
- $i \in Q$ is the initial state,
- $F \subseteq Q$ is a set of final states, and
- $E \subseteq \overbrace{Q \times ... \times Q}^{k} \times \Sigma^{(k)} \times Q$ is a finite set of transitions from a vector of k states to one state that reads a k-ary symbol.

Consider the following tree recognizer:

- $Q = \{p, q, r, s, t\}$
- $\Sigma = \{D/2, A/1, B/1, C/1\}^2$
- $i = p$
- $F = \{t\}$
- $E = \{(\langle p \rangle, A, q), (\langle p \rangle, B, r), (\langle p \rangle, B, s), (\langle p \rangle, C, s), (\langle q, r \rangle, D, t), (\langle q, s \rangle, D, t)\}$

As with string automata, it is helpful to have a visualization to understand what the recognizer is recognizing. Figure 2 provides a visualization of the recognizer above. Notice that some members of E are drawn as arcs with multiple (and ordered) tails. This is the key difference in visualization between string and tree automata – to capture the arity of the symbol being read we must visualize the automata as an ordered hypergraph.

The function of the members of E in the hypergraph visualization leads us to refer to the vector of k states as an *input vector* of states, and the single state as an *output state*. We will refer to E as the *transition set* of the recognizer.

In string automata, a *path* through a recognizer consists of a sequence of edges that can be followed from a start to an end state. The concatenation of labels of the edges of a path, typically in a left-to-right order, forms a string in the recognizer's language. In tree automata, however, a *hyperpath* through a recognizer consists of a sequence of hyperedges that can be followed, sometimes in parallel, from a start to an end state. We arrange the labels of the hyperedges to form a tree in the recognizer's language but must now consider proper order in two dimensions. The proper vertical order is specified by the order of application of transitions, i.e., the labels of transitions followed earlier are placed lower in the tree than the labels of transitions followed later. The proper horizontal order within one level of the tree is specified by the order of states in a transition's input vector. In the example recognizer, the trees $\overset{D}{\widehat{A \quad B}}$ and $\overset{D}{\widehat{A \quad C}}$ are valid. Notice that $\overset{D}{\widehat{A \quad B}}$ may be recognized in two different hyperpaths.

Like tree recognizers, tree transducers read tree input and decide whether the input is in the language, but they simultaneously produce some output as well. Since we wish to associate a weight with every acceptable tree in a language, we will consider transducers that produce weights as their output. Note that in transitioning from recognizers to transducers we are following the convention established in (Mohri, 1997) where a transducer with weight outputs is used to represent a weighted recognizer. One may consider the determinization of tree-to-weight transducers as equivalent to the determinization of weighted tree recognizers.

Formally, a bottom-up tree-to-weight transducer T is defined by $T = (Q, \Sigma, i, F, E, \lambda, \rho)$ where Q, Σ, i, and F are defined as for recognizers, and:

- $E \subseteq \overbrace{Q \times ... \times Q}^{k} \times \Sigma^{(k)} \times Q \times \mathcal{R}_+$ is a finite set of transitions from a vector of k states to one state, reading a k-ary symbol and outputting some weight
- λ is the initial weight function mapping i to $\mathcal{R}_+$
- ρ is the final weight function mapping F

[2] The number denotes the arity of the symbol.

to $\mathcal{R}_+$.

We must also specify a convention for propagating the weight calculated in every transition. This can be explicitly defined for each transition but we will simplify matters by defining the propagation of the weight to a destination state as the multiplication of the weight at each source state with the weight of the production.

We modify the previous example by adding weights as follows: As an example, consider the following tree-to-weight transducer (Q, Σ, i, and F are as before):

- $E = \{(\langle p \rangle, A, q, .3), (\langle p \rangle, B, r, .2), (\langle p \rangle, B, s, .6), (\langle p \rangle, C, s, .4), (\langle q, r \rangle, D, t, .2), (\langle q, s \rangle, D, t, .3)\}$
- $\lambda = \rho = 1$

Figure 3 shows the addition of weights onto the automata, forming the above transducer. Notice the tree $\begin{smallmatrix} & D & \\ A & & C \end{smallmatrix}$ yields the weight 0.036 ($0.3 \times 0.3 \times 0.4$), and $\begin{smallmatrix} & D & \\ A & & B \end{smallmatrix}$ yields the weight 0.012 ($0.2 \times 0.3 \times 0.2$) or 0.054 ($0.3 \times 0.3 \times 0.6$), depending on the hyperpath followed.

