Theoretical Computer Science 682 (2017) 30–41

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Finding the N best vertices in an infinite weighted hypergraph ☆, ☆☆

Johanna Björklund *, Frank Drewes *, Anna Jonsson *

Department of Computing Science, Umeå University, 901 87 Umeå, Sweden

ARTICLE INFO

Article history:
Received 21 September 2016
Received in revised form 24 February 2017
Accepted 14 March 2017
Available online 27 March 2017

Keywords:
Hypergraph
N-best problem
Idempotent semiring

ABSTRACT

We propose an algorithm for computing the N best vertices in a weighted acyclic hypergraph over a *nice* semiring. A semiring is nice if it is finitely-generated, idempotent, and has $\mathbb{1}$ as its minimal element. We then apply the algorithm to the problem of computing the N best trees with respect to a weighted tree automaton, and complement theoretical correctness and complexity arguments with experimental data. The algorithm has several practical applications in natural language processing, for example, to derive the N most likely parse trees with respect to a probabilistic context-free grammar.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

Suppose that we can solve an optimisation problem A by solving, in succession, the problems $A_1, \ldots, A_n$. One way of approaching the joint optimisation over the cascade $A_1, \ldots, A_n$ is to find the N best solutions to A_1, and take these as input to A_2. We then compute the N best solutions to A_2 for each of these inputs, and prune the combined output down to the N best alternatives. The computation continues in this fashion until we have the outputs for A_n, at which point we take the top-ranking one as the best solution to A. In general, this approach will not yield an optimal solution, but it is often a viable heuristic.

The problem of finding N top-scoring elements with respect to some ranking device is referred to as the N-best problem. Such rankings can be computed by weighted automata, which associate with each element a value in some ordered semiring. The ranking prefers cheaper elements to more expensive ones, that is, N distinct elements with as small values as possible are sought. Typically, such automata may be non-deterministic, in which case the values of the individual runs (that is, of the nondeterministic computations) are summed up. The N-best problem is thus a harder problem than the related *N-best derivations problem*, which asks for the N best individual runs of the automaton. In the most common situation, where the semiring is the min-plus semiring with addition being minimum, the best run is always a run on the best element, but among the N best runs several may actually be runs on the same element.

Mohri and Riley [7] provide an algorithm for solving the N-best problem for weighted string automata over the min-plus semiring $\mathcal{A} = (\mathbb{R}_+^\infty, \min, +, 0, \infty)$. To keep the running time polynomial, they use a combination of lazy determinisation and

☆ Dedicated to Prof. Dr. Jürgen Dassow on the occasion of his 70th birthday.

☆☆ Preliminary presentations of different parts of this work have been given at *Weighted Automata: Theory and Applications* (WATA 2016) and *Trends in Tree Automata and Tree Transducers* (TTATT 2016).

* Corresponding authors.
E-mail addresses: johanna@cs.umu.se (J. Björklund), drewes@cs.umu.se (F. Drewes), aj@cs.umu.se (A. Jonsson).

http://dx.doi.org/10.1016/j.tcs.2017.03.010
0304-3975/© 2017 Elsevier B.V. All rights reserved.

Dijkstra's N-shortest paths algorithm. In [2], we generalise this algorithm to work for weighted tree automata over $\mathcal{A}$.[1] Such an extension is of interest because weighted tree automata are widely used in natural language processing to represent the parse trees of probabilistic context-free grammars. The probability values can be computed in the Viterbi semiring, or alternatively in the min-plus semiring by moving to the domain of negative log likelihoods. The latter method is preferred in many practical applications as it leads to better numerical precision. The extension to weighted tree automata in [2] simplifies the search technique by working directly with the input automaton rather than an on-the-fly determinisation. To mitigate the added dimensionality caused by working with trees rather than strings, an additional pruning technique is applied in order to arrive at an efficient algorithm. The resulting running time is roughly comparable to that in [3] for computing the N-best derivations.

In this paper we consider the N-best problem for weighted hypergraphs over nice semirings, i.e. semirings which are idempotent, finitely generated, and have $\mathbb{1}$ as their smallest element. The convenience of idempotency in the context of search algorithms has already been shown in [6]. The hypergraphs may be infinite, but may not contain cycles. This provides an abstraction and generalisation of the approach in [2]. Roughly speaking, the hypergraph may be instantiated to represent the set of all trees over a given ranked alphabet. Each vertex corresponds to the root of a tree, and a hyperedge connects it to the vertices representing the roots of its direct subtrees. Naturally, the weight of each vertex should be the weight of the tree it represents, and hence the N-best problem is to find the N cheapest vertices of the hypergraph. The exact correspondence is somewhat more complex than that, because we also have to account for the states of the tree automaton and the distinction between final and non-final states.

The framework based on weighted hypergraphs over nice semirings can be adapted to weighted finite-state devices whose computation graphs are acyclic. These include finite-state string automata, unranked tree automata, and context-free grammars. Every extremal semiring is idempotent, so for example the Łukasiewicz-, Viterbi-, and Boolean semirings meet this condition. These semirings have also the property that the multiplicative $\mathbb{1}$ is their smallest element. Out of these, only the Boolean semiring is finitely generated and hence nice. However, for all three semirings, the subset of weights that actually appear in the computation graphs of the previously mentioned devices are generated by the devices' edge- and transition weights, which make up finite sets. This observation makes our framework applicable to, e.g., weighted string automata over the Viterbi semiring.

1.1. Outline

This paper is organised as follows. Section 2 recalls basics of graph theory and algebra, and introduces nice semirings and layered graphs. In Section 3, we present an N-best meta algorithm for weighted hypergraphs and prove its correctness. The algorithm is abstract in the sense that it builds on top of a small set of auxiliary procedures, and the realisation of these procedures depends on the type of graph being studied. The algorithm is used in Section 4 to solve the N-best problem for wta over the min-plus semiring and the theoretical results are complemented with practical experiments. Section 5 concludes this paper by summarizing our findings and mapping out future work.

2. Preliminaries

We write $\mathbb{N}$ for the set of non-negative integers, $\mathbb{N}^\infty$ for $\mathbb{N} \cup \{\infty\}$, $\mathbb{R}_+$ for the set of non-negative reals, and $\mathbb{R}_+^\infty$ for $\mathbb{R}_+ \cup \{\infty\}$. Given $n \in \mathbb{N}$, we let $[n] = \{1, \ldots, n\}$, and $[\infty] = \mathbb{N}$. In particular, $[0] = \emptyset$. Let S and S' be a pair of (possibly infinite) sets. The number of elements of S is written $|S|$, and the powerset of S is denoted by $pow(S)$. Given a k-tuple $v = (a_1, \ldots, a_k)$ we may denote its ith component a_i ($i \in [k]$) by $v(i)$. A sequence of elements of S is *non-repetitive* if it does not contain the same element in two distinct positions. Given a function $\pi : S \to S'$, we extend π to a function from sequences over S to sequences over S' in the usual elementwise fashion. Similarly, π is extended to a function $\pi : pow(S) \to pow(S)$. Note that we do not make a notational distinction between π and these two canonical extensions of π. Given a non-empty sequence w, $[w]$ denotes the smallest set S such that w is a sequence over S. The set of all strings (i.e., finite sequences), over S is denoted by S^*; it includes the empty string λ.

