Technische Universität Dresden

Masterarbeit

The Problem of Computing the Most Probable Tree of a Probabilistic Tree Automaton

Pius Friedrich Meinert

Fakultät Informatik
Institut für Theoretische Informatik
Lehrstuhl für Grundlagen der Programmierung

Betreuer:
Dipl.-Inf. Kilian Gebhardt

Erstgutachter:
Prof. Dr.-Ing. habil. Heiko Vogler

Zweitgutachter:
Dr.-Ing. Stefan Borgwardt

02. August 2019

Erklärung der Urheberschaft

Hiermit versichere ich, die vorliegende Arbeit selbstständig, ohne fremde Hilfe und ohne Benutzung anderer als der von mir angegebenen Quellen angefertigt zu haben. Alle aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche gekennzeichnet. Die Arbeit wurde noch keiner Prüfungsbehörde in gleicher oder ähnlicher Form vorgelegt.

Dresden, 02. August 2019

Unterschrift von Pius Friedrich Meinert

Aufgabenstellung

“The Problem of Computing the Most Probable Tree of a Probabilistic Tree Automaton”

Technische Universität Dresden
Fakultät Informatik

Student:	Pius Friedrich Meinert
Geburtsdatum:	8th January 1993
Matrikelnummer:	4661105
Studiengang:	Master Informatik
Immatrikulationsjahr:	2016
Modul:	Masterarbeit
Studienleistung:	Masterarbeit
Umfang:	22 weeks, 29 CP
Beginn am:	1st March 2019
Einzureichen am:	2nd August 2019
Verantw. Hochschullehrer:	Prof. Dr.-Ing. habil. Heiko Vogler
Betreuer:	Dipl.-Inf. Kilian Gebhardt

Weighted tree automata (WTA) are well-studied devices in the field of theoretical computer science [FV09]. Their application in *natural language processing* (NLP) is often with the probabilistic semiring, the Viterbi semiring, and the tropical semiring for training and decoding purposes: During weight training with the expectation/maximization algorithm, the probabilistic semiring and the inside/outside algorithm [Bak79] are used to compute the expected frequency of rule applications for the recognition of a certain training sample. However, during decoding, i.e., the application of the automaton to new inputs, the Viterbi semiring (or, equivalently the tropical semiring after conversion in a negative log-domain) is applied to find the most probable run [Knu77; Ned03].

The main reason why WTA over the probabilistic semiring (also called *probabilistic tree automata* (PTA)) are not used during decoding is the lack of a suitable algorithm to find the most probable tree and its anticipated run-time. The probability of some tree is the sum of the probabilities of all runs of the PTA on this tree. The probability of each run is the product of the probabilities of the rules it comprises. Apparently, optimizing this sum of products is NP-hard. This follows, on the one hand, from the NP-hardness of finding the most probable tree of a probabilistic tree-substitution grammar [Sim02] and, on the other hand, from the NP-hardness of finding the most probable string recognized by a *probabilistic finite (string) automaton* (PFA) [CdlH00].

Thus, in order to apply PTA one either assumes that there is no ambiguity, e.g., by ensuring that the PTA is deterministic and searching for the best run which coincides with the most probable tree in this case. Alternatively, one simply assumes that the best run is a good approximation of the most probable tree.

For some probabilistic finite automata A de la Higuera and Oncina [dlHO13b; dlHO13a] have analyzed the problem of finding the most probable string recognized by A. In particular, they

- device an algorithm that solves the decision problem whether given a PFA A, a bound $b \in \mathbb{N}$, and a probability p there exists a string w of length $\leq b$ with probability $> p$ in polynomial time [dlHO13b],
- show that there is a PFA A for which the most probable string has length exponential in the size of A [dlHO13b],
- show that a string w having probability p is of bounded length [dlHO13a],
- give an algorithm that computes the most probable string (with probability p) in time polynomial in the inverse of p [dlHO13a], and
- carry out experiments on synthetic automata to evaluate the quality of the bound and differences between most probable run and most probable string. [dlHO13a]

Thus, for PFA it is possible to compute the most probable string but the run-time of the presented algorithm depends on how probable the result is, which cannot be known in advance.

A careful analysis of the techniques applied by de la Higuera and Oncina [dlHO13b; dlHO13a] indicates that they can be generalized to probabilistic tree automata. Thus, the tasks of Pius Meinert to solve in his master thesis are as follows:

- He shall show that finding the most probable tree recognized by some PTA is NP-hard using a direct reduction of SAT similar to Sima'an [Sim02].
- He shall show that there is a PTA A for which the most probable tree has height exponential in the size of A.
- He shall show that a tree ξ having probability p is of bounded size.
- He shall give an algorithm that computes the most probable tree (with probability p) in time polynomial in the inverse of p.
- He shall implement this algorithm in a performant programming language and carry out experiments on suitable synthetic PTA to evaluate the quality of the bound and the differences between most probable run and most probable tree.

If the run-time of the implementation is empirically small enough to allow for experiments with real-world grammars from the domain of NLP, then a facultative task is to analyze differences in run-time and in quality compared to the most probable run (or other relevant decoding strategies).

Requirements. The student's work must satisfy the usual standards. The work must be self-contained and complete with all necessary definitions and references. The authorship of the content – including the own – must be clearly identifiable. Third-party content, e.g. algorithms, constructions, definitions, and ideas, must be clearly marked by appropriate references to the literature. Long literal citations shall be avoided. Where applicable, it must be explained to what extent and for which purpose third-party content was modified. The structure shall be clearly recognizable, and the reader shall be guided well through the work. The presentation of notions and methods shall be mathematically well-founded. The student shall provide explanations and examples for every major notion, method, and construction. Where appropriate, illustrations shall complete the presentation. Regarding diagrams, which illustrate phenomenons of experiments, it must be clearly explained, which values are shown by the various axes and which dependencies are shown between the values of these axes. Lemmas and theorems shall be proven as complete as possible. The proofs shall be presented in an easily comprehensible way.

The implementation shall be documented in detail. The documentation shall be reasonably distributed over the source code and the written part of the work. All dependencies and installation steps shall be documented. The installation shall be automated in a script if this is possible in a platform independent way. Alternatively, a platform independent containerized version of the program shall be compiled. The student must credibly show that the implemented programs and program parts function as required, which shall be documented by appropriate example runs. Reported experiments shall be automated in a single script including preprocessing steps to allow for reproducibility. If the run-time of experiments is very high, then this script shall be parameterized such that single experiments can be run independently, shared intermediate results are reused, and experiments on a subset of the data can be run. In these cases also the parametrization and results on the subset of the data shall be documented in the written report. The student agrees on a later release of the implementation under a free software license.

Dresden, 26th February 2019

Unterschrift von Heiko Vogler

Unterschrift von Pius Friedrich Meinert

References

[Bak79] James K Baker. “Trainable grammars for speech recognition”. In: *The Journal of the Acoustical Society of America* 65.S1 (1979), pp. 132–132.

[CdlH00] Francisco Casacuberta and Colin de la Higuera. “Computational Complexity of Problems on Probabilistic Grammars and Transducers”. In: *Grammatical Inference: Algorithms and Applications*. Ed. by Arlindo L. Oliveira. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 15–24. ISBN: 978-3-540-45257-7.

[dlHO13a] Colin de la Higuera and Jose Oncina. “Computing the Most Probable String with a Probabilistic Finite State Machine”. In: *Proceedings of the 11th International Conference on Finite State Methods and Natural Language Processing*. St Andrews, Scotland: Association for Computational Linguistics, July 2013, pp. 1–8. URL: `http://www.aclweb.org/anthology/W13-1801`.

[dlHO13b] Colin de la Higuera and Jose Oncina. “The most probable string: an algorithmic study”. In: *Journal of Logic and Computation* 24.2 (Jan. 2013), pp. 311–330. ISSN: 0955-792X. DOI: `10.1093/logcom/exs049`.

[FV09] Zoltán Fülöp and Heiko Vogler. “Weighted Tree Automata and Tree Transducers”. In: *Handbook of Weighted Automata*. Ed. by Manfred Droste, Werner Kuich, and Heiko Vogler. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 313–403. ISBN: 978-3-642-01492-5. DOI: `10.1007/978-3-642-01492-5_9`.

[Knu77] Donald E. Knuth. “A generalization of Dijkstra’s algorithm”. In: *Information Processing Letters* 6.1 (1977), pp. 1–5. ISSN: 0020-0190. DOI: `10.1016/0020-0190(77)90002-3`.

[Ned03] Mark-Jan Nederhof. “Weighted Deductive Parsing and Knuth’s Algorithm”. In: *Computational Linguistics* 29.1 (2003), pp. 135–143. DOI: `10.1162/089120103321337467`.

[Sim02] Khalil Sima’an. “Computational complexity of probabilistic disambiguation”. In: *Grammars* 5.2 (2002), pp. 125–151.

Abstract

In this work we show a number of results regarding the most probable tree (mpt) problem for probabilistic tree automata (pta) $\mathcal{A}$. We proof a bound on recognisable trees, showing that trees ξ with a high probability are shallow, i.e., $\mathrm{height}(\xi) \leq \frac{|\mathcal{A}|^2}{\mathrm{Pr}_{\mathcal{A}}(\xi)}$. Furthermore, we provide a pta construction for which the most probable tree is of superpolynomial size. The computation of the most probable tree itself is NP-hard. This we show by a polynomial-time reduction directly from 3-SAT. Lastly, present an algorithm for finding the most probable tree in time dependent on the mpt's probability.

Contents

1. Introduction

Natural language can be highly ambiguous: Even though an utterance is usually delivered with a certain analysis in mind, more often than not, it can be interpreted in a number of ways. Any attempt to model natural language should therefore incorporate such ambiguity (Chomsky 1956).

Nondeterministic finite state automata are one of the most basic formalisms for such purposes. Such an automaton employs state behaviour to gradually accept a sequence of words forming a sentence. A run of an automaton for a sentence is a succession of states where between these states the words of the sentence are recognised in order. In consequence, multiple different runs for the same sentence can represent different interpretations.

As not all of the various readings of a sentence are equiprobable, runs of an automaton should not be either. Accordingly, weights are assigned to transitions in an automaton, that is, to rules leading from one state to another while reading a word. Then, the weight of a run can be determined by multiplying all transition weights in the run. In addition, the score of a sentence is attained by summing up the weights of all its different runs. If weights are real numbers in the range of 0 to 1, they can be interpreted as probabilities. An automaton with this kind of weights is called probabilistic finite state automaton (pfa) and defines a probability distribution over a set of sentences (or strings).

A natural generalisation for pfa is to allow for branching and, thus, establish probabilistic tree automata (pta). Just as the former assign probabilities to strings, a pta defines a probability distribution over a set of trees. Probabilistic tree automata allow for modelling the syntactic structure of a sentence as it is for example captured in phrase structure trees that are provided by treebanks like the Penn Treebank (Marcus, Santorini, and Marcinkiewicz 1993). Tree automata in general have been proved to be useful in a variety of natural language processing applications (May and Knight 2006; Knight and May 2009; Koller and Kuhlmann 2011) and their formal properties have been studied extensively in the past (Engelfriet 2015; Gécseg and Steinby 1984; Fülöp and Vogler 2009).

One of the most interesting problems for probabilistic tree automaton is that of calculating the most probable tree. As this proves to be quite hard to determine, one typically settles for the computationally feasible approximation of finding the most probable run. Unfortunately, the tree with the best run and the most probable tree do not coincide in general.

In this work we examine the most probable tree problem and associated questions. This is motivated by a number of results for the corresponding problem for pfa, i.e., finding the most probable string. We show that these results are generalisable to probabilistic tree automata.

Subsequently, we provide an overview over the thesis' chapters: We start out with the presentation of necessary preliminaries in Chapter 2. This chapter is predominantly used to introduce the automata and related concepts that appear throughout the thesis. Whenever appropriate, definitions and notation are strongly based on the reference work for weighted tree automata by Fülöp and Vogler 2009.

Chapter 3 encompasses two results regarding the size of probable trees: The first, described in Section 3.1, is a bound on the height of trees depending on the size of the automaton and the tree's inverse probability. Essentially that implies that probable trees are shallow. This result is based on a proof by de la Higuera and Oncina 2013 (Section 3) for pfa and the size of strings.

The remaining sections deal more directly with the most probable tree problem: In Section 3.2 we provide a pta whose most probable tree is of a size greater than polynomial in the size of the automaton. By that we show that there are instances of pta for which determining the most probable tree cannot be accomplished in polynomial time. The equivalent for pfa has been shown in Section 4.4 of de la Higuera and Oncina 2014.

While the preceding demonstrates that the most probable tree problem is not in the class of problems that can be computed in nondeterministic polynomial time (NP), Chapter 4 shows that the problem is NP-hard, i.e., at least as difficult as the hardest problems in NP. Even though there exist proofs that already indicate NP-hardness for this problem (cf. Sima'an 2002; Casacuberta and de la Higuera 2000), we present a short, direct reduction from the NP-complete problem 3-SAT to the most probable tree problem.

Finally, albeit the problem being NP-hard, we develop an algorithm for the most probable tree problem in Chapter 5. The algorithm is an adaptation of an algorithm that finds the most probable string given a pfa (de la Higuera and Oncina 2013, Algorithm 1). Besides showing that it is theoretically possible to calculate the best tree, we implement the algorithm in the programming language Rust. With the help of that, we evaluate the practical applicability of the algorithm and how well the best run approximation holds up.

2. Preliminaries

In this chapter, we provide essential notation and definitions that are used throughout this work. We begin by stating more widely known mathematical concepts and how they are represented here. Afterwards, definitions for alphabets and trees are given which are required for the subsequent section. That section introduces definitions that lead to the notion of probabilistic tree automata. As a last point, a short section gives an introduction to some concepts of complexity theory.

2.1. General mathematical definitions and notation

For the notions of *sets* and *functions/mappings* we use the usual notation. As for generally known sets used here: $\mathbb{N}$ denotes the set of *natural numbers* including 0 and $\mathbb{R}$ the set of *real numbers* including 0. The sets $\mathbb{N}^+ = \mathbb{N} \setminus \{ 0 \}$ and $\mathbb{R}^+ = \mathbb{R} \setminus \{ 0 \}$ denote the sets of *positive natural numbers* and *positive real numbers*, respectively.

For every $k \in \mathbb{N}$, the set $\{ n \in \mathbb{N} \mid 1 \leq n \leq k \}$ is abbreviated by $[k]$. Similarly, for every $a, b \in \mathbb{R}$, the set $\{ r \in \mathbb{R} \mid a \leq r \leq b \}$ is abbreviated by $[a, b]$. A parentheses instead of a bracket indicates the exclusion of the bounding number, e.g., $(a, b] = [a, b] \setminus \{ a \}$.

Let I and A be sets. An *(I-indexed) family* is a function $f\colon I \to A$. Especially for families we often write a_i instead when we refer to an element $f(i) \in A$. Consequently, such a family f is introduced by $(a_i \mid i \in I)$.

Throughout this thesis, the symbols $+$ and $\cdot$ denote the usual addition and multiplication on the natural and real numbers. For an $r \in \mathbb{R}^+$, we denote the logarithm of r to base 2 by $\log(r)$. Let A, B be sets and $f\colon A \to B$ a mapping. We define $\max_{a \in A} f(a) = \{ f(a) \mid f(a') \leq f(a) \text{ for every } a' \in A \}$ and $\arg\max_{a \in A} f(a) = \{ a \mid f(a) \in \max_{a' \in A} f(a') \}$.

A *commutative semiring* is a tuple $(S, \oplus, \odot, \mathbf{0}, \mathbf{1})$ where

- $\oplus\colon S \times S \to S$ is an associative and commutative mapping,
- $\odot\colon S \times S \to S$ is an associative and commutative mapping that distributes over $\oplus$,
- $\mathbf{0} \in S$ is the identity element of $\oplus$ and the absorbing element of $\odot$, and
- $\mathbf{1} \in S$ is the identity element of $\odot$.

Oftentimes a semiring $(S, \oplus, \odot, \mathbf{0}, \mathbf{1})$ is referred to only by its carrier set S. The semiring that is used almost exclusively here, is the *probabilistic semiring* $([0,1], \cdot, +, 0, 1)$.

Probability Let Ω be a countable, nonempty set. A *σ-algebra* is a set $\mathcal{A} \subseteq \mathcal{P}(\Omega)$ where $\Omega \in \mathcal{A}$, for all $A \in \mathcal{A}$ it holds that $\Omega \setminus A \in \mathcal{A}$, and if $A_1, A_2, \ldots \in \mathcal{A}$ then $\bigcup_{i=1}^{\infty} A_i \in \mathcal{A}$. Let $\mathcal{A}$ be a σ-algebra. A *probability measure* is a function $\Pr\colon \mathcal{A} \to [0,1]$ such that $\Pr(\Omega) = 1$ and for every countable sequence $A_1, A_2, \ldots \in \mathcal{A}$ of pairwise disjoint sets, i.e., $A_i \neq A_j \implies A_i \cap A_j = \varnothing$,

$$\Pr\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \Pr(A_i).$$

Let $\omega \in \Omega$ and $A, B \subseteq \Omega$. We abbreviate $\Pr(\{\,\omega\,\})$ by $\Pr(\omega)$ and $\Pr(A \cap B)$ by $\Pr(A, B)$. Given pairwise disjoint sets $B_1, B_2, \ldots \subseteq \Omega$ with $\bigcup_{i=1}^{\infty} B_i = \Omega$: Due to the *law of total probability*, we have

$$\Pr(A) = \sum_{i=1}^{\infty} \Pr(A, B_i).$$

Since for our purposes more rudimentary probability definitions are sufficient, we omit the definition of random variables and do not delve deeper into probability theory. Instead, for a countable set A, we primarily use the definition of a *probability distribution* as a mapping $\Pr\colon A \to [0,1]$ such that $\sum_{a \in A} \Pr(a) = 1$.

2.2. Alphabets and trees

An *alphabet* is a finite, nonempty set whose elements are called *symbols*. A finite sequence of symbols from an alphabet Γ is called a *word over* Γ (often just referred to as *string*). The set of all words over Γ is denoted by Γ^*. Consider a word $w = w_1 w_2 \ldots w_n$ where $n \in \mathbb{N}, w_1, \ldots, w_n \in \Gamma$. If $n = 0$, we call w the empty word and denote it by ε.

A *ranked alphabet* is a tuple (Σ, rk) where Σ is an alphabet and $\mathrm{rk}\colon \Sigma \to \mathbb{N}$ is a mapping of symbols to natural numbers which we refer to as *rank*. For every $k \in \mathbb{N}$, we define $\Sigma^{(k)} = \{\,\sigma \in \Sigma \mid \mathrm{rk}(\sigma) = k\,\}$. Especially when introducing ranked alphabets we will directly provide the symbols including their ranks by writing $\sigma^{(k)}$, signifying that $\mathrm{rk}(\sigma) = k$. As the rank mapping is thereby implicitly given, we often identify the alphabet Σ with the tuple (Σ, rk).

Definition 1. Let Σ be a ranked alphabet and $X = \{ x_1, x_2, \dots \}$ be a set of variables disjoint from Σ. The *set of trees over* Σ *and* X, denoted by $T_\Sigma(X)$, is the smallest set $T \subseteq (\Sigma \cup X \cup \{ (,), , \})^*$ s.t.

1. $X \subseteq T$ and
2. if $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, and $\xi_1, \dots, \xi_k \in T$, then $\sigma(\xi_1, \dots, \xi_k) \in T$.

As a shorthand, $T_\Sigma(\varnothing)$ is denoted as T_Σ. Given a set of variables X, we often refer to a tree $\zeta \in T_\Sigma(X) \setminus T_\Sigma$ as *prefix* (in literature often called *contexts*) and to a tree $\xi \in T_\Sigma$ as *complete*.

