

A Comparison of Two N-Best Extraction Methods for Weighted Tree Automata

Johanna Björklund, Frank Drewes, and Anna Jonsson(✉)

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

Abstract. We conduct a comparative study of two state-of-the-art algorithms for extracting the N best trees from a weighted tree automaton (wta). The algorithms are Best Trees, which uses a priority queue to structure the search space, and Filtered Runs, which is based on an algorithm by Huang and Chiang that extracts N best runs, implemented as part of the Tiburon wta toolkit. The experiments are run on four data sets, each consisting of a sequence of wtas of increasing sizes. Our conclusion is that Best Trees can be recommended when the input wtas exhibit a high or unpredictable degree of nondeterminism, whereas Filtered Runs is the better option when the input wtas are large but essentially deterministic.

1 Introduction

Data-driven language processing involves as a rule weighted language models. Rather than providing a definite answer as to whether a sentence belongs to a target language, these return a probability or a fitness score. This reflects the inherently ambiguous nature of human language and is convenient for statistical machine learning, but it often complicates downstream processing. When the output of a machine translation system is not limited to a small set of possible translations, but is a weighted device ranking the universe of all possible outputs, efficient algorithms are needed to find the highest-scoring solutions. This problem is known as the N-best problem. The input is a weighted automaton M and a natural number N, and task is to find N best-ranking elements with respect to M. The difficulty of the problem, and indeed whether there is a unique or several interchangeable solutions, largely depends on the type of automata at hand and the domain from which weights are taken.

We consider the N-best problem for weighted tree automata [3,4], which are useful in natural language processing, owing to their capability to rank parse trees of context-free languages. This makes them useful for syntax-based forms of processing, as demonstrated in, e.g., machine translation and program verification. Weighted tree automata [2] are typically defined over algebras that have at least as much structure as a semiring, but semifields or even fields are often used. The weight of a computation (called a run) of an automaton on an input tree is the semiring product of the weights of the rules applied, and the weight

© Springer International Publishing AG, part of Springer Nature 2018
C. Câmpeanu (Ed.): CIAA 2018, LNCS 10977, pp. 97–108, 2018.
https://doi.org/10.1007/978-3-319-94812-6_9

of the tree is the semiring sum of the weights of all runs on this tree. We restrict ourselves to the so-called tropical semiring, which means that the weight of a run is the (ordinary) sum of the weights of the applied rules and the weight of a tree is the minimum of the weights of its runs.

Our main contribution is an empirical evaluation of two N-best algorithms for weighted tree automata (wta) M over the tropical semiring. Both algorithms represent the state of the art, but operate in quite different ways. The first of these is an indirect method based on an N-best runs (or derivations) algorithm proposed by Huang and Chiang [5] and implemented in the wta toolkit Tiburon [7]. The algorithm computes N best runs in time $O(mN \log N)$, where m is the number of transitions of the wta. This can be used to compute N best trees by generating a list of N' best runs of M, for some large enough $N' \geq N$. These runs are evaluated to the corresponding trees and duplicates are discarded to obtain a list of N best trees. Henceforth, we refer to this method by Filtered Runs. Filtered Runs thus takes a heuristic approach in the sense that an unlucky user may request too large or small a number of best runs, either wasting time or not gathering enough runs to find N unique best trees.

The second algorithm that we evaluate is Best Trees [1], a generalisation of an N-best algorithm for string automata [8]. Intuitively, Best Trees implements a breadth-first search, while making extensive use of pruning to avoid a combinatorial explosion. The running time of Best Trees is $O(N^2 \cdot (mn \log N + n^3))$ [1, slightly simplified], where m and n are the number of transitions and states of the input wta. Hence, the algorithm is less efficient than the pure N-best runs algorithm by Huang and Chiang, even though both are polynomial. However, Best Trees is guaranteed to produce exactly the desired number of trees without the need to discard duplicates, whereas Filtered Runs may require the enumeration of a large number of runs. The latter can happen if the input wtas exhibit a high degree of nondeterminism, i.e. if the number of distinct trees among the N best runs grows slowly (logarithmically in the worst case) with increasing N. Filtered Runs is thus expected to run faster even on large wtas if there is no or very little nondeterminism, while the asymptotic advantage of Best Trees should become apparent as the degree of nondeterminism increases. We perform empirical evaluations in order to (a) confirm this expected behaviour and (b) get an idea about how the algorithms compare on different kinds of wta and varying amounts of nondeterminism.