A commutative *semiring* is a tuple $\mathcal{A} = (A, \oplus, \otimes, \mathbb{0}, \mathbb{1})$ such that $(A, \oplus, \mathbb{0})$ and $(A, \otimes, \mathbb{1})$ are commutative monoids, $\otimes$ distributes both-sided over $\oplus$, and $\mathbb{0}$ is an absorbing element with respect to $\otimes$. In the following, we generally assume that $\otimes$ binds stronger than $\oplus$, so $a \oplus b \otimes c$ is interpreted as $a \oplus (b \otimes c)$.

Since all semirings considered in this paper are commutative, we do not explicitly mention commutativity throughout the rest of the paper.

A *quasi-order* on S is a reflexive, transitive, binary relation $\leq$. We write $a < b$ to express that $a \leq b$ but $b \not\leq a$. As usual, $\geq$ and $>$ denote the inverses of $\leq$ and $<$, resp. A quasi-order is a *partial order* if it is antisymmetric. It is *well-founded* if there are no infinite descending chains, i.e., there is no infinite sequence $a_1 > a_2 > a_3 > \cdots$. A stronger notion than that of well-foundedness is that of a *well quasi-order* (wqo). A quasi-order is a wqo if every infinite sequence $a_1, a_2, \ldots$ eventually increases, i.e., there are $i < j$ such that $a_i \leq a_j$.

[1] A version of [2] providing more detail can be found in [1].

A semiring $\mathcal{A}$ is *idempotent* if $a \oplus a = a$ for all $a \in A$. In this case, there is a partial order $\leq_{\mathcal{A}}$ on $\mathcal{A}$, called the *natural order* of $\mathcal{A}$, which is given by $a \leq_{\mathcal{A}} b \iff a \oplus b = a$. Idempotent semirings are *monotonic* with respect to their natural order [6, Lemma 2], in other words, $a \leq_{\mathcal{A}} b$ implies $a \odot c \leq_{\mathcal{A}} b \odot c$ for all $c \in A$ and $\odot \in \{\oplus, \otimes\}$.

The semiring $\mathcal{A}$ is *finitely generated* if there is a finite subset A' of A such that every $a \in A$ is a sum of products of elements in A'. A finitely generated idempotent semiring in which $\mathbb{1}$ is the smallest element is said to be *nice*. In almost all applications, the fact that nice semirings are required to be finitely generated is an insignificant restriction because weights are usually constructed on the basis of a finite subset of $\mathcal{A}$ (such as the weights of the rules of a weighted tree automaton), and the sub-semiring generated by this subset is by definition finitely generated.

Example 1. Clearly, every extremal semiring is idempotent, but not every idempotent semiring is extremal: The min-plus semiring over $\mathbb{R}_+^\infty$ is an idempotent and extremal semiring with $\min(a, b)$ serving as semiring addition and ordinary addition as semiring multiplication. However, by generalising the domain to vectors of length $k \in \mathbb{N}$, $k \geq 2$, over $\mathbb{R}_+^\infty$ and applying semiring addition and multiplication component-wise, we get an idempotent semiring that is not extremal. If we restrict the domain of the min-plus semiring to $\mathbb{N}^\infty$, it becomes finitely generated (by $\{\mathbb{0}, \mathbb{1}, \infty\}$) and, in fact, nice because $\mathbb{1} = 0$ is its smallest element. The extension to vectors is still nice.

Rather than working on ordinary graphs, in which edges have a single source and a single target, we consider hypergraphs in which hyperedges may have multiple sources ordered as a sequence, but still only one target. This is particularly convenient for representing sets of (ordered) trees: a hyperedge labelled f with n sources corresponds to an occurrence of the symbol f of rank n in a tree. The sources and the target represent the roots of the direct subtrees and of the tree itself, respectively.

Definition 1 *(Hypergraph).* A *hypergraph* is a tuple $G = (V, E, src, tar)$ such that

- V and E are disjoint sets of *vertices* and *hyperedges*, respectively,
- $src\colon E \to V^*$ assigns to each $e \in E$ a sequence of *sources* $src(e)$, and
- $tar\colon E \to V$ assigns to each $e \in E$ a *target* $tar(e)$.

A vertex $v \in V$ is called an *end* if $v \notin [src(e)]$ for all $e \in E$. A *path* in G is a non-empty sequence of edges $\pi = e_0 \cdots e_n$ such that, for all $i \in [n]$, $tar(e_{i-1}) \in [src(e_i)]$. The *sources* and the *target* of π are $src(\pi) = src(e_0) \in V^*$ and $tar(\pi) = tar(e_n) \in V$, respectively, and we say that $tar(\pi)$ is *reachable* from e_0. The set of all paths in G is denoted $paths(G)$. The set of vertices that are *descendants* of v is given by

$$descendants(v) = \{v\} \cup \bigcup_{\substack{\pi \in paths(G),\\ v = tar(\pi)}} [src(\pi)]\,.$$

If $v' \in descendants(v)$, then v' is a *descendant* of v, and v is an *ancestor* of v'.

With this definition, hypergraphs can be infinite structures and may have parallel hyperedges. Ordinary graphs are a special case of hypergraphs, namely the one where each hyperedge has exactly one source. In the following, we shall simply speak of graphs and edges instead of hypergraphs and hyperedges.

Given a graph $G = (V, E, src, tar)$, we define $hull\colon pow(V) \to pow(V)$ by

$$hull(U) = \{tar(e) \mid e \in E,\ src(e) \in U^*\}$$

for all $U \subseteq V$. Hence, $hull(U)$ yields the set of all vertices that can be reached in one step from edges all of whose sources are in U. In particular, $hull(\emptyset)$ is the set of vertices that are targets of edges without sources (the *leaves*). Furthermore, we let $hull^{\leq 0}(U) = U$ and $hull^{\leq n+1}(U) = hull^{\leq n}(hull(U) \cup U)$ for $n \in \mathbb{N}$.

Definition 2 *(Layered graph).* A graph G is *layered* if (a) $hull^{\leq n}(\emptyset)$ is finite for every $n \in \mathbb{N}$, (b) for every vertex $v \in V$ there are only finitely many paths π such that $tar(\pi) = v$, and (c) $V = tar(E)$.

In particular, layered graphs are acyclic by requirement (b). Requirement (c) can be guaranteed by attaching to every vertex that is not the target of any edge a dummy edge having the vertex in question as a target and no sources. However, this is only possible if there are only finitely many such vertices, because otherwise it would make $hull(\emptyset)$ infinite, thus violating (a). Requirement (c) makes sure that all of G can gradually be built up from the "bottom" by starting with the empty set of vertices and repeatedly applying *hull*, thus following edges upward from children to their parents. In this process requirement (a) makes sure that the subgraph obtained always stays finite.

Example 2 *(Layered graph).* Definition 2 identifies a class of graphs for which weight functions can be conveniently defined, and which can be searched for optimal vertices by starting at the leaves and gradually expanding the search area using the operation *hull*. Not all graphs are of this kind. Suppose for instance that we have a non-trivial algebra and consider the

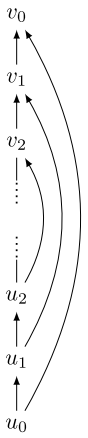


Fig. 1. A graph that satisfies condition (a) but not condition (b).

graph in which every term in this algebra is a vertex, and in which there is an edge from a term t to a term s if t is a proper subterm of s. Clearly, every vertex has infinitely many outgoing edges, so condition (a) of Definition 2 is not satisfied. However, as every term has finitely many subterms, condition (b) is fulfilled. (Condition (c) is not satisfied either, but this could be remedied in the way mentioned above.)