Let Σ be a ranked alphabet, $X = \{ x_1, x_2, \dots \}$ be a set of variables disjoint with Σ and $\xi \in T_\Sigma(X)$ a tree. In the following we define operations on trees (cf. Fülöp and Vogler 2009, Section 2.2) and use the abbreviation $\xi = \sigma(\xi_1, \dots, \xi_k)$ for: There are $k \in \mathbb{N}, \sigma \in \Sigma^{(k)}$, and $\xi_1, \dots, \xi_k \in T_\Sigma(X)$ such that $\xi = \sigma(\xi_1, \dots, \xi_k)$. We define the *height*, *size*, and *set of positions* of trees as functions height: $T_\Sigma(X) \to \mathbb{N}$, size: $T_\Sigma(X) \to \mathbb{N}$, and pos: $T_\Sigma(X) \to \mathcal{P}(\mathbb{N}^*)$. First, height$(\xi) \in \mathbb{N}$ is recursively defined as

$$\operatorname{height}(\xi) = \begin{cases} 1 & \text{if } \xi \in X \\ 1 + \max_{i \in [k]} \operatorname{height}(\xi_i) & \text{if } \xi = \sigma(\xi_1, \dots, \xi_k). \end{cases}$$

Secondly, size$(\xi) \in \mathbb{N}$ is defined as

$$\operatorname{size}(\xi) = \begin{cases} 1 & \text{if } \xi \in X \\ 1 + \sum_{i \in [k]} \operatorname{size}(\xi_i) & \text{if } \xi = \sigma(\xi_1, \dots, \xi_k). \end{cases}$$

Thirdly, the set pos$(\xi) \subseteq \mathbb{N}^*$ is inductively defined as

$$\operatorname{pos}(\xi) = \begin{cases} \{ \varepsilon \} & \text{if } \xi \in X \\ \{ \varepsilon \} \cup \{ ip \mid i \in [k], p \in \operatorname{pos}(\xi_i) \} & \text{if } \xi = \sigma(\xi_1, \dots, \xi_k). \end{cases}$$

A *path for a tree* ξ (path) is a sequence of positions $\pi = p_1 p_2 \dots p_k \in \operatorname{pos}(\xi)^k$ where for all $i \in \{2, \dots, k\}$ there is a $c \in \mathbb{N}^+$ with $p_i = p_{i-1}c$ (continuous path).

Let $p \in \operatorname{pos}(\xi)$. The *label of* ξ *at* p is defined by

$$\xi(p) = \begin{cases} \xi & \text{if } \xi \in X \\ \sigma & \text{if } \xi = \sigma(\xi_1, \dots, \xi_k) \text{ and } p = \varepsilon \\ \xi_i(p_i) & \text{if } \xi = \sigma(\xi_1, \dots, \xi_k) \text{ and } p = ip_i \text{ for } i \in [k], p_i \in \operatorname{pos}(\xi_i). \end{cases}$$

The *subtree of* ξ *at* p is recursively defined by

$$\xi|_p = \begin{cases} \xi & \text{if } p = \varepsilon \\ \xi_i|_{p_i} & \text{if } \xi = \sigma(\xi_1, \dots, \xi_k) \text{ and } p = ip_i \text{ for } i \in [k], p_i \in \text{pos}(\xi_i). \end{cases}$$

Let $\zeta \in T_\Sigma(X)$. The *replacement of the subtree of* ξ *at* p *by* ζ is defined by

$$\xi[\zeta]_p = \begin{cases} \zeta & \text{if } p = \varepsilon \\ \sigma(\xi_1, \dots, \xi_{i-1}, \xi_i[\zeta]_{p_i}, \xi_{i+1}, \dots, \xi_k) & \begin{array}{l}\text{if } \xi = \sigma(\xi_1, \dots, \xi_k) \text{ and} \\ \quad p = ip_i \text{ for } i \in [k], p_i \in \text{pos}(\xi_i).\end{array} \end{cases}$$

Let $\xi_1, \dots, \xi_n \in T_\Sigma$. We define substituting every occurrence of a variable x_i in ξ by tree ξ_i, *tree substitution on* ξ *with* $\xi_1, \dots, \xi_n$, by

$$\xi(\xi_1, \dots, \xi_n) = \begin{cases} \xi_i & \text{if } \xi \in X \text{ and } \xi = x_i \\ \xi & \text{if } \xi \in X \text{ and } \xi \neq x_i \\ \xi & \text{if } \xi \in T_\Sigma \\ \xi[\xi_i]_p(\xi_1, \dots, \xi_n) & \begin{array}{l}\text{if } \xi = \sigma(\xi_1, \dots, \xi_k), \\ \quad i \in [k], p \in \text{pos}(\xi) \text{ and } \xi(p) = x_i.\end{array} \end{cases}$$

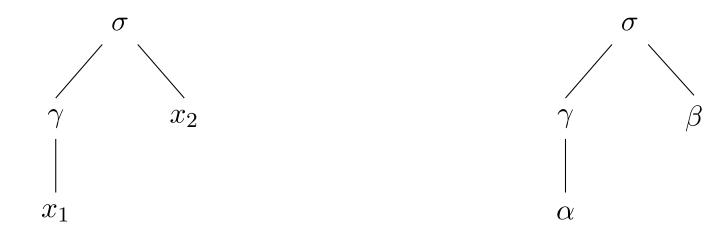
As a last operation for trees, we define the set of complete trees reachable from a prefix tree. That is, trees that that can be constructed by replacing all variables in prefix trees with complete trees.

Definition 2. Let Σ be a ranked alphabet, X a set variables and $\zeta \in T_\Sigma(X) \setminus T_\Sigma$ a prefix tree. We define the set of *reachable trees from* ζ as

$$T_\Sigma|_\zeta = \{\, \zeta(\xi_1, \dots, \xi_n) \in T_\Sigma \mid \xi_1, \dots, \xi_n \in T_\Sigma \,\}.$$

The concepts and the naming of prefix trees (normally *contexts*), complete trees (usually just *trees*), tree substitution and reachable trees are fairly uncommon. We therefore provide a short illustration in Example 1.

Example 1. Let $\Sigma = \{\, \sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}, \beta^{(0)} \,\}$ be a ranked alphabet and $X = \{\, x_1, x_2, \dots \,\}$ a set of variables. An example for a prefix tree in $T_\Sigma(X) \setminus T_\Sigma$ is given by $\zeta = \sigma(\gamma(x_1), x_2)$. Let $\xi_1 = \alpha$ and $\xi_2 = \beta$. The complete tree $\xi = \sigma(\gamma(\alpha), \beta)$ is the result of the tree substitution $\zeta(\xi_1, \xi_2)$. Consequently, ξ is reachable from prefix ζ, i.e., $\xi \in T_\Sigma|_\zeta$. A depiction of ξ and ζ is provided in Figure 1. $\triangle$



(a) $\zeta \in T_\Sigma(X) \setminus T_\Sigma$ (b) $\xi \in T_\Sigma|_\zeta$

Figure 1.: Prefix tree ζ and complete tree ξ of Example 1.

Lastly, we define tree series, assigning elements of a semiring set to trees:

Definition 3 (Fülöp and Vogler 2009, Section 2.4)**.** Let X be a set with $\Sigma \cap X = \varnothing$ and S a semiring. A *tree series over* Σ*,* X *and* S (or for short: tree series) is a mapping $r\colon T_\Sigma(X) \to S$. For every $\xi \in T_\Sigma(X)$, the element $r(\xi) \in S$ is called the *coefficient* of ξ and is denoted by (r, ξ).

The set of all tree series is denoted by $S\langle\langle T_\Sigma(X)\rangle\rangle$.

2.3. Automata

As most of our results are based on works about probabilistic finite automata, we initially state their definition as a reference. These automata recognise words over an alphabet (strings) and assign probabilities to them.

Definition 4 (cf. de la Higuera and Oncina 2013, Definition 1)**.** A *probabilistic finite automaton* (for short: pfa) is a tuple $\mathcal{B} = (Q, \Gamma, M, I, F)$ where:

- Q is a finite nonempty set, the *set of states.*
- Γ is the *input alphabet* with $\Gamma \cap Q = \varnothing$.
- $M\colon Q \times \Gamma \times Q \to [0,1]$ is the complete transition function.
- $I\colon Q \to [0,1]$ is a mapping of *initial probabilities.*
- $F\colon Q \to [0,1]$ is a mapping of *final probabilities.*

These pfa are a special case of weighted finite automata. A natural generalisation of such string automata is to build tree automata that accept tree series. An introduction to weighted tree automata and important results are represented by Fülöp and Vogler 2009. Our definitions are primarily based on theirs. In the following let S be a commutative semiring.

Definition 5 (Fülöp and Vogler 2009, Definition 3.2)**.** A *weighted tree automaton (over S)* (for short: wta) is a tuple $\mathcal{A} = (Q, \Sigma, S, \mu, \nu)$ where:

- Q is a finite nonempty set, the *set of states.*
- Σ is the *ranked input alphabet* with $\Sigma \cap Q = \varnothing$.
- $\mu = (\mu_\sigma \mid \sigma \in \Sigma)$ is a *family of transition mappings* $\mu_\sigma \colon Q^k \times Q \to S$ for $\sigma \in \Sigma^{(k)}$.
- $\nu \colon Q \to S$ is a mapping of *root weights.*

Let X be a set of variables. A *run of* $\mathcal{A}$ *on* $\xi \in T_\Sigma(X)$ is a mapping $\kappa \colon \mathrm{pos}(\xi) \to Q$; the *set of all runs of* $\mathcal{A}$ *on* ξ is denoted by $R_\mathcal{A}(\xi)$. For every $\kappa \in R_\mathcal{A}(\xi)$ and $p \in \mathrm{pos}(\xi)$, the *run induced by* κ *at position* p is the run $\kappa|_p \in R_\mathcal{A}(\xi|_p)$ and defined for every $p' \in \mathrm{pos}(\xi|_p)$ by $\kappa|_p(p') = \kappa(pp')$. For a tree $\xi \in T_\Sigma(X)$, we define the weight function of a run $\mathrm{wt} \colon R_\mathcal{A}(\xi) \to [0, 1]$ recursively as follows: If $x \in X$ and $\kappa \in R_\mathcal{A}(x)$, then $\mathrm{wt}(\kappa) = 1$. Let $k \in \mathbb{N}, \sigma \in \Sigma^{(k)}, \xi_1, \ldots, \xi_k \in T_\Sigma(X), \kappa \in R_\mathcal{A}(\sigma(\xi_1, \ldots, \xi_k))$. We define $\mathrm{wt}(\kappa) = \mathrm{wt}(\kappa|_1) \cdot \ldots \cdot \mathrm{wt}(\kappa|_k) \cdot \mu_\sigma((\kappa(1), \ldots, \kappa(k)), \kappa(\varepsilon))$. The *run semantics of* $\mathcal{A}$ is the tree series $r_\mathcal{A} \in S\langle\langle T_\Sigma(X)\rangle\rangle$ such that for every $\xi \in T_\Sigma(X)$

$$(r_\mathcal{A}, \xi) = \sum_{\kappa \in R_\mathcal{A}(\xi)} \mathrm{wt}(\kappa) \cdot \nu_{\kappa(\varepsilon)}.$$

We call a wta over the probabilistic semiring $S = ([0, 1], +, \cdot, 0, 1)$ a *probabilistic tree automaton* (pta) if the following two properties hold:

1. The root weight mapping ν forms a probability distribution over all states $q \in Q$, i.e., $\sum_{q \in Q} \nu_q = 1$ and for every $q \in Q$ the transition mappings form probability distributions over the input symbols $\sigma \in \Sigma$ and tuples of states $(q_1, \ldots, q_k) \in Q^k$, i.e., $\forall q \in Q : \sum_{\sigma \in \Sigma} \sum_{q_1, \ldots, q_k \in Q} \mu_\sigma((q_1, \ldots, q_k), q) = 1$ *(proper).*
2. The tree series $r_\mathcal{A} \in S\langle\langle T_\Sigma\rangle\rangle$ forms a probability distribution over all trees $\xi \in T_\Sigma$, i.e., $\sum_{\xi \in T_\Sigma} (r_\mathcal{A}, \xi) = 1$ *(consistent).*

Since all pta are defined over the probabilistic semiring, its declaration is omitted for a pta $\mathcal{A} = (Q, \Sigma, \mu, \nu)$. Additionally, for a pta $\mathcal{A}$ and tree $\xi \in T_\Sigma$ we denote the coefficient $(r_\mathcal{A}, \xi)$ as $\Pr_\mathcal{A}(\xi)$ and call it *probability of* ξ. Note that $\Pr_\mathcal{A}(\kappa, \xi) = \mathrm{wt}(\kappa) \cdot \nu_{\kappa(\varepsilon)}$ forms a probability distribution over $\xi \in T_\Sigma$ and $\kappa \in R_\mathcal{A}(\xi)$ and therefore $\sum_{\xi \in T_\Sigma} \sum_{\kappa \in R_\mathcal{A}(\xi)} \Pr(\kappa, \xi) = 1$.

Prefix probabilities Let $\mathcal{A} = (Q, \Sigma, \mu, \nu)$ be a probabilistic tree automaton, X a set of variables and $\zeta \in T_\Sigma(X) \setminus T_\Sigma$ a prefix tree. Due to properness and the definition of the weight function wt, we have

$$\Pr_{\mathcal{A}}(\zeta) = \sum_{\xi \in T_\Sigma|_\zeta} \Pr_{\mathcal{A}}(\xi).$$

Notation and nomenclature Whenever a pta is introduced, we only provide transitions and root weights that are non-zero. All other possible transitions or root weights for states have the value 0. Furthermore, states with non-zero root weight are often referred to as *final states*. For a pta $\mathcal{A}$ and tree $\xi \in T_\Sigma(X)$, if $\Pr_{\mathcal{A}}(\xi) \neq 0$, we say $\mathcal{A}$ *accepts* ξ or *recognises* ξ, interchangeably. In order to avoid having to mention the set of states Q for a pta $\mathcal{A}$ every time, we define the *size of* $\mathcal{A}$ as $|\mathcal{A}| = |Q|$.

Visual representation Whenever we visualise probabilistic tree automata, we depict them as functional hypergraphs: Each state is represented by a node (circle). Its identifier is provided within the circle. A doubly lined circle for a state $q \in Q$ indicates $\nu_q \neq 0$. Should q be the only final state, i.e., $\nu_q = 1$, further indication of root weights is omitted. Otherwise it is given in the description of the figure. A transition is depicted by a hyperedge (squares) with its symbol within the box and its probability next to it. The associated states can be inferred from the connected edges in a counterclockwise order. Should a transition be presented on its own, it is visualised as shown in Figure 3a. Similarly, runs are shown with the state for each position next to the position in tree itself (Figure 3b).

Example 2. Probabilistic tree automaton $\mathcal{A} = (Q, \Sigma, \mu, \nu)$ where

$$\begin{aligned}
Q &= \{ q_0, q_1, q_2 \}, \\
\Sigma &= \{ \sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}, \beta^{(0)} \}, \\
\mu_\sigma &= \{ ((q_1, q_2), q_0) \to 1.0,\ ((q_1, q_2), q_1) \to 0.1 \}, \\
\mu_\gamma &= \{ ((q_1), q_1) \to 0.5,\ ((q_2), q_1) \to 0.3 \}, \\
\mu_\alpha &= \{ ((\varepsilon), q_1) \to 0.1,\ ((\varepsilon), q_2) \to 0.5 \}, \\
\mu_\beta &= \{ ((\varepsilon), q_2) \to 0.5 \}, \\
\nu_{q_0} &= 0.9, \text{ and} \\
\nu_{q_1} &= 0.1.
\end{aligned}$$

A depiction of this pta is given in Figure 2.

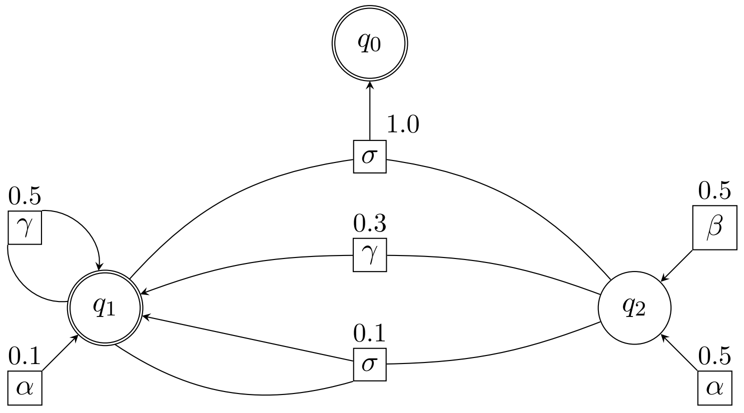


Figure 2.: A depiction of the probabilistic tree automaton as described in Example 2 with root weights $\nu_{q_0} = 0.9$ and $\nu_{q_1} = 0.1$.

We provide a short exemplary calculation for the weight of a run and the probability of a tree: Let $\xi = \sigma(\gamma(\alpha), \beta) \in T_\Sigma$ be a tree and $\kappa \in R_\mathcal{A}(\xi)$ a run of $\mathcal{A}$ on ξ where $\kappa(\varepsilon) = q_0$, $\kappa(1) = q_1$, $\kappa(2) = q_2$, and $\kappa(11) = q_2$. The weight of κ is

$$\begin{aligned}\mathrm{wt}(\kappa) &= \mathrm{wt}(\kappa|_1) \cdot \mathrm{wt}(\kappa|_2) \cdot \mu_\sigma((q_1, q_2), q_0)\\ &= \mathrm{wt}(\kappa|_{11}) \cdot \mu_\gamma((q_2), q_1) \cdot \mu_\beta((\varepsilon), q_2) \cdot 1.0\\ &= \mu_\alpha((\varepsilon), q_2) \cdot 0.3 \cdot 0.5 \cdot 1.0\\ &= 0.5 \cdot 0.3 \cdot 0.5 \cdot 1.0 = 0.075.\end{aligned}$$

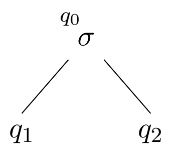
The probability of ξ is

$$\begin{aligned}\mathrm{Pr}_\mathcal{A}(\xi) &= \mathrm{wt}(\kappa) \cdot \nu_{q_0} + \sum_{\kappa' \in R_\mathcal{A}(\xi) \setminus \{\kappa\}} \mathrm{wt}(\kappa') \cdot \nu_{\kappa'(\varepsilon)}\\ &= 0.075 \cdot 0.9 + 0.0235 = 0.091.\end{aligned}$$

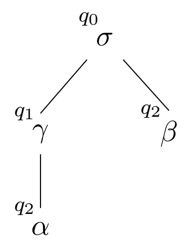
Run κ and a transition from μ_σ are visualised in Figure 3. $\triangle$

2.4. Complexity theory

We reduce the section regarding complexity theory to the presentation of relevant problems and requirements for the proof of NP-hardness. Each problem consists of an instance, describing the parameters of the problem, and a question, expressing the problem to solve. Presented here are two kinds of problems: Optimisation problems for which the best answer from all possible answers is sought and decision problems that have only one



(a) Transition $((q_1, q_2), q_0) \to 1.0 \in \mu_\sigma$.



(b) Run $\kappa \in R_{\mathcal{A}}(\sigma(\gamma(\alpha), \beta))$.

Figure 3.: Visualisation of a transition and a run for the pta from Example 2.

correct answer for the given input. We can transform optimisation problems into decision problems by repeatedly answering the decision problem while modifying its instance parameters. The eponymous problem of this work is the most probable tree problem (MPT) for pta:

Problem 1 (most probable tree problem (MPT))**.**

Instance A pta $\mathcal{A} = (Q, \Sigma, \mu, \nu)$.

Question What is the most probable tree ξ given $\mathcal{A}$, i.e., $\xi \in \arg\max_{\xi' \in T_\Sigma} \Pr_{\mathcal{A}}(\xi')$?

MPT is an optimisation problem. As such, we can transform it into a decision problem (MPT-decision). There, we do not directly seek the most probable tree but a tree whose probability exceeds a given threshold. Repeatedly increasing the threshold and checking if the answer to the decision problem is still yes, eventually provides the answer to the optimisation problem MPT.

Problem 2 (most probable tree decision problem (MPT-decision))**.**

Instance A pta $\mathcal{A} = (Q, \Sigma, \mu, \nu)$ and a real number $\lambda \in [0, 1]$.

Question Is there a tree $\xi \in T_\Sigma$, such that $\Pr_{\mathcal{A}}(\xi) \geq \lambda$?

