An Efficient Best-Trees Algorithm for Weighted Tree Automata over the Tropical Semiring

Johanna Björklund, Frank Drewes, and Niklas Zechner

Department of Computing Science, Umeå University, 901 87 Umeå, Sweden
{johanna,drewes,zechner}@cs.umu.se

Abstract. We generalise a search algorithm by Mohri and Riley from strings to trees. The original algorithm takes as input a weighted automaton A over the tropical semiring, together with an integer N, and outputs N strings of minimal weight with respect to A. In our setting, the input automaton defines a weighted tree language, again over the tropical semiring, 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, m, n, and r, where m and n are the number of transitions and the number of states of A, respectively, and r is the maximum rank of symbols in the input alphabet.

UMINF 14.22 ISSN 0348-0542

1 Introduction

Tree automata [6, 7, 3] are useful in natural language processing (NLP), not least to describe the derivation trees of context-free grammars and, in fact, any other type of context-free device in the sense of Mezei and Wright [14]. Since data-driven approaches were made feasible through the availability of large-scale corpora, weighted grammars have increased in popularity, and with them, so have weighted tree automata [12]. The weights assigned to transitions in these devices allow analyses to be computed together with, for example, an associated confidence level or a probability. This is helpful when we want to assess the quality of an analysis, or when there are several competing analyses to choose between.

At a higher level of abstraction, selecting the right analysis consists in evaluating some objective function f over the set A of all possible analyses, and choosing the analysis that maximises the value of f. Huang and Chiang [10] observe that it may not be tractable to compute $f(a)$ for every single $a \in A$, but that we may obtain a satisfactory approximation by ranking the elements of A, computing an N-best list $a_1, \ldots, a_N$ according to this ranking, and optimising f over $\{a_1, \ldots, a_N\}$. Examples include the reranking of the hypotheses produced by parsers or translation systems, based on auxiliary language models or evaluation scores orthogonal to the first round of analysis [2, 17, 9, 13].

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 [10]. 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, and propagate them downstream. In their paper Huang and Chiang provide several examples of this technique, including joint parsing and semantic role labeling [8], and combined information extraction and coreference resolution [18].

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 $(\mathbb{R} \cup \{\infty\}, \min, +, 0, \infty)$ an appropriate structure for transition weights [15]. The min-plus semiring is also known as the tropical semiring, and in an alternative formulation, the domain is $\mathbb{R} \cup \{-\infty\}$, and max replaces min as the additive operation.

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. Huang and Chiang [10] 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 D is the size of the largest run in S. However, as pointed out by Mohri and Riley [16], 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 weighted languages recognisable by deterministic weighted automata is a proper subset of those recognisable by nondeterministic weighted automata (see also Example 1). When the standard determinisation algorithm is applied to an automaton of this kind, the algorithm will not terminate but continue forever to build up an ever-increasing state space.

Example 1. Consider the weighted string language $\mathcal{W}$ over the alphabet $\Sigma = \{a, b, c\}$, defined by $\mathcal{W}(w) = \min_{\alpha \in \Sigma} |w|_\alpha$, where $|w|_\alpha$ denotes the number of occurrences of the symbol α in the string w. The language $\mathcal{W}$ is computable by a weighted string automaton (wsa) M over the tropical semiring: The automaton makes an initial nondeterminstic choice whether to count as or bs, and then proceeds accordingly, transitioning with weight 1 on the chosen symbol, and with weight 0 on all other symbols. However, there is no deterministic wsa *over the same semiring* that recognises the language $\mathcal{W}$.

Since every weighted string language may be viewed as a weighted tree language (of monadic trees), this gap in descriptive power does not close in the tree domain. To mention a non-monadic example, it is easy to find the length the shortest root-to-leaf path through a tree t with a nondeterministic weighted tree automaton (wta) over the tropical semiring. Each run computes the length of a nondeterministically chosen path by simple counting, so that the minimum of the weights of all runs is the length of the shortest path. Finding a deterministic wta to compute the same weighted language is however impossible.

Let us now return to the problem of finding the N best trees. Mohri and Riley [16] solve this problem for the string case, where the input is a wsa over the tropical semiring (and the number N). To avoid computing redundant paths, they apply Dijkstra's N-shortest paths algorithm [4] 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 (recall that the state-space of the determinised automaton may be infinite). The property of the tropical semiring that makes this possible is that paths of minimal weight are relatively short, something which is not true for other commonly used rings and semirings, e.g. the integers under normal addition and multiplication.

We generalise the search algorithm by Mohri and Riley [16] 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 of the tree case means that a straightforward adaptation of [16] no longer guarantees a polynomial run-time. We therefore propose to use a pruning technique to obtain an efficient algorithm. Finally, we prove correctness and include a formal complexity analysis, two elements that are missing in [16].

2 Preliminaries

We write $\mathbb{N}$ for the set of nonnegative integers and $\mathbb{N}^\infty$ for $\mathbb{N} \cup \{\infty\}$. For $n \in \mathbb{N}^\infty$, $[n] = \{i \in \mathbb{N} \mid 1 \leq i \leq n\}$. Thus, in particular, $[0] = \emptyset$ and $[\infty] = \mathbb{N}$. The cardinality of a set S is written $|S|$, and the powerset of S is denoted by $pow(S)$. We abbreviate the Cartesian product $S \times \cdots \times S$ with n identical factors by S^n, and the inclusion $d_i \in D_i$ for all $i \in [k]$ by $d_1 \cdots d_k \in D_1 \cdots D_k$. 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). This gives rise to the technical problem that the expression $O(\cdots \log r \cdots)$ evaluates to an incorrect $O(0)$ in the case $r = 1$ (i.e., when the input automaton is actually a weighted finite-state string automaton). Moreover, the expression yields an undefined result for $r = 0$. To avoid having to handle the case $r \leq 1$ explicitly as a special case, we thus use the convention that, throughout this paper, $\log r$ is used as an abbreviation of $\max(1, \log r)$.