Consider then the graph in which the set of vertices is $\{u_0, u_1, u_2, \dots\} \cup \{v_0, v_1, v_2, \dots\}$ and where there are edges from u_i to u_{i+1}, from v_{i+1} to v_i, and from u_i to v_i, for every $i \in \mathbb{N}$ (see Fig. 1). Thus, u_0 is the unique leaf and v_0 is the unique end. Since no vertex has more than two outgoing edges, the hull of any finite set of vertices can be nothing but a finite set. As $hull(\emptyset)$ is finite as well, condition (a) of Definition 2 is fulfilled. On the other hand, every vertex v_i can be reached from every u_j such that $j \geq i$ and is, hence, the target of infinitely many paths. Thus condition (b) is not met. In a weighted setting such as the one developed below, such graphs do not seem to admit reasonably defined weights unless the semiring is assumed to have infinite sums.

Finally, suppose that we have a non-deterministic Turing machine M and a fixed initial configuration c_0. We consider the graph in which the vertices are the pairs (c, n) such that c is a configuration of M that is reachable from c_0 in n steps. There is an edge from (c, n) to $(c', n+1)$ if c' can be reached in one step of M from c. Since the number of outgoing edges leaving each vertex is finite and the number of configurations reached in at most n steps is also finite, both condition (a) and (b) are satisfied. If we, in addition, mark $(c_0, 0)$ with a sourceless edge, condition (c) is fulfilled as well.

Let $\mathcal{A} = (A, \oplus, \otimes, \mathbb{0}, \mathbb{1})$ be a semiring. A *graph with weights in* $\mathcal{A}$, also called a *weighted graph*, is a tuple $G = (V, E, src, tar, wgt)$ such that (V, E, src, tar) is a layered graph and $wgt\colon E \to \mathcal{A}$ is its *weight assignment*. The induced *weight function* $w\colon V \cup E \to A$ determined by the following conditions:

- $w(v) = \bigoplus_{e \in tar^{-1}(v)} w(e)$ for every vertex v and
- $w(e) = wgt(e) \otimes \bigotimes_{i \in [n]} w(v_i)$ for every edge e, where $src(e) = v_1 \cdots v_n$.

To see that w is uniquely determined, let $\ell(v) = 2m$ for every $v \in V$, where m is the maximum length of all paths π such that $tar(\pi) = v$. Due to conditions (b) and (c) of Definition 2, the set of those paths is non-empty and finite, and hence $\ell(v)$ is well defined. For an edge e, let $\ell(e) = \ell(tar(e)) - 1$. Now, induction on $\ell(x)$ uniquely determines $w(x)$ for all $x \in V \cup E$, as follows. For $v \in V$, if $e_1, \dots, e_m$ are the (finitely many!) pairwise distinct edges having v as their target, then $\ell(e_i) < \ell(v)$ for all $i \in [m]$. Hence, the induction hypothesis yields that $w(e_i)$ is uniquely determined for every $i \in [m]$, which determines $w(v) = \bigoplus_{i \in [m]} w(e_i)$. Similarly, for every edge e with $src(e) = v_1 \cdots v_n$, we have $\ell(v_i) < \ell(e)$ and thus, by the induction hypothesis, $w(v_i)$ is uniquely determined for every $i \in [n]$, which gives us $w(e) = wgt(e) \otimes \bigotimes_{i \in [n]} w(v_i)$.

For $m, m' \in \mathbb{N}^k$, we let $m \leq m'$ if $m(i) \leq m'(i)$ for all $i \in [k]$. An element $m \in M$ of a finite set $M \subseteq \mathbb{N}^k$, is *minimal* if there is no $m' \in M$ such that $m' < m$. The subset of M of minimal elements is denoted $min(M)$.

Given a semiring $\mathcal{A} = (A, \oplus, \otimes, \mathbb{0}, \mathbb{1})$ and $k \in \mathbb{N}$, we denote by $\mathcal{A}^k$ the semiring whose domain is A^k and in which $\oplus$ and $\otimes$ are applied element-wise and zero and one are the vectors $\mathbb{0}^k = (\mathbb{0}, \dots, \mathbb{0})$ and $\mathbb{1}^k = (\mathbb{1}, \dots, \mathbb{1})$, respectively. It is easy to verify that $\mathcal{A}^k$ is a semiring, and that $\mathcal{A}^k$ is nice if $\mathcal{A}$ is nice.

3. Computing N-best vertices

Let us now define the N-best vertices problem. As is common for many search applications where N-best problems are of interest (for example, shortest path, maximum likelihood, least weight) one may restrict oneself to semirings that are idempotent and choose their natural order as the order to be considered. If the natural order of such an idempotent semiring $\mathcal{A}$ is furthermore well-founded, we say that $\mathcal{A}$ is *well-founded*.

Definition 3 *(N-best vertices).* The *N-best vertices problem* is defined as follows. An instance is a pair (G, V^T, N) consisting of

- a layered weighted graph $G = (V, E, src, tar, wgt)$ with weights in a well-founded semiring $\mathcal{A}$,
- a set of *target vertices* $V^T \subseteq V$ such that each $v \in V^T$ is an end, and
- an integer $N \in \mathbb{N}^\infty$ such that $N \leq \left|V^T\right|$.

A solution is a sequence of N pairwise distinct elements $v_1, v_2, \cdots$ of V^T such that there do not exist $i \in [N]$ and $v \in V^T \setminus \{v_1, \ldots, v_i\}$ with $w(v) <_{\mathcal{A}} w(v_i)$.

The requirement that each $v \in V^T$ is an end deserves commenting on. The N-best vertices problem could equally well be formulated for arbitrary sets of target vertices. However, in this case there could be $v, v' \in V^T$ such that $v \in hull^{\leq i}(\emptyset)$ and $v' \in hull^{\leq j}(\emptyset) \setminus hull^{\leq i}(\emptyset)$ with $i < j$ but $w(v) > w(v')$. The algorithm devised below would then become more involved, as it would have to account for the possibility to visit v on its way to discovering and outputting v', later on potentially having to reconsider v in order to output v itself. With each vertex in V^T being an end, there are fewer cases to consider.

The restriction of target vertices to ends is not severe, but mainly serves to simplify the presentation. If a vertex $v \in V^T$ violates the restriction, one may modify G by adding a new vertex $\hat{v}$ and a single edge with weight $\mathbb{1}$ (i.e., the multiplicative unit) pointing from v to $\hat{v}$, and then replacing v by $\hat{v}$ in V^T. This rewriting process essentially creates new 'dummy' vertices that are ends, to replace target vertices that are not. Clearly, if $\hat{v}_1, \ldots, \hat{v}_N$ is a solution to the modified problem instance, then $v_1, \ldots, v_N$ is a solution to the original one.

In practical instantiations of Definition 3, the graph G can be used to model the runs of an automaton, and V^T to identify accepting states. This is for instance the case in Section 4, where we compute the N best trees with respect to a weighted tree automaton.

We note here that Definition 3 does not specify how G and V^T are represented. Regarding V^T, it is only assumed that there is a procedure that computes the characteristic function of V^T, in other words, that it decides for a vertex $v \in V$ whether $v \in V^T$. As for the graph G, since it is layered we know that for each vertex v the subgraph G^v is finite, where G^v consists of all vertices and edges lying on a path π such that $tar(\pi) = v$. Below, we will add the assumption that there are procedures that construct G^v or subgraphs thereof, for certain vertices v. The algorithm will use this in order to construct the portion of G needed to solve the N-best vertices problem.