This transducer is an example of a *nonsubsequential* transducer. A tree transducer is *subsequential* if for each vector $\mathbf{v}$ of k states and each $x \in \Sigma^{(k)}$ there is at most one transition in E with input vector $\mathbf{v}$ and label x. These restrictions ensure a subsequential transducer yields a single output for each possible input, that is, it is deterministic in its output.

Because we will reason about the destination state of a transducer transition and the weight of a transducer transition separately, we make the following definition. For a given $e = (\mathbf{v}, x, q, w) \in E$ where $\mathbf{v}$ is a vector of k states, $x \in \Sigma^{(k)}$, $q \in Q$, and $w \in \mathcal{R}_+$, let $\delta(\mathbf{v}, x) = q$ and $\sigma(\mathbf{v}, x) = w$. Equivalent shorthand forms are $\delta(e)$ and $\sigma(e)$.

4 Determinization

The determinization algorithm is presented as Algorithm 1. It takes as input a bottom-up tree-to-weight transducer τ_1 and returns as output a subsequential bottom-up tree-to-weight transducer τ_2 such that the tree language recognized by τ_2 is equivalent to that of τ_1 and the output weight given input tree t on τ_2 is equal to the sum of all possible output weights given t on τ_1. Like the algorithm of (Mohri, 1997), this algorithm will terminate for automata that recognize finite tree languages. It may terminate on some automata that recognize infinite tree languages, but we do not consider any of these cases in this work.

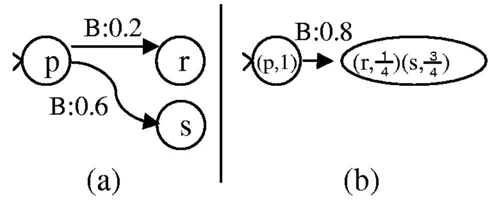

Figure 4: a) Portion of a transducer before determinization; b) The same portion after determinization

Determinizing a tree-to-weight transducer can be thought of as a two-stage process. First, the structure of the automata must be determined such that a single hyperpath exists for each recognized input tree. This is achieved by a classic powerset construction, i.e., a state must be constructed in the output transducer that represents all the possible reachable destination states given an input and a label. Because we are working with tree automata, our input is a vector of states, not a single state. A comparable powerset construction on unweighted tree automata and a proof of correctness can be found in (Comon, et. al., 1997).

The second consideration to weighted determinization is proper propagation of weights. For this we will use (Mohri, 1997)'s concept of the *residual weight*. We represent in the construction of states in the output transducer not only a subset of states of the input transducer, but also a number associated with each of these states, called the residual. Since we want τ_2's hyperpath of a particular input tree to have as its associated weight the sum of the weights of the all of τ_1's hyperpaths of the input tree, we replace a set of hyperedges in τ_1 that have the same input state vector and label with a single hyperedge in τ_2 bearing the label and the sum of τ_1's hyperedge weights. The destination state of the τ_2 hyperedge represents the states reachable by τ_1's applicable hyperedges and for each state, the proportion of the weight from the relevant τ_1 transition.

Figure 4 shows the determinization of a portion of the example transducer. Note that the hyperedge

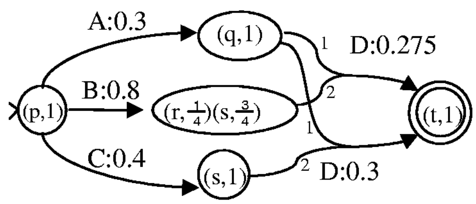

Figure 5: Determinized bottom-up tree-to-weight transducer

leading to state r in the input transducer contributes $\frac{1}{4}$ of the weight on the output transducer hyperedge and the hyperedge leading to state s in the input transducer contributes the remaining $\frac{3}{4}$. This is reflected in the state construction in the output transducer. The complete determinization of the example transducer is shown in Figure 5.