Another problem we need is a version of the boolean satisfiability problem. We do not go into detail but regarding notation we use the following: For some $n \in \mathbb{N}^+$, a set of *variables* is given by a set $X = \{ x_1, \ldots, x_n \}$. The set of corresponding *literals* is defined by $\mathrm{Lit}(X) = X \cup \{ \bar{x} \mid x \in X \}$. As a representation for the truth values *true* and *false* we often use the symbols $\top$ and $\bot$, respectively. We make use of the decision problem 3-SAT. For a $k \in \mathbb{N}^+$, a 3-SAT formula is a formula $f = \bigwedge_{i \in [k]} C_i$ with clauses $C_i = (u_1^i \vee u_2^i \vee u_3^i)$ and literals $u_j^i \in \mathrm{Lit}(X)$.

Problem 3 (3-SAT)**.**

Instance A boolean formula f in conjunctive normal form with at most 3 literals per clause over variables $x_1, \dots, x_n$.

Question Is the formula f satisfiable, i.e., is there an assignment of $\top$ or $\bot$ to each variable such that f evaluates to $\top$?

Of interest here, is the class of NP-hard problems. Problems in this class are at least as hard as any problem that is in NP. That is, these problems are not necessarily in NP but are as least as hard as any problem that can by solved by a nondeterministic turing machine in polynomial-time.

Polynomial-time reduction In order to show that a decision problem D is NP-hard it suffices to proof that it is at least as hard is an NP-complete decision problem, that is, a problem that is both in NP and NP-hard. 3-SAT, for example, is NP-complete. Showing this can be done by a *polynomial-time reduction* from an NP-complete problem Q to D which is denoted by $Q \preceq_p D$. For that, we provide a method to transform an arbitrary instance of Q to an instance of D, show that the transformation can be done in deterministic polynomial-time and prove that the reduction is answer-preserving. The latter meaning that for every instance of NP-complete problem Q we have: The instance's answer is yes if and only if the answer to the transformed instance (of problem D) is yes.

3. Tree size properties

For probabilistic finite automata, de la Higuera and Oncina provided two results on the length of recognisable words. We generalise these results to probabilistic tree automata. First, in Section 3.1, we give an upper bound on the height of recognisable trees. Subsequently, we show the construction of a pta for which the most probable tree is of superpolynomial size in the size of the automaton (Section 3.2).

3.1. Probable tress are shallow

This first theorem is a generalisation of de la Higuera and Oncina 2013, Proposition 1. With that, we show a relation between the structure of a tree, in particular its height, and its probability. Their proposition states that probable strings are short. We adapt their proof and essentially show that probable trees are shallow.

Theorem 1. Let $\mathcal{A}$ be a pta and $\xi \in T_\Sigma$ a tree with $\Pr_\mathcal{A}(\xi) > 0$. Then $\operatorname{height}(\xi) \leq \frac{|\mathcal{A}|^2}{\Pr_\mathcal{A}(\xi)}$.

The theorem follows from the subsequent lemma. But instead of having a fixed tree, we show a bound on a tree's height given a lower bound of its probability.

Lemma 1. Let $\mathcal{A}$ be a pta and $p \in \mathbb{R}^+$ a real number. If there is a tree with probability at least p, then its height is at most $b = \frac{|\mathcal{A}|^2}{p}$.

Proof. Let ξ be a tree of height h, with probability $\Pr_\mathcal{A}(\xi) = p$, and $n = |\mathcal{A}|$. Without loss of generality let $Q = \{ q_1, \dots, q_n \}$ be the set of states for $\mathcal{A}$. We fix a path $\pi = p_1 p_2 \dots p_h \in \operatorname{pos}(\xi)^h$ of length h. In case there are multiple such paths in ξ, we arbitrarily choose the leftmost. For each $i \in [n]$, let $R^i_\mathcal{A}(\xi) = \{ \kappa \in R_\mathcal{A}(\xi) \mid i = \min(\arg\max_{k \in [n]} |\{ p \in \pi \mid \kappa(p) = q_k \}|) \}$ be the set of runs for which $q_i \in Q$ is the most used state on path π. Runs that have no unique most used state on π are arbitrarily assigned to the set with the smallest index i. Then $R^1_\mathcal{A}, \dots, R^n_\mathcal{A}$ are pairwise disjoint and $\bigcup_{i \in [n]} R^i_\mathcal{A}(\xi) = R_\mathcal{A}(\xi)$ holds.

There is at least one index $j \in [n]$ such that $\sum_{\kappa \in R^j_{\mathcal{A}}(\xi)} \Pr_{\mathcal{A}}(\kappa, \xi) \geq \frac{p}{n}$. This can be easily shown for $j \in \arg\max_{i \in [n]} \sum_{\kappa \in R^i_{\mathcal{A}}(\xi)} \Pr_{\mathcal{A}}(\kappa, \xi)$:

$$\begin{aligned}
\Pr_{\mathcal{A}}(\xi) = \quad & \sum_{\kappa \in R_{\mathcal{A}}(\xi)} \Pr_{\mathcal{A}}(\kappa, \xi) \geq p \\
\Leftrightarrow \sum_{i=1}^{n} & \sum_{\kappa \in R^i_{\mathcal{A}}(\xi)} \Pr_{\mathcal{A}}(\kappa, \xi) \geq p \\
\Rightarrow \sum_{i=1}^{n} & \sum_{\kappa \in R^j_{\mathcal{A}}(\xi)} \Pr_{\mathcal{A}}(\kappa, \xi) \geq p \\
\Leftrightarrow \quad & \sum_{\kappa \in R^j_{\mathcal{A}}(\xi)} \Pr_{\mathcal{A}}(\kappa, \xi) \geq \frac{p}{n}.
\end{aligned} \tag{1}$$

Let $\kappa \in R^j_{\mathcal{A}}(\xi)$ and $k \in [h]$ the smallest index such that $\kappa(p_k) = q_j$. For each of the other indices $k' \in \{ k+1, \dots, h \}$ with $\kappa(p_{k'}) = q_j$ we define a run $\kappa_{k'}$ of $\mathcal{A}$ on the tree $\xi[\xi_{p_{k'}}]|_{p_k}$ where for each position $p \in \mathrm{pos}(\xi[\xi_{p_{k'}}]|_{p_k})$:

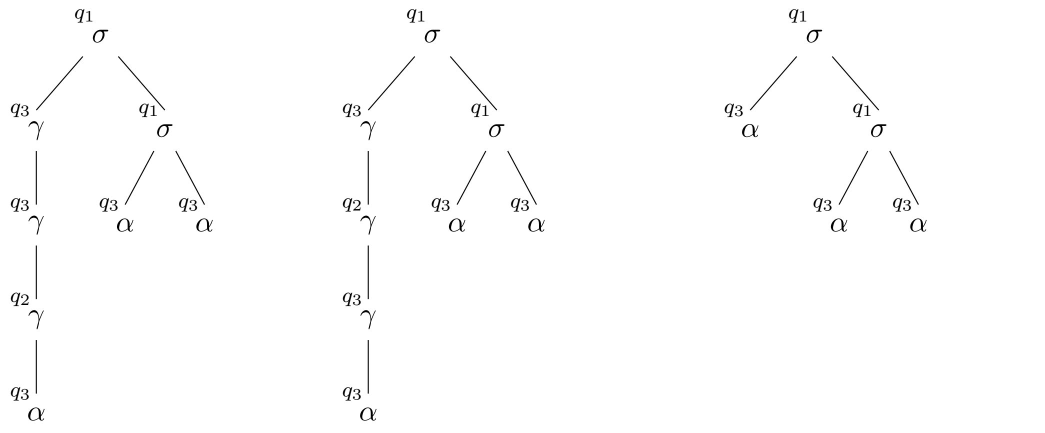
$$\kappa_{k'}(p) = \begin{cases} \kappa|_{p_{k'}}(c) & \text{if } p = p_k c \\ \kappa(p) & \text{otherwise.} \end{cases}$$

We denote this set of new (run, tree)-pairs as $\mathrm{Alt}(\kappa, j)$. Each element $(\kappa', \xi') \in \mathrm{Alt}(\kappa, j)$ is at least as likely as (κ, ξ) because only a subset of transitions is applied. Furthermore, since with κ the state q_j appears at least $\frac{h}{n}$ times on π, i.e., $|\{ p \in \pi \mid \kappa(p) = q_j \}| \geq \frac{h}{n}$, we have that

$$|\mathrm{Alt}(\kappa, j)| \geq \frac{h}{n} - 1. \tag{2}$$

Next up, we will form a sum over all alternative (run, tree)-pairs for runs in $R^j_{\mathcal{A}}(\xi)$. Unfortunately, there may be cases where different runs yield the same pair when modified (Example 3). In order to avoid adding an alternative pair multiple times, we only consider unique pairs. Previously shown bounds are still applicable due to the following lemma (proven later) which shows that an alternative pair has more probability mass than that of all the (run, tree)-pairs it originated from combined.

Lemma 2. Let $\kappa_1, \dots, \kappa_k$ be different runs in $R^j_{\mathcal{A}}(\xi)$, and (κ', ξ') be a pair belonging to $\bigcap_{i=1}^{k} \mathrm{Alt}(\kappa_i, j)$. Then $\Pr_{\mathcal{A}}(\kappa', \xi') \geq \sum_{i=1}^{k} \Pr_{\mathcal{A}}(\kappa_i, \xi)$.



(a) $\kappa_1 \in R^3_{\mathcal{A}}(\xi)$. (b) $\kappa_2 \in R^3_{\mathcal{A}}(\xi)$. (c) $(\kappa', \xi') \in \mathrm{Alt}(\kappa_1, 3) \cap \mathrm{Alt}(\kappa_2, 3)$.

Figure 4.: Two different runs with the same alternative (run, tree)-pair (Example 2).

Example 3. Let $\mathcal{A} = (\{ q_1, q_2, q_3 \}, \{ \sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)} \}, \mu, \nu)$ be a probabilistic tree automaton and $\xi = \sigma(\gamma(\gamma(\gamma(\alpha))), \sigma(\alpha, \alpha)) \in T_\Sigma$ a tree. Given runs $\kappa_1, \kappa_2 \in R^3_{\mathcal{A}}(\xi)$, Figure 4 shows that both runs share an alternative (run, tree)-pair $(\kappa', \xi') \in \mathrm{Alt}(\kappa_1, 3) \cap \mathrm{Alt}(\kappa_2, 3)$ when cut from position $p_k = 1$ to position $p_{k'} = 1111$. $\triangle$

Let $U = \bigcup_{\kappa \in R^j_{\mathcal{A}}(\xi)} \mathrm{Alt}(\kappa, j)$ be the set of all unique alternative (run, tree)-pairs for runs in $R^j_{\mathcal{A}}(\xi)$. Summing over the probability of all pairs in U yields:

$$\begin{aligned}
\sum_{(\kappa',\xi')\in U} \mathrm{Pr}_{\mathcal{A}}(\kappa', \xi') &\geq \sum_{(\kappa',\xi')\in U} \sum_{\substack{\kappa \in R^j_{\mathcal{A}}(\xi): \\ (\kappa',\xi')\in \mathrm{Alt}(\kappa,j)}} \mathrm{Pr}_{\mathcal{A}}(\kappa, \xi) && \text{(Lemma 2)} \\
&= \sum_{\kappa \in R^j_{\mathcal{A}}(\xi)} \sum_{(\kappa',\xi')\in \mathrm{Alt}(\kappa,j)} \mathrm{Pr}_{\mathcal{A}}(\kappa, \xi) \\
&\geq \sum_{\kappa \in R^j_{\mathcal{A}}(\xi)} \left(\frac{h}{n} - 1 \right) \cdot \mathrm{Pr}_{\mathcal{A}}(\kappa, \xi) && \text{(Equation 2)} \\
&\geq \left(\frac{h}{n} - 1 \right) \cdot \frac{p}{n}. && \text{(Equation 1)}
\end{aligned}$$

Each pair $(\kappa', \xi') \in U$ represents a run κ' of the pta $\mathcal{A}$ on ξ'. Note that all of these are distinct and none represent a run on the original tree ξ. Since the tree series $r_{\mathcal{A}} \in S\langle\langle T_\Sigma \rangle\rangle$ is consistent $\left(\sum_{\xi \in T_\Sigma} \mathrm{Pr}_{\mathcal{A}}(\xi) = 1 \right)$, we can safely add the probability $\mathrm{Pr}_{\mathcal{A}}(\xi) = p$ to $\mathrm{Pr}_{\mathcal{A}}(U)$ without exceeding 1:

$$\begin{aligned}
& & \sum_{(\kappa',\xi')\in U} \Pr_{\mathcal{A}}(\kappa',\xi') + \Pr_{\mathcal{A}}(\xi) &\leq 1 \\
&\Rightarrow & \left(\frac{h}{n}-1\right)\cdot\frac{p}{n} + p &\leq 1 \\
&\Leftrightarrow & (h-n)\cdot p + pn^2 &\leq n^2 \\
&\Leftrightarrow & h-n &\leq \frac{n^2\cdot(1-p)}{p} \\
&\Leftrightarrow & h &\leq \frac{n^2\cdot(1-p)}{p} + n \\
& & &\leq \frac{n^2}{p}.
\end{aligned}$$

□

It remains to provide the proof for Lemma 2. As this requires another helping statement, we shall give that first. The following lemma states that the weight of all runs with the same root state, does not exceed one.

Lemma 3. Let $\mathcal{A} = (Q, \Sigma, \mu, \nu)$ be a pta and $X = \{ x_1, x_2, \dots \}$ a set of variables. Then, for every tree $\xi \in T_\Sigma(X)$ and $q \in Q$ the sum over all possible runs on ξ with root state q is less than one:

$$\sum_{\substack{\kappa\in R_{\mathcal{A}}(\xi):\\ \kappa(\varepsilon)=q}} \mathrm{wt}(\kappa) \leq 1.$$

Proof. We prove the statement by structural induction on ξ.
Induction hypothesis: Let $\xi \in T_\Sigma(X)$. Then it holds that

$$\sum_{\substack{\kappa\in R_{\mathcal{A}}(\xi):\\ \kappa(\varepsilon)=q}} \mathrm{wt}(\kappa) \leq 1. \tag{IH}$$

Induction base: For every $x \in X, q \in Q$:

$$\sum_{\substack{\kappa\in R_{\mathcal{A}}(x):\\ \kappa(\varepsilon)=q}} \mathrm{wt}(\kappa) = 1. \qquad \text{(Definition of wt}(x)\text{ for } x \in X)$$

Induction step: Let $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, $\xi_1, \ldots, \xi_k \in T_\Sigma(X)$ and $\xi = \sigma(\xi_1, \ldots, \xi_k) \in T_\Sigma(X)$. Then,

$$\begin{aligned}
\sum_{\substack{\kappa \in R_\mathcal{A}(\xi):\\ \kappa(\varepsilon)=q}} \mathrm{wt}(\kappa) &= \sum_{q_1,\ldots,q_k \in Q} \mu_\sigma((q_1,\ldots,q_k),q) \cdot \sum_{\substack{(\kappa_1,\ldots,\kappa_k)\in\\ R_\mathcal{A}(\xi_1)\times\cdots\times R_\mathcal{A}(\xi_k):\\ \forall i\in[k]:\kappa_i(\varepsilon)=q_i}} \prod_{i=1}^{k} \mathrm{wt}(\kappa_i)\\
&= \sum_{q_1,\ldots,q_k \in Q} \mu_\sigma((q_1,\ldots,q_k),q) \cdot \sum_{\substack{\kappa_1\in R_\mathcal{A}(\xi_1):\\ \kappa_1(\varepsilon)=q_1}} \cdots \sum_{\substack{\kappa_k\in R_\mathcal{A}(\xi_k):\\ \kappa_k(\varepsilon)=q_k}} \mathrm{wt}(\kappa_1)\cdot\ldots\cdot\mathrm{wt}(\kappa_k)\\
&= \sum_{q_1,\ldots,q_k \in Q} \mu_\sigma((q_1,\ldots,q_k),q) \cdot \left(\sum_{\substack{\kappa_1\in R_\mathcal{A}(\xi_1):\\ \kappa_1(\varepsilon)=q_1}} \mathrm{wt}(\kappa_1)\right)\cdot\ldots\cdot\left(\sum_{\substack{\kappa_k\in R_\mathcal{A}(\xi_k):\\ \kappa_k(\varepsilon)=q_k}} \mathrm{wt}(\kappa_k)\right)\\
&= \sum_{q_1,\ldots,q_k \in Q} \mu_\sigma((q_1,\ldots,q_k),q) \cdot \prod_{i=1}^{k} \sum_{\substack{\kappa_i\in R_\mathcal{A}(\xi_i):\\ \kappa_i(\varepsilon)=q_i}} \mathrm{wt}(\kappa_i)\\
&\le \sum_{q_1,\ldots,q_k \in Q} \mu_\sigma((q_1,\ldots,q_k),q) \cdot \prod_{i=1}^{k} 1 && \text{(IH)}\\
&\le 1. && (\mu \text{ proper})
\end{aligned}$$

□

Proof of Lemma 2. Let $\widetilde{\kappa} \in R^j_\mathcal{A}(\xi)$ be a run. Note that each pair $(\widetilde{\kappa}', \widetilde{\xi}') \in \mathrm{Alt}(\widetilde{\kappa}, j)$ has $\widetilde{\kappa}'(\varepsilon) = \widetilde{\kappa}(\varepsilon)$ because building an alternative (run, tree)-pair never changes the state at the root position ε. Since all runs $\kappa_1, \ldots, \kappa_k$ share an alternative (run, tree)-pair $(\kappa', \xi') \in \bigcap_{i=1}^k \mathrm{Alt}(\kappa_i, j)$, we can infer that for all $i \in [k] : \kappa_i(\varepsilon) = \kappa'(\varepsilon)$.

For an alternative run (where q_j is the most used state), a run is cut from the first position with state q_j to a subsequent occurrence of q_j. Therefore, the first time q_j is seen on the specified path in a run corresponds to the first occurrence of the state in one of its alternative runs. Let p be the first position in κ' such that $\kappa'(p) = q_j$. In consequence, for all $i \in [k]$, we have $\kappa_i(p) = q_j$. As $(\kappa', \xi') \in \bigcap_{i=1}^k \mathrm{Alt}(\kappa_i, j)$, all κ_i are cut starting from position p and, in order to yield the same alternative (run, tree)-pair, are cut up to the same position p'.

Hence, the parts unique to each κ_i can be defined as runs $\kappa_i^{p'}|_p$ $(\forall p'' \in \mathrm{pos}(\xi[x]_{p'}) : \kappa_i^{p'}(p'') = \kappa_i(p''))$ on the same tree $\xi[x]_{p'}|_p \in T_\Sigma(\{\, x \,\})$ and their weight $\mathrm{wt}(\kappa_i^{p'}|_p)$ is composed of those transition weights that are part of $\mathrm{wt}(\kappa_i)$ but not $\mathrm{wt}(\kappa')$. In consequence, the probability of κ_i is separable as follows:

$$\begin{aligned}\Pr_{\mathcal{A}}(\kappa_i, \xi) &= \mathrm{wt}(\kappa_i) \cdot \nu_{\kappa_i(\varepsilon)} \\ &= \mathrm{wt}(\kappa') \cdot \frac{\mathrm{wt}(\kappa_i)}{\mathrm{wt}(\kappa')} \cdot \nu_{\kappa'(\varepsilon)} \\ &= \mathrm{wt}(\kappa') \cdot \mathrm{wt}(\kappa_i^{p'}|_p) \cdot \nu_{\kappa'(\varepsilon)}. \end{aligned} \tag{3}$$

The runs $\kappa_i^{p'}|_p$ are all different runs for the same tree and have the same state at the root position. Due to Lemma 3 we conclude that $\sum_{i=1}^{k} \mathrm{wt}(\kappa_i^{p'}|_p) \leq 1$ and can deduce a bound on $\Pr_{\mathcal{A}}(\kappa', \xi')$:

$$\begin{aligned}\Pr_{\mathcal{A}}(\kappa', \xi') &= \mathrm{wt}(\kappa') \cdot \nu_{\kappa'(\varepsilon)} \\ &\geq \mathrm{wt}(\kappa') \cdot \nu_{\kappa'(\varepsilon)} \cdot \sum_{\substack{\kappa \in R_{\mathcal{A}}(\xi[x]_{p'}|_p): \\ \kappa(\varepsilon) = \kappa'(\varepsilon)}} \mathrm{wt}(\kappa) && \text{(Lemma 3)} \\ &\geq \mathrm{wt}(\kappa') \cdot \nu_{\kappa'(\varepsilon)} \cdot \sum_{i=1}^{k} \mathrm{wt}(\kappa_i^{p'}|_p) \\ &= \mathrm{wt}(\kappa') \cdot \nu_{\kappa'(\varepsilon)} \cdot \sum_{i=1}^{k} \frac{\mathrm{wt}(\kappa_i)}{\mathrm{wt}(\kappa')} && \text{(Equation 3)} \\ &= \sum_{i=1}^{k} \mathrm{wt}(\kappa_i) \cdot \nu_{\kappa_i(\varepsilon)} \\ &= \sum_{i=1}^{k} \Pr_{\mathcal{A}}(\kappa_i, \xi). \end{aligned}$$

□

Even though the proofs are quite similar, in some places non-trivial adjustments had to be made as the original proof is somewhat awry. Nevertheless, we have shown that the original proposition for pfa is generalisable to pta. A stronger bound, that depends on the tree's size instead of its height, would have been preferable. Especially, since the bound proves to be useful in the time complexity analysis of Section 5.3. The current proof technique of cutting trees on a path to get a set whose probability we know, seems to be inadequate for that task, though.