Before continuing, let us verify that the N-best vertices problem is well defined in the sense that a solution always exists. This is what the assumption of well-foundedness is needed for.

Lemma 1. *Every instance of the N-best vertices problem has a solution.*

Proof. Choose (not necessarily constructively) any element u_0 of V^T and build a sequence of vertices $u_0, u_1, u_2, \ldots$ in V^T with strictly decreasing weights. By assumption, the natural order of $\mathcal{A}$ is well-founded. Therefore, every such sequence is finite. Thus, the process eventually arrives at a vertex v_1 such that no vertex of strictly lesser weight exists in V^T. Now, fix v_1 and repeat the argument with $V^T \setminus \{v_1\}$ instead of V^T. Continue until N elements $v_1, \ldots, v_N$ have been found (or *ad infinitum* in case $N = \infty$). By construction, $v_1, \ldots, v_N$ is a solution. □

Not all idempotent semirings are well-founded; examples showing this are easy to construct. However, as we shall prove next, nice semirings are indeed well-founded.

Lemma 2. *Every nice semiring $\mathcal{A}$ is well-founded.*

Proof. The proof is divided into two parts. We first show that every element in $\mathcal{A}$ can be written as a sum of products of elements in a generating set $\{a_1, \ldots, a_k\}$. Due to idempotency and the definition of the natural order, the value of this sum is decided by its minimal summands, and these can be represented as vectors in $\mathbb{N}^k$. In the second part, we show that if there were an infinite strictly decreasing sequence of elements in $\mathcal{A}$, then there would also be an infinite non-increasing sequence of vectors in $\mathbb{N}^k$. Dickson's lemma, which states that $(\mathbb{N}^k, \leq)$ is a wqo, ensures us that this is not possible.

Let $\mathcal{A}$ be generated by $\{a_1, \ldots, a_k\}$. For $m \in \mathbb{N}^k$ let $\varphi(m) = \bigotimes_{j \in [k]} a_j^{m(j)}$, and for a finite set $M \subseteq \mathbb{N}^k$ let $\Phi(M) = \bigoplus_{m \in M} \varphi(m)$, where $\Phi(\emptyset) = \mathbb{0}$. Then, since $\otimes$ distributes over $\oplus$, every $a \in A$ is represented by at least one finite set $M \subseteq \mathbb{N}^k$ in the sense that $a = \Phi(M)$. We call such a set a *representative* of a.

Let us now prove that the value of $\Phi(M)$ is determined by the minimal elements of M, i.e., that the following statement holds:

$$\text{For finite } M \subseteq \mathbb{N}^k,\ \Phi(M) = \Phi(min(M)). \tag{1}$$

Indeed, if $m, m' \in M$ are such that $m < m'$, then we have $\varphi(m') = \varphi(m) \otimes a$ for $a = \varphi(m' - m)$. Since $\mathcal{A}$ is monotonic, $\varphi(m) = \varphi(m) \otimes \mathbb{1} \leq_{\mathcal{A}} \varphi(m) \otimes a = \varphi(m')$ and thus $\varphi(m) \oplus \varphi(m') = \varphi(m)$ by the definition of $\leq_{\mathcal{A}}$. Every such non-minimal element $m' \in M$ can therefore be removed from M with no effect on $\Phi(M)$, which proves (1).

Suppose there is a descending sequence $s = b_0 >_{\mathcal{A}} b_1 >_{\mathcal{A}} b_2 >_{\mathcal{A}} \cdots$ and let M_i be a representative of b_i for all $i \in \mathbb{N}$. By the definition of the natural order, $b_i >_{\mathcal{A}} b_{i+1}$ implies that $\Phi(M_i \cup M_{i+1}) = \Phi(M_i) \oplus \Phi(M_{i+1}) = \Phi(M_{i+1})$, so we may assume that $M_0 \subseteq M_1 \subseteq \cdots$ and thus $M_0 \subsetneq M_1 \subsetneq \cdots$ because $b_i \neq b_{i+1}$ for all $i \in \mathbb{N}$. Now, pick $m_i \in min(M_i) \setminus min(M_{i-1})$ for all $i > 0$. Then $m_i \not\leq m_j$ for $0 < i < j$ since $m_i \leq m_j$ would imply $m_j \notin min(M_j) \setminus min(M_i)$. In other words, the sequence $m_1, m_2, \ldots$ is non-increasing and must thus be finite because $(\mathbb{N}^k, \leq)$ is a wqo. This shows that the sequence s is finite, and thus that $\mathcal{A}$ is well-founded. □

We shall now devise a simple algorithm that solves the N-best vertices problem. For the remainder of this section, we consider an instance (G, V^T, N), where $G = (V, E, src, tar, wgt)$, and take $\mathcal{A}$ to be nice. As mentioned earlier, we require that G can be explored effectively by means of a few procedures that represent G. What we provide is therefore a meta algorithm, in which different realisations of the procedures yield different instantiations of the overall algorithm. These procedures are as follows (where $u \in V$):

- There is a procedure that computes $hull(U)$ for every finite set $U \subseteq V$.
- The procedure $minWeight(u)$ returns a minimal element of the set of all $w(v)$ such that $v \in V^T$ and v is reachable from u. If no vertex in V^T is reachable from u, then an error element $\top$ is returned, which is considered to be larger than all elements of $\mathcal{A}$. Note that $minWeight(u)$ is well-defined owing to the well-foundedness of $\mathcal{A}$ (cf. the proof of Lemma 1).
- If $minWeight(u) \neq \top$ then $bestAncestor(u)$ returns some $v \in V^T$ with $w(v) = minWeight(u)$ that is reachable from u.
- Finally, $descendants(u)$ returns the set of vertices v of which u is an ancestor (including u itself). Note that this set is finite as G is layered.

Algorithm 1 Solving the generic N-best vertices problem.

```
1: procedure BestVertices(G, V^T, N)
2:     U ← ∅;
3:     for i = 1, ..., N do
4:         H ← hull(U) \ U;
5:         select u ∈ H such that minWeight(u) is minimal;
6:         v ← bestAncestor(u);
7:         output v;
8:         U ← U ∪ descendants(v)
9:     end for
10: end procedure
```

The pseudocode of our algorithm is given in Algorithm 1. We now show that it is correct.

Lemma 3. *After $i \leq N$ executions of the loop body in Algorithm 1, it will have outputted i distinct vertices $v_1, \ldots, v_i \in V^T$ such that*

(1) there are no $v \in V^T \setminus \{v_1, \ldots, v_i\}$ and $j \in [i]$ such that $v <_{\mathcal{A}} v_j$, and
(2) $U = \bigcup_{j \in [i]} descendants(v_j)$.

Proof. We proceed by induction on i. For $i = 0$ the assertions are trivially true as U is initialised to $\emptyset$. Thus, assume that (1) and (2) hold for some $i - 1 < N$ and consider the ith execution of the loop. Let $H^{\uparrow}$ be the set of all vertices that are reachable from some vertex in $H = hull(U) \setminus U$. Thus, $H^{\uparrow}$ is the set of vertices the algorithm chooses v_i from. We show that exactly those vertices not yet in U are reachable from a vertex in H, i.e. that $H^{\uparrow} = V \setminus U$.