For a set A, an A-labelled *tree* is a partial function $t\colon \mathbb{N}^* \to A$ such that the domain $dom(t)$ of t is a finite prefix-closed set, and for every $v \in dom(t)$ there exists a $k \in \mathbb{N}$ such that $\{i \in \mathbb{N} \mid vi \in dom(t)\} = [k]$. 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 $\Sigma = \bigcup_{k\in\mathbb{N}} \Sigma_{(k)}$ which is 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 consists of all Σ-labelled trees t such that the rank of every node $v \in dom(t)$ coincides with the rank of $t(v)$. For a set T of trees we denote by $\Sigma(T)$ the set of trees

$$\{f[t_1, \ldots, t_k] \mid k \in \mathbb{N}, f \in \Sigma_k, \text{ and } t_1, \ldots, t_k \in T\} \ .$$

In particular, T can be given by a set Q where every $q \in Q$, as mentioned above, is identified with the tree t such that $dom(t) = \{\lambda\}$ and $t(\lambda) = q$. Thus, in this case, $\Sigma(Q)$ consists of all trees of height 1 which have a symbol from Σ in their root and children in Q.

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

$$C_\Sigma = \{c \in T_{\Sigma\cup\{\Box\}} \mid \text{ there is exactly one } v \in dom(c) \text{ with } c(v) = \Box\} \ ,$$

Consider a context $c \in C_\Sigma$ and let $v \in dom(c)$ be the unique node such that $c(v) = \Box$. The *substitution* of a tree t into c is defined by $dom(c[\![t]\!]) = dom(c) \cup \{vu \mid u \in dom(t)\}$ and

$$c[\![t]\!](w) = \begin{cases} c(w) \text{ if } w \in dom(c) \setminus \{v\}, \text{ and} \\ t(u) \ \text{ if } w = vu \text{ for some } u \in dom(t) \ . \end{cases}$$

A *weighted tree language* over the tropical semiring is a mapping $L\colon T_\Sigma \to \mathbb{N}^\infty$, 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 a few slightly different variants. In particular, one may either use *final weights* or *final states*. As shown by Borchardt [1] these two variants are equivalent. Moreover, 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. For technical convenience, we therefore work with wta having final states.

Formally, a *weighted tree automaton* is a system $M = (Q, \Sigma, \delta, Q_\mathrm{f})$ where

- Q is a finite set of *states*;
- Σ is a ranked alphabet of *input symbols* disjoint with Q;
- $\delta\colon \Sigma(Q) \times Q \to \mathbb{N}^\infty$ is the *transition function*; and

– $Q_\mathrm{f} \subseteq Q$ is the set of *final states*.

Note that the transition function δ can be specified as a set of all transition rules $f[q_1, \ldots, q_k] \xrightarrow{w} q$ such that $\delta(f[q_1, \ldots, q_k], q) = w \neq \infty$. In particular, transition rules whose weight is ∞ are not mentioned explicitly. In the following, we let $|\delta|$ denote the number of transition rules describing δ.

For technical 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 $\pi(\lambda) \in Q_\mathrm{f}$.

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

$$w(\pi) = \begin{cases} \delta(f[\pi(1), \ldots, \pi(k)], \pi(\lambda)) + \sum_{i=1}^{k} w(\pi/i) & \text{if } f \in \Sigma, \text{ and} \\ 0 & \text{if } f \in Q \ . \end{cases}$$

Put differently,

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

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 $\mathcal{W}_M \colon T_\Sigma \to \mathbb{N}^\infty$ *recognised by* M, namely $\mathcal{W}_M(t) = M(t)$ for all $t \in T_\Sigma$.

In summary, in a wta M every run on a tree t has as its weight the sum of the weights of its transitions. The weight $M(t)$ of t is the minimum of the weights of the accepting runs on t. The problem we are concerned with in this paper is to compute N trees of minimal weight according to M. Thus, an acceptable solution is a set $T = \{t_1, \ldots, t_N\} \in T_\Sigma$ such that there does not exist any tree $t \notin T$ such that $M(t) < M(t_i)$ for some $i \in [N]$.

3 The Algorithm

We now develop our algorithm for computing N minimal trees with respect to a given wta. This will be done in two steps: First a basic version is developed, and second it is turned into a more efficient one by means of a pruning strategy. Correctness and efficiency will be studied in Section 5. Throughout the rest of this paper, let $M = (Q, \Sigma, \delta, Q_\mathrm{f})$ be the wta given as input to the search algorithm. Finding a single tree with minimum weight, that is, the case $N = 1$, boils down to finding a best run and returning the tree corresponding to this run. For this, we can use the generalisation of Dijkstra's algorithm proposed by Knuth [11]. Formulated in terms of wta, this result can be stated as follows.

Lemma 1 (Knuth [11]). *A tree t such that $M(t) \leq M(t')$ for all trees $t' \in T_\Sigma$ can be computed in time $O(m \cdot (\log n + r))$, where $m = |\delta|$, $n = |Q|$, and r is the maximum rank of symbols in Σ.*

Below, we will continue to use the letters m, n, and r to denote the number of transition rules, the number of states, and the maximum rank of symbols of the wta in question.

3.1 The Basic Algorithm

Our algorithm explores its search space recursively. The frontier of the explored part is organised as a priority queue. The algorithm iteratively selects a tree t with a minimal value $\Delta(t)$ 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. The function that computes this expansion based on T and t is defined as follows.

Definition 1 (Expansion). *Given a set of trees T and a single tree $t \in T$,*

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

To define our algorithm, it is convenient to consider two wta 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 $M_q = (Q, \Sigma \cup \{\Box\}, \delta \cup \{\Box \xrightarrow{0} q\}, Q_\mathrm{f})$. Note that $M_q(c) = M(c[\![q]\!])$ for every state $q \in Q$.

The priority of a tree t in our queue is primarily determined 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, the c that minimises it can be calculated in advance, and $M^q(t)$ can be obtained through a dynamic programming technique, as we shall see.

Definition 2 (Smallest completion). *A* smallest completion *of a state $q \in Q$ is a context c_q such that $M_q(c_q) = \min\{M_q(c) \mid c \in C_\Sigma\}$.*

Before continuing, let us note that smallest completions can be computed efficiently.

Lemma 2. *For each state $q \in Q$, a smallest completion c_q can be computed in time $O(mr \cdot (\log n + r))$.*

Proof. We modify the wta M in order to compute c_q by Lemma 1. The idea is to add a leaf symbol that represents q, and make sure that trees not containing this symbol exactly once will be assigned the weight ∞.