3.2. A most probable tree of superpolynomial size

The most probable tree of a probabilistic tree automaton does not necessarily have to be small and can in fact be quite large. We provide a pta for which the size of the most probable tree is more than polynomial in the size of the automaton. The construction is based on a probabilistic finite automaton which is presented by de la Higuera 1997 and, more recently, by de la Higuera and Oncina 2014.

For the proof, we begin by building probabilistic tree automata that recognise monadic trees over ranked alphabet $\{\, \alpha^{(0)}, \gamma^{(1)} \,\}$. More specifically, these pta $\mathcal{A}_i$ only give non-zero probabilities to trees whose size is one plus a multiple of $\psi_i \in \mathbb{N}^+$. Afterwards, we combine such automata to a new pta. That pta assigns the highest probabilities to trees whose sizes (minus one) is a multiple of most ψ_i.

Theorem 2. There is a pta $\mathcal{A}$ for which the size of the most probable tree is greater than polynomial in the size of $\mathcal{A}$.

Proof. Let $\theta \in \mathbb{R}^+$ be a positive real number, $\Sigma = \{\, \alpha^{(0)}, \gamma^{(1)} \,\}$ a ranked alphabet and $\Psi = \{\, \psi_1, \dots, \psi_z \,\}$ a set of prime numbers. For each $i \in [z]$ we construct a probabilistic tree automaton $\mathcal{A}_i = (Q^i, \Sigma, \mu^i, \nu^i)$ where

$$\begin{aligned}
Q^i &= \{\, i_j \mid j \in [\psi_i] \,\}, \\
\mu^i_\gamma &= \{\, ((i_1), i_{\psi_i}) \to 1 - \theta \,\}, \\
&\quad \cup \{\, ((i_{u+1}), i_u) \to 1 \mid \forall u \in [\psi_i - 1] \,\}, \\
\mu^i_\alpha &= \{\, ((\varepsilon), i_{\psi_i}) \to \theta \,\}, \text{ and} \\
\nu^i_{i_{\psi_i}} &= 1.
\end{aligned}$$

The pta $\mathcal{A}_i$ for $\psi_i = 5$ is depicted in Figure 5.

Note, that for all trees $\xi \in T_\Sigma \setminus \{\, \gamma^{k\psi_i}(\alpha) \mid k \in \mathbb{N} \,\}$ a pta $\mathcal{A}_i$ assigns a probability of zero, i.e., $\Pr_{\mathcal{A}_i}(\xi) = 0$. On the other hand, for $k \in \mathbb{N}$, trees that have exactly k times ψ_i the symbol γ have a probability of $\Pr_{\mathcal{A}_i}(\gamma^{k\psi_i}(\alpha)) = \theta(1-\theta)^k$.

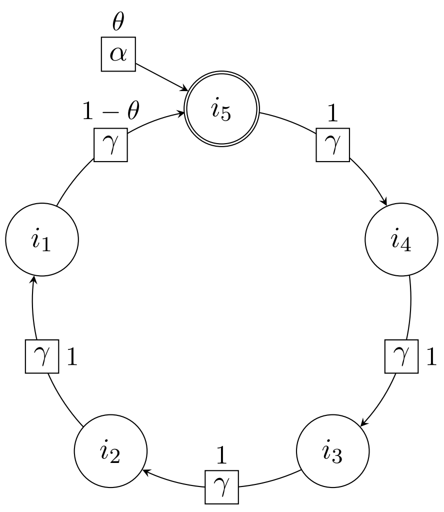


Figure 5.: The pta for $\psi_i = 5$ with the only non-zero root weight $\nu^i_{i_5} = 1$.

Next, we combine the probabilistic tree automata with a new final state. This results in the pta $\mathcal{A} = (Q, \Sigma, \mu, \nu)$ where

$$\begin{aligned}
Q &= \bigcup_{i\in[z]} Q^i \cup \{ f \}, \\
\mu_\gamma &= \bigcup_{i\in[z]} \mu^i_\gamma \cup \left\{ ((i_1), f) \to \frac{1}{z} \mid \forall i \in [z] \right\}, \\
\mu_\alpha &= \bigcup_{i\in[z]} \mu^i_\alpha, \text{ and} \\
\nu_f &= 1.
\end{aligned}$$

The construction for a set of prime numbers $\{ 2, 3, 5, \ldots, \psi_z \}$ is illustrated in Figure 6.

In the following we consider a specific $k = \prod_{i\in[z]} \psi_i$. The probability of a tree in T_Σ with a size of $k + 1$ can easily be calculated by examining, how much each part (pta for a prime ψ_i) contributes:

$$\Pr_{\mathcal{A}}(\gamma^k(\alpha)) = \sum_{i\in[z]} \frac{1}{z} \cdot \theta(1-\theta)^{\frac{k}{\psi_i}-1} = \frac{\theta}{z} \cdot \sum_{i\in[z]} (1-\theta)^{\frac{k}{\psi_i}-1}.$$

Since k is the least common multiple of all primes ψ_i, any tree of size less or equal than k is not accepted by at least one of the pta. Therefore, for a $k' < k$:

$$\Pr_{\mathcal{A}}(\gamma^{k'}(\alpha)) \leq \frac{\theta}{z} \cdot \sum_{i\in[z-1]} (1-\theta)^{\frac{k'}{\psi_i}-1} < \frac{\theta}{z} \cdot \sum_{i\in[z-1]} 1 = \theta \cdot \frac{z-1}{z}.$$

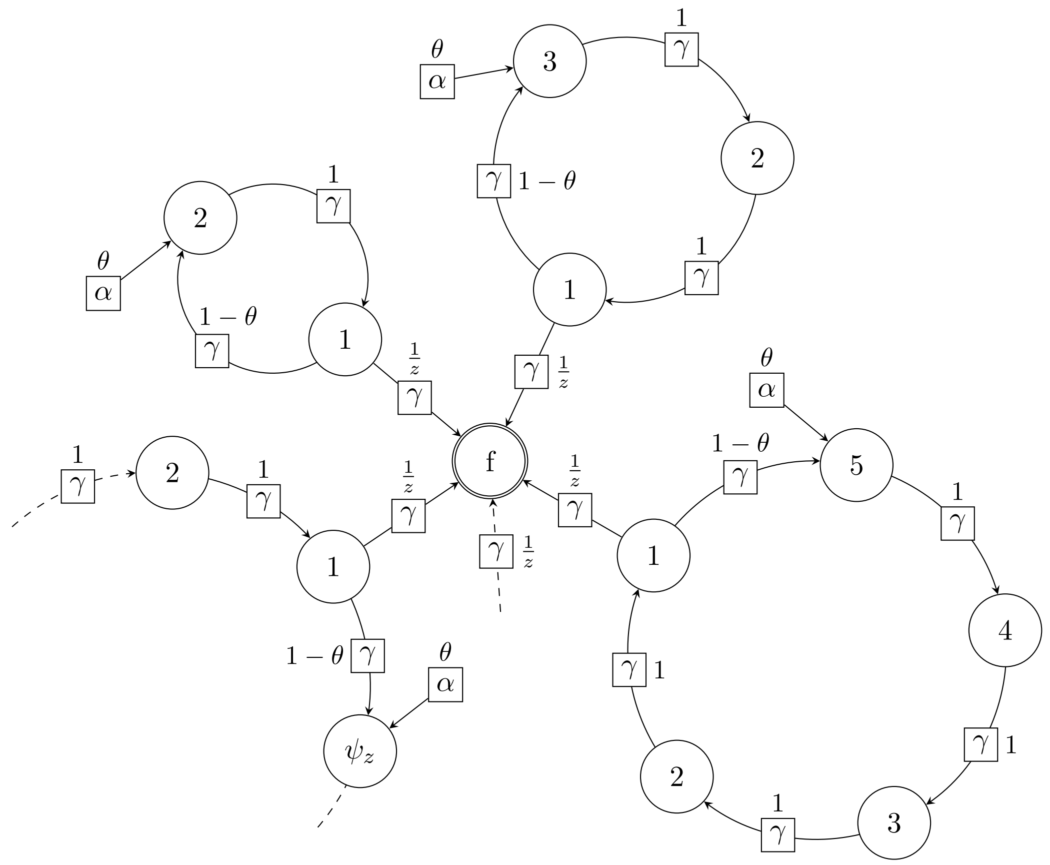


Figure 6.: The pta for a set of prime numbers $\{ 2, 3, 5, \ldots, \psi_z \}$. The only non-zero root weight is $\nu_f = 1$. Note that the identifiers for the states are simplified here. For example: Instead of state i_3 (for a prime ψ_i), we just write 3.

For a good value of θ we can show $\Pr_{\mathcal{A}}(\gamma^k(\alpha)) > \theta \cdot \frac{z-1}{z}$. To this end we first show

$$\begin{aligned}
\Pr_{\mathcal{A}}(\gamma^k(\alpha)) &= \frac{\theta}{z} \cdot \sum_{i \in [z]} (1-\theta)^{\frac{k}{\psi_i}-1} \\
&\geq \frac{\theta}{z} \cdot \sum_{i \in [z]} (1-\theta)^{k} && (k \geq \tfrac{k}{\psi_i} - 1) \\
&= \theta \cdot (1-\theta)^k
\end{aligned}$$

and subsequently determine the required value of θ:

$$\begin{aligned}
& \theta \cdot (1-\theta)^k > \theta \cdot \frac{z-1}{z} \\
\Leftrightarrow \quad & (1-\theta)^k > \frac{z-1}{z} \\
\Leftrightarrow \quad & (1-\theta) > \sqrt[k]{\frac{z-1}{z}} \\
\Leftrightarrow \quad & \theta < 1 - \sqrt[k]{\frac{z-1}{z}}.
\end{aligned}$$

Accordingly, for $\theta < 1 - \sqrt[k]{\frac{z-1}{z}}$ and $k' < k$ we have $\Pr_{\mathcal{A}}(\gamma^k(\alpha)) > \Pr_{\mathcal{A}}(\gamma^{k'}(\alpha))$, that is, no tree of size less than $k+1$ can have a higher probability than the tree $\gamma^k(\alpha)$. Consequently, the most probable tree is of size greater than or equal $k+1$. Now it remains to show that the size $k+1 = \prod_{i \in z} \psi_i + 1$ of such a probable tree increases faster than polynomial in the automaton's size $|\mathcal{A}| = \sum_{i \in z} \psi_i + 1$.

We apply the idea of de la Higuera 1997 and construct a specific set of primes. First, note that for all $n \in \mathbb{N}^+$ there exists a number $\psi \in [n, n + n^{\frac{11}{20}}]$ that is prime (Heath-Brown and Iwaniec 1979). In consequence, there is a prime ψ_i in every interval $[2^i, 2^{i+1})$. For a $j \in \mathbb{N}^+$ we construct the set $\Psi^j = \{\, \psi_i \mid i \in [j] \,\}$ that contains j primes each of which is part of a different interval $[2^i, 2^{i+1})$. With such a more specific set of primes, we can infer an upper bound on the size $|\mathcal{A}|$ of the constructed automaton and a lower bound on the size of the tree $\gamma^k(\alpha)$:

$$\begin{aligned}
|\mathcal{A}| &= 1 + \sum_{\psi \in \Psi^j} \psi < 1 + \sum_{i \in [j]} 2^{i+1} < 2^{j+2} \\
\mathrm{size}(\gamma^k(\alpha)) &= 1 + \prod_{\psi \in \Psi^j} \psi \geq 1 + \prod_{i \in [j]} 2^i = 1 + 2^{\sum_{i \in [j]} i} > 2^{\frac{j^2}{2}}.
\end{aligned}$$

Finally, for any fixed $n \in \mathbb{N}$ there exists a $j_0 \in \mathbb{N}^+$ such that for all $j > j_0$ it holds that

$$|\mathcal{A}|^n < \left(2^{j+2}\right)^n < 2^{\frac{j^2}{2}} < \text{size}(\gamma^k(\alpha)).$$

Let $j_0 > 6n$. Then, for any $j > j_0$, we have

$$\begin{aligned}
2^{\frac{j^2}{2}} &> 2^{\frac{j}{2}\cdot j_0} && (j > j_0)\\
&> 2^{\frac{j}{2}\cdot 6n} && (j_0 > 6n)\\
&= 2^{n\cdot 3j} = \left(2^{3j}\right)^n\\
&\geq \left(2^{j+2}\right)^n && (j > j_0 > 0)
\end{aligned}$$

and in conclusion, the size of the tree $\gamma^k(\alpha)$ grows faster than any polynomial of the number of states $|\mathcal{A}|$. $\square$

Originally, de la Higuera and Oncina 2014 claimed their string to be exponential in the size of the constructed pfa. We think that with the proof at hand one can only show superpolynomial size. However, this suffices to indicate the complexity class of the mpt problem: Without a special encoding, it would take more than polynomial-time to read or output the most probable tree. Therefore, the mpt problem cannot be in the class of problems that are computable in nondeterministic polynomial time on a nondeterministic turing machine (NP).

4. NP-hardness of the most probable tree problem

In this chapter we provide a short proof for the NP-hardness of the most probable tree problem for probabilistic tree automata. Theoretically, there are a couple of ways to prove MPT to be NP-hard: We could relate the most probable parse problem of stochastic tree substitution grammars, which has been proved NP-hard (Sima'an 2002), to pta. And alternatively, since the most probable string problem (MPS) for probabilistic finite automaton is already shown to be NP-hard by Casacuberta and de la Higuera 2000, we could argue that pta can simulate pfa and therefore that MPT is at least as difficult to solve as MPS.

Here, we provide a direct reduction from the NP-complete problem 3-SAT to MPT-decision. The reduction, as it is presented in Section 4.2, is based on Sima'an 2002 (Section 5.2). It represents the proof to the following theorem:

Theorem 3. The most probable tree problem (MPT) for pta is NP-hard.

Proof. The proof is given by a polynomial-time reduction from the NP-complete decision problem 3-SAT to MPT-decision, i.e., 3-SAT $\preceq_p$ MPT-decision. The reduction and an accompanying example are provided in Section 4.2. □

4.1. Further notation

Preliminary for the subsequent section we need to modify the pta definition. We allow a pta to have transitions that may accept greater parts of a tree with their state behaviour instead of just a single symbol:

Definition 6. A probabilistic tree automaton $\mathcal{A} = (Q, \Sigma, \mu, \nu)$ is in *extended form* if the family of transition mappings $\mu = (\mu_\sigma \mid \sigma \in \Sigma)$ contains for $\sigma \in \Sigma^{(k)}$ transition mappings

$$\mu_\sigma \colon T_\Sigma(Q)^k \times Q \to [0, 1].$$

This extended form is for convenience. Otherwise, in order to recognise more than one symbol at once with the same state behaviour, we would be required to introduce a number of new states. Such states would only be used to remember a certain condition while the bigger, desired tree is recognised.

Basically, such pta in extended form are the equivalent to probabilistic regular tree grammars (prtg) such as pta are the equivalent to prtg in normal form (cf. Engelfriet 2015, Definition 3.21). Even though prtg and pta can be used interchangeably (cf. Engelfriet 2015, Theorem 3.25) we want to avoid the additional overhead of introducing prtg.

In the following section a pta in extended form is constructed but we will often simply refer to it as pta. This does not harm the result of the reduction because in a similar manner as one can show that for each regular tree grammar there exists a regular tree grammar in normal form (Engelfriet 2015, Theorem 3.22), it is possible to show the following:

Corollary 1. Each probabilistic tree automaton in extended form has an equivalent probabilistic tree automaton.

Proof sketch. Given pta $\mathcal{A} = (Q, \Sigma, \mu, \nu)$ in extended form. Assume, for a $\sigma \in \Sigma$, the mapping μ_σ contains a transition $((\xi_1, \ldots, \xi_i, \ldots, \xi_k), q) \to p$ where $i \in [k]$ and $\xi_i \notin Q$. Construct a new pta $\mathcal{B} = (Q \cup \{ q' \}, \Sigma, \mu', \nu \cup \{ q' \to 0 \})$ where $q' \notin Q$ and μ' is obtained from μ by replacing the transition $((\xi_1, \ldots, \xi_i, \ldots, \xi_k), q) \to p$ in μ_σ by the two following transitions: $((\xi_1, \ldots, q', \ldots, \xi_k), q) \to p$ in μ_σ and $((\xi_i|_1, \ldots, \xi_i|_{k'}), q') \to 1$ in $\mu_{\xi_i(\varepsilon)}$ where $\mathrm{rk}(\xi_i(\varepsilon)) = k'$. The associated tree series is preserved, i.e., $r_\mathcal{A} = r_\mathcal{B}$. This process is repeated finitely often until the resulting pta is not in extended form. $\square$

4.2. Reduction

In this section we provide a polynomial-time reduction from the NP-complete problem 3-SAT to the most probable tree decision problem. Thereby, we provide the proof for Theorem 3. After stating the general approach for this reduction, we show how to construct a pta in extended form for an arbitrary 3-SAT instance. The specific probabilities for the pta and the resulting threshold for the decision problem are derived subsequently. Finally, we briefly demonstrate that the construction is indeed achievable in polynomial-time and answer preserving.

Intuition For a 3-SAT formula to be satisfiable, two conditions have to be met: First, each clause has at least one literal that is evaluated to true and thus, the whole clause evaluates to true. Secondly, each variable is consistently assigned the same value for each clause. The pta we construct in the following accepts trees that represent the 3-SAT instance and truth assignments to each literal. There are two kinds of runs for each such tree: Runs that ensure a consistent assignment and runs that guarantee each clause to be satisfied.