To study this in a setting where the type, size, and amount of nondeterminism of input wtas can be varied in a controlled way, we run our experiments on a range of synthesized wtas designed for this purpose rather than using "real life" wtas. While the results mostly confirm the theoretical expectations, they show that the precise behaviour is not as simple as the theoretical worst case analysis suggests. In particular, the running time of Filtered Runs depends on aspects other than the pure amount of nondeterminism, such as the order in which the transition rules of the input wta are given.

2 Preliminaries

We write $\mathbb{N}$ for the set of nonnegative integers, $\mathbb{N}_+$ for $\mathbb{N}\setminus\{0\}$, and $\mathbb{R}_+$ for the set of non-negative reals; $\mathbb{N}^\infty$ and $\mathbb{R}_+^\infty$ denote $\mathbb{N}\cup\{\infty\}$ and $\mathbb{R}_+\cup\{\infty\}$, respectively. For $n\in\mathbb{N}$, $[n]=\{i\in\mathbb{N}\mid 1\le i\le n\}$. Thus, in particular, $[0]=\emptyset$ and $[\infty]=\mathbb{N}$. The cardinality of a (countable) set S is written $|S|$, and the powerset of S is denoted by $pow(S)$. The n-fold Cartesian product of a set S with itself is denoted by S^n. As usual, the set of all finite sequences over S is denoted by S^*, and the empty sequence by λ.

For a set A, an A-labelled *tree* is a partial function $t\colon \mathbb{N}_+^*\to A$ whose domain $dom(t)$ is a finite non-empty set that is prefix-closed and closed to the left: whenever $vi\in dom(t)$ for some $v\in\mathbb{N}_+^*$ and $i\in\mathbb{N}_+$, it holds that $v\in dom(t)$ (prefix-closedness) and $vj\in dom(t)$ for all $1\le j\le i$ (closedness to the left). The size of t is $|t|=|dom(t)|$. An element v of $dom(t)$ is called a *node of* t, and $|\{i\in\mathbb{N}_+\mid vi\in dom(t)\}|$ 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 $f[t_1,\dots,t_k]$, which may be identified with f if $k=0$.

A *ranked alphabet* is a disjoint union of finite sets of symbols, $\Sigma=\bigcup_{k\in\mathbb{N}}\Sigma_{(k)}$. For $f\in\Sigma$, the $k\in\mathbb{N}$ such that $f\in\Sigma_{(k)}$ is the *rank* of f, denoted by $rank(f)$. The set T_Σ of ranked trees over Σ consists of all Σ-labelled trees t in which the rank of every node $v\in dom(t)$ equals the rank of $t(v)$. For a set T of trees we denote by $\Sigma(T)$ the set of trees which have a symbol from Σ in their root, with direct subtrees in T, i.e., $\{f[t_1,\dots,t_k]\mid k\in\mathbb{N}, f\in\Sigma_{(k)},\text{ and } t_1,\dots,t_k\in T\}$.

In the following, let $\Box\notin\Sigma$ be a special symbol of rank 0. The set of *contexts over* Σ is the set C_Σ of trees $c\in T_{\Sigma\cup\{\Box\}}$ containing exactly one node $v\in dom(c)$ with $c(v)=\Box$. The *substitution* of another tree t into c results in the tree $c[\![t]\!]$ given by $dom(c[\![t]\!])=dom(c)\cup\{vu\mid u\in dom(t)\}$ and

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

for all $w\in dom(c[\![t]\!])$.

Recall that the domain of the *tropical semiring* is $\mathbb{R}_+^\infty$, with min serving as addition and real-valued addition as multiplication. 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 a *weighted tree automaton with final states* (wta). A wta is a system $M=(Q,\Sigma,R,Q_\mathrm{f})$ consisting of

- a finite set Q of symbols of rank 0 called *states*;
- a ranked alphabet Σ of *input symbols* disjoint with Q;
- a finite set R of *transition rules* $f[q_1,\dots,q_k]\xrightarrow{w}q$, where $q,q_1,\dots,q_k\in Q$, $f\in\Sigma_{(k)}$, and $w\in\mathbb{R}_+$; and
- a set $Q_\mathrm{f}\subseteq Q$ of *final states*.

