On Computing Best Trees for Weighted Tree Automata

Johanna Björklund Frank Drewes
Niklas Zechner
Department of Computing Science, Umeå University, 901 87 Umeå, Sweden
{johanna,drewes,zechner}@cs.umu.se

We generalise a search algorithm by Mohri and Riley from strings to trees. The original algorithm takes as input a weighted automaton M over the tropical semiring, together with an integer N, and outputs N strings of minimal weight with respect to M. In our setting, M defines a weighted tree language, and the output is a set of N trees with minimal weight. We prove that the algorithm is correct, and that its time complexity is a low polynomial in N and the relevant size parameters of M.

1 Introduction

Tree automata are useful in natural language processing (NLP), not least to describe the derivation trees of context-free grammars in an automata-theoretic way. To allow analyses to be computed together with an associated confidence level or a probability, we can choose to equip transitions and final states with weights, i.e., to work with weighted tree automata (wta) [4]. This is convenient when there is a set of competing analyses to choose from, and we want to find an analysis that optimises some objective function f. Huang and Chiang [5] observe that even when it is not tractable to compute f for every possible analysis, we may still obtain a satisfactory approximation by first ranking the candidate analyses according to a simpler function, such as can be computed by a wta, and then finding an N-best list $a_1, \ldots, a_N$ according to this ranking, and finally optimising f over $\{a_1, \ldots, a_N\}$. Examples include reranking the hypotheses produced by parsers or translation systems, where the reranking is based on auxiliary language models or evaluation scores orthogonal to the first round of analysis; see, e.g. [3, 8].

There are other situations in which an N-best analysis can be used for approximation. Suppose for instance that the analysis is computed by a cascade of computational modules, a common architecture for NLP systems [5]. Each module typically comes with its own objective function, and the goal is to optimise these jointly. Although it might not be possible to compute the full set of outputs from each module, we may again settle for the N best outputs from each module, and propagate them downstream. In their paper, Huang and Chiang provide several examples of this technique, including (i) joint parsing and semantic role labeling, and (ii) combined information extraction and coreference resolution.

In the majority of the above-mentioned applications, the weights represent probabilities and are as such taken from the interval of real values between zero and one. However, for the sake of numerical precision, negative log likelihoods are used in the actual computations, and the min operation is used to find the most likely analysis. This makes the min-plus semiring (or *tropical semiring*) $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$ an appropriate structure for transition weights. Alternatively, the max-plus semiring $(\mathbb{R}_+ \cup \{-\infty\}, \max, +, -\infty, 0)$ may be used.

In this paper, we focus on the case where trees are associated with weights by means of a weighted tree automaton (wta) over the tropical semiring. Thus, the weight of a computation, called a run, is the sum of the weights of the rules applied, and the weight of a tree is the minimum of the set of all runs on that tree. Note that the latter is only relevant if the automaton is nondeterministic. In [5] Huang

Submitted to:
TTATT 2015

and Chiang give an $O(m + D \cdot N \log N)$ algorithm for (essentially) finding a set S of N best runs in an acyclic wta, where m is the number of transitions and D is the size of the largest run in S. However, as pointed out by Mohri and Riley [7], one would usually rather determine the N best *trees*, because the trees correspond to the analyses and it is not very useful to obtain the same analysis twice in an N-best list just because it corresponds to several distinct runs of the nondeterministic automaton that implements the weight assignment. Unfortunately, determining the N best trees is a harder problem. Part of the difficulty lies in the fact that weighted automata are not closed under determinisation. In fact, both in the string and in the tree case the set of weighted languages recognisable by deterministic weighted automata is a proper subset of those recognisable by nondeterministic weighted automata.