Construction Let $X = \{ X_1, \ldots, X_n \}$ be a set of variables and $k \in \mathbb{N}^+$. For $i \in [k], j \in [3]$, let $u^i_j \in \mathrm{Lit}(X)$ be literals and $c_i = (u^i_1 \vee u^i_2 \vee u^i_3)$ clauses. Given 3-SAT formula $f = \bigwedge_{i\in[k]} c_i$, let for each $j \in [n]$ the number of occurrences of literals x_j or $\bar{x}_j$ in f be denoted by n_j. Furthermore, let $p_0 = 1 - 2\sum_{j\in[n]} \theta \cdot \left(\frac{1}{2}\right)^{n_j}$ and for $j \in [n]$ let $p_j = \theta \cdot \left(\frac{1}{2}\right)^{n_j}$ for a $\theta \in \mathbb{R}^+$ with constraints that are determined later on. In order to associate parts of the construction more easily with the original formula, its symbols for literals and clauses are reused. For formula f, we construct pta $\mathcal{A}_f = (Q, \Sigma, \mu, \nu)$ where

$$Q = \{ F \} \cup \{ C_i \mid i \in [k] \} \cup \{ U^i_j \mid i \in [k], j \in [3] \},$$

$$\Sigma = \left\{ f^{(k)} \right\} \cup \left\{ {c_i}^{(3)} \mid i \in [k] \right\} \cup \left\{ {u^i_j}^{(1)} \mid i \in [k], j \in [3] \right\} \cup \left\{ \top^{(0)}, \bot^{(0)} \right\},$$

$$\mu_f = \{ ((C_1, \ldots, C_k), F) \to p_0 \} \qquad \text{(Type 1)}$$

$$\cup \bigcup_{j\in[n]} \bigcup_{b\in\{\top,\bot\}} \left\{ ((c_1(\xi^1_1, \xi^1_2, \xi^1_3), \ldots, c_k(\xi^k_1, \xi^k_2, \xi^k_3)), F) \to p_j \right\} \qquad \text{(Type 2)}$$

such that, for each $i' \in [k]$ and $j' \in [3]$, we have

$$\xi^{i'}_{j'} = \begin{cases} u^{i'}_{j'}(\top) & \text{if } u^{i'}_{j'} = x_j \wedge b = \top \\ u^{i'}_{j'}(\bot) & \text{if } u^{i'}_{j'} = x_j \wedge b = \bot \\ u^{i'}_{j'}(\bot) & \text{if } u^{i'}_{j'} = \bar{x}_j \wedge b = \top \\ u^{i'}_{j'}(\top) & \text{if } u^{i'}_{j'} = \bar{x}_j \wedge b = \bot \\ U^{i'}_{j'} & \text{otherwise,} \end{cases}$$

$$\mu_{c_i} = \left\{ ((u^i_1(\top), U^i_2, U^i_3), C_i) \to \frac{1}{3} \right\}$$

$$\cup \left\{ ((U^i_1, u^i_2(\top), U^i_3), C_i) \to \frac{1}{3} \right\}$$

$$\cup \left\{ ((U^i_1, U^i_2, u^i_3(\top)), C_i) \to \frac{1}{3} \right\} \text{ for each } i \in [k], \qquad \text{(Type 3)}$$

$$\mu_{u^i_j} = \left\{ ((b), U^i_j) \to \frac{1}{2} \mid b \in \{ \top, \bot \} \right\} \text{ for each } i \in [k] \text{ and } j \in [3], \text{ and} \qquad \text{(Type 4)}$$

$$\nu_F = 1.$$

A demonstration of how the transitions are built for a specific 3-SAT formula is presented in Example 4. Figure 7 shows the accompanying visual representations of these transitions and compatible probabilities.

Example 4. For the following specific instance of a 3-SAT formula (Barton, Berwick, and Ristad 1987), we exemplarily show what transitions are constructed for the reduction:

$$\begin{aligned} f &= c_1 \wedge c_2 \\ &= (u_1^1 \vee u_2^1 \vee u_3^1) \wedge (u_1^2 \vee u_2^2 \vee u_3^2) \\ &= (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3). \end{aligned}$$

Note that for this example we replace the abstract symbols for literals by the (negated) variables themselves, e.g., $\bar{x}_2$ instead of u_2^1. Visual representations of the transitions with suitable probabilities are shown in Figure 7. These transitions are described subsequently:

(a) For each $j \in [3]$ and $b \in \{\top, \bot\}$, we add a transition that consistently assigns b to variable x_j at each occurrence of x_j. The transition that consistently assigns $\top$ to variable x_j, for example, accepts the tree with root f that has children c_1 and c_2. Furthermore, for $k \in [2], i \in [3]$, the children of c_k are states U_i^k, except if the corresponding literal is x_j or $\bar{x}_j$: In case it is x_j we add child $x_j(\top)$ and if it is $\bar{x}_j$ we add $\bar{x}_j(\bot)$. Transitions of this kind are depicted in Figure 7a.

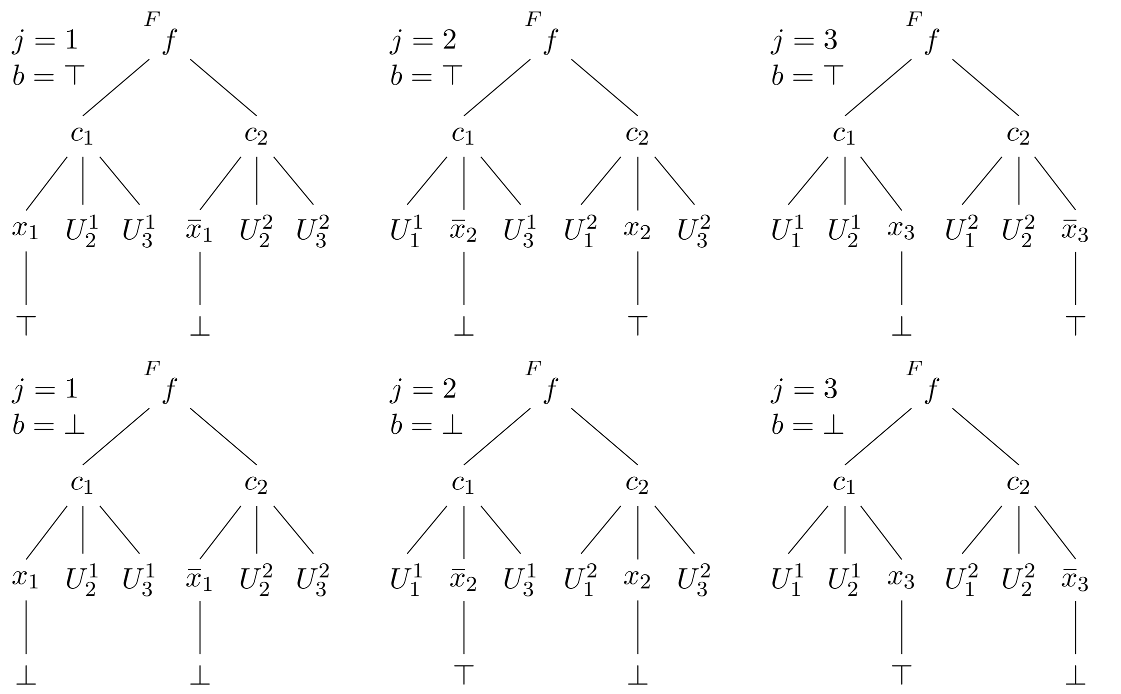
(b, c) In order to guarantee that the formula is satisfied, we first include a transition that recognises $f(C_1, C_2)$ (Figure 7b). Subsequently, we add transitions that accept for a state C_k ($k \in [2]$), a tree that ensures that at least one of its literals evaluates to $\top$. For C_2 we add for example a transition that accepts a tree with c_2 at its root and U_1^2, U_2^2 and $\bar{x}_3(\top)$ as children. Figures 7b and 7c show the corresponding illustrations.

(d) Finally, there are positions where the truth assignment does not matter. These are signified by states U_i^k ($k \in [2], i \in [3]$). For these we include transitions that accept a tree consisting of the literal and either assignment, e.g., for state U_2^1, the trees $\bar{x}_2(\top)$ and $\bar{x}_2(\bot)$. These transitions are shown in Figure 7d.

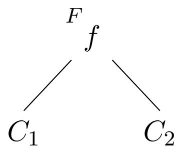
$\triangle$

Deriving probabilities and the threshold The probabilities for transitions in $\mu_{u_j^i}$ and μ_{c_i} ($i \in [k], j \in [n]$) are $\frac{1}{2}$ and $\frac{1}{3}$, respectively. Their choice is relatively straightforward as the resulting pta is proper and therefore the probabilities of all transitions with the same target state sum up to 1.

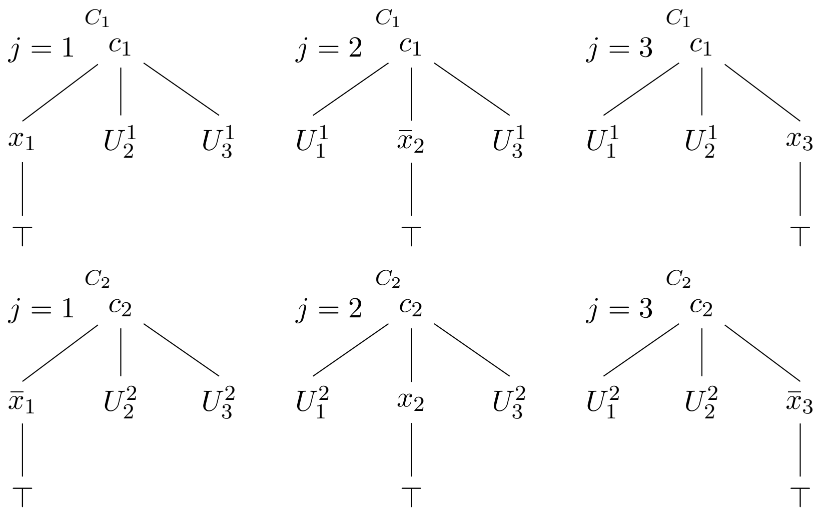
The values $p_0 = 1 - 2\sum_{j\in[n]} \theta \cdot \left(\frac{1}{2}\right)^{n_j}$ and $p_j = \theta \cdot \left(\frac{1}{2}\right)^{n_j}$ ($j \in [n]$) for transitions in μ_f are based on the following observations: First, the pta for a formula f accepts trees that



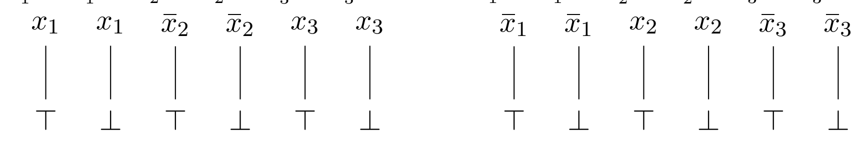
(a) $p_j = \frac{2}{13}$



(b) $p_0 = \frac{1}{13}$

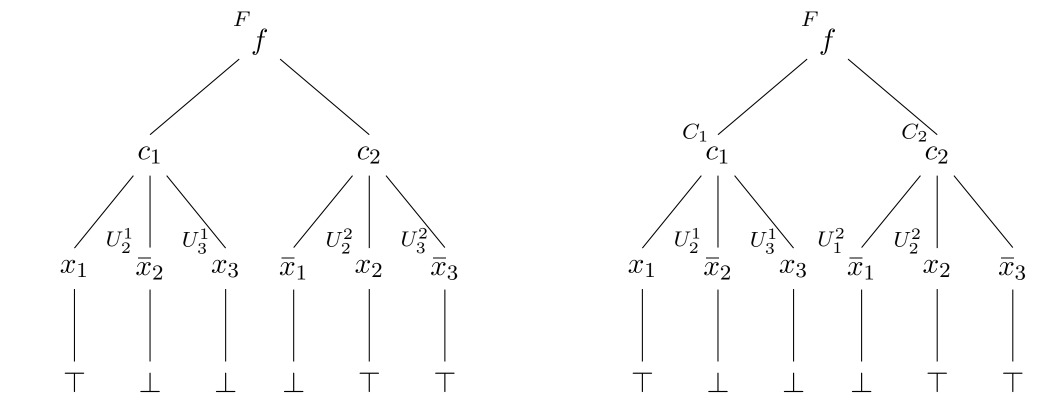


(c) $p = \frac{1}{3}$



(d) $p = \frac{1}{2}$

Figure 7.: Visual representation of all transitions of the pta created for 3-SAT formula $f = c_1 \wedge c_2 = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$.



(a) Run for consistent assignment of variable x_1.

(b) Run ensuring each clause is satisfied by assigning $\top$ to x_1 and $\bot$ to x_3.

Figure 8.: Trees with a consistent assignment for x_1, x_2 and x_3 that satisfy formula f.

only differ at the leaf positions, i.e., in their assignment of truth values. Secondly, there are two kinds of runs for each tree ξ:

1. For a $j \in [n]$: A transition of type 2 is followed by $3k - n_j$ transitions of type 3. The resulting tree has a consistent assignment of variable x_j. As this can be done for all variables, there are at most n runs of this kind for ξ (Figure 8a). These runs have a probability of
$$p_j \cdot \left(\frac{1}{2}\right)^{3k-n_j} = \theta \cdot \left(\frac{1}{2}\right)^{3k}.$$
2. For the second kind of run (Figure 8b), we have a transition of type 1 followed by k transitions of type 3 such that each clause is satisfied and finally, $2k$ transitions of type 4 for the assignment of the remaining literals. There are at most $3k$ runs of this kind for ξ because we have 3 different options of satisfying a clause for each clause. The probability of such runs is
$$p_0 \cdot \left(\frac{1}{3}\right)^{k} \cdot \left(\frac{1}{2}\right)^{2k} = \left(1 - 2\sum_{j\in[n]} \theta \cdot \left(\frac{1}{2}\right)^{n_j}\right) \cdot \left(\frac{1}{3}\right)^{k} \cdot \left(\frac{1}{2}\right)^{2k}.$$

The values for parameter θ and the threshold λ are chosen such that for a tree ξ we have $\Pr_{\mathcal{A}f}(\xi) \geq \lambda$ if and only if ξ represents a satisfying and consistent variable assignment for formula f. Such a tree ξ has n runs of the first kind (consistent assignment) and at least one run of the second kind (satisfied clauses). For that reason we set the threshold at

$$\lambda = n \cdot \theta \cdot \left(\frac{1}{2}\right)^{3k} + \left(1 - 2\sum_{j\in[n]} \theta \cdot \left(\frac{1}{2}\right)^{n_j}\right) \cdot \left(\frac{1}{3}\right)^{k} \cdot \left(\frac{1}{2}\right)^{2k}.$$

There are, however, two constraints on parameter θ:

1. The pta is proper and therefore we have for all $j \in [n] : 0 \le p_j \le 1$ and $0 \le p_0 \le 1$. In consequence, $0 \le \theta \le 2^{n_j}$ and $0 \le \theta \le \left(2\sum_{j'\in[n]} \cdot (\frac{1}{2})^{n_{j'}}\right)^{-1}$. As a result of

$$\begin{aligned} 2\sum_{j'\in[n]} \cdot \left(\frac{1}{2}\right)^{n_{j'}} &\ge \left(\frac{1}{2}\right)^{n_j} \\ \Leftrightarrow \left(2\sum_{j'\in[n]} \cdot \left(\frac{1}{2}\right)^{n_{j'}}\right)^{-1} &\le 2^{n_j}, \end{aligned}$$

   the latter upper bound is stricter and therefore the first constraint on θ is:

$$0 \le \theta \le \left(2\sum_{j'\in[n]} \cdot \left(\frac{1}{2}\right)^{n_{j'}}\right)^{-1}. \qquad \text{(upper bound)}$$

2. In order to infer from the probability of the tree whether it represents a consistent and satisfying alignment, we have to make sure that runs of the second kind (satisfied clauses) cannot make up for missing runs of the first kind (consistent assignment). Since there can be at most 3^k of the second kind of runs, we require that

$$\begin{aligned} 3^k \cdot \left(1 - 2\sum_{j\in[n]} \theta \cdot \left(\frac{1}{2}\right)^{n_j}\right) \cdot \left(\frac{1}{3}\right)^{k} \cdot \left(\frac{1}{2}\right)^{2k} &< \theta \cdot \left(\frac{1}{2}\right)^{3k} \\ \Leftrightarrow \qquad \left(1 - 2\sum_{j\in[n]} \theta \cdot \left(\frac{1}{2}\right)^{n_j}\right) &< \theta \cdot \left(\frac{1}{2}\right)^{k} \\ \Leftrightarrow \qquad \left(\left(\frac{1}{2}\right)^{k} + 2\sum_{j\in[n]} \left(\frac{1}{2}\right)^{n_j}\right)^{-1} &< \theta. \qquad \text{(lower bound)} \end{aligned}$$

The lower bound is strictly smaller than the upper bound. Hence, it is feasible to find a value for θ satisfying both bounds.

Polynomiality Given a formula with k clauses and n variables, the construction of the corresponding pta is achievable in deterministic polynomial-time in k and n: For the pta in extended form we have $|Q| = 1 + k + 3k$, $|\Sigma| = 1 + k + 3k + 2$ and $1 + 2n + 3k + 2 \cdot 3k$ transitions. Each transition is of size at most $1 + 5k$ (counting the involved states and symbols). Furthermore, the transformation of the pta in extended form to its normal form can be done in time polynomial in k and n as well. This is because for a transition at most $3k$ new states and transitions have to be added. What's more, the parameter θ and threshold λ are calculable in polynomial-time and as all accepted trees are of size $1 + 7k$, computing their probabilities is possible in polynomial-time, too.

Answer preserving Finally, it has to be shown that the reduction preserves the answers from the original 3-SAT instance. Therefore, whenever there is a satisfying and consistent assignment of variables for formula f, there is a tree in pta $\mathcal{A}_f$ with a probability higher than the threshold and vice versa.

Corollary 2. Let f be a 3-SAT formula and $\mathcal{A}_f = (Q, \Sigma, \mu, \nu)$ be the corresponding pta from the reduction with threshold λ. Formula f is satisfiable if and only if there is a tree $\xi \in T_\Sigma$ with $\mathrm{Pr}_{\mathcal{A}f}(\xi) \geq \lambda$.

Proof. We show that, first, formula f is satisfiable implies there is a tree ξ with probability $\mathrm{Pr}_{\mathcal{A}f}(\xi) \geq \lambda$ and, secondly, a tree ξ with $\mathrm{Pr}_{\mathcal{A}f}(\xi) \geq \lambda$ implies f is satisfiable:

"$\Rightarrow$": If f is satisfiable there is a consistent assignment of variables and for each clause there is at least one literal that evaluates to true. A tree ξ that corresponds to the assignment has n runs of the first kind that ensure a consistent assignment for each variable and at least one second kind run where each clause is guaranteed to be satisfied. Due to the choice of λ, we have $\mathrm{Pr}_{\mathcal{A}f}(\xi) \geq \lambda$.

"$\Leftarrow$": Having a tree ξ for which $\mathrm{Pr}_{\mathcal{A}f}(\xi) \geq \lambda$ implies that ξ has n runs of the first kind (consistency) and at least one of the second kind (satisfying). This is because parameter θ has been chosen such that the probability of additional runs of the second kind cannot make up for missing runs of the first kind. Hence, the assignment of variables represented by ξ is a consistent and satisfying assignment of variables for formula f. □

This concludes the polynomial-time reduction of the NP-complete problem 3-SAT to the most probable tree decision problem, i.e., 3-SAT $\preceq_p$ MPT-decision. In effect, this proves MPT to be NP-hard (Theorem 3). Provided the tree is not specially encoded, Section 3.2 shows that MPT cannot be in NP and therefore not NP-complete: There may be an instance where the mpt is of a size greater than a polynomial of the size of the pta.

5. Most probable tree algorithm

The problem of finding the most probable tree is one of the most interesting problems for probabilistic tree automata. Unfortunately, as is shown in Chapter 4, this problem is NP-hard. This result suggests that finding a solution is computationally expensive and hence, practically intractable. Oftentimes one settles for calculating the best run and assumes that it is a good approximation of the most probable tree. Probabilistic finite state automata (pfa) are subject to the same problem since finding the most probable string has been proved to be NP-hard as well (Casacuberta and de la Higuera 2000). Nevertheless, de la Higuera and Oncina 2013 devised an algorithm (Algorithm 1) that computes the most probable string. We adapt this algorithm in order to generalise it to probabilistic tree automata.

First, we present the algorithm for calculating the most probable string in Section 5.1. Using this as a foundation, we subsequently generalise the algorithm for use with probabilistic tree automata and highlight differences (Section 5.2). In Section 5.3 the corresponding time complexity analysis for the mpt algorithm can be found. How well this holds up in an implementation is illustrated by experiments in Section 5.4. Some details regarding the implementation itself follow immediately afterwards (Section 5.5).

5.1. Calculating the most probable string

The algorithm for computing the most probable string of a probabilistic finite automaton by de la Higuera and Oncina 2013 is based on two concepts: The potential probability of a string that defines an upper bound to the probability of a string and a priority queue that helps to track what strings are worth extending.