By (2) all descendants of every $u \in U$ are in U as well. But $H \cap U = \emptyset$, which implies that no vertex in U is reachable from H, i.e., that $U \cap H^{\uparrow} = \emptyset$. (If a vertex $u \in U$ would be reachable from $h \in H$ then $h \in descendants(u) \subseteq U$, contradicting the fact that $H \cap U = \emptyset$.) To show that $H^{\uparrow} = V \setminus U$, it thus remains to check that $v \in H^{\uparrow}$ for all $v \in V \setminus U$. As G is layered, the set Π of all paths π that satisfy $tar(\pi) = v$ is non-empty (by condition (c)) and finite (by condition (b)). It follows that one can construct a path $\pi = e_1 \cdots e_n \in \Pi$ such that $tar(e_1) \notin U$ and $[src(e_1)] \subseteq U$. (Start with any path of length 1 in Π. If the current path is $e_1 \cdots e_n$ but there is $u \in [src(e_1)] \setminus U$, extend it to $e_0 e_1 \cdots e_n$ where $tar(e_0) = u$; note that e_0 exists by condition (c), and that this process must eventually stop by condition (b).) Thus $tar(e_1) \in H$, which proves that $v \in H^{\uparrow}$. This proves that, indeed, $H^{\uparrow} = V \setminus U$. In particular, since all vertices in V^T are ends,

$$V^T \setminus \{v_1, \ldots, v_{i-1}\} = V^T \setminus \bigcup_{j \in [i]} descendants(v_j) \subseteq V \setminus U = H^{\uparrow} \ .$$

Since a solution does exist, this means that there is a $v_i \in (V^T \cap H^{\uparrow}) \setminus U$ of minimal weight, which implies that $\{v_1, \ldots, v_i\}$ satisfies (1). It follows that Line 5 assigns u this vertex v_i (or another vertex in V^T of the same weight), that Line 6 assigns v the vertex $bestAncestor(u)$, and that Line 7 outputs v. Thus, (2) is now satisfied for i. □

It may be worthwhile noting that, since the set U in Algorithm 1 is always of the form $\bigcup_{j\in[i]} descendants(v_j)$ for the already outputted ends $v_1, \ldots, v_i$, the procedures $hull(U)$, $minWeight(u)$, and $bestAncestor(u)$ only need to be implemented for this special case.

4. Application to weighted tree automata

Let us now discuss in which sense Algorithm 1 generalises the N-best algorithm in [2] for weighted tree automata (wta) over the min-plus semiring $\mathcal{T} = (\mathbb{R}^{\infty}_{+}, min, +, \infty, 0)$ (also called the *tropical semiring*). To regain the original algorithm in [2] from Algorithm 1, the auxiliary procedures *hull*, *minWeight*, *bestAncestor*, and *descendants* can be realised in the domain of computation graphs generated by wta. For this it is useful to recall parts of the theoretical framework of [2].

4.1. Weighted tree automata

For a set Σ, a Σ-labelled *tree* is a partial function $t\colon \mathbb{N}^* \to \Sigma$ 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 *vertex of* t, and k is the *rank of* v.

A *ranked alphabet* is a finite non-empty 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 $\mathbb{T}_\Sigma$ of all trees over Σ consists of all Σ-labelled trees t such that the rank of every vertex $v \in dom(t)$ equals the rank of $t(v)$.

The *subtree* of $t \in \mathbb{T}_\Sigma$ 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 $t(\lambda)$.

For a set T of trees we denote by $\Sigma(T)$ the set of trees

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

Thus, $\Sigma(T)$ consists of all trees which have a symbol from Σ at their root, with direct subtrees in T.

We let $\square \notin \Sigma$ be a special symbol of rank 0. The set of *contexts over* Σ is

$$C_\Sigma = \{c \in \mathbb{T}_{\Sigma\cup\{\square\}} \mid \text{ there is exactly one } v \in dom(c) \text{ with } c(v) = \square\}\ .$$

Given a context $c \in C_\Sigma$, where $v \in dom(c)$ is the unique vertex such that $c(v) = \square$, we define the *substitution* of a tree t into c to be the tree $c[\![t]\!]$ given as follows: $dom(c[\![t]\!]) = dom(c) \cup \{vu \mid u \in dom(t)\}$ and, for $w \in dom(c[\![t]\!])$,

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

A *weighted tree language* (over the min-plus semiring $\mathcal{T}$) is simply a mapping $\mathcal{W}\colon \mathbb{T}_\Sigma \to \mathcal{T}$, where Σ is a ranked alphabet. A *weighted tree automaton* over $\mathcal{T}$ is a system $M = (Q, \Sigma, \delta, \rho)$ where

- Q is a ranked alphabet of *states*, all of rank 0, called *states*;
- Σ is a ranked alphabet of *input symbols* disjoint with Q;
- $\delta\colon \Sigma(Q) \times Q \to \mathcal{T}$ is the *transition function*; and
- $\rho\colon Q \to \mathcal{T}$ is the assignment of *final weights*.

For $q \in Q$, δ inductively gives rise to a function $\delta_q\colon \mathbb{T}_\Sigma \to \mathcal{T}$, as follows: For all $t = f[t_1, \ldots, t_k] \in \mathbb{T}_\Sigma$

$$\delta_q(t) = min_{q_1,\ldots,q_k\in Q} \left(\delta(f[q_1, \ldots, q_k], q) + \sum_{i\in[k]} \delta_{q_i}(t_i) \right)\ . \tag{2}$$

Now, the weighted tree language $\mathcal{W}_M\colon \mathbb{T}_\Sigma \to \mathcal{T}$ recognised by M is given by

$$\mathcal{W}_M(t) = min_{q\in Q}\ \big(\rho(q) + \delta_q(t)\big) \text{ for all } t \in \mathbb{T}_\Sigma\ .$$

We extend δ_q, and thus $\mathcal{W}_M$ to $\mathbb{T}_{\Sigma\cup Q}$ by defining $\delta_q(q) = 0$ and $\delta_q(q') = \infty$ for all $q' \in Q \setminus \{q\}$. It follows that, for all trees t and all contexts c,

$$\mathcal{W}_M(c[\![t]\!]) = min_{q\in Q}\ \big(\mathcal{W}_M(c[\![q]\!]) + \delta_q(t)\big)\ . \tag{3}$$

For the sake of completeness, and for later reference, let us now recall the N-best trees problem for this device. An instance of this problem is a pair (M, N) consisting of a wta $M = (Q, \Sigma, \delta, \rho)$ over $\mathcal{T}$ and an integer $N \in \mathbb{N}^{\infty}$. A solution is a sequence of N pairwise distinct trees $t_1, t_2, \cdots$ such that there do not exist $i \in [N]$ and $t \in \mathbb{T}_\Sigma \setminus \{t_1, \ldots, t_i\}$ with $\mathcal{W}_M(t) <_{\mathcal{T}} \mathcal{W}_M(t_i)$. To make sure that a solution always exists, we may for simplicity assume that $\Sigma \neq \Sigma_{(0)} \neq \emptyset$, in other words, that $\mathbb{T}_\Sigma$ is an infinite set.