A transition rule $r\colon f[q_1,\dots,q_k]\xrightarrow{w}q$ will also be viewed as a symbol of rank k, so that R becomes a ranked alphabet. In addition, we view every state $q\in Q$

as a symbol of rank 0. We define the set $runs_M^q \subseteq T_{R \cup Q}$ of *q-runs of* M, their resulting trees $result_M(\rho)$, and their weights $wt_M(\rho)$ (for $\rho \in runs_M^q$) inductively, as follows:

1. For every state $q \in Q$, $q \in runs_M^q$ with $result_M(q) = q$ and $wt_M(q) = 0$.
2. For every $r\colon f[q_1, \ldots, q_k] \xrightarrow{w} q$ in R and all $\rho_1 \in runs_M^{q_1}, \ldots, \rho_k \in runs_M^{q_k}$, $\rho = r[\rho_1, \ldots, \rho_k] \in runs_M^q$ with $result_M(\rho) = f[result_M(\rho_1), \ldots, result_M(\rho_k)]$, and $wt_M(\rho) = w + \sum_{i \in [k]} wt_M(\rho_i)$.

The set of *accepting* runs is $runs_M = \{runs_M^q \mid q \in F\}$.

Now, the weighted tree language $M\colon T_\Sigma \to \mathbb{R}_+^\infty$ *recognised by* M is given by

$$M(t) = \min\{wt_M(\rho) \mid \rho \in runs_M^q \text{ is accepting and } result_M(\rho) = t\}$$

for all $t \in T_\Sigma$ (where, by convention, $\min \emptyset = \infty$). In other words, $M(t)$ is the minimal weight of an accepting run of t. Note that we, by a slight abuse of notation, denote by M both the wta and the weighted tree language it computes.

For a wta M and an $N \in \mathbb{N}^\infty$ as input, the *N-best runs problem* is the problem to compute a sequence of N accepting runs of minimal weight according to M. More precisely, an algorithm solving the problem outputs a sequence $\rho_1, \rho_2, \ldots$ of N pairwise distinct accepting runs such that there are no $i \in [N]$ and $\rho \in runs_M \setminus \{\rho_1, \ldots, \rho_i\}$ with $wt_M(\rho) < wt_M(\rho_i)$. (If the total number N' of accepting runs is smaller than N, the algorithm only outputs N' runs.)

Similarly, the *N-best trees problem* asks to compute pairwise distinct trees $t_1, t_2, \ldots$ in T_Σ of minimal weight, i.e., such that there are no $i \in [N]$ and $t \in T_\Sigma \setminus \{t_1, \ldots, t_i\}$ with $M(t) < M(t_i)$.

3 Previous Work

The difference between the two N-best problems is that, in the nondeterministic case, distinct runs may result in the same tree. The wta toolkit Tiburon provides an implementation of the N-best runs algorithm by Huang and Chiang [5]. This yields an obvious procedure for solving the N-best trees problem: one simply computes N' best runs $\rho_1, \rho_2, \ldots$ for large enough N', and outputs those $result_M(\rho_i)$ for which $result_M(\rho_i) \notin \{result_M(\rho_1), \ldots, result_M(\rho_{i-1})\}$. This procedure is guaranteed to produce the desired result because any given tree has at most an exponential number of runs, which means that the next tree will be encountered after at most exponentially many steps. (If there are no more accepting runs one can simply continue to enumerate arbitrary ones of the remaining trees, whose weight will by definition be ∞.)

The N-best trees algorithm developed in [1] avoids the detour via N' best runs. We now give a short summary of the reasoning that leads to this algorithm. Let the size parameters of the input wta M be the following:

- m is the number $|R|$ of transition rules of M,
- n is the number of states, and

– r is the maximum rank of symbols.

The algorithm explores its search space by maintaining a priority queue K of trees that are candidates of output trees. The trees in the queue mark the frontier of the search space, the priority being determined primarily by the minimal value of $M(c[\![t]\!])$, where c ranges over all possible contexts. To determine this value, note that the definition of $M(t)$ works also for trees $t \in T_{\Sigma \cup Q}$. In particular, t can be of the form $c[\![q]\!]$, where c is a context and $q \in Q$. This is useful because of the following. For a run of the form $\rho[\![\rho']\!]$ where $\rho' \in runs_M^q$ we clearly have $wt_M(\rho[\![\rho']\!]) = wt_M(\rho) + wt_M(\rho')$. Hence, if we denote by M^q the wta obtained from M by replacing its set of final states by $\{q\}$, then

$$M(c[\![t]\!]) = \min_{q \in Q} (M(c[\![q]\!]) + M^q(t))$$

for all contexts c and all trees t.