Let $\mathcal{B} = (Q, \Gamma, M, I, F)$ be a pfa and $w \in \Gamma^*$ a word over alphabet Γ. The potential probability of w is defined as $\mathrm{PP}_{\mathcal{B}}(w) = \min(\Pr_{\mathcal{B}}(w\Gamma^*), \frac{|\mathcal{B}|^2}{|w|})$ (de la Higuera and Oncina 2013, Definition 2). While the second part is useful during the runtime analysis, the first part is essential to the algorithm: $\Pr_{\mathcal{B}}(w\Gamma^*)$ defines the prefix probability of string $w = w_1 \dots w_n$. It can be easily calculated by means of matrix multiplication (cf. Cognetta, Han, and Kwon 2018, Section 2.1):

$$\Pr_{\mathcal{B}}(w\Gamma^*) = I \cdot M(w_1) \cdot \ldots \cdot M(w_n) \cdot M(\Gamma^*) \cdot F,$$

where $M(\Gamma^*) = \sum_{i=0}^{\infty} M(\Gamma)^i = (\mathbf{1} - M(\Gamma))^{-1}$ and $\mathbf{1}$ denotes the identity matrix of suitable dimensions.

Algorithm 1: most probable string alg. (de la Higuera and Oncina 2013, Alg. 1)

```
Input: probabilistic finite automaton $\mathcal{B} = (Q, \Gamma, M, I, F)$
Data: priority queue Q containing strings $w \in \Gamma^*$ sorted by $\mathrm{PP}_{\mathcal{B}}(w)$
Output: most probable string $\widehat{w} \in \arg\max_{w \in \Gamma^*} \Pr_{\mathcal{B}}(w)$
1  Q ← [$\varepsilon$]
2  $\widehat{w} \leftarrow \varepsilon$
3  while not Q.is_empty do
4      $w \leftarrow$ Q.pop_best()
5      if $\mathrm{PP}_{\mathcal{B}}(w) > \Pr_{\mathcal{B}}(\widehat{w})$ then
6          if $\Pr_{\mathcal{B}}(w) > \Pr_{\mathcal{B}}(\widehat{w})$ then
7              $\widehat{w} \leftarrow w$
8          foreach $a \in \Gamma$ do
9              if $\mathrm{PP}_{\mathcal{B}}(wa) > \Pr_{\mathcal{B}}(\widehat{w})$ then
10                 Q.insert($wa, \mathrm{PP}_{\mathcal{B}}(wa)$)
11     else
12         return $\widehat{w}$
13 return $\widehat{w}$
```

In Algorithm 1 we gradually build strings and keep a priority queue of those strings which is sorted according to their respective potential probabilities. Basically, the algorithm keeps extending the string in the queue with the highest potential probability until it is worse than the best string found so far. In detail it does the following: First, the priority queue Q and the current best word $\widehat{w}$ are initialised with the empty string (`lines 1-2`). In each iteration of the main loop (`line 3-13`) the string w with the highest potential probability in the queue is examined (`line 4`). In case w has the potential to be better than the current best string $\widehat{w}$ (`line 5`), extensions of w are considered (`lines 8-10`) and should the string w already have a higher probability than $\widehat{w}$, the current best string is updated (`lines 6-7`). In consequence, a string w cannot only be taken into consideration as a candidate for the most probable string but one may even extend w with a symbol $a \in \Gamma$ should wa's potential probability be high enough. Should the queue's best word w have a lower potential probability than the probability of the current best $\widehat{w}$, we return $\widehat{w}$ as the most probable string.

5.2. Generalisation to trees

The most challenging element in the adaptation of the most probable string algorithm to probabilistic tree automata was the definition of potential probability for trees. Even though the probability bound of de la Higuera and Oncina 2013 (Proposition 1) proved to be generalisable to pta, the calculation of prefix probabilities does not work as similar for trees. We failed to find a way to use matrix multiplication to determine the probability of a prefix tree. Mostly since we could not appropriately incorporate branching transitions in the multiplication.

In fact, the calculation of prefix probabilities proved to be even simpler for trees. For a prefix tree $\xi \in T_\Sigma(X) \setminus T_\Sigma$ the difficulty was: There is at least one position $p \in \text{pos}(\xi)$ with $\xi(p) \in X$ and at some point we have to determine $\text{wt}(\kappa|_p)$. Now we simply define $\text{wt}(\kappa|_p) = 1$. An intuitive explanation for this is that, due to properness, we have

$$\text{wt}(\kappa|_p) = \sum_{\sigma \in \Sigma} \sum_{q_1,\dots,q_k} \mu_\sigma((q_1, \dots, q_k), \kappa|_p(\varepsilon)) = 1.$$

Using the result of Theorem 1, the potential probability of trees is therefore defined as follows:

Definition 7. Let pta $\mathcal{A} = (Q, \Sigma, \mu, \nu)$ and $\xi \in T_\Sigma(X)$. The *potential probability of* ξ *given* $\mathcal{A}$ is

$$\text{PP}_\mathcal{A}(\xi) = \min\left(\Pr_\mathcal{A}(\xi), \frac{|\mathcal{A}|^2}{\text{height}(\xi)}\right).$$

As for the algorithm itself: Finding the most probable tree (Algorithm 2) differs from the most probable string algorithm due to the fact that a tree is either a prefix or complete. A string can be a candidate for the most probable string and extended at the same time. A tree is only one of the two: Either a complete tree that could be a candidate for the mpt or a prefix tree whose variables we can substitute (thus extending the tree). As a result the mpt algorithm differs from the original algorithm as follows.

Initially, since we cannot instantiate the priority queue with an empty tree, the priority queue is filled with trees $\sigma(x_1, \dots, x_k)$ for $k \in \mathbb{N}, \sigma \in \Sigma^{(k)}$ (`lines 2-3`). In the main loop (`lines 5-15`) we have to check if either of two cases is true for the best element in the queue ξ: If $\xi \in T_\Sigma$ (`lines 7-8`), it is complete and the most probable tree we were looking for. The latter holds because extending other trees in the priority queue cannot yield a complete tree $\xi' \in T_\Sigma$ such that $\Pr_\mathcal{A}(\xi') > \Pr_\mathcal{A}(\xi)$ since extending a tree can never improve its probability. This is because at the position in each run where a variable is, the weight is 1 but by extending a tree, we apply at that position a transition with a weight of at most 1 and the total weight can at best stay the same. In case

$\xi \in T_\Sigma(X) \setminus T_\Sigma$ (`lines 9-15`), we look for the first position $p \in \mathrm{pos}(\xi)$ where $\xi(p) \in X$ in a breadth-first manner and for each $k \in \mathbb{N}, \sigma \in \Sigma^{(k)}$ extended trees $\xi' = \xi[\sigma(x_1, \ldots, x_k)]_p$ are created (`lines 10-11`). This tree ξ' is then inserted into the priority queue given that its potential probability is not smaller than the current best complete tree $\widehat{\xi} \in T_\Sigma$ in the priority queue (`lines 12-15`). If also $\xi' \in T_\Sigma$, then the current best tree is updated (`lines 13-14`). An exemplary execution of the algorithm on the pta from Example 2 (cf. Figure 2) is given in Example 5.

```
Algorithm 2: most probable tree algorithm
   Input: probabilistic tree automaton $\mathcal{A} = (Q, \Sigma, \mu, \nu)$
   Data: priority queue Q containing trees $\xi \in T_\Sigma(X)$ sorted by $\mathrm{PP}_\mathcal{A}(\xi)$
   Output: most probable tree $\widehat{\xi} \in \arg\max_{\xi \in T_\Sigma} \mathrm{Pr}_\mathcal{A}(\xi)$
 1 Q ← [ ]
 2 foreach $\sigma \in \Sigma$ do
 3     Q.insert($\sigma(x_1, \ldots, x_k), \mathrm{PP}_\mathcal{A}(\sigma(x_1, \ldots, x_k))$)
 4 $\widehat{\xi}$ ← undefined
 5 while not Q.is_empty do
 6     $\xi$ ← Q.pop_best()
 7     if $\xi \in T_\Sigma$ then
 8         return $\xi$
 9     else
10         foreach $\sigma \in \Sigma$ do
11             $\xi'$ ← $\xi$.extend($\sigma$)
12             if $\mathrm{PP}_\mathcal{A}(\xi') > \mathrm{PP}_\mathcal{A}(\widehat{\xi})$ or $\widehat{\xi}$.is_undefined then
13                 if $\xi' \in T_\Sigma$ then
14                     $\widehat{\xi}$ ← $\xi'$
15                 Q.insert($\xi', \mathrm{PP}_\mathcal{A}(\xi')$)
16 return $\widehat{\xi}$
```

Example 5. Exemplary execution of Algorithm 2 that calculates the most probable tree for the pta $\mathcal{A}$ as it is provided in Example 2 (cf. Figure 2). Each row of the chart describes the contents of the priority queue Q, the current best complete tree $\widehat{\xi} \in T_\Sigma$ and the queues most probable tree $\xi \in T_\Sigma(X)$ after each iteration of the main loop (after `line 15`). Except for the first row which displays the initialisation of those values (after `line 4`).

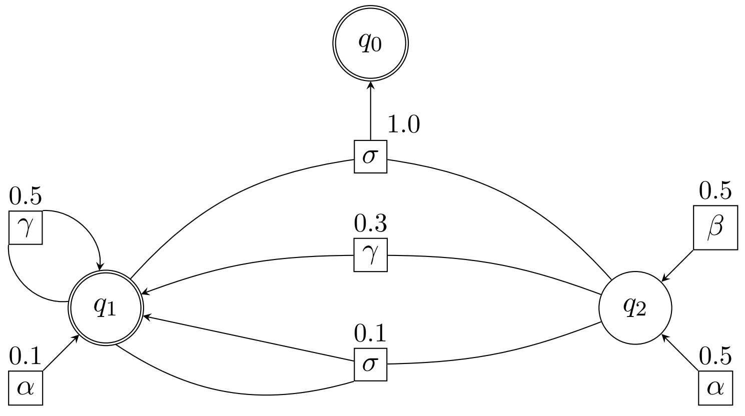


Figure 2.: A depiction of the probabilistic tree automaton as described in Example 2 with root weights $\nu_{q_0} = 0.9$ and $\nu_{q_1} = 0.1$. (Repeated from Chapter 2 for Example 5.)

it.	Q		$\widehat{\xi}, \mathrm{PP}_{\mathcal{A}}(\widehat{\xi})$	$\xi, \mathrm{PP}_{\mathcal{A}}(\xi)$
0	$[\,(\sigma(x_1, x_2), 0.91)$, $(\gamma(x_1), 0.08)$,	$(\alpha, 0.01)$, $(\beta, 0)\,]$	/	/
1	$[\,(\sigma(\gamma(x_3), x_2), 0.728)$, $(\sigma(\alpha, x_2), 0.091)$, $(\sigma(\sigma(x_3, x_4), x_2), 0.091)$,	$(\gamma(x_1), 0.08)$, $(\alpha, 0.01)$, $(\beta, 0)\,]$	/	$\sigma(x_1, x_2)$, 0.91
2	$[\,(\sigma(\gamma(x_3), \beta), 0.364)$, $(\sigma(\gamma(x_3), \alpha), 0.364)$, $(\sigma(\alpha, x_2), 0.091)$, $(\sigma(\sigma(x_3, x_4), x_2), 0.091)$,	$(\gamma(x_1), 0.08)$, $(\alpha, 0.01)$, $(\beta, 0)\,]$	/	$\sigma(\gamma(x_3), x_2)$, 0.728
3	$[\,(\sigma(\gamma(x_3), \alpha), 0.364)$, $(\sigma(\gamma(\gamma(x_4)), \beta), 0.182)$, $(\sigma(\gamma(\alpha), \beta), 0.091)$, $(\sigma(\alpha, x_2), 0.091)$, $(\sigma(\sigma(x_3, x_4), x_2), 0.091)$,	$(\gamma(x_1), 0.08)$, $(\sigma(\gamma(\beta), \beta), 0.06825)$, $(\alpha, 0.01)$, $(\beta, 0)\,]$	$\sigma(\gamma(\alpha), \beta)$, 0.091	$\sigma(\gamma(x_3), \beta)$, 0.364
4	$[\,(\sigma(\gamma(\gamma(x_4)), \beta), 0.182)$, $(\sigma(\gamma(\gamma(x_4)), \alpha), 0.182)$, $(\sigma(\gamma(\alpha), \beta), 0.091)$,	$(\sigma(\alpha, x_2), 0.091)$, $(\sigma(\sigma(x_3, x_4), x_2), 0.091)$, $\ldots\,]$	$\sigma(\gamma(\alpha), \beta)$, 0.091	$\sigma(\gamma(x_3), \alpha)$, 0.364
5	$[\,(\sigma(\gamma(\gamma(x_4)), \alpha), 0.182)$, $(\sigma(\gamma(\alpha), \beta), 0.091)$, $(\sigma(\alpha, x_2), 0.091)$,	$(\sigma(\sigma(x_3, x_4), x_2), 0.091)$, $\ldots\,]$	$\sigma(\gamma(\alpha), \beta)$, 0.091	$\sigma(\gamma(\gamma(x_4)), \beta)$, 0.182
6	$[\,(\sigma(\gamma(\alpha), \beta), 0.091)$, $(\sigma(\alpha, x_2), 0.091)$,	$(\sigma(\sigma(x_3, x_4), x_2), 0.091)$, $\ldots\,]$	$\sigma(\gamma(\alpha), \beta)$, 0.091	$\sigma(\gamma(\gamma(x_4)), \alpha)$, 0.182

As the top of the queue is a complete tree, the algorithm terminates in iteration 7. This best tree in Q is returned as the tree that is recognised with the highest probability by pta $\mathcal{A}$: $\sigma(\gamma(\alpha), \beta) \in \arg\max_{\xi' \in T_\Sigma} \mathrm{Pr}_\mathcal{A}(\xi')$ with $\mathrm{Pr}_\mathcal{A}(\sigma(\gamma(\alpha), \beta)) = 0.091$. $\triangle$

5.3. Complexity analysis

For the complexity analysis of Algorithm 2, we begin by eliciting what parts of it dominate the runtime. After that, we look into what operations are used there and what their time complexity is. The overall theoretical runtime depends on the size of two values which we determine subsequently.

Coarse analysis At the beginning of the algorithm (`lines 1-4`) there are only $|\Sigma|$ insertions into the empty priority queue Q. The algorithm's runtime is therefore dominated by the main loop (`lines 5-15`). This main loop terminates either if the mpt $\widehat{\xi}$ has been found (`lines 7-8`) or if Q is empty (`line 5`). Since the latter only happens if the pta $\mathcal{A}$ has no mpt, we determine how many loops there can be until the mpt is found: Let $V = \{\, \xi \in T_\Sigma(X) \mid \mathrm{PP}_\mathcal{A}(\xi) \geq \widehat{p} \}$ be a set of *viable* trees whose potential probability is at least as high as the highest probability $\widehat{p}$. In case the best tree ξ in Q (`line 6`) has potential probability $\mathrm{PP}_\mathcal{A}(\xi) \geq \widehat{p}$ it is viable. If $\mathrm{PP}_\mathcal{A}(\xi) < \widehat{p}$, then it is not in V and we cannot improve its probability by extending (as we mentioned before). As all trees in Q have a lower probability, the mpt must have been found already. In conclusion, until the mpt is found, every best element in Q we look at has to be viable. Therefore, the loop is executed at most $|V|$ many times.

Operations of the loop The following operations are executed during the main loop:

- ξ`.extend` and $\xi \in T_\Sigma$
  Extending a tree can be accomplished in constant time: Since a tree is extended in a breadth-first manner we simply keep a list of variables in the tree. Whenever we extend ξ with a tree $\sigma(x_1, \ldots, x_k)$, the first variable in the list will be substituted by that tree and variables $x_1, \ldots, x_k$ are appended to the list. An empty list of variables indicates that ξ is complete and therefore $\xi \in T_\Sigma$ can be done in constant time as well.

- $\mathrm{PP}_\mathcal{A}(\xi)$
  Let m be the number of transitions in a pta $\mathcal{A}$. By memorising the probability of each tree, one can calculate the probability of a newly created, extended tree ξ in time $\mathcal{O}(|\mathcal{A}| \cdot m \cdot \mathrm{height}(\xi))$.

- `Q.insert`$(\xi, \mathrm{PP}_{\mathcal{A}}(\xi))$ and `Q.pop_best`$()$
  A simple implementation of a priority queue with a binary heap (Williams 1964) has a worst case time complexity for `pop_best` and `insert` of $\mathcal{O}(\log(|\mathsf{Q}|))$.

As the priority queue `Q` grows quite large (cf. Section 5.3.2), `pop_best` and `insert` clearly dominate the runtime of the other operations. Let $|\mathsf{Q}_{\max}|$ be the maximum size of the priority queue. Taking the loop over all symbols (`lines 10-15`) into account, we get a time complexity of:

$$\mathcal{O}(|V| \cdot (|\Sigma| + 1) \cdot \log(|\mathsf{Q}_{\max}|)).$$

In the following, we determine how many viable trees ($|V|$) there can be and how large the priority queue can grow ($|\mathsf{Q}_{\max}|$).

5.3.1. Size of viable set

In order to asses how many viable trees there can be, we need to introduce the concept of ununifiable sets of trees. This is borrowed from the idea of substituting variables by terms in formal expressions. Here, ununifiability deals with prefix trees that cannot be extended to the same complete tree and the substitution of variables takes place independently of other variables. The concept is defined as follows:

Definition 8. A set of trees $U \subseteq T_\Sigma(X)$ is called *ununifiable* if for two different trees in U their respective sets of reachable trees (Definition 2) are disjoint, i.e.,

$$\forall \xi, \zeta \in U : \xi \neq \zeta \implies T_\Sigma|_\xi \cap T_\Sigma|_\zeta = \varnothing.$$

Since each tree $\xi \in T_\Sigma$ has at most one prefix in an ununifiable set of trees, we can deduce that such sets adhere to the consistency constraint of pta. With that one can show a bound on the size of ununifiable sets of trees whose elements' probability has a known lower bound.

Lemma 4. Let $U \subseteq T_\Sigma(X)$ be an ununifiable set of trees and $\mathcal{A}$ a pta. Then $\sum_{\zeta \in U} \mathrm{Pr}_{\mathcal{A}}(\zeta) \leq 1$.

Proof.

$$
\begin{aligned}
\sum_{\zeta \in U} \Pr_{\mathcal{A}}(\zeta) &= \sum_{\zeta \in U} \sum_{\xi \in T_\Sigma|_\zeta} \Pr_{\mathcal{A}}(\xi) && \text{(prefix probability, Section 2.3)}\\
&= \sum_{\xi \in T_\Sigma} \sum_{\substack{\zeta \in U:\\ \xi \in T_\Sigma|_\zeta}} \Pr_{\mathcal{A}}(\xi)\\
&\le \sum_{\xi \in T_\Sigma} \Pr_{\mathcal{A}}(\xi) && \text{(U ununifiable)}\\
&= 1. && \text{($\mathcal{A}$ consistent)}
\end{aligned}
$$

□

Corollary 3. Let $U \subseteq T_\Sigma(X)$ be an ununifiable set of trees such that $\forall \zeta \in U : \Pr_{\mathcal{A}}(\zeta) > p$. Then $|U| < \frac{1}{p}$.

Proof.

$$
\begin{aligned}
& \sum_{\zeta \in U} \Pr_{\mathcal{A}}(\zeta) \le 1 && \text{(Lemma 4)}\\
\Rightarrow \quad & |U| \cdot p < 1 && (\forall \zeta \in U : \Pr_{\mathcal{A}}(\zeta) > p)\\
\Leftrightarrow \quad & |U| < \frac{1}{p}.
\end{aligned}
$$

□

Let $\mathcal{A} = (Q, \Sigma, \mu, \nu)$ be a pta, $\widehat{p} = \max_{\xi \in T_\Sigma} \Pr_{\mathcal{A}}(\xi)$ be the probability of the mpt, and let $\widehat{k} = \max_{\sigma \in \Sigma} \mathrm{rk}(\sigma)$ be the maximum rank of $\mathcal{A}$. We can conclude the following:

(1) Let $\xi \in V$. Due to the definition of potential probability (Definition 7), $\mathrm{PP}_{\mathcal{A}}(\xi) \ge \widehat{p}$ implies $\frac{|\mathcal{A}|^2}{\mathrm{height}(\xi)} \ge \widehat{p}$ and accordingly, $\mathrm{height}(\xi) \le \frac{|\mathcal{A}|^2}{\widehat{p}}$. Furthermore, the size of ξ is bounded by

$$
\begin{aligned}
\mathrm{size}(\xi) &\le \sum_{i=0}^{\mathrm{height}(\xi)} \widehat{k}^i\\
&\le \frac{\widehat{k}^{\frac{|\mathcal{A}|^2}{\widehat{p}}+1} - 1}{\widehat{k} - 1}. && \text{(Partial sum of geometric series)}
\end{aligned}
$$

(2) A tree is always extended in a breadth-first manner (from left to right). For example, the tree $\sigma(x_1, x_2)$ could be extended to $\sigma(\alpha, x_2)$ but never to $\sigma(x_1, \alpha)$. As a result,

for a complete tree $\xi \in T_\Sigma$, there exist at most $\operatorname{size}(\xi)$ many trees $\zeta \in T_\Sigma(X)$ such that ξ can be reached from ζ, that is, $|\{\, \zeta \in T_\Sigma(X) \mid \xi \in T_\Sigma|_\zeta \,\}| \leq \operatorname{size}(\xi)$.