4.2. The computation graph of a weighted tree automaton

Let $M = (Q, \Sigma, \delta, \rho)$ be a wta over $\mathcal{T}$. In the following, we abbreviate $Q \times \mathbb{T}_\Sigma$ by $Q\langle\mathbb{T}_\Sigma\rangle$ and a pair $(q,t) \in Q\langle\mathbb{T}_\Sigma\rangle$ by $q\langle t\rangle$. The weighted graph $G_M = (V, E, src, tar, wgt)$ associated with M, which also has weights in $\mathcal{T}$, is given by the following components:

- $V = Q\langle\mathbb{T}_\Sigma\rangle \cup \mathbb{T}_\Sigma$,
- For every $t = f[t_1, \ldots, t_k] \in \mathbb{T}_\Sigma$ and all $q, q_1, \ldots, q_k \in Q$, there is an edge $e \in E$ with $src(e) = q_1\langle t_1\rangle \cdots q_k\langle t_k\rangle$, $tar(e) = q\langle t\rangle$, and $wgt(e) = \delta(f[q_1, \ldots, q_k], q)$. Below, we denote this edge e by $e^t_{q_1\cdots q_k, q}$.
- Further, for every $q \in Q$ and $t \in \mathbb{T}_\Sigma$, there is an edge e such that $src(e) = q\langle t\rangle$, $tar(e) = t$, and $wgt(e) = \rho(q)$.

It is straightforward to show that G_M is layered. The ends of G_M are the vertices in $\mathbb{T}_\Sigma$, and thanks to the edges of the second kind we have $w(t) = min_{q\in Q}\,(\rho(q) + w(q\langle t\rangle))$ for every tree $t \in \mathbb{T}_\Sigma$. Thus, in order to establish the desired equality $w(t) = \mathcal{W}_M(t)$ we need to verify that $w(q\langle t\rangle) = \delta_q(t)$. We prove this by induction on $t = f[t_1, \ldots, t_k]$. Thus, assume that the claim holds for all trees smaller than t. Then the definition of the weight function of G_M (on both vertices and edges) yields

$$\begin{aligned} w(q\langle t\rangle) &= \min_{q_1,\ldots,q_k\in Q}\ w(e^t_{q_1\cdots q_k,q}) \\ &= \min_{q_1,\ldots,q_k\in Q}\ \left(wgt(e^t_{q_1\cdots q_k,q}) + \sum_{i\in[k]} w(q_i\langle t_i\rangle)\right) \\ &= \min_{q_1,\ldots,q_k\in Q}\ \left(\delta(f[q_1,\ldots,q_k],q) + \sum_{i\in[k]} \delta_{q_i}(t_i)\right) \\ &= \delta_q(t) \end{aligned}$$

as required. We have thus shown the following theorem:

Theorem 1. *Let $M = (Q, \Sigma, \delta, \rho)$ be a wta over $\mathcal{T}$ and $N \in \mathbb{N}^\infty$. Then a solution to the N-best vertices problem for $(G_M, \mathbb{T}_\Sigma, N)$ is also a solution to the N-best trees problem for (M, N).*

4.3. Specialisation of Algorithm 1

Since the min-plus semiring $\mathcal{T}$ is extremal, it is idempotent, and $\mathbb{1} = 0$ is its smallest element. However, it is not nice because it is not finitely generated. Upon closer consideration, we see that the subset $\mathcal{F}$ of $\mathcal{T}$ that is actually used in the computations of M is generated by 0, 1, the set $\{\rho(q) \mid q \in Q\}$ of final weights, and the set $\{\delta(f[q_1, \ldots, q_k], q) \mid k \in \mathbb{N}, f \in \Sigma_{(k)}, q, q_1, \ldots, q_k \in Q\}$ of edge weights. In other words, $\mathcal{F}$ is finitely generated. This makes Algorithm 1 applicable to $(G_M, \mathbb{T}_\Sigma, N)$, and it follows from Theorem 1 that the result of running it on $(G_M, \mathbb{T}_\Sigma, N)$ is a solution to the N-best trees problem for (M, N). What remains to obtain a concrete algorithm is to provide efficient implementations of *hull*, *minWeight*, *bestAncestor*, and *descendants*.

For simplicity, let us assume that there are at least N trees of finite weight. We give only a rough description of how Algorithm 1 can be specialised to this case, focusing on the main points rather than reiterating all the arguments from [2].

For the sake of efficiency, an explicit search of $H = hull(U) \setminus U$ should be avoided in line 5 of Algorithm 1. To make this easier, we can assume that $\mathcal{W}_M(s) = \infty$ for all proper subtrees s of trees of finite weight. As argued in [2] this is easy to achieve by adding a special root symbol r to $\Sigma_{(1)}$. As a consequence we can simplify our further considerations by removing all ends (i.e., all vertices in $\mathbb{T}_\Sigma$) from H, as their weight is ∞ (due to the fact that t is a subtree of a tree of finite weight for all $q\langle t\rangle \in U$). In other words, from now on let $H = (hull(U) \setminus U) \cap Q\langle\mathbb{T}_\Sigma\rangle$.

The algorithm in [2] was made to run in polynomial time by exploiting the regular structure of wta computations. Consider a tree t, and suppose that we want to minimise $\mathcal{W}_M(c[\![t]\!]) = w(c[\![t]\!])$ over all $c \in C_\Sigma$. From Equation (3) we know that $w(c[\![t]\!]) = min_{q\in Q}\mathcal{W}_M(c[\![q]\!]) + \delta_q(t)$. Therefore, we precompute a set of contexts c_q such that, for each $q \in Q$, $\mathcal{W}_M(c_q[\![q]\!]) = min_{c\in C_\Sigma}\mathcal{W}_M(c[\![q]\!])$. As shown in [2] this can be done efficiently using an algorithm by Knuth [5]. Let then $w_q = \mathcal{W}_M(c_q[\![q]\!])$ for all $q \in Q$, a quantity that we may also precompute. Then it is very cheap to compute $minWeight(p\langle t\rangle) = min_{c\in C_\Sigma} w(c[\![t]\!]) = min_{q\in Q} w_q + \delta_q(t)$ (which is, in fact, independent of the state p). Moreover, if q is the state that minimises the rightmost expression, then $bestAncestor(p\langle t\rangle) = c_q[\![t]\!]$. This yields Algorithm 2. Note that the computation of $\delta_q(t)$ in Line 5 is inexpensive because the algorithm may, along with every tree $t \in \mathbb{T}_\Sigma$ encountered during the run of the algorithm, keep the vector of weights $\delta_q(t)$, $q \in Q$. When a tree t is encountered for the first time, the weight vectors of its subtrees are already available, which means that the weight vector of t is obtained by $|Q|$ applications of Equation (2), not requiring any recursion.

Algorithm 2 Solving the N-best trees problem.

```
1: procedure BestVertices(M, N)
2:     U ← ∅;
3:     for i = 1, ..., N do
4:         H ← Q⟨Σ(U ∩ T_Σ)⟩ \ U;
5:         select q⟨t⟩ ∈ H such that w_q + δ_q(t) is minimal;
6:         s ← c_q[[t]];
7:         output s;
8:         U ← U ∪ {p⟨s'⟩ | p ∈ Q, s' ∈ S} ∪ S where S = {s/v | v ∈ dom(s)}
9:     end for
10: end procedure
```

To enable a quick selection of $q\langle t\rangle$ in line 5 of Algorithm 2, the vertices in H can be maintained in a priority queue K, where the priority of an element $q\langle t\rangle$ is determined by $\Delta(q\langle t\rangle) = w_q + \delta_q(t)$, with lower-weighted vertices given precedence. Since $minWeight(q\langle t\rangle) = min_{p\in Q} w_p + \delta_p(t)$, it holds for the first queue element $q\langle t\rangle$ (though not in general) that $\Delta(q\langle t\rangle) = minWeight(q\langle t\rangle)$. Furthermore, this $minWeight(q\langle t\rangle)$ is minimal over all enqueued trees. The selection of an appropriate $q\langle t\rangle \in H$ thus boils down to a dequeuing operation, and a subsequent removal of all remaining elements in $\{p\langle t\rangle \mid p \in Q\}$ from K.[2] The procedure *descendants* is easily implemented, as shown in Line 8.