More precisely, let M' be obtained from M as follows. For every state p, add a new state p'. Moreover, add a new symbol $\overline{q}$ of rank 0 to Σ and a transition rule $\overline{q} \xrightarrow{0} q'$. For every original transition rule $f[q_1, \ldots, q_k] \xrightarrow{w} p$ and every $i \in [k]$, add the transition rule

$$f[q_1, \ldots, q_{i-1}, q_i', q_{i+1}, \ldots, q_k] \xrightarrow{w} p' \ .$$

Finally, make $\{p' \mid p \in Q_\mathrm{f}\}$ the set of final states of M'. Clearly, $M'(t) \neq \infty$ only if t contains exactly one occurrence of $\overline{q}$. The states of the form p' are used to remember that the subtree that has been processed contained exactly one occurrence of $\overline{q}$. Furthermore, for every context $c \in C_\Sigma$, we have $M'(c[\![\overline{q}]\!]) = M_q(c)$. Hence, Lemma 1 yields the result because there are only twice as many states in M' as in M, and only r times as many transition rules. □

Algorithm 1 Enumerate N trees of minimal weight in ascending order for a wta M such that $\mathcal{W}_M$ 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
```

The following lemma shows that $M^q(f[t_1, \ldots, t_k])$ can be computed from $M^p(t_i)$ for $p \in Q$ and $i \in [k]$. We omit the straightforward proof.

Lemma 3. *For every $t = f[t_1, \ldots, t_k] \in T_\Sigma$ and $q \in Q$,*

$$M^q(t) = \min_{q_1 \cdots q_k \in Q^k} \left\{ \delta(f[q_1, \ldots, q_k], q) + \sum M^{q_i}(t_i) \right\} \ .$$

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 3 (Optimal state). *Choose an arbitrary but fixed ordering of Q. The mappings $optset\colon T_\Sigma \to pow(Q)$ and $opt\colon T_\Sigma \to Q$ are defined by*

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

and $opt(t)$ is the minimal element in $optset(t)$, for every $t \in T_\Sigma$.

We can now give our basic algorithm. Rather than formulating the algorithm for arbitrary wta, we formulate it 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 $L(c[\![t]\!]) \geq L(t)$. In Section 3.3 it will be shown that the algorithm can easily be extended to avoid this restriction.

Our basic algorithm is presented in Algorithm 1. The algorithm maintains three data structures, namely T, K, and C:

- T is a set of trees that represents the explored search space;

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
```

- K is a priority queue of trees in $\Sigma(T)$ that represents the frontier of the search space. When convenient, we identify K with the set of all trees that it contains; and
- C is a table containing the value $M^q(t)$, for all $q \in Q$ and $t \in T \cup K$.

The queue K initially contains the trees in $\Sigma_{(0)}$. Let the ordering $\leq_K$ of K be such that for all trees t and t' in K

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

Here, $<_{lex}$ is any lexical order that orders trees first by size and then by alphabet symbols. It would, in fact, be sufficient to define $\leq_K$ as the partial order given by the condition above, not resolving ties with $<_{lex}$. However, assuming the order to be total simplifies some arguments, as does the following property, which will turn out to be useful in the proofs of Section 5.

Observation 1 *For all $s, t \in T_\Sigma$ and $c \in C_\Sigma$, if $s <_{lex} t$, then $c[\![s]\!] <_{lex} c[\![t]\!]$.*

3.2 An Efficient Algorithm Based on Pruning

As mentioned above, 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 t can be discarded from the queue if there are, for every state $q \in Q$, at least N other trees s_i, $i \in [N]$, with higher priority that are taken by M to q charging less weight than t. As it turns out, it suffices to focus on the states in $optset(t)$, which makes the pruning more efficient. However, to make the proofs easier, we shall require that q is also in $optset(s_i)$ for every $i \in [N]$, even though this is worse for the pruning.

Intuitively, the pruning condition is fulfilled if there are sufficiently many known good alternatives to t in the formation of a set of minimal trees. A polynomial runtime is thus obtained from the new procedure *Prune* (see Algorithm 2) at Lines 3 and 12 of Algorithm 1. This leads to Algorithm 3.

Algorithm 3 Compute N trees of minimal weight for a wta M s.t. $\mathcal{W}_M$ is monotone

```
1: procedure BestTrees(M, N)
2:     T ← ∅;
3:     Prune(T, 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:        Prune(T, enqueue(K, expand(T, t)))
13:    end while
14: end procedure
```

3.3 Removing the Monotonicity Assumption

To finish this section, let us see how Algorithm 3 can be used to compute N minimal trees even in the case where $\mathcal{W}_M$ is not monotone. For this, we modify the input wta in order to make it monotone. We 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.

4 Example

Before studying the correctness and efficiency of the algorithms in Section 5, let us have a look at an example. We consider the input automaton M illustrated in Figure 1, where $\Sigma = \Sigma_{(0)} \cup \Sigma_{(1)}$ with $\Sigma_{(0)} = \{a, b\}$ and $\Sigma_{(2)} = \{\circ\}$. Assume in the following that the lexical ordering places a before b before $\circ$. Let $\|t\|_\sigma$ ($\sigma \in \Sigma_{(0)}$) denote the number of occurrences of σ in a tree $t \in T_\Sigma$, and let $\|t\|$ denote the total number of leaves of t. Then we have

$$M(t) = \begin{cases} \|t\| + \min(\|t\|_a, \|t\|_b) & \text{if } \|t\| \text{ is even} \\ \infty & \text{otherwise .} \end{cases}$$

The algorithm indicated in the proof of Lemma 2 may yield $c_{p_a} = \circ[\Box, b]$, $c_{p_b} = \circ[\Box, a]$, and $c_{q_a} = c_{q_b} = \Box$. We note that

- $M_{p_\sigma}(c_{p_\sigma}) = 1$ and $M_{q_\sigma}(c_{q_\sigma}) = 0$ for $\sigma \in \Sigma_{(0)}$, and
- letting $\overline{a} = b$ and $\overline{b} = a$ we have, for all $t \in T_\Sigma$,

$$optset(t) = \begin{cases} \{p_\sigma \mid \sigma \in \Sigma_{(0)} \text{ and } \|t\|_\sigma \leq \|t\|_{\overline{\sigma}}\} & \text{if } \|t\| \text{ is odd} \\ \{q_\sigma \mid \sigma \in \Sigma_{(0)} \text{ and } \|t\|_\sigma \leq \|t\|_{\overline{\sigma}}\} & \text{if } \|t\| \text{ is even .} \end{cases}$$