As $M(c[\![p]\!])$ is independent of t, a context c that minimises it can be calculated in advance. Such a context c, which we call a *cheapest context*[1] of q and which is henceforth denoted by c_q, is thus a cheapest context into which a subtree t can be embedded in order to reach a final state, once the state q has been reached at the root of t. As was shown in [1], a family $(c_q)_{q \in Q}$ of cheapest contexts can efficiently be computed given M.

To solve the N-best trees problem, when looking at 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. The smaller this weight is, the higher should the priority of t be. Clearly, when comparing trees with regard to this, c can be assumed to be one of the cheapest contexts c_p. Thus, our aim has to be to determine the state q that minimises the weight of $c_q[\![t]\!]$. We call such a state an *optimal state* for t, and denote it by $opt(t)$ (breaking ties arbitrarily). In the algorithm, optimal states can efficiently be computed in an incremental way as trees are assembled from subtrees, provided that a small amount of bookkeeping information is stored along with each tree.

Now, the N-best trees algorithm maintains data structures T and K, where

– T is a set of trees that have already been processed and
– K is a priority queue of trees in $\Sigma(T)$, the frontier of the search space.

The queue K initially contains the trees in Σ_0. Its priority order $<_K$ is defined by $t <_K t' \iff \Delta(t) < \Delta(t')$, where $\Delta(s) = M(c_{opt(s)}[\![s]\!])$ for all $s \in T_\Sigma$.

We reproduce the pseudocode of the base algorithm from [1] in Algorithm 1. As discussed in detail in [1], the set of trees enqueued in line 13 can be pruned because for every state q at most N trees for which q is an optimal state may ever become relevant. An additional optimisation was used in the implementation used for the experiments of this paper: once the algorithm has outputted $i \leq N$ trees, it suffices to keep $N - i$ rather than N trees in K for each optimal state. Hence, the queue shrinks as progress is made. While this does not affect the asymptotic running time, it does yield a significant improvement in practise.

[1] In [1] the term *smallest completion* was used.

Algorithm 1. Compute $N \in \mathbb{N}^\infty$ trees of minimal weight according to a wta M

```
1: procedures Best Trees(M, N)
2:     compute cheapest contexts for all states
3:     T ← ∅; K ← ∅
4:     enqueue(K, Σ_0)
5:     i ← 0
6:     while i < N ∧ K is nonempty do
7:         t ← dequeue(K)
8:         T ← T ∪ {t}
9:         if M(t) = Δ(t) then
10:            output(t)
11:            i ← i + 1
12:        end if
13:        enqueue(K, expand(T, t))
14:    end while
15: end procedures
```

4 Experiments

In this section, we experimentally verify the time complexity of Best Trees, and then compare its performance with the indirect method Filtered Runs.

It is easy to construct worst-case scenarios in which Best Trees works exponentially faster than Filtered Runs. For example, let $\Sigma = \Sigma_{(0)} \cup \Sigma_{(1)}$ with $\Sigma_{(0)} = \{a\}$ and $\Sigma_{(1)} = \{f\}$, and consider the wta M with two states q_1, q_2, and the rules $a \xrightarrow{0} q_i$ and $f[q_i] \xrightarrow{1} q_j$ for all $i, j \in [2]$, where q_1 is final. Then $M(t) = |t|$ for every tree $t \in T_\Sigma$ and thus Best Trees simply enumerates trees by size. However, each tree t has $2^{|t|}$ accepting runs, all of weight $|t|$, and thus Filtered Runs needs to generate $2^{|t|} - 1$ best runs to discover t.

In the following, we conduct experiments on synthesized sets of wtas which, rather than triggering this kind of worst-case behaviour, are designed to shed light on particular aspects of the algorithms in less extreme (and thus perhaps practically more relevant) cases. An annotated collection containing all of these wta sets is available on the project web page[2], along with the measured running times for each wta. Due to space restrictions, we limit ourselves for the present to brief descriptions of the data.