Mohri and Riley [7] solve the problem of finding the N best strings, where the input is a weighted string automaton (wsa) over the tropical semiring (and the number N). To avoid computing redundant paths, they apply Dijkstra's N-shortest paths algorithm to a determinised version of the input automaton. Their algorithm applies the determinisation algorithm under a lazy evaluation scheme to guarantee termination and keep the running time polynomial. We generalise this algorithm to weighted tree languages, while simplifying the technique by working directly with the input automaton rather than an on-the-fly determinisation. The frontier is no longer a set of paths, but rather a set of trees that are combined and recombined into new trees to drive the search. This increased dimensionality creates an efficiency problem which we solve by a pruning technique. Owing to space limitations, the proofs have been left out, but a detailed treatment is given in [1] and available as a technical report[1].

2 Preliminaries

We write $\mathbb{N}$ for the set of non-negative integers and $\mathbb{R}_+$ for the set of non-negative reals; $\mathbb{R}_+^\infty$ denotes $\mathbb{R}_+ \cup \{\infty\}$. For $n \in \mathbb{N}$, $[n] = \{1, \ldots, n\}$. In particular, $[0] = \emptyset$. The number of elements of a (possibly infinite) set S is written $|S|$. The empty string is denoted by λ.

The estimation of the running time of our algorithm contains the factor $\log r$, where r is the maximum rank of symbols in the ranked alphabet considered (see below for the definitions). To avoid the technical problem that $\log 1 = 0$ we use the convention that, throughout this paper, $\log r$ abbreviates $\max(1, \log r)$.

For a set A, an A-labelled *tree* is a function $t\colon D \to A$ where $D \subseteq \mathbb{N}^*$ is such that, for every $v \in D$, there exists a $k \in \mathbb{N}$ with $\{i \in \mathbb{N} \mid vi \in D\} = [k]$. We call D the *domain* of t and denote it by $dom(t)$. An element v of $dom(t)$ is called a *node of* t, and k is the *rank of* v. The *subtree* of $t \in T_\Sigma$ rooted at v is the tree t/v defined by $dom(t/v) = \{u \in \mathbb{N}^* \mid vu \in dom(t)\}$ and $t/v(u) = t(vu)$ for every $u \in \mathbb{N}^*$. If $t(\lambda) = f$ and $t/i = t_i$ for all $i \in [k]$, where k is the rank of λ in t, then we denote t by $f[t_1, \ldots, t_k]$. If $k = 0$, then $f[]$ is usually abbreviated as f. In other words, a tree t with domain $\{\lambda\}$ is identified with $t(\lambda)$.

A *ranked alphabet* is a finite set of symbols $\Sigma = \bigcup_{k \in \mathbb{N}} \Sigma_{(k)}$, partitioned into pairwise disjoint subsets $\Sigma_{(k)}$. For every $k \in \mathbb{N}$ and $f \in \Sigma_{(k)}$, the *rank* of f is $rank(f) = k$. The set T_Σ of all trees over Σ contains all Σ-labelled trees t such that the rank of every $v \in dom(t)$ coincides with the rank of $t(v)$. For a set T of trees we denote by $\Sigma(T)$ the set of all trees $f[t_1, \ldots, t_k]$ such that $f \in \Sigma_{(k)}$ and $t_1, \ldots, t_k \in T$.

Let Σ be a ranked alphabet and let $\Box \notin \Sigma$ be a special symbol of rank 0. The set of *contexts over* Σ is the set C_Σ consisting of all $c \in T_{\Sigma \cup \{\Box\}}$ such that there is exactly one $v \in dom(c)$ with $c(v) = \Box$. The substitution of a tree t for $\Box$ in c is defined as usual, and is denoted by $c[\![t]\!]$.

A *weighted tree language* over the tropical semiring is a mapping $L\colon T_\Sigma \to \mathbb{R}_+^\infty$, where Σ is a ranked alphabet. Such languages can be specified by the use of so-called weighted tree automata (wta), of which there exist variants with *final weights* and with *final states*. As shown by Borchardt [2] these two variants

[1] See `http://www8.cs.umu.se/research/uminf/index.cgi?year=2014&number=22`

are equivalent, and going from final weights to final states only requires a single additional state (which becomes the unique final state) and, in the worst case, twice as many transitions. This means that all results shown in this paper, including the running time estimations, hold for both types of wta.