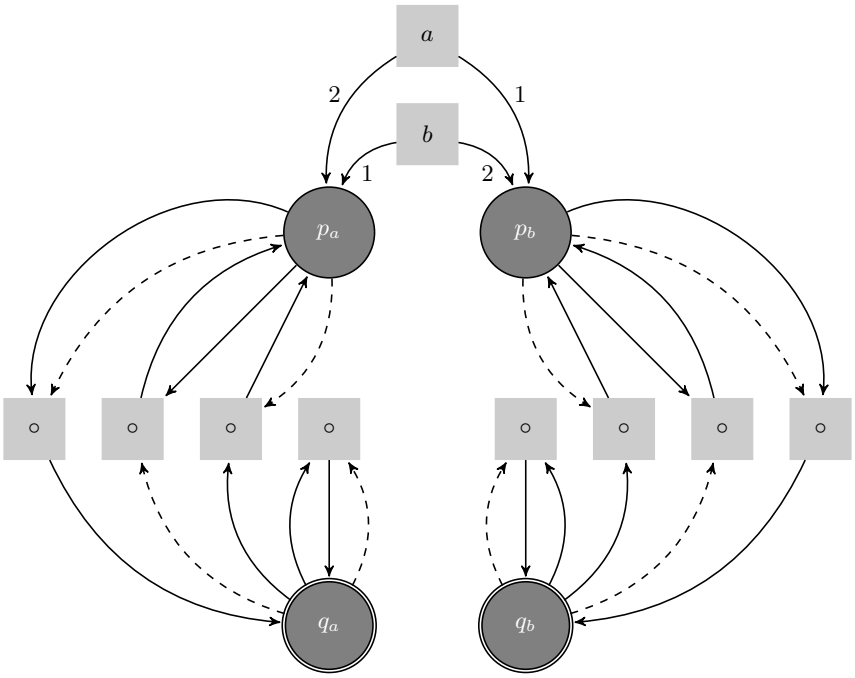

Fig. 1. The input wta considered as an example. The input alphabet is $\Sigma_{(0)} \cup \Sigma_{(2)}$, where $\Sigma_{(0)} = \{a, b\}$ and $\Sigma_{(2)} = \{\circ\}$. Round nodes (with double circles if final) represent states, and squares represent transitions. The consumed input symbols are shown inside the squares. Solid arcs point to the right-hand side of the transition in question and are labelled with the weight of the transition unless it is zero. In the case of input symbol $\circ$ the two states in the left-hand side of a transition are indicated by incoming solid and dashed arcs. Since the wta is symmetric, the latter distinction is, in fact, irrelevant.

To increase readability, let us denote trees in T_Σ without the binary symbol $\circ$. For example, $\circ[\circ[a, b], b]$ will be denoted by $[[a, b], b]$. To find $N = 3$ minimal trees with respect to M, Algorithm 3 proceeds as follows:

After initialisation, $T = \emptyset$ and K contains a and b, where $a <_K b$ because $\Delta(a) = 2 = \Delta(b)$ and $a <_{lex} b$. The iterations of the 'while' loop proceed as follows:

Step 1: dequeue a	
Output: none (as $\Delta(a) = 2 \neq M(a)$)	Expand with: $[a, a]$
T: $\boxed{a}$	K: $\boxed{b}$, $\boxed{[a, a]}$

Step 2: dequeue b	
Output: none (as $\Delta(b) = 2 \neq M(b)$)	Expand with: $\boxed{[b,b]}$, $\boxed{[b,a]}$, $\boxed{[a,b]}$
T: $\boxed{a}$, $\boxed{b}$	K: $\boxed{[a,a]}$, $\boxed{[b,b]}$, $\boxed{[a,b]}$, $\boxed{[b,a]}$

Step 3: dequeue $[a,a]$	
Output: $[a,a]$ (as $\Delta([a,a]) = 2 = M([a,a])$)	Expand with: $\boxed{[[a,a],[a,a]]}$, $\boxed{[[a,a],a]}$, $\boxed{[[a,a],b]}$, $\boxed{[a,[a,a]]}$, $\boxed{[b,[a,a]]}$
T: $\boxed{a}$, $\boxed{b}$, $\boxed{[a,a]}$	K: $\boxed{[b,b]}$, $\boxed{[a,b]}$, $\boxed{[b,a]}$, $\boxed{[a,[a,a]]}$, $\boxed{[[a,a],a]}$, $\boxed{[[a,a],[a,a]]}$, $\boxed{[b,[a,a]]}$, $\boxed{[[a,a],b]}$

Here, the greyed out boxes indicate trees that are pruned away. In this case, for each of them the only optimal state is p_b, which is also an optimal state of the trees a, $[a,[a,a]]$, and $[[a,a],a]$ having a higher priority. (Pruned trees of Step 3 are not added to the next table.)

Step 4: dequeue $[b,b]$	
Output: $[b,b]$ (as $\Delta([b,b]) = 2 = M([b,b])$)	Expand with: $\boxed{[[b,b],[b,b]]}$, $\boxed{[[b,b],a]}$, $\boxed{[[b,b],b]}$, $\boxed{[[b,b],[a,a]]}$, $\boxed{[a,[b,b]]}$, $\boxed{[b,[b,b]]}$, $\boxed{[[a,a],[b,b]]}$
T: $\boxed{a}$, $\boxed{b}$, $\boxed{[a,a]}$, $\boxed{[b,b]}$	K: $\boxed{[a,b]}$, $\boxed{[b,a]}$, $\boxed{[a,[a,a]]}$, $\boxed{[[a,a],a]}$, $\boxed{[b,[b,b]]}$, $\boxed{[[b,b],b]}$, $\boxed{[[b,b],[b,b]]}$, $\boxed{[a,[b,b]]}$, $\boxed{[[b,b],a]}$, $\boxed{[[a,a],[b,b]]}$, $\boxed{[[b,b],[a,a]]}$

In the next step, $[a,b]$ is dequeued and written to the output, and the algorithm terminates.