To encapsulate the representation of states from the input transducer and associated residual weights, we define a state in the output transducer as a set of (q, w) tuples, where $q \in Q_1$ and $w \in \mathcal{R}_+$. Since the algorithm builds new states progressively, we will need to represent a vector of states from the output transducer, typically depicted as $\mathbf{v}$. We may construct the vector pair $(\mathbf{q}, \mathbf{w})$ from $\mathbf{v}$, where $\mathbf{q}$ is a vector of states of the input transducer and $\mathbf{w}$ is a vector of residual weights, by choosing a (state, weight) pair from each output state in $\mathbf{v}$. For example, let $Q_1 = \{a, b, c\}$. Then two possible output transducer states could be $x = ((a, 1))$ and $y = ((a, 0.25), (c, 0.75))$. If we choose $\mathbf{v} = \langle y, x \rangle$ then a valid vector pair $(\mathbf{q}, \mathbf{w})$ is $\mathbf{q} = \langle c, a \rangle$, $\mathbf{w} = \langle 0.75, 1 \rangle$.

The sets $\Gamma(\mathbf{v}, x)$, $\gamma(\mathbf{v}, x)$, and $\nu(\mathbf{v}, x)$ are defined as follows:

- $\Gamma(\mathbf{v}, x) = \{(\mathbf{q}, \mathbf{w})$ from $\mathbf{v} : \exists e = (\mathbf{q}, x, \delta_1(e), \sigma_1(e)) \in E_1\}$.
- $\gamma(\mathbf{v}, x) = \{(\mathbf{q}, \mathbf{w}, e)$ from $\mathbf{v} \times E_1 : \exists e = (\mathbf{q}, x, \delta_1(e), \sigma_1(e)) \in E_1\}$.
- $\nu(\mathbf{v}, x) = \{q' : \exists(\mathbf{q}, \mathbf{w})$ from $\mathbf{v}, \exists e = (\mathbf{q}, x, q', \sigma_1(e)) \in E_1\}$.

.

$\Gamma(\mathbf{v}, x)$ is the set of vector pairs $(\mathbf{q}, \mathbf{w})$ constructed from $\mathbf{v}$ where each $\mathbf{q}$ is an input vector in a transition $e \in E_1$ with label x. $\gamma(\mathbf{v}, x)$ is the set of unique transitions e paired with the appropriate pair for each $(\mathbf{q}, \mathbf{w})$ in $\Gamma(\mathbf{v}, x)$. $\nu(\mathbf{v}, x)$ is the set of states reachable from the transitions in $\Gamma(\mathbf{v}, x)$.

The consideration of *vectors* of states on the incident edge of transitions effects two noticeable changes on the algorithm as it is presented in (Mohri, 1997). The first, relatively trivial, change is the inclusion of the residual of multiple states in the calculation of weights and residuals on lines 16 and 17. The second change is the production of vectors for consideration. Whereas the string-based algorithm considered newly-created states in turn, we must consider newly-available vectors. For each newly created state, newly available vectors can be formed by using that state with the other states of the output transducer. This operation is performed on lines 7 and 22 of the algorithm.

5 Empirical Studies

We now turn to some empirical studies. We examine the practical impact of the presented work by showing:

- That the multiple derivation problem is pervasive in practice and determinization is effective at removing duplicate trees.
- That duplication causes misleading weighting of individual trees and the summing achieved from weighted determinization corrects this error, leading to re-ordering of the n-best list.
- That weighted determinization positively affects end-to-end system performance.

We also compare our results to a commonly used technique for estimation of n-best lists, i.e., summing over the top $k > n$ derivations to get weight estimates of the top $m \leq k$ unique elements.

5.1 Machine translation

We obtain packed-forest English outputs from 116 short Chinese sentences computed by a string-to-tree machine translation system based on (Galley, et. al., 2004). The system is trained on all Chinese-English parallel data available from the Linguistic Data Consortium. The decoder for this system is a CKY algorithm that negotiates the space described in (DeNeefe, et. al., 2005). No language model was used in this experiment.

The forests contain a median of 1.4×10^{12} English parse trees each. We remove cycles from each

Algorithm 1: Weighted Determinization of Tree Automata

Input: **BOTTOM-UP TREE-TO-WEIGHT TRANSDUCER** $\tau_1 = (Q_1, \Sigma, i_1, F_1, E_1, \lambda_1, \rho_1)$.
Output: **SUBSEQUENTIAL BOTTOM-UP TREE-TO-WEIGHT TRANSDUCER** $\tau_2 = (Q_2, \Sigma, i_2, F_2, E_2, \lambda_2, \rho_2)$.