(3) Let $U \subseteq V$ be ununifiable. Then all trees $\xi \in U$ have a potential probability of $\mathrm{PP}_{\mathcal{A}}(\xi) > \widehat{p}$ and therefore a probability of $\Pr_{\mathcal{A}}(\xi) > \widehat{p}$. Due to Corollary 3 it follows that $|U| < \frac{1}{\widehat{p}}$, i.e., there are at most $\frac{1}{\widehat{p}}$ trees in V that can be extended into pairwise disjoint sets of complete trees.

Thus, for each tree ξ from the ununifiable subset of Item (3) there at most $\operatorname{size}(\xi)$ many prefixes from which ξ is reachable according to Item (2) and the tree size is bounded (Item (1)). This leads to the upper bound on the number of different viable trees:

$$|V| \leq \frac{1}{\widehat{p}} \cdot \frac{\widehat{k}^{\frac{|\mathcal{A}|^2}{\widehat{p}}+1}}{\widehat{k}-1}.$$

5.3.2. Size of priority queue and total time complexity

Let ξ be the first element of the priority queue Q at some iteration of the main loop. Should ξ not be in V, i.e., $\mathrm{PP}_{\mathcal{A}}(\xi) < \widehat{p}$, every other element in Q has a smaller potential probability as well. Complete trees $\xi \in T_\Sigma$ therefore are too unlikely to be the mpt and prefix trees $\xi \in T_\Sigma(X) \setminus T_\Sigma$ can only worsen their probability by extending. In consequence, until the most probable tree is found, the first element of Q is always viable, i.e., $\xi \in V$.

In every iteration of the main loop, the first element of the priority queue Q causes at most $|\Sigma|$ many insertions. Since we know an upper bound on the number of viable trees, we can infer the maximum number of insertions into Q and therefore its maximum size:

$$|\mathsf{Q}_{\max}| \leq |V| \cdot |\Sigma| \leq \frac{|\Sigma|}{\widehat{p}} \cdot \frac{\widehat{k}^{\frac{|\mathcal{A}|^2}{\widehat{p}}+1}}{\widehat{k}-1}.$$

and consequently the time complexity the queue operations `insertion` and `pop_best` are in

$$\mathcal{O}\left(\log\left(\frac{|\Sigma|}{\widehat{p}} \cdot \frac{\widehat{k}^{\frac{|\mathcal{A}|^2}{\widehat{p}}+1}}{\widehat{k}-1}\right)\right).$$

Finally, the overall worst case runtime complexity of Algorithm 2 is in

$$\mathcal{O}\left(\frac{1}{\widehat{p}} \cdot \frac{\widehat{k}^{\frac{|\mathcal{A}|^2}{\widehat{p}}+1}}{\widehat{k}-1} \cdot (|\Sigma|+1) \cdot \log\left(\frac{|\Sigma|}{\widehat{p}} \cdot \frac{\widehat{k}^{\frac{|\mathcal{A}|^2}{\widehat{p}}+1}}{\widehat{k}-1}\right)\right).$$

5.4. Experiments

Even though Algorithm 2 is theoretically capable of calculating the most probable tree, its runtime is dependent on a variety of properties: The size of the alphabet $|\Sigma|$, the highest rank $\widehat{k}$, the size of the automaton/the number of states $|\mathcal{A}|$ and the probability of the mpt $\widehat{p}$. That raises the question of how performant the algorithm is for different tree automata.

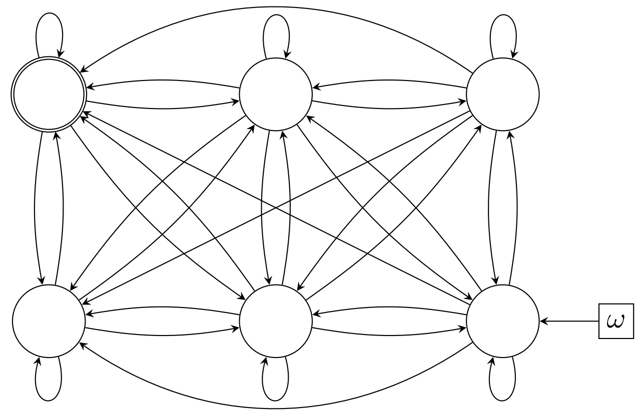
Building pta from corpora in the natural language processing domain, like the Penn Treebank (Marcus, Santorini, and Marcinkiewicz 1993), would require handling possibly quite large alphabets ($|\Sigma|$) and a lot of branching ($\widehat{k}$). Furthermore, depending on the method of constructing a probabilistic tree automaton from a (subset of a) corpus, the number of states ($|\mathcal{A}|$) and transitions can be immense. A simple *read-off pta* (Dietze 2019, Definition 4.3.1), for example, would contain as many states as there are symbols in the ranked alphabet. Lastly, the probabilities of trees recognised by such pta can get rather small which would heavily impact the runtime of the algorithm ($\widehat{p}$).

In this section we aim to examine the effect of these properties on the algorithm's runtime. For this purpose, we first build a set of synthetic pta with which the performance of the mpt algorithm is evaluated afterwards. Additionally, it is determined how good of an approximation the best run is for the most probable tree.

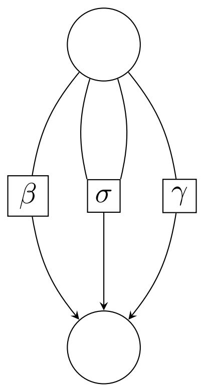
5.4.1. Constructing synthetic automata

In order to ensure that the pta we construct for testing provide a challenge for our algorithm, we employ the construction of de la Higuera and Oncina 2013, Section 5.2 and modify it to allow for branching automata. The goal of using these structures is to ensure that the most probable tree is not trivially small and, more importantly, that there are multiple different runs for most trees and there is a considerable number of trees whose probability is close to the mpt.

The states of these automata are arranged in levels where each level has a fixed number of states (multiplicity). Each state of a level $n \in \mathbb{N}^+$ has transitions coming from all states of level $n' \in [n]$ with $n' \geq n - 1$ and one initial transition with the special end symbol ω leads to a state of the highest level. The topology of such a pta is depicted in Figure 9a. In that drawing, each connection between two nodes stands for multiple transitions as can be seen in Figure 9b. The probability of each transition is chosen randomly while adhering to properness. All states have a root weight of 0 except one state of the first level which has a root weight of 1. The *average rank* of a pta is defined as the arithmetic mean of the ranks of all symbols excluding the start symbol ω. More formally, constructing with a number of levels $l \in \mathbb{N}^+$ and multiplicity $m \in \mathbb{N}^+$ yields a



(a) The topology of a test pta with 3 levels and a multiplicity of 2.



(b) Each connection between two nodes in Figure 9a represents $|\Sigma|$ transitions, where $\Sigma = \{ \sigma^{(2)}, \gamma^{(1)}, \beta^{(1)} \}$.

Figure 9.: A test pta with 3 levels, multiplicity 2, an alphabet size of 3 and average rank of $1\frac{1}{3}$.

pta $\mathcal{A} = (Q, \Sigma \cup \{ \omega \}, \mu, \nu)$ where $\omega \notin \Sigma$,

$$\begin{aligned} Q &= \{ q_{ij} \mid i \in [l], j \in [m] \}, \\ \mu_\sigma &= \bigcup_{i \in [l]} \bigcup_{j \in [m]} \bigcup_{\substack{i' \in [l]: \\ i' \geq i-1}} \bigcup_{j' \in [m]} \{ ((q_{i'j'}, \ldots, q_{i'j'}), q_{ij}) \to p \} \\ &\qquad \forall k \in \mathbb{N}, \sigma \in \Sigma^{(k)} \text{ where } p \in (0,1], \\ \mu_\omega &= \{ ((\varepsilon), q_{lm}) \to p \} \text{ where } p \in (0,1], \text{ and} \\ \nu_{q_{11}} &= 1. \end{aligned}$$

An example construction with 3 levels, multiplicity of 2, alphabet size of 3 and average rank of $1\frac{1}{3}$ (not including special symbol ω) is depicted in Figure 9.

For testing we created random pta with varying parameters. We oriented ourselves by the test set of de la Higuera and Oncina 2013 in order to assess for what kind of automata the calculations are feasible. Though, since branching is introduced, smaller automata had to be used to keep the priority queue from growing to big. Overall, the configurations include level $l \in \{ 2, 3, 4 \}$, multiplicity $m \in \{ 2, 3 \}$, alphabet size $|\Sigma| \in \{ 2, 3, 4, 5 \}$ and average rank $\overline{k} \in \{ 1.0, 1.5, 2.0, 2.5 \}$. The latter may deviate a little ($\overline{k} \pm 0.2$) depending on whether the alphabet size allows for an exact average rank of $\overline{k}$. For each of these configuration 10 random pta have been built, resulting in a test set of size 960. For 302 pta of those, the mpt could not be calculated since the size of the priority queue could not be handled. The calculations exceeded a maximum of 20^7 insertions which led to a queue size that required too much memory.

5.4.2. Results

By evaluating the most probable tree algorithm we are mainly interested in how well the complexity analysis of Section 5.3 holds up against tests on synthetic automata. Of central importance is of course the total number of insertions into the priority queue until the mpt is found because this forms the basis of the theoretical runtime analysis. Additionally, as this algorithm is an adaption of an algorithm for probabilistic string automata, the effect of branching on the number of insertions is of special interest. Lastly, due to having the first algorithm for computing the mpt at hand, it becomes possible to evaluate the quality of the best run approximation.

The mpt algorithm has been implemented in the programming language Rust (Section 5.5 provides some details of the implementation). In addition, we implemented Algorithm 3 that determines the probability of the best run for a tree which has been proposed by Maletti and Satta 2009, Figure 3 and is an adaptation of Knuth 1977, searching for the shortest path in a functional hypergraph. We, in turn, adapted the algorithm to support multiple final states: For each state the run with the highest probability is sought until the best run for all states with non-null root weight have been found. At the end, considering the root weights themselves, the probability of the tree with the best run is returned.

Algorithm 3: best run algorithm (Maletti and Satta 2009, cf. Figure 3)

```
  Input: pta $\mathcal{A} = (Q, \Sigma, \mu, \nu)$
  Output: probability of the best run $\widetilde{p} = \max_{\xi \in T_\Sigma} \max_{\kappa \in R_{\mathcal{A}}(\xi)} \Pr_{\mathcal{A}}(\kappa, \xi)$
1 $\mathcal{E} \leftarrow \varnothing$
2 $\mathcal{R} \leftarrow \{ q \in Q \mid \nu(q) \neq 0 \}$
3 while $\mathcal{R} \not\subseteq \mathcal{E}$ do
4     $\mathcal{D} \leftarrow \{ q \mid \mu_\sigma(q_1 \dots q_k, q) > 0, q \notin \mathcal{E}, q_1, \dots, q_k \in \mathcal{E} \}$
5     foreach $q \in \mathcal{D}$ do
6         $\mathrm{wt}(q) \leftarrow \max_{k \in \mathbb{N}, \sigma \in \Sigma^{(k)}, q_1, \dots, q_k \in \mathcal{E}} \mu_\sigma(q_1 \dots q_k, q) \cdot \prod_{i=1}^{k} \mathrm{wt}(q_i)$
7     $\mathcal{E} \leftarrow \mathcal{E} \cup \{ \arg\max_{q \in \mathcal{D}} \mathrm{wt}(q) \}$
8 return $\max_{q \in \mathcal{E}} \mathrm{wt}(q) \cdot \nu(q)$
```

As mentioned before, of foremost interest is how good the runtime bound holds up in testing. Since the runtime complexity is proportional to the number of insertions into the priority queue, we mainly investigate the influences on it. Hence, we start by looking into whether the previously established bound on the priority queue Q,

$$|\mathsf{Q}| \leq \frac{|\Sigma|}{\widehat{p}} \cdot \frac{\widehat{k}^{\frac{|\mathcal{A}|^2}{\widehat{p}}+1}}{\widehat{k}-1},$$

is close to the real bound: In Figure 10 the total number of insertions is plotted against the inverse probability of the mpt for each of the 658 automata that did not exceed 20^7 insertions and whose mpt could therefore be calculated. This plot shows indeed that, at least empirically, the bound on the number of insertions is far smaller:

$$|\mathsf{Q}| \leq \frac{2}{\widehat{p}}.$$

This is coincides with the results of de la Higuera and Oncina 2013, Section 5.2 and similarly, a theoretical bound of $|\mathsf{Q}| \leq \frac{1}{\widehat{p}^2}$ seems likely. Obtaining results this similar suggests that either the test set is not suitable for testing tree automata, i.e., branching is not incorporated enough or the number of insertions is even for the mpt algorithm mainly dependent on the mpt's probability.

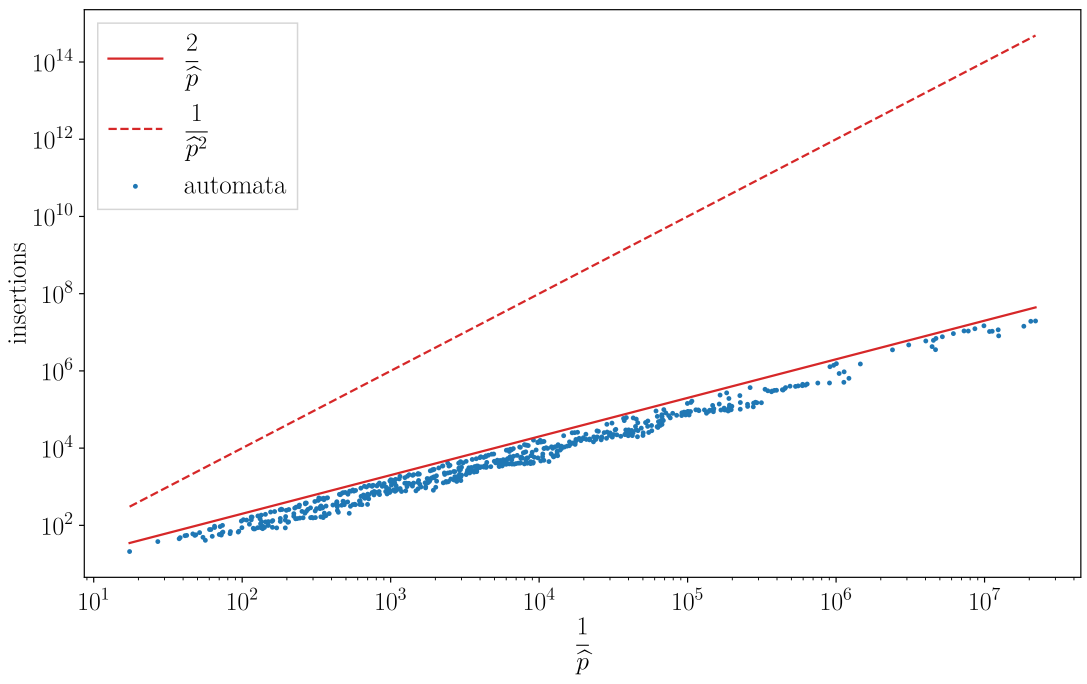


Figure 10.: Number of insertions in the priority queue in relation to the inverse probability $\frac{1}{\widehat{p}}$ of the most probable tree.

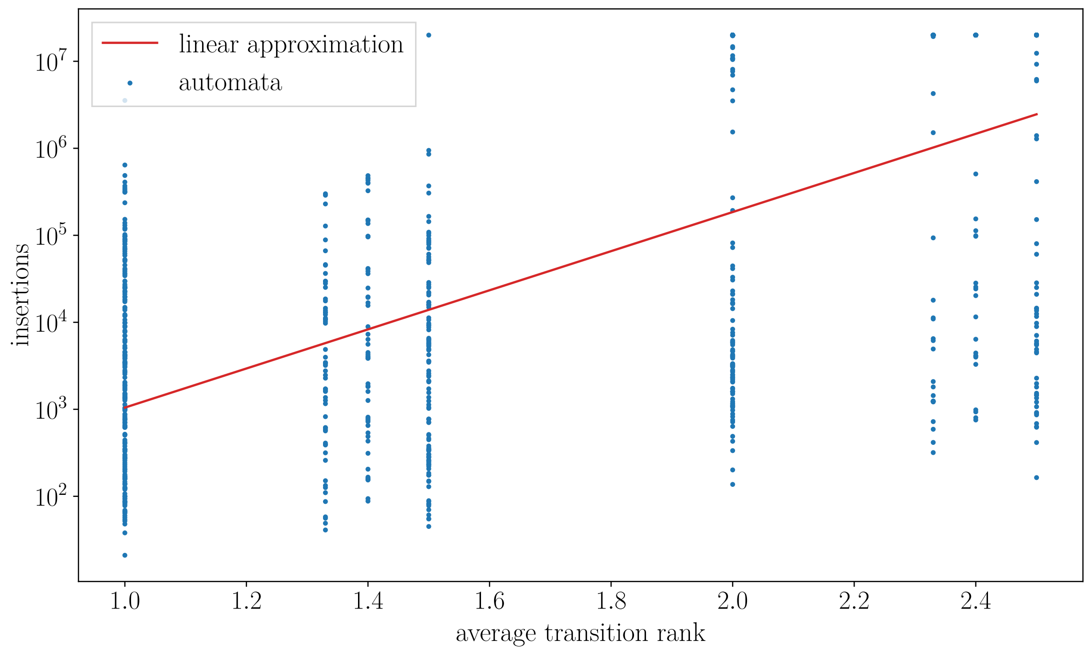


Figure 11.: Number of insertions relative to the average rank of a transition (including 148 (153) instances of average rank 2.0 (~ 2.5) that exceed 20^7 insertions).

Regarding the branching factor: Figure 11 shows that a higher average rank does influence the number of insertions made. Note that in this plot we included those automata that exceeded the maximum of 20^7 insertions. As a consequence, the impact of branching might be even higher than the depiction suggests: Of the 302 automata that exceeded the maximum, 148 had an average rank of 2.0 and 153 of ~ 2.5. To allow for examining automata with higher average ranks, smaller automata would be necessary but we refrained from doing so due to the already relatively small size of the test pta.