5 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 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.

5.1 Correctness of Algorithm 1

We begin our correctness considerations by an easy lemma.

Lemma 4. *Algorithm 1 never dequeues the same tree twice.*

Proof. We prove that Algorithm 1 never *enqueues* the same tree twice, which implies the desired result. The proof is by induction on the number of executions of the loop body.

Base case: Prior to the first execution of the loop body, Algorithm 1 enqueues elements in Line 3. Since the queue is initially empty, the trees in Σ_0 have not been enqueued before.

Inductive case: Consider the ith execution of the loop body, in which the trees in $\mathit{expand}(T, t)$ are enqueued. We have to show that this set is disjoint with all sets $\mathit{expand}(T', t')$ of trees enqueued in earlier executions of the loop body, and with the set of trees enqueued in Line 3. As for the latter it suffices to notice that $\mathit{expand}(T, t)$ only contains trees of height at least 2. Thus, it remains to consider $\mathit{expand}(T, t) \cap \mathit{expand}(T', t')$.

Consider a tree $s' = f[t_1, \ldots, t_k] \in \mathit{expand}(T', t')$. By the induction hypothesis, no tree has hitherto been enqueued twice. Hence, when t is dequeued in Line 6, this is the first time this happens for this particular tree. As T' consists only of trees that have previously been dequeued, this implies that $t \notin T' \cup \{t'\}$. It follows that $t \notin \{t_1, \ldots, t_k\}$. In contrast, t is a direct subtree of every tree in $\mathit{expand}(T, t)$ so $s' \notin \mathit{expand}(T, t)$, which completes the proof. □

Lemma 5. *If Algorithm 1 dequeues a tree in $t \in T_\Sigma$, then it has previously dequeued all trees in $s \in T_\Sigma$ 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.*

Proof. It suffices to show that Algorithm 1 *enqueues* all trees $s \in T_\Sigma$ with $s \leq_K t$ before it dequeues t. To prove this, let $s = f[s_1, \ldots, s_k]$ and assume that s is *not* enqueued before t is dequeued. Assume furthermore that s is a minimal tree with this property. Then $s_i \leq_K s \leq_K t$ for all $i \in [k]$. Hence, since s is a minimal counterexample, $s_1, \ldots, s_k$ are enqueued before t is dequeued. It follows that $s_1, \ldots, s_k$ are dequeued before t is. As all trees that are dequeued are inserted into T, it follows that s is enqueued in Line 12 when the last one of $s_1, \ldots, s_k$ has been dequeued, contradicting the assumption that s is not enqueued before t is dequeued. □

We can now prove the correctness of Algorithm 1.

Theorem 2 (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 $t \in T_\Sigma \setminus \{t_1, \ldots, t_i\}$, $M(t) \geq M(t_i)$.*

Proof. Clearly, the first statement of the theorem is a consequence of the second. By Lemma 4, the output trees of $\mathit{BestTreesBasic}(M, \infty)$ are pairwise distinct. To

prove that *BestTreesBasic*(M, ∞) outputs an infinite sequence of trees we show that, after any number of iterations of the loop in Algorithm 1, only a finite number of additional iterations can be made until a tree is written to the output (i.e., until Line 9 is reached). Suppose that a tree t is dequeued. Then the tree $t' = c_{opt(t)}[\![t]\!]$ satisfies $M(t') = \Delta(t') = \Delta(t)$. By Lemma 5 and the definition of $\leq_K$, no tree s with $|s| > |t'|$ will be dequeued before t'. By Lemma 4 this means that t' will eventually be dequeued (unless the algorithm terminates before this happens). Since $M(t') = \Delta(t')$, the condition in Line 8 is satisfied and t' is written to the output in Line 9. Thus the algorithm is guaranteed to terminate.

To complete the proof, assume that *BestTreesBasic*$(M, \infty) = t_1, t_2, \ldots$ and consider some $i \in \mathbb{N}$ and a tree $t \in T_\Sigma \setminus \{t_1, \ldots, t_i\}$. We have to show that $M(t) \geq M(t_i)$. For this, assume that $M(t) < \infty$, because otherwise the assertion is trivially true. Now, recall the assumption that $\mathcal{W}_M$ is monotone. Since $M(t) < \infty$, it implies that $M(c[\![t]\!]) \geq M(t)$ for all contexts c, which means that $M(t) = \Delta(t)$ (because $M(\Box[\![t]\!]) = M(t)$). We also have $M(t_i) = \Delta(t_i)$, because Algorithm 1 outputs t_i only if the condition in Line 8 is satisfied. However, by Lemma 5 we have $\Delta(t) \geq \Delta(t_i)$ since t would otherwise have been dequeued before t_i, and thus written to the output at that stage (because $M(t) = \Delta(t)$). Hence, $M(t) = \Delta(t) \geq \Delta(t_i) = M(t_i)$, thus finishing the proof. □

5.2 Correctness of Algorithm 3

Based on the correctness of Algorithm 1 we can now go on to prove the correctness of Algorithm 3 and study its efficiency. In the following, let us say that a tree $s \in T_\Sigma$ is *discarded* in a run of *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, call a tree $t \in T_\Sigma$ *inactive* (with respect to the considered run of *BestTrees*(M, N)) if it contains a discarded subtree. Naturally, a tree that is not inactive is called *active*.

Lemma 6. *Algorithm 3 never dequeues an inactive tree.*

Proof. The argument in the proof of Lemma 4, showing that no tree is ever enqueued twice, is valid for Algorithm 3 as well. Thus, when a tree is discarded from K it will not be enqueued again and, therefore, not be dequeued. In particular, T never contains a discarded tree. Now, consider an inactive tree $t \in T_\Sigma \setminus \Sigma_{(0)}$ which is not itself a discarded tree. On the one hand, this means that t contains a discarded tree as a proper subtree. On the other hand, if t is enqueued in K, it follows by a straightforward induction that all its proper subtrees are in T. Thus, no subtree of t is discarded, which means that t must be active. □

Lemma 7. *Let* *BestTreesBasic*$(M, \infty) = t_1, t_2, \ldots$ *and consider the execution of* *BestTrees*(M, N) *for some* $N > 0$. *Let* $l \in \mathbb{N}^\infty$ *be the number of active trees among* $t_1, t_2, \ldots$, *and let* i_j *be such that* t_{i_j} *is the* j*th active tree in* $t_1, t_2, \ldots$, *for all* $j \in [l]$. *Then* *BestTrees*$(M, N) = t_{i_1}, t_{i_2}, \ldots, t_{i_{\min(l,N)}}$.