Formally, a *weighted tree automaton* is a system $M = (Q, \Sigma, \delta, Q_f)$ where Q is a finite set of *states* which are considered as symbols of rank 0; Σ is a ranked alphabet of *input symbols* disjoint with Q; $\delta\colon \Sigma(Q) \times Q \to \mathbb{R}_+^\infty$ is the *transition function*; and $Q_f \subseteq Q$ is the set of *final states*. Note that the transition function δ can be specified as the set of all transition rules $f[q_1,\dots,q_k] \xrightarrow{w} q$ such that $\delta(f[q_1,\dots,q_k],q) = w \neq \infty$. In particular, transition rules whose weight is ∞ are not represented explicitly.

For convenience, we define the behaviour of M on trees in $T_{\Sigma\cup Q}$ as opposed to just T_Σ, where states are considered to be symbols of rank 0: The set of *runs* of M on $t \in T_{\Sigma\cup Q}$ is the set of all Q-labelled trees $\pi\colon dom(t) \to Q$ such that $\pi(v) = t(v)$ for all $v \in dom(t)$ with $t(v) \in Q$. A run π is *accepting* if $\pi(\lambda) \in Q_f$.

The *weight* of a run π on a tree $t = f[t_1,\dots,t_k]$ is defined as

$$w(\pi) = \sum_{v \in dom(t), t(v) \in \Sigma_{(k)}} \delta(t(v)[\pi(v1)\cdots\pi(vk)], \pi(v)) \ .$$

Now, let $M(t) = \min\{w(\pi) \mid \pi \text{ is an accepting run of } M \text{ on } t\}$ for every tree $t \in T_{\Sigma\cup Q}$. This defines the weighted tree language $\mathscr{W}_M\colon T_\Sigma \to \mathbb{R}_+^\infty$ *recognised by* M, namely $\mathscr{W}_M(t) = M(t)$ for all $t \in T_\Sigma$.

The problem we are concerned with in this paper is to compute N trees such that, according to M, there are no tress outside this set with smaller weight. For $N \in \mathbb{N}$, an acceptable solution is a set $T = \{t_1,\dots,t_N\} \subseteq T_\Sigma$ such that $M(t_i) \leq M(t)$ for all $i \in [N]$ and $t \in T_\Sigma \setminus T$. Similarly, for $N = \infty$, we seek an infinite set $T = \{t_1, t_2, \dots\} \subseteq T_\Sigma$ with $M(t_i) \leq M(t)$ for all $i \geq 1$ and $t \in T_\Sigma \setminus T$.

3 The Algorithm

We now present our algorithm for computing N minimal trees with respect to a given wta. This is done in two steps: First a basic version is developed, which is later turned into a more efficient one by means of a pruning strategy. Correctness and efficiency are studied in Section 4. Throughout the paper, let $M = (Q, \Sigma, \delta, Q_f)$ be the wta given as input to the search algorithm. The letters m, n, and r denote the number $|\delta|$ of transition rules, the number $|Q|$ of states, and the maximum rank of symbols in Σ.

Our algorithm explores its search space recursively. The frontier of the explored part is organised as a priority queue. The algorithm iteratively selects a promising tree t from the queue, considers t for output, puts it into a set T of explored trees, and finally expands the frontier by all trees in $\Sigma(T)$ which have at least one occurrence of t as a direct subtree. For $t \in T \subseteq T_\Sigma$ this expansion is defined as

$$expand(T,t) = \{f[t_1,\dots,t_k] \in \Sigma(T) \mid t_i = t \text{ for at least one } i \in [k]\} \ .$$

To define our algorithm, it is convenient to consider two wtas M^q and M_q, for every $q \in Q$. The wta M^q is simply given by $M^q = (Q, \Sigma, \delta, \{q\})$, i.e. q becomes the unique final state. The wta M_q is given by $M_q = (Q, \Sigma \cup \{\Box\}, \delta \cup \{\Box \xrightarrow{0} q\}, Q_f)$. Note that $M_q(c) = M(c[\![q]\!])$ for all $c \in C_\Sigma$ and $q \in Q$.