Lastly, we investigate the quality of the best run approximation $\widetilde{p}$ (Algorithm 3). Naturally, only the 658 instances that yielded a probability $\widehat{p}$ for the mpt are considered. Of those, 524 ($79,635\%$) pta have trees associated with the best run that coincide with the mpt but in only 282 ($42,857\%$) cases do the probabilities match, i.e., $\widehat{p} = \widetilde{p}$. Figure 12 shows the relative error $\frac{\widehat{p}-\widetilde{p}}{\widehat{p}}$ of using the best parse probability $\widetilde{p}$ as opposed to the mpt probability $\widehat{p}$. This demonstrates that the approximation can be off by quite a lot and suggests that the relative error increases as the probability $\widehat{p}$ decreases. For probabilistic finite state automata (de la Higuera and Oncina 2013, Figure 4) the probabilities never coincide. This discrepancy is due to the smaller automata we used: de la Higuera and Oncina 2013 constructed automata of level 3 to 5, we on the other hand from 2 to 4 and

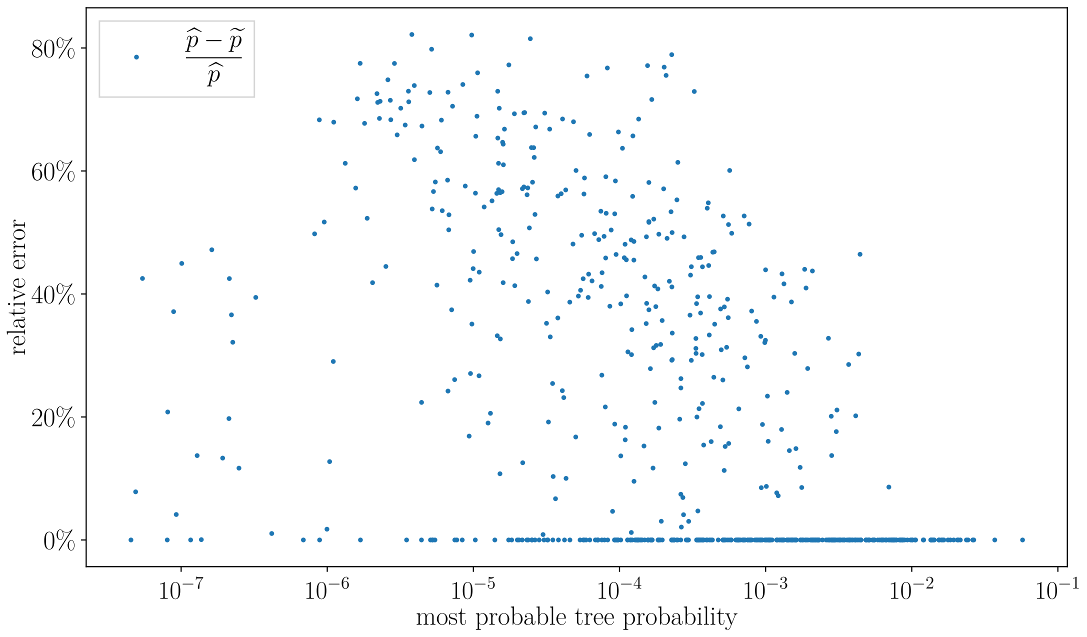


Figure 12.: Relative error of the best run probability $\widetilde{p}$ given the probability $\widehat{p}$ of the mpt.

in a pta of level 2, the mpt often only has a single run. In all instances with a number of levels greater than 2 the probabilities differ.

The applicability of the most probable tree algorithm in practice is highly questionable: The experiments have been conducted on an Intel® Xeon® Silver 4114 CPU with 2.20GHz and even for our simple test automata, some instances took up to an hour to compute (cf. Figure 13). More importantly the maximum size of the priority queue is sometimes as big as $40 - 80\%$ of the total number of insertions (mostly depending on the size of the alphabet). This can result in allocating more than 200GB of memory which makes the whole algorithm decidedly impractical. Though, given that finding the most probable tree is NP-hard, this is to be expected.

5.5. Implementation

The algorithms for computing the most probable tree (Algorithm 2) and determining the best run (Algorithm 3) are implemented in the relatively young and fast programming language Rust[1]. The implementation is available here[2]. Additionally, the corresponding `README` file that gives insight into the usage of the program and how to reproduce

[1]`https://www.rust-lang.org/`
[2]`https://github.com/cirdeirf/mpt/`

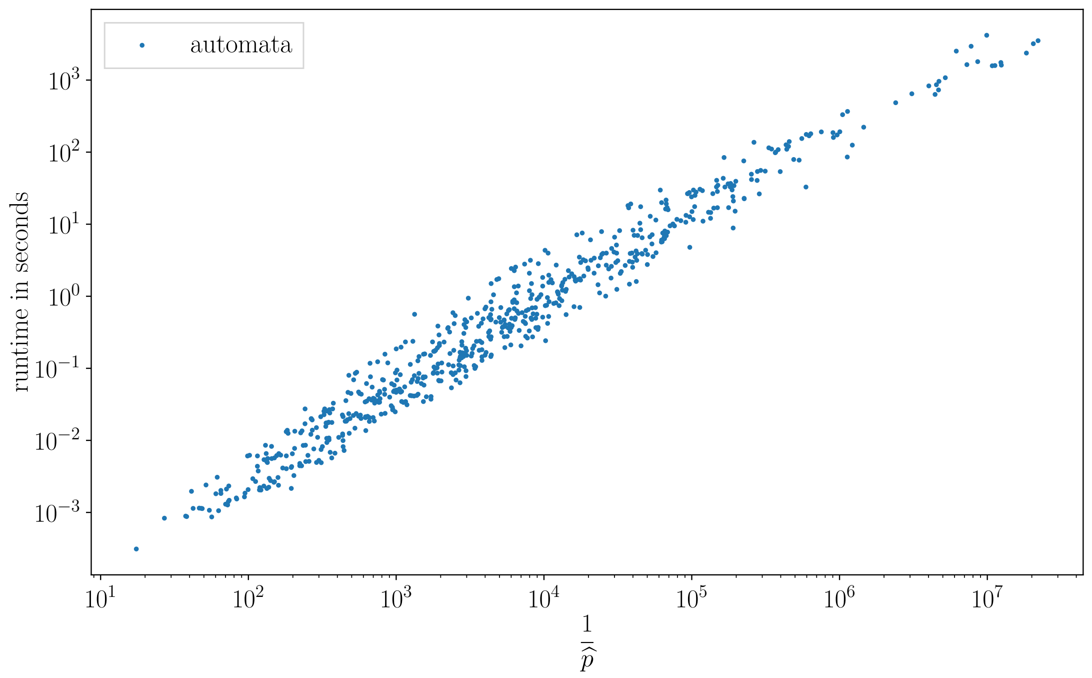


Figure 13.: Runtime in seconds in relation to the inverse probability $\frac{1}{\widehat{p}}$ of the most probable tree.

the results presented in Section 5.4.2 is provided in Appendix A. Since most of the implementation is relatively straightforward, we will only highlight the use of some external libraries and differences to the theoretical work.

First of all, all probabilities are represented internally on a logarithmic scale. For this purpose we make use of a log-domain library[3] for Rust. With the help of this representation we intend to speed up calculation of joint probabilities as we can form sums over logarithmic probabilities instead of products over probabilities and usually, additions can be computed more efficiently than multiplications. Additionally, using the logarithmic presentation increases accuracy for small probabilities due to internal approximation of real numbers by computers.

Furthermore we employ the so-called integeriser library[4]. Thereby, we may represent all states $q \in Q$ and symbols $\sigma \in \Sigma$ internally as integers. In doing so we obtain a better runtime and have to use less space since we do not have to work with strings for example.

The structure that implements a probabilistic tree automaton can be found in `pta/mod.rs`. Among its associated methods are the implementations for the algorithms

[3]`https://github.com/tud-fop/rust-log-domain/`
[4]`https://github.com/tud-fop/rust-integeriser/`

and for computing the probability of a tree. For the most part, these implementations are quite close to their specifications as shown in this work. The most important difference is that even though we implemented a method for calculating the potential probability of a tree, the most probable tree algorithm does not use it. This is supposed to speed up the algorithm because for a pta $\mathcal{A}$ and tree ξ, the bound $\frac{|\mathcal{A}|^2}{\text{height}(\xi)}$ never takes effect for our test set: Our smallest automaton has $|\mathcal{A}|^2 = (\text{level} \cdot \text{multiplicity})^2 = 4^2$ and ξ would have to have a height of at least 16 such that the bound even can be a probability in $[0, 1]$. The bound proves to be useful for analysing the runtime complexity of the algorithm but is not tight enough for practical application.

An important detail regarding the runtime of the mpt algorithm is the memoisation of probabilities: Each tree ξ (defined in `pta/tree.rs`) stores for all states $q \in Q$ the probability to be recognised with this state, i.e., it stores $\sum_{\kappa \in R_{\mathcal{A}}(\xi):\kappa(\varepsilon)=q} \text{Pr}_{\mathcal{A}}(\kappa)$. Through that, the probability of each tree is only calculated once even if it appears as a subtree of another tree. This is necessary as the algorithm would get too slow otherwise. Unfortunately, quite a lot of trees are stored in memory while searching for the mpt and the probabilities for all these have to be memorised as well. Thus, the computation needs a lot of memory which becomes evident in the results of the previous section.

Lastly, although it is possible to implement the operation of extending a tree ξ (Algorithm 2, `line 11`) such that it takes only constant time, our implementation of `extend` has a worse runtime. It looks at each position in a breadth-first manner until a variable is found that can be replaced. Despite this taking in theory up to size(ξ) many steps, the runtime of this method is negligible. The benefit of not remembering all variable positions for each tree is decreased memory consumption which seems to be the critical resource.

6. Conclusion

In this thesis, we have shown an number of results for probabilistic tree automata related to the problem of calculating the most probable tree. Most of these are based on previous results for probabilistic finite automata by de la Higuera and Oncina 2013 and de la Higuera and Oncina 2014. We managed to generalise these to pta.

As a first theorem, we proved trees with a high probability in pta to be shallow (Section 3.1). Apart from the necessary steps to accomodate for tree structure instead of strings, some corrections in the original proof had to be made. Furthermore, we would have liked to get a bound dependent on the size of a tree instead of only its height. Especially, since the inferred bound turned out to be useful during the complexity analysis of our most probable tree algorithm.

Regarding the size of probable trees: We have shown that there are cases where the most probable tree can be of superpolynomial size in the size of the pta (Section 3.2). The original proof for pfa implied exponential size but the current proof does not yield such a result. Still, the proof implies that the problem of finding the most probable tree is not part of the complexity class NP.

In consequence, NP-completeness for mpt should be unattainable. NP-hardness, on the other hand can be shown be a direct polynomial-time reduction from the NP-complete problem 3-SAT which we have provided in Chapter 4. Due to previous NP-hardness results for probabilistic grammars (Sima'an 2002; Casacuberta and de la Higuera 2000), it would have been possible to relate pta to these and proof NP-hardness that way.

Finally, we generalised an algorithm for finding the most probable string in a pfa to probabilistic tree automata (Chapter 5). We additionally provided an implementation for the mpt algorithm and conducted tests on synthetic pta. Even though, the algorithm works, it is highly impractical; primarily due its high memory consumption. Moreover, the theoretical runtime complexity depends on the mpt's probability. As this is the only parameter unknown before executing the algorithm, it would be of interest to infer a bound on this probability given the properties of the probabilistic tree automaton.

Bibliography

Barton, G. Edward, Robert C. Berwick, and Eric S. Ristad (1987). *Computational Complexity and Natural Language*. Cambridge, MA, USA: MIT Press. ISBN: 0262022664.

Casacuberta, Francisco and Colin de la Higuera (2000). “Computational Complexity of Problems on Probabilistic Grammars and Transducers”. In: *Grammatical Inference: Algorithms and Applications, 5th International Colloquium, ICGI 2000, Lisbon, Portugal, September 11-13, 2000, Proceedings*, pp. 15–24. DOI: `10.1007/978-3-540-45257-7\_2`.

Chomsky, Noam (1956). “Three models for the description of language”. In: *IRE Trans. Information Theory* 2.3, pp. 113–124. DOI: `10.1109/TIT.1956.1056813`.

Cognetta, Marco, Yo-Sub Han, and Soon Chan Kwon (2018). “Incremental Computation of Infix Probabilities for Probabilistic Finite Automata”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 2732–2741. URL: `https://aclanthology.info/papers/D18-1293/d18-1293`.

Dietze, Toni (2019). “A Formal View on Training of Weighted Tree Automata by Likelihood-Driven State Splitting and Merging”. Dissertation. TU Dresden.

Engelfriet, Joost (2015). *Tree Automata and Tree Grammars*. Vol. abs/1510.02036. arXiv: `1510.02036`. URL: `http://arxiv.org/abs/1510.02036`.

Fülöp, Zoltán and Heiko Vogler (2009). “Weighted Tree Automata and Tree Transducers”. In: *Handbook of Weighted Automata*. Ed. by Manfred Droste, Werner Kuich, and Heiko Vogler. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 313–403. ISBN: 978-3-642-01492-5. DOI: `10.1007/978-3-642-01492-5_9`. URL: `https://doi.org/10.1007/978-3-642-01492-5_9`.

Gécseg, Ferenc and Magnus Steinby (1984). *Tree Automata*. Akadéniai Kiadó, Budapest, Hungary. ISBN: 963-05-3170-4.

Heath-Brown, D.R. and H. Iwaniec (Dec. 1979). “On the difference between consecutive primes”. In: *Inventiones mathematicae* 55.1, pp. 49–69. ISSN: 1432-1297. DOI: `10.1007/BF02139702`. URL: `https://doi.org/10.1007/BF02139702`.

de la Higuera, Colin (1997). “Characteristic Sets for Polynomial Grammatical Inference”. In: *Machine Learning* 27.2, pp. 125–138. DOI: `10.1023/A:1007353007695`. URL: `https://doi.org/10.1023/A:1007353007695`.

de la Higuera, Colin and José Oncina (2013). “Computing the Most Probable String with a Probabilistic Finite State Machine”. In: *Proceedings of the 11th International Conference on Finite State Methods and Natural Language Processing, FSMNLP 2013, St. Andrews, Scotland, UK, July 15-17, 2013*, pp. 1–8. URL: `http://aclweb.org/anthology/W/W13/W13-1801.pdf`.

– (2014). “The most probable string: an algorithmic study”. In: *J. Log. Comput.* 24.2, pp. 311–330. DOI: `10.1093/logcom/exs049`. URL: `https://doi.org/10.1093/logcom/exs049`.

Knight, Kevin and Jonathan May (2009). “Applications of Weighted Automata in Natural Language Processing”. In: *Handbook of Weighted Automata*. Ed. by Manfred Droste, Werner Kuich, and Heiko Vogler. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 571–596. ISBN: 978-3-642-01492-5. DOI: `10.1007/978-3-642-01492-5_14`. URL: `https://doi.org/10.1007/978-3-642-01492-5_14`.

Knuth, Donald E. (1977). “A Generalization of Dijkstra’s Algorithm”. In: *Inf. Process. Lett.* 6.1, pp. 1–5. DOI: `10.1016/0020-0190(77)90002-3`. URL: `https://doi.org/10.1016/0020-0190(77)90002-3`.

Koller, Alexander and Marco Kuhlmann (2011). “A Generalized View on Parsing and Translation”. In: *Proceedings of the 12th International Conference on Parsing Technologies, IWPT 2011, October 5-7, 2011, Dublin City University, Dubin, Ireland*, pp. 2–13. URL: `http://www.aclweb.org/anthology/W11-2902`.

Maletti, Andreas and Giorgio Satta (2009). “Parsing Algorithms based on Tree Automata”. In: *Proceedings of the 11th International Workshop on Parsing Technologies (IWPT-2009), 7-9 October 2009, Paris, France*, pp. 1–12. URL: `http://www.aclweb.org/anthology/W09-3801`.

Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz (1993). “Building a Large Annotated Corpus of English: The Penn Treebank”. In: *Computational Linguistics* 19.2, pp. 313–330.

May, Jonathan and Kevin Knight (2006). “Tiburon: A Weighted Tree Automata Toolkit”. In: *Implementation and Application of Automata, 11th International Conference, CIAA 2006, Taipei, Taiwan, August 21-23, 2006, Proceedings*, pp. 102–113. DOI: `10.1007/11812128\_11`. URL: `https://doi.org/10.1007/11812128%5C_11`.

Sima’an, Khalil (2002). “Computational Complexity of Probabilistic Disambiguation”. In: *Grammars* 5.2, pp. 125–151. DOI: `10.1023/A:1016340700671`. URL: `https://doi.org/10.1023/A:1016340700671`.

Williams, J. W. J. (1964). “Algorithm 232: Heapsort”. In: *Communications of the ACM* 7.6, pp. 347–348.

List of Figures

A. Readme file

In the following we append the contents of the `README.md` file the is included in the implementation.

Most Probable Tree Algorithm

This is a Rust-based implementation of the most probable tree (mpt) algorithm introduced in "The Problem of Computing the Most Probable Tree of a Probabilistic Tree Automaton" and the best run algorithm presented in "Parsing Algorithms based on Tree Automata" by Andreas Maletti and Giorgio Satta, 2009. More information about the mpt algorithm and the classes can be found in the thesis itself, in the comments of the source code and in the rustdoc documentation which can be generated and opened by:

```
cargo doc --document-private-items --open
```

This project is licensed unter the terms of the GNU General Public License v3.0.

Setup

This program requires Rust and Cargo (Rust build tool and package manager). Both are available from the Rust programming language website (`https://www.rust-lang.org/`). For even simpler use, we provide a binary (`mpt`).

Example

The example pta from the thesis is available in
`experiments/pta/manually_constructed/example.pta`:

```
root: q0 # 0.9
root: q1 # 0.1
transition: q1 -> α() # 0.1
transition: q2 -> α() # 0.5
transition: q2 -> β() # 0.5
transition: q1 -> γ(q1) # 0.5
```

```
transition: q1 -> γ(q2) # 0.3
transition: q1 -> σ(q1, q2) # 0.1
transition: q0 -> σ(q1, q2) # 1.0
```

Every root weight/transition not mentioned is assumed to have a probability of zero. For states and symbols all strings are allowed that do not contain any of the following characters: `'"'`, `' '`, `'-'`, `'>'`, `'→'`, `','`, `';'`, `'('`, `')'`, `'['`, `']'`, `'%'`.

The most probable tree for this example pta can be calculated by calling

```
cargo run
```

which should print

```
mpt:               σ( γ( α ), β )
probability:       0.09100000000000004
insertions:        13
time:              1.226144ms
```

to the standard output.

Similarly, the best run for the example is computed by calling

```
cargo run -- --best-parse
```

which yields

```
best parse:        σ( γ( β ), β )
probability:       0.06749999999999999
time:              227.69µs
```

Ambiguity

Note that the implementation does not choose symbols $\sigma \in \Sigma$ in the same order every time it is called. Therefore it can happen that fewer insertions are necessary when a new current best complete tree has been found that prevents the insertion of another tree, e.g., for the example: We extend tree $\xi = \sigma(\gamma(x_1), \beta)$ to a tree $\xi' = \sigma(\gamma(\sigma(x_2, x_3)), \beta)$ with probability 0.02275 and afterwards, we extend ξ to $\sigma(\gamma(\alpha), \beta)$ with the higher probability of 0.091. If we would have extended with α first, we would not have created ξ'. Additionally, there might be multiple mpt. For the example, the following output is therefore valid as well:

```
mpt:                σ( γ( α ), α )
probability:        0.09100000000000004
insertions:         12
time:               1.887662ms
```

Experiments

The experiments have been conducted on a set of synthetic automata. How these are constructed is described in Section 5.4.1. A new test set with the parameters from the thesis (level, multiplicity, alphabet size) can be generated by invoking:

```
cargo run -- --generate
```

The specific set of synthetic pta we used can be found in `experiments/pta/test1/`. In order to automatically calculate the most probable tree for all these automata, one has to call the following command:

```
cargo run --release -- --experiments
```

Analogously, for the best run with the `--best-parse` flag. Note that `--release` is not necessary but improves the runtime (the binary is compiled with this flag).

The experiments were conducted on an Intel® Xeon® Silver 4114 CPU with 2.20GHz and the ouput is available in `experiments/test1_mpt.log` and `experiments/test1_best_parse.log`.

Help

The following command line arguments are available:

```
> cargo run -- --help

mpt 0.1
Pius Meinert <yrr+work@pm.me>
Most probable tree and best parse algorithms for probabilistic
tree automata (pta). Implementation for my master's thesis:
"The Problem of Computing the Most Probable Tree of a
Probabilistic Tree Automaton". By default, the most probable
tree algorithm is executed.

```

```
USAGE:
    mpt [FLAGS] <INPUT>

FLAGS:
    -b, --best-parse     Calculate the tree with the best run
                         (cf. Figure 3, Maletti and Satta, 2009)
    -e, --experiments    Calculate the most probable tree/best
                         run for all pta in the test set:
                         experiments/pta/test1/.
    -g, --generate       Generate a number of synthetic
                         automata with the parameters (level,
                         multiplicity, alphabet size) as used
                         during testing for the thesis. They
                         are saved in: experiments/pta/.
    -h, --help           Prints help information
    -V, --version        Prints version information
    -v, --verbose        Set level of verbosity:
                         v: print the pta (root weight mapping
                            and transitions),
                         vv: + output current best tree
                             (only for mpt),
                         vvv: + print variables after each
                              iteration of the while loop
                              (only mpt).

ARGS:
    <INPUT>    Set the input file to use [default:
               experiments/pta/manually_constructed/example.pta]
```