The experiments were run on a computer with 8 Intel i7 processors, each at a speed of 3.6 GHz and with 16 GB memory allocated for the JVM. The efficiency results are based on repeated experimentation, and each reported running time is a mean value of five runs. As both Best Trees and Filtered Runs are deterministic algorithms, the existing (but relatively low) variance in the running times is due to variations in the execution environment, e.g., overall system load. All plots show the running times in milliseconds as a function of N or wta size, which is why we exclude the y axis labels from the figures.

[2] http://people.cs.umu.se/aj/besttrees_experiments/.

4.1 Data

Below follows a short presentation of the wta sets used in this paper. Each set consists of a sequence of language-equivalent wtas of increasing sizes. The relatively small wta sizes are due to the limitations of Tiburon: increasing the number of states causes out-of-memory errors when adding more nondeterminism.

Basic Example. A sequence of 20 wtas, starting with the example wta in [1]. Each subsequent automaton was derived from its predecessor by mirroring an existing run for some tree t by the addition of new states and rules resulting in an alternative run on t. Thus, the numbers of states and transition rules increase at the same rate, which allows us to check the running time as a function of the number of states and rules on the one hand, and of N on the other hand. The amount of nondeterminism, however, does not significantly increase as only t gets one more accepting run.

Different Weights. A sequence of 16 wtas over a, b of rank 0 and 0, resp., and states $q_1, \ldots, q_4$, in which all of the transition rules have different weights. All states are final. The transition rules have the weight of the index of their target state. The weight of transition rule $b[q_i, q_j] \to q_k$ is $ijk/100$, where ijk is interpreted as a number in decimal notation. The ℓ-th wta ($\ell \in [16]$) consists of the first 4ℓ of these rules if ordered according to decreasing weight. As a consequence, the degree of nondeterminism is moderate throughout, but is changing as the wta sizes grow, as the later rules result in the best runs. In particular, the degree of nondeterminism of the final wta including all transition rules is low because only the rules $a \to q_1$ and $b[q_1, q_1] \to q_1$ result in cheapest runs.

Modified Different Weights. Similar to Different Weights, but transition rules on b are added in a different order, starting with $b[q_1, q_1] \to q_1$ (see the project web page (see footnote 1) for details). Hence, the best runs in all 16 wtas are those which assign q_1 to all nodes, which means that the degree of nondeterminism is small and grows moderately with growing wta sizes.

Equal Weights. The wta set Equal Weights is also similar to the set Different Weights, but the weights are equal (all 0) apart from the rules on a, which have weight 4. Thus, this example has the highest possible degree of nondeterminism.

4.2 Running Time of Best Trees

Let us first compare the measured running times of Best Trees with the theoretical bound $O(N^2 \cdot (mn \log N + n^3))$ derived in [1].[3]

A somewhat unexpected outcome of our experiments was that our implementation of the computation of the cheapest contexts in line 2 of Algorithm 1, which makes use of an algorithm by Knuth [6], turned out to be slightly less efficient

[3] For the sake of clarity, the expression is slightly simplified. In particular, the maximum rank r of symbols is taken to be constant, as it is typically small in practise.