begin
  $\lambda_2 \longleftarrow \lambda_1(i_1)$
  $i_2 \longleftarrow \{(i_1, 1)\}$
  PRIORITY QUEUE $P \longleftarrow \emptyset$
  $Q_2 \longleftarrow \{i_2\}$
  $F_2 \longleftarrow \emptyset$
  ENQUEUE$(P, \langle i_2 \rangle)$
  while $P \neq \emptyset$ **do**
    $\mathbf{v} \longleftarrow$ *head*$[P]$
    $k \longleftarrow |\mathbf{v}|$
    for *each* $p \in \boldsymbol{v}$ *such that* $p \notin F_2$ **do**
      if $\exists (q, w) \in p$ *such that* $q \in F_1$ **then**
        $\rho_2(p) \longleftarrow \sum_{(q,w) \in p \text{ s.t. } q \in F_1} w \times \rho_1(q)$
        $F_2 \longleftarrow F_2 \cup p$
    for *each* $x \in \Sigma^{(k)}$ *such that* $\Gamma(\boldsymbol{v}, x) \neq \emptyset$ **do**
      $\sigma_2(\mathbf{v}, x) \longleftarrow \sum_{((q_1,w_1),\ldots,(q_i,w_i)) \in \Gamma(\mathbf{v},x)} [w_1 \times \ldots \times w_i \times \sum_{e=(\langle q_1,\ldots,q_i \rangle, x, \delta_1(e), \sigma_1(e)) \in E_1} \sigma_1(e)]$
      $\delta_2(\mathbf{v}, x) \longleftarrow \bigcup_{q' \in \nu(\mathbf{v},x)} \{(q', \frac{1}{\sigma_2(\mathbf{v},x)} \times \sum_{((q_1,w_1),\ldots,(q_i,w_i),e) \in \gamma(\mathbf{v},x), \text{ s.t. } \delta_1(e)=q'} w_1 \times \ldots \times w_i \times \sigma_1(e))\}$
      $E_2 \longleftarrow E_2 \cup (\mathbf{v}, x, \delta_2(\mathbf{v}, x), \sigma_2(\mathbf{v}, x))$
      `/*` ***RANK***(i) `returns the largest hyperedge size that can leave state` i.
      COMBINATIONS(Q, N, k) `returns all possible vectors of length` $1..k$
      `containing members of` Q `and at least one member of` N. `*/`
      if $\delta_2(\boldsymbol{v}, x)$ *is a new state* **then**
        for *each* $\boldsymbol{u} \in$ **COMBINATIONS**$(Q_2, \{\delta_2(\boldsymbol{v}, x)\}, \max_{q \in p \in Q_2 \cup \{\delta_2(\boldsymbol{v},x)\}}$ **RANK**$(q))$ **do**
          if $\boldsymbol{u}$ *is a new vector* **then**
            ENQUEUE$(P, \mathbf{u})$
        $Q_2 \longleftarrow Q_2 \cup \{\delta_2(\mathbf{v}, x)\}$
    DEQUEUE(P)
end

forest,[3] apply our determinization algorithm, and extract the n-best trees using a variant of (Huang and Chiang, 2005). The effects of weighted determinization on an n-best list are obvious to casual inspection. Figure 7 shows the improvement in quality of the top 10 trees from our example translation after the application of the determinization algorithm.

The improvement observed circumstantially holds up to quantitative analysis as well. The forests obtained by the determinized grammars have between 1.39% and 50% of the number of trees of their undeterminized counterparts. On average, the determinized forests contain 13.7% of the original number of trees. Since a determinized forest contains no repeated trees but contains exactly the same unique trees as its undeterminized counterpart, this indicates that an average of 86.3% of the trees in an undeterminized MT output forest are duplicates.

Weighted determinization also causes a surprisingly large amount of n-best reordering. In 77.6% of the translations, the tree regarded as "best" is different after determinization. This means that in a large majority of cases, the tree with the highest weight is not recognized as such in the undeterminized list because its weight is divided among its multiple derivations. Determinization allows these instances and their associated weights to combine and puts the highest weighted tree, not the highest weighted derivation, at the top of the list.

[3] As in (Mohri, 1997), determinization may be applicable to some automata that recognize infinite languages. In practice, cycles in tree automata of MT results are almost never desired, since these represent recursive insertion of words.

method	Bleu
undeterminized	21.87
top-500 “crunching”	23.33
determinized	24.17

Figure 6: Bleu results from string-to-tree machine translation of 116 short Chinese sentences with no language model. The use of best derivation (undeterminized), estimate of best tree (top-500), and true best tree (determinized) for selection of translation is shown.