The priority of a tree t in our queue is primarily decided by the minimal value of $M(c[\![t]\!])$, where c ranges over all possible contexts. To determine this, we compute for every $q \in Q$ the minimal value of $M_q(c) + M^q(t)$. Since M^q denotes the wta obtained from M by taking q as the unique final state, $M^q(t)$ is the minimal weight of all runs on t whose root state is q. Since $M_q(c)$ is independent of t, a c that minimises it can be calculated in advance using, e.g., Knuth's extension of Dijkstra's algorithm [6] (which, roughly speaking, computes the best derivation in a weighted context-free grammar). This yields the following lemma.

Lemma 1 *A family of contexts $(c_q)_{q\in Q}$ such that $M_q(c_q) = \min\{M_q(c) \mid c \in C_\Sigma\}$ for each $q \in Q$, can be computed in time $O(mr \cdot (\log n + r))$.*

In the rest of the paper, we frequently make use of the contexts c_q, assuming that they have been computed for all $q \in Q$. For a tree t in the frontier of our search space we are, intuitively, interested in the tree $c[\![t]\!]$ that has the least possible weight. Clearly, c can be assumed to be one of the contexts c_q. Thus, our aim has to be to determine the state q that minimises the weight of $c_q[\![t]\!]$.

Definition 1 (Optimal state) *The mapping optset: $T_\Sigma \to pow(Q)$, where $pow(Q)$ is the powerset of Q, is defined by*

$$optset(t) = \{q \in Q \mid M_q(c_q) + M^q(t) = \min_{c\in C_\Sigma} M(c[\![t]\!])\} \ .$$

In addition, let $opt(t)$ denote an arbitrary but fixed element of $optset(t)$, for every $t \in T_\Sigma$.

We can now give our basic algorithm, which we formulate only for wta computing *monotone* weighted tree languages. Here, a weighted tree language L is called *monotone* if, for all trees $t \in T_\Sigma$ and all $c \in C_\Sigma \setminus \{\Box\}$, $L(t) \neq \infty$ implies $L(c[\![t]\!]) \geq L(t)$. To see that this does not diminish the usefulness of the algorithm, notice that an arbitrary input wta M can be made monotone as follows: Introduce a new symbol *out* of rank 1 and turn M into M' such that $M'(t) = \infty$ and $M'(out[t]) = M(t)$ for all $t \in T_\Sigma$. This can easily be achieved by adding a new state q_f, which becomes the unique final state, and transitions $out[q] \xrightarrow{0} q_\mathrm{f}$ for $q \in Q_\mathrm{f}$. Then M' is monotone and if $out[t_1], \ldots, out[t_N]$ are N trees of minimal weight with respect to M', then $t_1, \ldots, t_N$ are minimal with respect to M.

Our basic algorithm is presented in Algorithm 1. It maintains three data structures: T is a set of trees that represents the explored search space, K is a priority queue of trees in $\Sigma(T)$, and C is a table containing the value $M^q(t)$, for all $q \in Q$ and $t \in T \cup K$. The table C can easily be updated whenever new trees are added to K. The priority order $\leq_K$ of K is given by

$$t <_K t' \Rightarrow \Delta(t) < \Delta(t') \text{ or } \Delta(t) = \Delta(t') \text{ and } t <_{lex} t'$$
$$\text{where } \Delta(s) = M(c_{opt(s)}[\![s]\!]) \text{ for all } s \in T_\Sigma.$$

Here, $<_{lex}$ is any lexiographical order that orders trees first by size and then lexiographically. Note that the output condition in Line 8 cannot be replaced by the more intuitive $M(t) < \infty$ because it has to cover the case where $\Delta(t) = \infty$ (which happens if there are fewer than N trees of finite weight).

Unfortunately, Algorithm 1 builds a large number of trees and is thus not very efficient. Therefore, we now give a more efficient version that works by repeatedly pruning the priority queue.