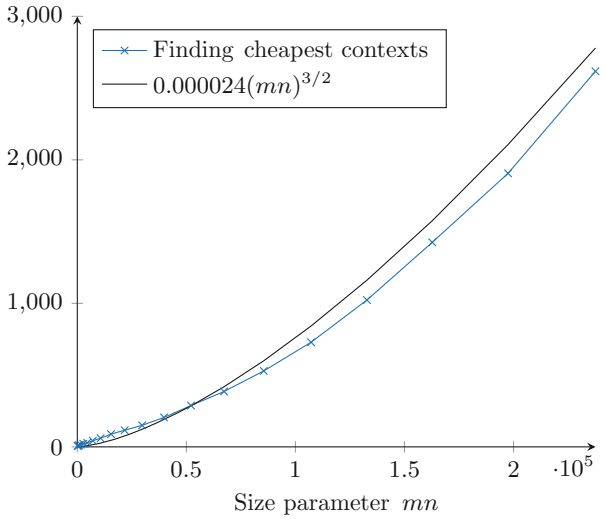
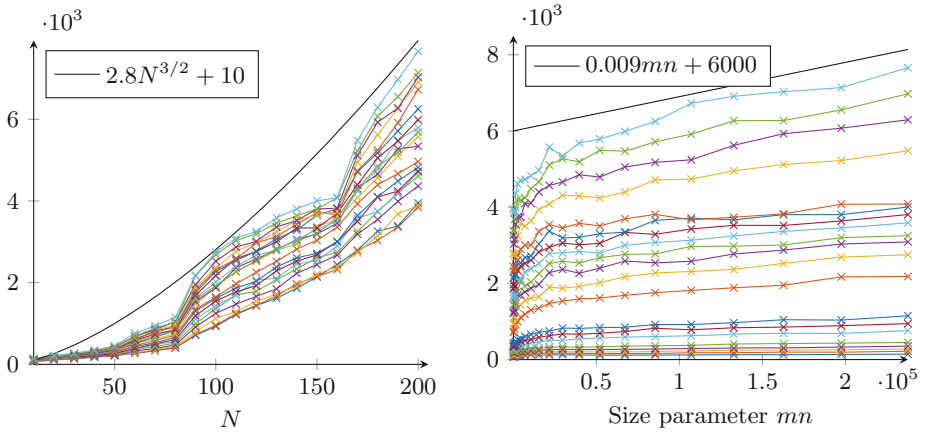


Fig. 1. Running time of finding the cheapest contexts for wtas of increasing size in the set Basic Example.

than the theoretical upper bound $O(mn \log n)$. Plotted against the parameter mn, $O(mn \log n)$ should basically become linear when finding the cheapest contexts for Basic Example. As can be seen in Fig. 1, the practical running time appears to instead be proportional to $(mn)^{3/2}$ on this type of input automata. Further experiments revealed that when m was increased while n was kept constant (suggesting a running time of $\Theta(m)$), we instead acquired figures resembling $\Theta(m^2)$ very closely. Optimising the computation of the cheapest contexts could therefore give a certain gain when N is small, but since the computation time is independent of N, its influence vanishes as N grows. Due to this not being optimised, we from here on disregard the time for finding the cheapest contexts when presenting running times for Best Trees. This, however, only affects the runs on Basic Example significantly.

The numbers of rules and states increase at a similar rate in the wta sequence Basic Example; this behaviour allows us to vary the two parameters N and mn. Based on the theoretical upper bounds, the running times should be in $O(N^2 \log N)$ and $O((mn)^{3/2})$ (the latter because of the term n^3 in the theoretical estimation, which equals $(mn)^{3/2}$ for $m = n$). This is confirmed by our results, which are visualised in Fig. 2a for increasing N, and in Fig. 2b for increasing mn. As may be expected, these running times are slightly better than the theoretical worst-case estimations.

As N varies, the theoretical worst-case running time of $O(N^2 \log N)$ is only reached when the priority queue used by the algorithm is at its maximal length throughout most of the execution. Simply put, this only happens when most trees can reach most states. In practise, the running time is therefor likely to be lower. This is for instance the case for the Basic Example test set.



(a) Running time depending on N; the multiple lines represent the increasingly large wtas of Basic Example.

(b) Running time depending on the size of wtas; the multiple lines represent $N \in \{10, 20, ..., 200\}$.

Fig. 2. Running time of Best Trees on the Basic Example set.

Here, the algorithm exhibits the behaviour shown in Fig. 2a, which is roughly proportional to $N^{3/2}$.

The waviness of the plot is explained by the pattern of the language recognised by the input wtas: it only contains trees with an even number of symbols of rank 0. By there only being a binary symbol in addition to the rank 0 symbols, all of the trees in the language have an odd number of binary symbols. When all of the trees with $2i-1$ binary symbols have been found, the next accepted tree is amongst the trees with $2i+1$ binary symbols. However, all of the possibilities for $2i$ binary symbols have to be processed as well since a rule application adds at most one binary symbol to the resulting tree. Thus, the jumps in the plot represent going from finding the trees with $2i-1$ binary symbols to finding the trees with $2i+1$ binary symbols.