Finally, let us discuss briefly the pruning technique introduced in [2] to make the algorithm run faster. The efficiency of the basic algorithm is less than optimal because the queue K usually contains many more elements than necessary if we populate it by all of H. The key observation is that, since we are only interested in outputting N trees, for each $q \in Q$ at most N of the most promising (i.e. lowest-weight) vertices $q\langle t\rangle \in U$ may indeed be needed. In other words, U can be pruned down by keeping, for every $q \in Q$, only a set of N vertices $q\langle t_1\rangle, \ldots, q\langle t_N\rangle \in U$. These must be chosen in such a way that $\sum_{i=1}^{N} w_q(t_i)$ is minimised. (If U contains fewer than N vertices $q\langle t\rangle$ for the state q in question, no pruning takes place.) The usage of pruning makes the algorithm much more efficient without affecting its correctness, leaving the non-pruned version superfluous for this particular specialisation of Algorithm 1.

We omit a correctness proof and the accompanying running time estimations because they carry over from [2] in a straightforward way. In particular, assuming that the input alphabet is either fixed or the maximum rank of its symbols is small compared to N, the time complexity of the resulting, pruning, algorithm is $O\left(N^2 \log N \cdot n \cdot m\right)$, where m and n are the number of transitions and the number of states of M, respectively. The complexity of the less efficient, non-pruning version has not been formally investigated. It is likely exponential unless particular care is taken in the graph-exploration step, that is, in the computation of what is called *hull* in the present work and *expand* in [2].

4.4. Experiments

An implementation of both the pruning and the non-pruning version of the N-best algorithm can be found in the *BestTrees* GitHub repository.[3] The distribution consists of the Java source code along with a number of wta examples. There is also a runnable `.jar` file of the latest version for convenience.

The experiments were run on a family of wtas, language equivalent with the wta M of Fig. 2. Here, $\Sigma = \Sigma_{(0)} \cup \Sigma_{(2)}$ with $\Sigma_{(0)} = \{a, b\}$ and $\Sigma_{(2)} = \{\circ\}$. Let $\|t\|_\sigma$ $(\sigma \in \Sigma_{(0)})$ denote the number of occurrences of σ in $t \in T_\Sigma$, and let $\|t\|$ denote the total number of leaves of t. Then we have

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

Algorithm 1 was first run on M itself, allowing N to range between 0 and 80. The average run times are summarised in Fig. 3 and, with a logarithmic scale, in Fig. 4. As expected, the pruning version is considerably more efficient. In fact, without pruning the algorithm exhibits an exponential behaviour. For the pruning version, the run time remained below 10 seconds for the entire test series, whereas the running time of the non-pruning version exceeded a quarter of an hour for the higher values of N. While the theoretical worst case running time of the pruning version is $O(N^2 \log N)$ for a fixed wta, Fig. 4 indicates that it may be nearly linear in practice, at least in this particular case. This is strengthened by Fig. 5, which clearly shows the almost linear behaviour.

Further tests were then conducted on different-sized automata derived from M by introducing redundancy. More specifically, we added new states and transitions to count the height of subtrees up to a varying bound k. This means that for any two distinct automata M' and M'' thus obtained, the automaton for the smaller k is a subautomaton of the one for the larger k. The size of the largest of these automata, measured as $m + n$, was 1500. Again the value of N was varied, though now on the smaller range between 0 and 50, and the algorithm was run both with and without pruning. The results are shown in Figs. 6 and 7. We see that the size of the wta does not matter significantly for the version of the algorithm

[2] In [2], the priority queue would not contain the trees $q\langle t\rangle$, but only the trees $t \in \mathbb{T}_\Sigma$. This difference does not affect the result because, as noted above, $minWeight(q\langle t\rangle)$ and $bestAncestor(q\langle t\rangle)$ are independent of q.

[3] https://github.com/tm11ajn/besttrees.

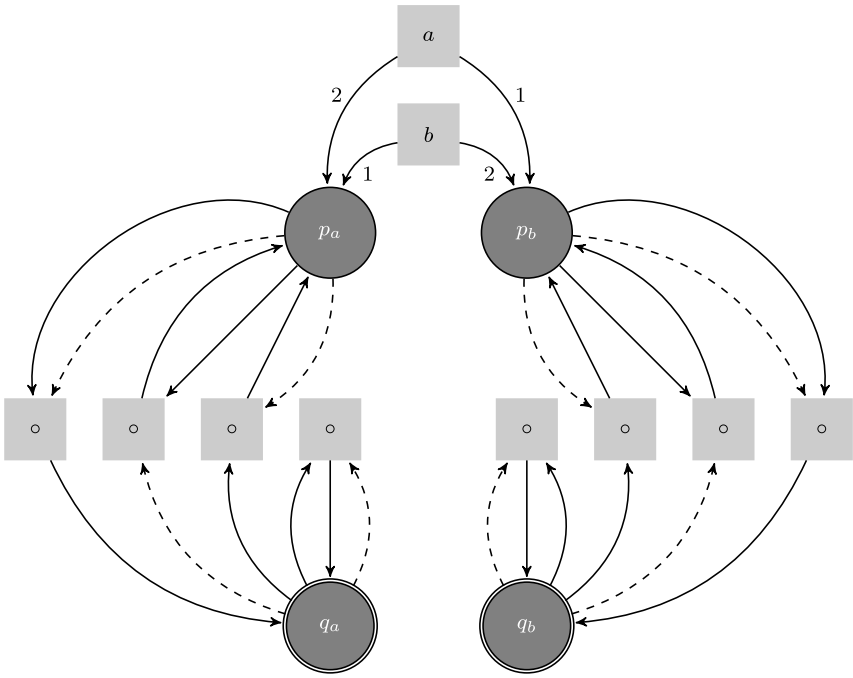


Fig. 2. The input wta used in our experiments, taken from [2]. Round nodes (with double circles if final) represent states, and squares represent transitions. The consumed input symbols are shown inside the squares. Solid arcs point to the right-hand side of the transition in question and are labelled with the weight of the transition unless it is zero. In the case of input symbol ∘ 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, unnecessary.

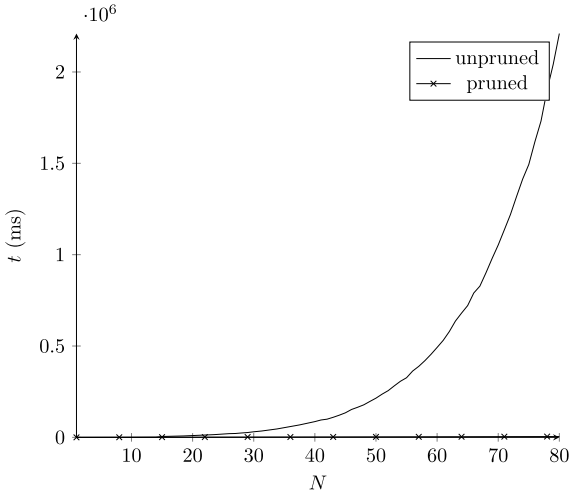


Fig. 3. The running time of both versions of the algorithm applied to the wta in Fig. 2 for different values of N.