The idea of the pruning step is that a tree s can be discarded from the queue if we already have, for every state $q \in optset(s)$, at least N other trees $t <_K s$ such that $q \in optset(t)$. Intuitively, in this case we have sufficiently many good alternatives to s in the formation of a set of minimal trees, so that s will not be needed. A polynomial runtime is thus obtained through the addtion of a new procedure *Prune* (see Algorithm 2). In Algorithm 1, we replace Line 3 by $Prune(T, \text{enqueue}(K, \Sigma_0))$, and Line 12 by $Prune(T, \text{enqueue}(K, expand(T,t)))$, thereby obtaining Algorithm 3 *BestTrees* (not listed explicitly, again due to space limitations).

4 Correctness and Efficiency

Let us now establish the correctness of Algorithms 1 and 3, and then study the efficiency of the latter. For this, we assume that $\Sigma \neq \Sigma_{(0)}$, so that T_Σ is infinite and hence N trees of minimal weight can always

Algorithm 1 Enumerate N trees of minimal weight for a wta M such that $\mathscr{W}_M$ is monotone

```
1:  procedure BestTreesBasic(M,N)
2:      T ← ∅; K ← ∅
3:      enqueue(K,Σ_0)
4:      i ← 0
5:      while i < N ∧ K nonempty do
6:          t ← dequeue(K)
7:          T ← T ∪ {t}
8:          if M(t) = Δ(t) then
9:              output(t)
10:             i ← i+1
11:         end if
12:         enqueue(K, expand(T,t))
13:     end while
14: end procedure
```

Algorithm 2 Prune the priority queue

```
1: procedure Prune(T,K)
2:     for s ∈ K do
3:         if |{t ∈ T ∪ K | q ∈ optset(t) and t <_K s}| ≥ N for all q ∈ optset(s) then
4:             discard(K,s)
5:         end if
6:     end for
7: end procedure
```

be found. It is clear that Algorithm 1 is correct if $\Sigma = \Sigma_{(0)}$ and terminates within $O(m)$ steps in this case. Throughout this section we will write $\mathit{BestTreesBasic}(M,N) = t_1,t_2,\ldots,t_l$ or $\mathit{BestTreesBasic}(M,N) = t_1,t_2,\ldots$ (and similarly for *BestTrees*) if running Algorithm 1 with the inputs M and N results in the (finite or infinite) sequence $t_1,t_2,\ldots,t_l$ or $t_1,t_2,\ldots$ of output trees.

Using the following simple lemma, we can prove the correctness of Algorithm 1.

Lemma 2 *Algorithm 1 never dequeues the same tree twice. Furthermore, if Algorithm 1 dequeues a tree t in T_Σ, then it has previously dequeued all trees s in T_Σ such that $s <_K t$. In particular, if a tree in $t \in T_\Sigma$ is dequeued, then all trees $s \in T_\Sigma$ with $\Delta(s) < \Delta(t)$ have been dequeued earlier.*

Theorem 1 (Correctness of Alg. 1) *For all $N \in \mathbb{N}$, $\mathit{BestTreesBasic}(M,N)$ terminates and returns N trees of minimal weight according to the wta M. Moreover, $\mathit{BestTreesBasic}(M,\infty) = t_1,t_2,\ldots$ consists of pairwise distinct trees such that, for each $i \in \mathbb{N}$ and every tree $t \in T_\Sigma \setminus \{t_1,\ldots,t_i\}$, $M(t) \geq M(t_i)$.*

In the following, let us say that a tree $s \in T_\Sigma$ is *discarded* in a run of $\mathit{BestTrees}(M,N)$ if it, at some stage, is considered in Line 2 of Algorithm 2, fulfills the pruning condition in Line 3, and is consequently removed from the queue in Line 4. Further, a tree $t \in T_\Sigma$ is *active* (with respect to the considered run of $\mathit{BestTrees}(M,N)$) if it contains no discarded subtrees.