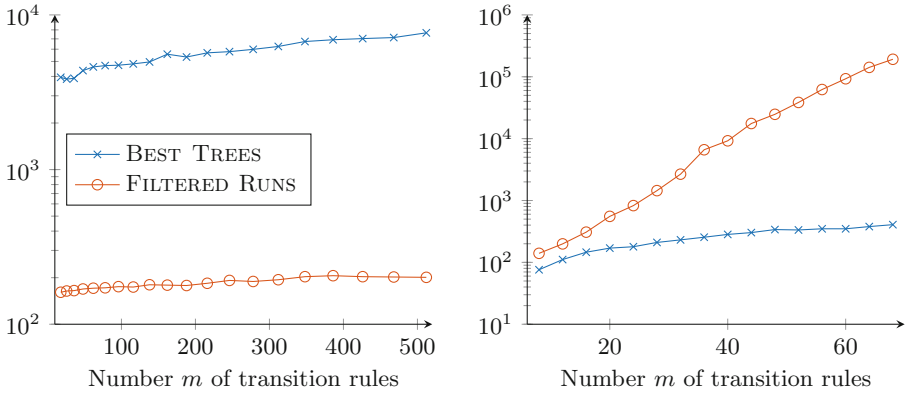
The measured running time as a function of the size of wtas is also lower than the theoretical upper bound $O((mn)^{3/2})$ in this example, namely linear.

4.3 Comparison of Best Trees and Filtered Runs

In these experiments, we used Tiburon v.1.0. We ran Tiburon on the input wta to get d_N runs where d_N is the smallest number of runs that produces N distinct trees. The number d_N was found manually for each combination of input wta and value of N. Then, the list of runs was filtered using a Python script such that only the N best trees remained. The filtering was done by going through the list top-down and collecting only the unseen trees (by comparing each to the previously collected ones) until N trees were gathered. The time spent on

filtering is included in the reported running times but negligible compared to the running time of Tiburon.

Figure 3 shows how the running times of Filtered Runs and Best Trees compare on the two extreme example sets. Even though Best Trees is polynomial on Basic Example (as confirmed above), it is much less efficient than Filtered Runs, which can be explained with the insignificant degree of nondeterminism of the wtas in Basic Example. Turning to Equal Weights instead, the situation changes: as seen in Fig. 3b, Best Trees remains polynomial whereas Filtered Runs appears to exhibit the expected exponential behaviour as it is sensitive to the high (and growing) degree of nondeterminism in Equal Weights.



(a) Comparison of Best Trees and Filtered Runs when run on the Basic Example wta set for $N = 200$.

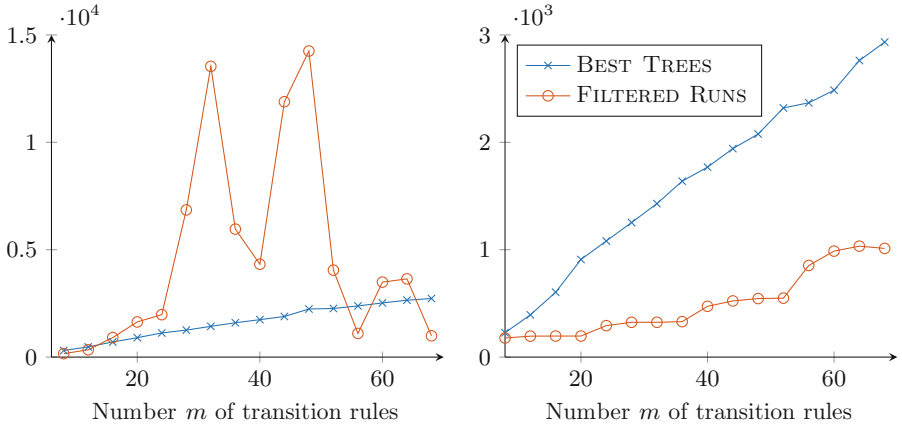
(b) Comparison of Best Trees and Filtered Runs when run on the Equal Weights wta set for $N = 30$.

Fig. 3. Best Trees and Filtered Runs when run on Basic Example and Equal Weights for fixed N.

The results for the less extreme sets Different Weights and Modified Different Weights are shown in Fig. 4. On the former, the running time of Filtered Runs is quite erratic, which can be explained with the changing degree of nondeterminism. In particular, the running time drops towards the end because the degree of nondeterminism does. Because of the low, and slowly but steadily growing degree of nondeterminism of Modified Different Weights, Filtered Runs runs faster on this set and the behaviour is much smoother as wta sizes increase.