We can compare our method with the more commonly used methods of “crunching” k-best lists, where $k > n$. The duplicate sentences in the k trees are combined, hopefully resulting in at least n unique members with an estimation of the true tree weight for each unique tree. Our results indicate this is a rather crude estimation. When the top 500 derivations of the translations of our test corpus are summed, only 50.6% of them yield an estimated highest-weighted tree that is the same as the true highest-weighted tree.

As a measure of the effect weighted determinization and its consequential re-ordering has on an actual end-to-end evaluation, we obtain Bleu scores for our 1-best translations from determinization, and compare them with the 1-best translations from the undeterminized forest and the 1-best translations from the top-500 “crunching” method. The results are tabulated in Figure 6. Note that in 26.7% of cases determinization did not terminate in a reasonable amount of time. For these sentences we used the best parse from top-500 estimation instead. It is not surprising that determinization may occasionally take a long time; even for a language of monadic trees (i.e. strings) the determinization algorithm is NP-complete, as implied by (Casacuberta and de la Higuera, 2000) and, e.g. (Dijkstra, 1959).

5.2 Data-Oriented Parsing

Weighted determinization of tree automata is also useful for parsing. Data-Oriented Parsing (DOP)’s methodology is to calculate weighted derivations, but as noted in (Bod, 2003), it is the highest ranking parse, not derivation, that is desired. Since (Sima’an, 1996) showed that finding the highest ranking parse is an NP-complete problem, it has been common to estimate the highest ranking parse by the previously described “crunching” method.

method	Recall	Precision	F-measure
undeterminized	80.23	80.18	80.20
top-500 “crunching”	80.48	80.29	80.39
determinized	81.09	79.72	80.40

Figure 8: Recall, precision, and F-measure results on DOP-style parsing of section 23 of the Penn Treebank. The use of best derivation (undeterminized), estimate of best tree (top-500), and true best tree (determinized) for selection of parse output is shown.

We create a DOP-like parsing model[4] by extracting and weighting a subset of subtrees from sections 2-21 of the Penn Treebank and use a DOP-style parser to generate packed forest representations of parses of the 2416 sentences of section 23. The forests contain a median of 2.5×10^{15} parse trees. We then remove cycles and apply weighted determinization to the forests. The number of trees in each determinized parse forest is reduced by a factor of between 2.1 and 1.7×10^{14}. On average, the number of trees is reduced by a factor of 900,000, demonstrating a much larger number of duplicate parses prior to determinization than in the machine translation experiment. The top-scoring parse after determinization is different from the top-scoring parse before determinization for 49.1% of the forests, and when the determinization method is “approximated” by crunching the top-500 parses from the undeterminized list only 55.9% of the top-scoring parses are the same, indicating the crunching method is not a very good approximation of determinization. We use the standard F-measure combination of recall and precision to score the top-scoring parse in each method against reference parses. The results are tabulated in Figure 8. Note that in 16.9% of cases determinization did not terminate. For those sentences we used the best parse from top-500 estimation instead.

6 Conclusion

We have shown that weighted determinization is useful for recovering n-best unique trees from a weighted forest. As summarized in Figure 9, the

[4]This parser acquires a small subset of subtrees, in contrast with DOP, and the beam search for this problem has not been optimized.

```
31.87:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBD(had) VP-C(VBN(aroused) NP-C(NPB(DT(the) JJ(american) NNS(protests))))) .(.))
32.11:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBD(had) VP-C(VBN(caused) NP-C(NPB(DT(the) JJ(american) NNS(protests))))) .(.))
32.15:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBD(had) VP-C(VB(arouse) NP-C(NPB(DT(the) JJ(american) NNS(protests))))) .(.))
32.55:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBD(had) VP-C(VB(cause) NP-C(NPB(DT(the) JJ(american) NNS(protests))))) .(.))
32.60:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBD(had) VP-C(VBN(attracted) NP-C(NPB(DT(the) JJ(american) NNS(protests))))) .(.))
33.16:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBD(had) VP-C(VB(provoke) NP-C(NPB(DT(the) JJ(american) NNS(protests))))) .(.))
33.27:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBG(causing) NP-C(NPB(DT(the) JJ(american) NNS(protests)))) .(.))
33.29:  S(NP-C(NPB(DT(this) NN(case))) VP(VBD(had) VP-C(VBN(aroused) NP-C(NPB(DT(the) JJ(american) NNS(protests))))) .(.))
33.31:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBD(had) VP-C(VBN(aroused) NP-C(NPB(DT(the) NN(protest)) PP(IN(of) NP-C(NPB(DT(the)
NNS(united_states))))))) .(.))
33.33:  S(NP-C(NPB(DT(this) NNS(cases))) VP(VBD(had) VP-C(VBN(incurred) NP-C(NPB(DT(the) JJ(american) NNS(protests))))) .(.))
```