Lemma 3 *Let $\mathit{BestTreesBasic}(M,\infty) = t_1,t_2,\ldots$ and consider the execution of $\mathit{BestTrees}(M,N)$ for some $N > 0$. Let $l \in \mathbb{N} \cup \{\infty\}$ be the number of active trees among $t_1,t_2,\ldots$, and let i_j be such that t_{i_j} is the jth active tree in $t_1,t_2,\ldots$, for all $j \leq l$. Then $\mathit{BestTrees}(M,N) = t_{i_1},t_{i_2},\ldots,t_{i_{\min(l,N)}}$.*

Using this, the correctness of Algorithm 3 is established.

Theorem 2 (Correctness of Alg. 3) *For all $N \in \mathbb{N}$, BestTrees(M,N) terminates and returns N trees of minimal weight according to the input wta M. Moreover, BestTrees$(M,\infty) = t_1, t_2, \ldots$ consists of pairwise distinct trees such that, for each $i \in \mathbb{N}$ and every tree $t \in T_\Sigma \setminus \{t_1, \ldots, t_i\}$, $M(t) \geq M(t_i)$.*

Let us now discuss the worst-case efficiency of *BestTrees*. A consequence of the pruning is that T can only grow to contain $N \cdot n$ trees, since at this point, the pruning will discard everything that is left in the queue. Since each execution of the 'while' loop increases the size of T, the body of the 'while' loop in *BestTrees* is executed at most $N \cdot n$ times. Using Lemma 1 together with Lemma 4 below, as well as the fact that the main loop of Algorithm 3 is executed at most Nn times, we obtain Theorem 3.

Lemma 4 *Prune$(K, Expand(T,t))$ is computable in time $O\big(\max(m \cdot (Nr + r\log r + N\log N), Nn^2)\big)$.*

Theorem 3 *BestTrees(M,N) runs in time $O\big(\max(Nmn \cdot (Nr + r\log r + N\log N), N^2n^3, mr^2)\big)$.*

It may be worthwhile to notice that the set T of Algorithm 3 is subtree closed, meaning that $t_1, \ldots, t_k \in T$ for every tree $f[t_1, \ldots, t_k] \in T$. Since all output trees of Algorithm 3 are in T, this means that the output of Algorithm 3 can be represented as a packed forest with $|T|$ nodes, i.e., of size $\leq N \cdot n$.

5 Conclusion and future work

Future work includes the implementation and integration of the algorithm into an open-source library for formal tree languages. On the theoretical side, we are interested in seeing further generalisations of the search algorithm, for example, from trees to directed acyclic graphs, or from the tropical semiring to some encompassing family of extremal semirings.

References

[1] Johanna Björklund, Frank Drewes & Niklas Zechner (2015): *An Efficient Best-Trees Algorithm for Weighted Tree Automata over the Tropical Semiring*. In: Language and Automata Theory and Applications – 9th International Conference, LATA 2015, Nice, France, Lecture Notes in Computer Science, Springer.

[2] Björn Borchardt (2004): *A Pumping Lemma and Decidability Problems for Recognizable Tree Series*. Acta Cybernetica 16, pp. 509–544.

[3] Michael Collins (2000): *Discriminative Reranking for Natural Language Parsing*. In: Computational Linguistics, Morgan Kaufmann, pp. 175–182.

[4] Zoltàn Fülöp & Heiko Vogler (2009): *Weighted Tree Automata and Tree Transducers*. In Manfred Droste, Werner Kuich & Heiko Vogler, editors: Handbook of Weighted Automata, Springer, pp. 313–403.

[5] Liang Huang & David Chiang (2005): *Better K-best Parsing*. In: Proceedings of the Conference on Parsing Technology 2005, Association for Computational Linguistics, pp. 53–64.

[6] Donald E. Knuth (1977): *A Generalization of Dijkstra's Algorithm*. Information Processing Letters 6, pp. 1–5.

[7] Mehryar Mohri & Michael Riley (2002): *An Efficient Algorithm for the n-Best-Strings Problem*. In: Proceedings of the Conference on Spoken Language Processing.

[8] Libin Shen (2004): *Discriminative reranking for machine translation*. In: Proceedings of HLT-NAACL 2004, pp. 177–184.