4.4 Discussion

An obvious advantage of Best Trees is that its argument N is simply the number of best trees desired. In contrast, using Filtered Runs with the current



(a) Comparison of Best Trees and Filtered Runs when run on the Different Weights wta set.

(b) Comparison of Best Trees and Filtered Runs when run on the Modified Different Weights wta set.

Fig. 4. Best Trees and Filtered Runs when run on Different Weights and Modified Different Weights for $N = 100$.

interface of Tiburon makes it necessary to guess the number d_N of runs needed to produce sufficiently many best runs. To avoid a trial and error procedure, one would have to compute an appropriate – and thus exponentially large – upper bound N' on d_N from N, resulting in a running time that is guaranteed to be exponentially less efficient than Best Trees even in cases where the actual d_N equals N. However, note that this is not an intrinsic weakness of Filtered Runs because the best runs algorithm of [5] allows for a lazy implementation that outputs runs one by one upon request. It should thus not be difficult to modify the Tiburon implementation so as to allow for $N' = \infty$, resulting in an infinite (lazy) list of runs that can be inspected in Filtered Runs to extract any number of best trees. The more serious limitation of Filtered Runs is that its running time is directly related to the number N' of best runs required, which is difficult to predict. The running times reported in our experiments are therefore indicative of how many runs had to be computed for each value of N.

Overall, Best Trees shows a more predictable and smoother behaviour than Filtered Runs, allowing us to predict its time consumption more reliably if the structure of input wtas is not well known (see Fig. 4a).

During the experiments, the memory usage of Tiburon became too high when running it on the nondeterministic wta sets for $N > 10$, forcing us to increase the memory allocated for the JVM from 2–4 GB to 16 GB. In contrast, Best Trees did not encounter such problems and thus seems to use less memory.

5 Conclusion

We have experimentally validated the running time of Best Trees, the N-best algorithm for wtas proposed in [1], and the practical results were in line with the theoretical predictions. We then continued to compare Best Trees with Filtered Runs. Whereas Best Trees can be said to take a direct approach, Filtered Runs is indirect in the sense that it computes a large number of best runs, and then discards those that duplicate previously outputted trees.

As it is easy to create "artificial" examples that make Filtered Runs exponentially less efficient than Best Trees, we used more benign input automata in our experiments. The experimental results confirm that the degree of nondeterminism has a decisive influence on the relative efficiency of both methods. The greater simplicity of Filtered Runs makes it preferable if the degree of nondeterminism is low and the input wtas are large. Conversely, if the degree of nondeterminism is high in comparison to the size of the wta, Best Trees is the more efficient algorithm.

Another, more general, advantage of Best Trees is that it provides guarantees, and that it avoids unpredictable behaviour such as the one seen in Fig. 4a, which may be important in applications where the structure of the input automata is varied or unclear. Using the currently available implementation provided by Tiburon, one can also add that it is inconvenient to be forced to guess the number of runs needed to get N distinct trees.

References

1. Björklund, J., Drewes, F., Zechner, N.: Efficient enumeration of weighted tree languages over the tropical semiring. J. Comput. Syst. Sci. (2017)
2. Fülöp, Z., Vogler, H.: Weighted tree automata and tree transducers. In: Droste, M., Kuich, W., Vogler, H. (eds.) Handbook of Weighted Automata. Monographs in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01492-5_9
3. Gécseg, F., Steinby, M.: Tree Automata. Akadémiai Kiadó (1984). https://arxiv.org/abs/1509.06233
4. Gécseg, F., Steinby, M.: Tree languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, Chap. 1, pp. 1–68. Springer, Heidelberg (1997). https://doi.org/10.1007/978-3-642-59126-6_1
5. 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)
6. Knuth, D.E.: A generalization of Dijkstra's algorithm. Inf. Process. Lett. **6**, 1–5 (1977)
7. May, J., Knight, K.: Tiburon: a weighted tree automata toolkit. In: Ibarra, O.H., Yen, H.-C. (eds.) CIAA 2006. LNCS, vol. 4094, pp. 102–113. Springer, Heidelberg (2006). https://doi.org/10.1007/11812128_11
8. Mohri, M., Riley, M.: An efficient algorithm for the n-best-strings problem. In: Proceedings of the Conference on Spoken Language Processing (2002)