Figure 7: Ranked list of machine translation results with no repeated trees.

experiment	undeterminized	determinized
machine translation	1.4×10^{12}	2.0×10^{11}
parsing	2.5×10^{15}	2.3×10^{10}

Figure 9: Median trees per sentence forest in machine translation and parsing experiments before and after determinization is applied to the forests, removing duplicate trees.

number of repeated trees prior to determinization was typically very large, and thus determinization is critical to recovering true tree weight. We have improved evaluation scores by incorporating the presented algorithm into our MT work and we believe that other NLP researchers working with trees can similarly benefit from this algorithm.

Further advances in determinization will provide additional benefit to the community. The translation system detailed here is a string-to-tree system, and the determinization algorithm returns the n-best unique trees from a packed forest. Users of MT systems are generally interested in the string yield of those trees, and not the trees per se. Thus, an algorithm that can return the n-best unique *strings* from a packed forest would be a useful extension.

We plan for our weighted determinization algorithm to be one component in a generally available tree automata package for intersection, composition, training, recognition, and generation of weighted and unweighted tree automata for research tasks such as the ones described above.

Acknowledgments

We thank Liang Huang for fruitful discussions which aided in this work and David Chiang, Daniel Marcu, and Steve DeNeefe for reading an early draft and providing useful comments. This work was supported by NSF grant IIS-0428020.

References

Rens Bod. 1992. A Computational model of language performance: data oriented parsing. In *Proc. COLING*

Rens Bod. 2003. An efficient implementation of a new DOP model. In *Proc. EACL*,

Björn Borchardt and Heiko Vogler. 2003. Determinization of finite state weighted tree automata. *Journal of Automata, Languages and Combinatorics*, 8(3).

W. S. Brainerd. 1969. Tree generating regular systems. *Information and Control*, 14.

F. Casacuberta and C. de la Higuera. 2000. Computational complexity of problems on probabilistic grammars and transducers. In *Proc. ICGI*.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proc. ACL*.

H. Comon and M. Dauchet and R. Gilleron and F. Jacquemard and D. Lugiez and S. Tison and M. Tommasi. 1997 *Tree Automata Techniques and Applications*.

S. DeNeefe and K. Knight and H. Chan. 2005. Interactively exploring a machine translation model. Poster in *Proc. ACL*.

Edsger W. Dijkstra 1959. A note on two problems in connexion with graphs *Numerische Mathematik*, 1.

J. E. Doner 1970. Tree acceptors and some of their applications *J. Comput. System Sci.*, 4.

M. Galley and M. Hopkins and K. Knight and D. Marcu. 2004. What's in a translation rule? In *Proc. HLT-NAACL*.

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

Liang Huang and David Chiang 2005. Better k-best parsing In *Proc. IWPT*.

Irene Langkilde and Kevin Knight 1998 The Practical Value of N-Grams in Generation In *Proc. INLG*.

M. Magidor and G. Moran. 1969. Finite automata over finite trees *Technical Report 30*. Hebrew University, Jerusalem.

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

Mehryar Mohri and Michael Riley. 2002. An efficient algorithm for the n-best strings problem. In *Proc. ICSLP*.

M. O. Rabin. 1969. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141.

Yves Schabes. 1990. *Mathematical and computational aspects of lexicalized grammars.* Ph.D. thesis. University of Pennsylvania, Philadelphia, PA.

Khalil Sima'an. 1996. Computational complexity of probabilistic disambiguation by means of tree-grammars. In *Proc. COLING*.

J. W. Thatcher and J. B. Wright. 1968. Generalized finite automata theory with an application to a decision problem of second order logic. *Mathematical Systems Theory*, 2.