Proof. For $i \in \mathbb{N}$, denote the contents of T and K directly before the ith loop execution in Algorithm 1 and Algorithm 3 by T_i, K_i, T'_i, and K'_i, respectively. We prove the following claim:

Claim. Let $s_1, s_2, \dots$ be the sequence of trees dequeued in Line 6 of Algorithm 1 during a run of *BestTreesBasic*(M, ∞). Consider a run of *BestTrees*(M, N) and let $A = \{s_{i_1}, s_{i_2}, \dots\}$ be the set of active trees with respect to this run, where $i_1 < i_2 < \cdots$. Then, for all $l > 0$, the first l trees dequeued by Algorithm 3 are $s_{i_1}, s_{i_2}, \dots, s_{i_l}$ (provided that the main loop is executed at least l times). Furthermore $K_{i_l} \cap A \subseteq K'_l \subseteq K_{i_l}$.

Clearly, the statement of the lemma follows from the correctness of this claim, because the condition for outputting a tree is the same in both algorithms.

To prove the claim, we proceed by induction on l. We have $i_1 = 1$ because the call of Algorithm 2 in Line 3 of Algorithm 3 does not discard the least element of $\Sigma_{(0)}$ (with respect to $\leq_K$). Hence, for $l = 1$, $s_{i_1} = s_1$ is the first tree dequeued by Algorithm 3. Further, $K_1 \cap A = \Sigma_{(0)} \cap A = K'_1 \subseteq K_1$ since $K'_1 = \{a \in \Sigma_{(0)} \mid a \text{ is not discarded}\}$. Now, assume that the claim holds for some $l > 0$. The trees $s_{i_l+1}, \dots, s_{i_{l+1}-1}$ dequeued during the next $i_{l+1} - i_l - 1$ iterations in Algorithm 1 are all inactive, and thus also the trees enqueued are inactive as each contains one of the dequeued inactive trees as a subtree. It follows that $K_{i_{l+1}-1} \cap A = K_{i_l} \cap A \subseteq K'_l \subseteq K_{i_l} \subseteq K_{i_{l+1}-1}$.

Since $s_{i_{l+1}}$ is the highest priority tree in $K_{i_{l+1}-1}$ and is active, this means that it is also the highest priority tree in K'_l, and is therefore the next one dequeued by Algorithm 3. Hence the first statement of the claim holds for $l+1$. In particular, $T'_{l+1} = \{s_{i_1}, \dots, s_{i_{l+1}}\} = T_{i_{l+1}} \cap A$. Further, since $K'_l \subseteq K_{i_{l+1}-1}$, the call in Line 12 of Algorithm 3 enqueues only trees that are also enqueued in Line 12 of Algorithm 1, which shows that $K'_{l+1} \subseteq K_{i_{l+1}}$.

It remains to be shown that $K_{i_{l+1}} \cap A \subseteq K'_{l+1}$. A tree $s \in K_{i_{l+1}} \cap A$ is either in $K_{i_{l+1}-1} \cap A \subseteq K'_l$ and hence in K'_{l+1} because it is not discarded, or we have

$$\begin{aligned} s \in\ & \mathit{expand}(T_{i_{l+1}}, s_{i_{l+1}}) \cap A \\ &= \mathit{expand}(T_{i_{l+1}} \cap A, s_{i_{l+1}}) \cap A \\ &= \mathit{expand}(T'_{l+1}, s_{i_{l+1}}) \cap A \\ &\subseteq K'_{l+1} \end{aligned}$$

where the second line is correct because trees with inactive subtrees are inactive. □

Theorem 3 (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$(M, \infty) = t_1, t_2, \dots$ consists of pairwise distinct trees such that, for each $i \in \mathbb{N}$ and every tree $t \in T_\Sigma \setminus \{t_1, \dots, t_i\}$, $M(t) \geq M(t_i)$.*

Proof. The second statement is correct by Theorem 2, because the behaviors of both algorithms are obviously identical for $N = \infty$.

To prove the first statement, assume that *BestTreesBasic*$(M, \infty) = t_1, t_2, \dots$ and, using Lemma 7, that *BestTrees*$(M, N) = t_{i_1}, \dots, t_{i_l}$ for some $l \leq N$. We

show that $\{t_{i_1}, \ldots, t_{i_l}\} = \{t_1, \ldots, t_N\}$. Let $\Theta = \{t_1, \ldots, t_N\} \setminus \{t_{i_1}, \ldots, t_{i_l}\}$. By Lemma 7 each tree in Θ is inactive. Let us assume that $\Theta \neq \emptyset$, and let k be the least index such that $t_k \in \Theta$. In other words, t_k is the first tree among $t_1, \ldots, t_N$ containing a discarded subtree. Since t_k is one of the output trees of Algorithm 1 we have $\Delta(t_k) = M(t_k)$. Let $t_k = c[\![s]\!]$, where s is one of the discarded subtrees of t_k. Thus, s is inactive but all its proper subtrees are active (by Lemma 6 and the fact that all proper subtrees of trees in K have once been dequeued).

To finish the proof, let $v \in dom(c)$ be the node of c such that $c/v = \Box$, and consider a minimal run π on t_k, where $q = \pi(v)$. We know that $M(c'[\![s]\!]) \geq M(t_k)$ for all $c' \in C_\Sigma$ because otherwise $c'[\![s]\!] \in \{t_1, \ldots, t_{k-1}\}$, which would contradict the choice of k since $c'[\![s]\!]$ contains the discarded subtree s. In other words,

$$q \in optset(s),\ M_q(c) = \min\{M_q(c') \mid c' \in C_\Sigma\} \text{ and } M(t_k) = M(c_q[\![s]\!])\ . \quad (1)$$

Since s was discarded during the execution of Algorithm 3, we know further that $T \cup K$, from that point onward, always contained N pairwise distinct trees $u_1, \ldots, u_N$ such that $u_i <_K s$ and $q \in optset(u_i)$.