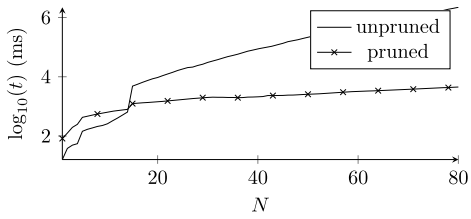


Fig. 4. The running time of both versions of the algorithm applied to the wta in Fig. 2 for different values of N on a logarithmic scale.

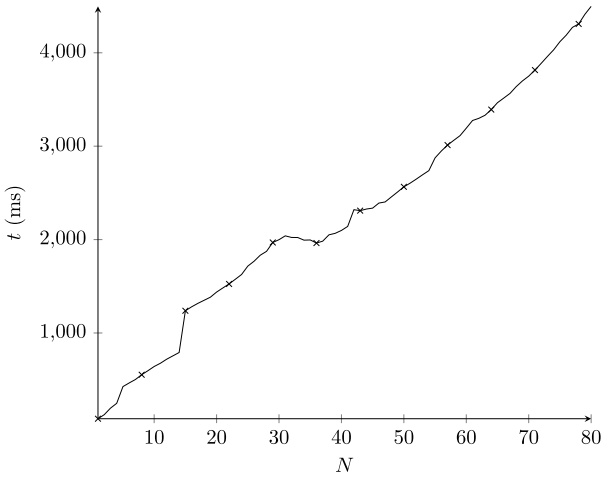


Fig. 5. The running time of the pruning version of the algorithm applied to the wta in Fig. 2 for different values of N.

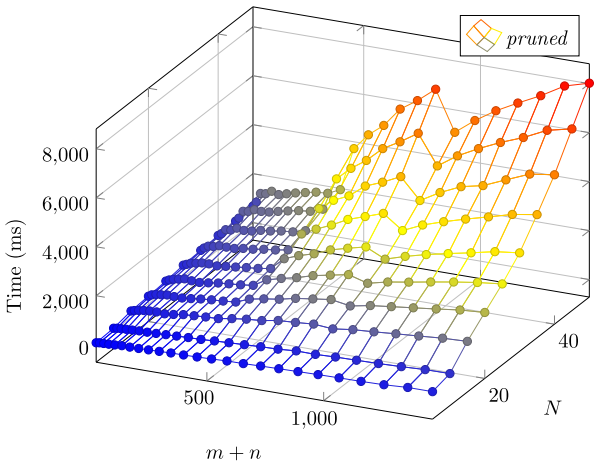


Fig. 6. The running time of the pruned version of the algorithm for different values of N on different sized wtas (measured in $m+n$).

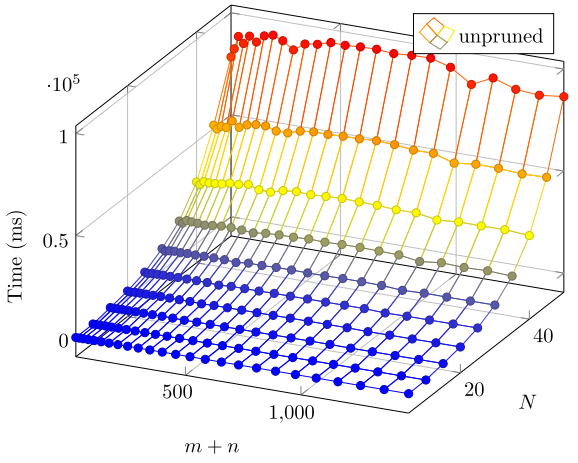


Fig. 7. The running time of the unpruned version of the algorithm for different values of N on different sized wtas (measured in $m+n$).

that does not use pruning. This is not surprising since the same hulls are calculated during the execution of the algorithm, regardless of the size of the wta (see footnote 2), and the number of iterations is determined solely by N. As with the initial

tests, the run time for the pruning version was below 10 seconds on all instances, whereas the version without pruning took minutes when N was close to 50.

5. Conclusion and future work

We have defined the N-best problem for layered graphs with edge-weights in a nice semiring. Whenever the graph representation permits the procedures *hull, minWeight, bestAncestor*, and *descendants* to be (efficiently) implemented, so can our N-best algorithm. As we have seen, this is the case for weighted tree automata over the min-plus semiring. Our initial experiments on this restricted domain indicated furthermore that the pruning technique proposed in [2], which carries over to the setting of the present paper, has the expected positive impact on the running time of the algorithm.

To understand the practical applicability of Algorithm 1 to problems in natural language processing, a study on real-world data is needed. Rather than using the non-pruning version of the algorithm as a baseline, it seems reasonable to use the state-of-the-art-approach, which is computing the N' best runs for some $N' \gg N$ and discarding duplicates. On the theoretical side, we would like to weaken the assumptions on the underlying graph and semiring. In particular, it would be interesting to see whether there is a class of semirings beyond the min-plus semiring which allows for an efficient implementation of the procedures *hull, minWeight, bestAncestor*, and *descendants* in the case of wta, thus yielding efficient solutions to the N-best trees problem based on Theorem 1.

A potentially rewarding line of future work is the instantiation of Algorithm 1 to work with less traditional types of objects, for example pictures. Suppose that we are given an algebra consisting of a domain of pictures and a set Σ of operations on them.[4] Under suitable well-definedness conditions, a wta M over Σ then defines a weighted picture language, the weight of a picture P being the sum of all $\mathcal{W}_M(t)$ such that t evaluates to P. (Alternative definitions are possible as well, such as defining the weight of P to be equal to $\mathcal{W}_M(t)$, and requiring that different trees representing the same picture yield the same weight.) Then the nodes of the input graph to the N-best vertices problem would be pictures rather than trees, and the output would be a set of N best pictures. Future work could try to make this rough idea more precise, in order to discover solvable variants of the N-best pictures problem.

Acknowledgements

We thank Andreas Maletti and the participants at the *4th International Workshop on Trends in Tree Automata and Tree Transducers (TTATT 2016)*, where a preliminary and abridged version of the paper was presented, for their insightful comments that led to various improvements. Last but by no means least, we are grateful to the anonymous reviewers for pointing out flaws and suggesting significant improvements in various places.

References

[1] Johanna Björklund, Frank Drewes, Niklas Zechner, An Efficient Best-Trees Algorithm for Weighted Tree Automata Over the Tropical Semiring, Report UMINF 14.22, Umeå University, 2014.
[2] Johanna Björklund, Frank Drewes, Niklas Zechner, An efficient best-trees algorithm for weighted tree automata over the tropical semiring, in: Proc. 9th Intl. Conf. on Language and Automata Theory and Applications, LATA 2015, in: LNCS, vol. 8977, 2015, pp. 97–108.
[3] Matthias Büchse, Daniel Geisler, Torsten Stüber, Heiko Vogler, n-Best parsing revisited, in: Proceedings of the 2010 Workshop on Applications of Tree Automata in Natural Language Processing, Association for Computational Linguistics, Uppsala, Sweden, July 2010, pp. 46–54.
[4] Frank Drewes, Grammatical Picture Generation – A Tree-Based Approach, Texts in Theoretical Computer Science, An EATCS Series, Springer, 2006.
[5] Donald E. Knuth, A generalization of Dijkstra's algorithm, Inform. Process. Lett. 6 (1977) 1–5.
[6] Mehryar Mohri, Semiring frameworks and algorithms for shortest-distance problems, J. Autom. Lang. Comb. 7 (3) (2002) 321–350.
[7] Mehryar Mohri, Michael Riley, An efficient algorithm for the n-best-strings problem, in: Proceedings of the Conference on Spoken Language Processing, 2002.

[4] See [4] for various types of such picture algebras.