We distinguish two cases, deriving a contradiction in each case and thus proving that t_k cannot exist:

1. If $M(c_q[\![u_i]\!]) < M(c[\![u_i]\!])$ for some $i \in [N]$, then it follows from the equations $M(c_q[\![u_i]\!]) = M_q(c_q) + M^q(u_i)$ and $M(c[\![u_i]\!]) \leq M_q(c) + M^q(u_i)$ that $M_q(c_q) < M_q(c)$ and thus $M(c_q[\![s]\!]) < M(t_k)$, contradicting (1).
2. If $M(c[\![u_i]\!]) = M(c_q[\![u_i]\!]) = \Delta(u_i)$ for all $i \in N$, we distinguish two sub-cases.
   (a) If $s <_{lex} u_i$ despite the fact that $u_i <_K s$, then $\Delta(u_i) < \Delta(s)$, so $M(c[\![u_i]\!]) = M(c_q[\![u_i]\!]) < M(t_k)$, which gives us $c[\![u_i]\!] <_K t_k$ for all $i \in [N]$. By Lemma 5, all of these N pairwise distinct trees occur among $t_1, \ldots, t_{k-1}$, which is impossible because $k \leq N$.
   (b) If, on the contrary, $u_i <_{lex} s$, then Observation 1 together with the equation

$$\Delta(c[\![u_i]\!]) \leq M(c[\![u_i]\!]) = \Delta(u_i) \leq \Delta(s) = M(t_k) = \Delta(t_k)$$

   gives us again $c[\![u_i]\!] <_K t_k$ for all $i \in [N]$ and thus the same contradiction as before. □

5.3 Efficiency of Algorithm 3

Let us now have a look at the worst-case efficiency of *BestTrees*. We first show why it is unnecessary to prune the set T along with K in order to make Algorithm 3 efficient. Intuitively, it is because once a tree t has been dequeued and added to T, the algorithm will never discover a better way to reach the states in $optset(t)$ than t itself.

In the following, $s <_D t$ denotes that trees s and t are both eventually dequeued in a run of $BestTrees(M, \infty)$, and s is dequeued before t.

Lemma 8. *If $s <_D t$, then $M^q(s) \leq M^q(t)$ for every $q \in optset(s)$.*

Proof. By the claim in the proof of Lemma 7, the sequence of trees dequeued by Algorithm 3 forms a subsequence of the sequence of trees dequeued by Algorithm 1. Hence, by Lemma 5, $s <_D t$ implies $s <_K t$, and thus $\Delta(s) \leq \Delta(t)$. Suppose, in contradiction to the statement of Lemma 8, that $M^q(t) < M^q(s)$ for some $q \in \mathit{optset}(s)$. This leads to a contradiction through the following sequence of (in-)equalities:

$$\Delta(t) \leq M(c_q[\![t]\!]) = M_q(c_q) + M^q(t) < M_q(c_q) + M^q(s) = M(c_q[\![s]\!]) = \Delta(s) \quad \square$$

A consequence of Lemma 8 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.

Lemma 9. *The body of the 'while' loop in BestTrees is executed at most $N \cdot n$ times.*

Proof. We show that T never grows beyond $N \cdot n$ elements. Since each execution of the 'while' loop increases the size of T, this limits the number of iterations.

Consider the sequence $W = t_1, \ldots, t_k$ of trees dequeued during the execution of $\mathit{BestTrees}(M, N)$. In the light of Lemma 8, for every state $q \in Q$ the subsequence $W_q = t_{i_1}, \ldots, t_{i_l}$ given by $\{i_1, \ldots, i_l\} = \{j \in [k] \mid q \in \mathit{optset}(t_{i_j})\}$ satisfies $M^q(t_{i_1}) \leq \cdots \leq M^q(t_{i_l})$. Hence each W_q is of length at most N, because every further tree would have fulfilled the pruning condition and would thus have been discarded from the queue instead of eventually being dequeued. As each tree in W occurs in at least one of the subsequences W_q, this proves the statement of the lemma. $\square$

Lemma 10. *$\mathit{Prune}(K, \mathit{Expand}(T, t))$ is computable in time*

$$O\big(\max(m \cdot (Nr + r \log r + N \log N), Nn^2)\big) \quad .$$

Proof. In order to implement pruning efficiently, we have to avoid the explicit computation of $\mathit{Expand}(T, t)$. Let us denote the subset of transition rules in δ that lead to the state q by δ_q. We first compute, for every $q \in Q$, an ordered list of (at most) N trees $s_1, \ldots, s_N$ in $\mathit{Expand}(T, t)$ such that $M^q(s_1) \leq \cdots \leq M^q(s_N)$ and $M^q(s) \geq M^q(s_N)$ for all $s \in \mathit{Expand}(T, t) \setminus \{s_1, \ldots, s_N\}$. To do this, consider every rule $\rho = (f[q_1, \ldots, q_k] \xrightarrow{w} q) \in \delta_q$ in turn and build a weighted edge-labelled digraph G_ρ having nodes $u_0, \ldots, u_k$ and $v_0, \ldots, v_k$ and the following edges for every $i \in [k]$:[1]

> For every $s' \in T \setminus \{t\}$ there are edges with label s' and weight $M^{p_i}(s')$ from u_{i-1} to u_i and from v_{i-1} to v_i. In addition, there are edges with label t and weight $M^{p_i}(t)$ from both u_{i-1} and v_{i-1} to v_i.

See Figure 2 for an illustration.

A path from u_1 to v_{k+1} in G_ρ which is labelled $t_1 \cdots t_k$ corresponds to the tree $f[t_1, \ldots, t_k] \in \mathit{Expand}(T, t)$. Note that t occurs among $t_1, \ldots, t_k$ since only

[1] The nodes v_0 and u_k are superfluous but simplify the description of G_ρ.

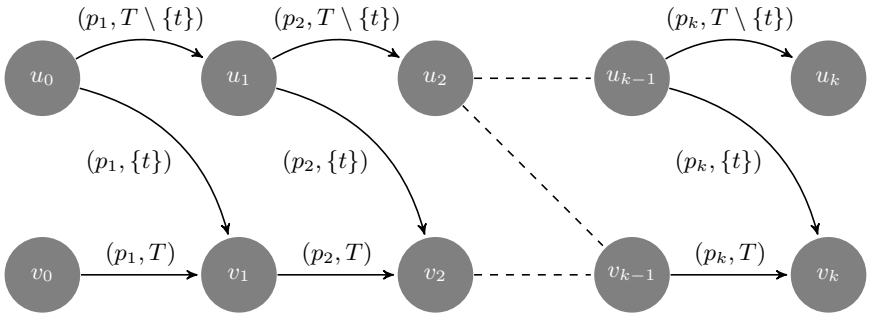

Fig. 2. The graph constructed in the proof of Lemma 10, to discover the N best ways of instantiating a rule $\rho = (f[q_1, \ldots, q_k] \xrightarrow{w} q) \in \delta_q$ with trees in T, in such a way that the tree t is used at least once. Here, (p, S) with $p \in Q$ and $S \subseteq T_\Sigma$ denotes the set of tuples $\{(s, M^p(s)) \mid s \in S\}$.

t-labelled edges lead from u_i to v_{i+1}. The weight of the path is the weight of a minimal run π on $f[t_1, \ldots, t_k]$ with $\pi(\lambda) = q$ and $\pi(i) = q_i$ for all $i \in [k]$. Since G_ρ has $O(r)$ nodes and $O(Nnr)$ edges, N paths of minimal weight can be computed by Eppstein's algorithm [5] in time $O(Nnr + r \log r + N \log N)$. We can improve this to $O(Nr + r \log r + N \log N)$ by including in G_ρ, for every pair of nodes, only N edges of minimal weight between those nodes. Clearly, only these edges can be on the N paths of minimal weight. For every rule ρ, this gives rise to an ordered list L_ρ of (at most) N trees. The time required for this is $O(m \cdot (Nr + r \log r + N \log N))$ in total. Together with each tree in the computed lists L_ρ, we keep track of the corresponding weight in order to be able to implement the following steps.

In the next step, the lists obtained for rules with the same right-hand side q are merged into one list L_q of length N of trees $s \in \mathit{Expand}(T, t)$, ordered according to $M^q(s)$. Trees not among the first N are discarded. Since the lists obtained in the previous step are sorted, this step takes time $O(m_q N)$ for each $q \in Q$, where m_q is the number of rules having q as their right-hand side, and hence $O(mN)$ in total, which is dominated by the time required for the previous step. Each tree t in L_q is now associated with the weight $M^q(t)$.

Finally, a similar procedure merges the n lists obtained in the previous step with the trees in K in order to construct $\mathit{Prune}(K, \mathit{Expand}(T, t))$. Since the queue contains at most Nn elements, the time required for this is $O\big(Nn^2\big)$ if we implement K as a linked list. □

Combining Lemma 2, 9 and 10, we obtain Theorem 4.

Theorem 4. *BestTrees*(M, N) *runs in time*

$$O\big(\max(Nmn \cdot (Nr + r \log r + N \log N), N^2 n^3, mr^2)\big) \quad .$$

It may be worthwhile to notice that the set T 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 yields the following observation.

Observation 5 *Algorithm 3 computes a set of N minimal trees that can be represented as a directed acyclic graph (or hypergraph) of size $N \cdot n \cdot r$.*

6 Conclusion and future work

We have lifted the algorithm by Mohri and Riley [16] to the domain of trees. Since the frontier of the search space grows exponentially if no additional measures are taken, pruning becomes an essential technique. Our suggested pruning strategy may be overly cautious, so it would be interesting to learn whether it can be improved through the addition of a preprocessing step, in which the topology of the input wta is explored.

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.

Bibliography

[1] Borchardt, B.: A pumping lemma and decidability problems for recognizable tree series. Acta Cybernetica 16, 509–544 (2004)

[2] Collins, M.: Discriminative reranking for natural language parsing. In: Computational Linguistics. pp. 175–182. Morgan Kaufmann (2000)

[3] Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: *Tree Automata Techniques and Applications* (2002), internet publication available at http://www.grappa.univ-lille3.fr/tata

[4] Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)

[5] Eppstein, D.: Finding the k shortest paths. SIAM J. Computing 28(2), 652–673 (1998)

[6] Gécseg, F., Steinby, M.: Tree Automata. Akadémiai Kiadó (1984)

[7] Gécseg, F., Steinby, M.: Tree languages. In: Handbook of Formal Languages, vol. 3, chap. 1, pp. 1–68. Springer Verlag (1997)

[8] Gildea, D., Jurafsky, D.: Automatic labeling of semantic roles. Computational Linguistics 28(3), 245–288 (sep 2002)

[9] Goodman, J.: Parsing inside-out. Tech. Rep. cmp-lg-9805007, Computing Research Repository (1998)

[10] Huang, L., Chiang, D.: Better k-best parsing. In: Proceedings of the Conference on Parsing Technology 2005. pp. 53–64. Association for Computational Linguistics (2005)

[11] Knuth, D.E.: A generalization of Dijkstra's algorithm. Information Processing Letters 6, 1–5 (1977)

[12] Kuich, W., Droste, M., Vogler, H. (eds.): Handbook of Weighted Automata. Springer (2009)

[13] Kumar, S., Byrne, W., Processing, S.: Minimum Bayes-risk decoding for statistical machine translation. In: Proceedings of HLT-NAACL 2004 (2004)

[14] Mezei, J., Wright, J.B.: Algebraic automata and context-free sets. Information and Control 11, 3–29 (1967)

[15] Mohri, M., Riley, M.: A weight pushing algorithm for large vocabulary speech recognition. In: European conference on speech communication and technology. pp. 1603–1606 (2001)

[16] Mohri, M., Riley, M.: An efficient algorithm for the n-best-strings problem. In: Proceedings of the Conference on Spoken Language Processing (2002)

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

[18] Wellner, B., McCallum, A., Peng, F., Hay, M.: An integrated, conditional model of information extraction and coreference with application to citation matching. In: Proceedings of the Conference on Uncertainty in Artificial Intelligence 2004. pp. 593–601. AUAI Press (July 2004)