Probabilistic Arithmetic Automata and Their Applications

Tobias Marschall, Inke Herms, Hans-Michael Kaltenbach, and Sven Rahmann

Abstract—We present a comprehensive review on *probabilistic arithmetic automata (PAAs)*, a general model to describe chains of operations whose operands depend on chance, along with two algorithms to numerically compute the distribution of the results of such probabilistic calculations. PAAs provide a unifying framework to approach many problems arising in computational biology and elsewhere. We present five different applications, namely 1) pattern matching statistics on random texts, including the computation of the distribution of occurrence counts, waiting times, and clump sizes under hidden Markov background models; 2) exact analysis of window-based pattern matching algorithms; 3) sensitivity of filtration seeds used to detect candidate sequence alignments; 4) length and mass statistics of peptide fragments resulting from enzymatic cleavage reactions; and 5) read length statistics of 454 and IonTorrent sequencing reads. The diversity of these applications indicates the flexibility and unifying character of the presented framework. While the construction of a PAA depends on the particular application, we single out a frequently applicable construction method: We introduce *deterministic arithmetic automata (DAAs)* to model deterministic calculations on sequences, and demonstrate how to construct a PAA from a given DAA and a finite-memory random text model. This procedure is used for all five discussed applications and greatly simplifies the construction of PAAs. Implementations are available as part of the MoSDi package. Its application programming interface facilitates the rapid development of new applications based on the PAA framework.

Index Terms—Probabilistic automaton, text model, hidden Markov model, pattern matching, statistics, clump, string algorithm, analysis of algorithms, alignment seed, peptide mass fingerprinting, DNA sequencing, dynamic programming

1 Introduction

In many applications in computational biology and elsewhere, processes can be modeled as chains of operations working on operands that are drawn probabilistically. One is then often interested in the *distribution of results* of such computations under suitable null models. Furthermore, the *distribution of the waiting time* for reaching a particular value can be of interest. By computing the distribution of a random variable X, we mean tabulating the values $\mathbb{P}(X = x)$ for all x of interest.

The goal of this paper is to establish a general formal framework, referred to as *probabilistic arithmetic automata (PAAs)*, to directly model such systems and answer the posed questions. We emphasize that we are not interested in simulation studies or approximations to these distributions, but in an exact computation up to machine accuracy. Problems from diverse applications, especially from computational biology, can be conveniently solved with PAAs in a unified way, whereas they are so far treated heterogeneously in the literature. As a result of this unification, the same software implementation of the core algorithms can be used to solve a broad range of problems.

- *T. Marschall is with the Life Sciences Group, Centrum Wiskunde & Informatica (CWI), Science Park 123, 1098 XG Amsterdam, The Netherlands. E-mail: T.Marschall@cwi.nl.*
- *I. Herms is with the Genome Informatics Group, Faculty of Technology, Bielefeld University, 33594 Bielefeld, Germany. E-mail: Inke.Herms@gmx.de.*
- *H.-M. Kaltenbach is with the Computational Systems Biology Group, Department of Biosystems Science and Engineering, Swiss Federal Institute of Technology (ETH), Mattenstrasse 26, CH-4058 Basel, Switzerland. E-mail: hans-michael.kaltenbach@bsse.ethz.ch.*
- *S. Rahmann is with the Genome Informatics, Faculty of Medicine, Institute of Human Genetics, University of Duisburg-Essen, Hufelandstr. 55, 45122 Essen, Germany. E-mail: Sven.Rahmann@uni-due.de.*

Manuscript received 5 Apr. 2011; revised 29 June 2012; accepted 14 July 2012; published online 1 Aug. 2012.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBB-2011-04-0089. Digital Object Identifier no. 10.1109/TCBB.2012.109.

1.1 Organization of the Paper

In the first part of this paper we set up the PAA framework. Specifically, we formally introduce PAAs in Section 2 and give generic algorithms in Section 3. We consider waiting time problems on PAAs in Section 4. In Section 5, *deterministic arithmetic automata* and *finite-memory text models* are defined and shown to be a convenient means of specifying a PAA.

Having the framework in place, we explore several application domains. Section 6 deals with the statistics of patterns on random texts. In Section 7, PAAs are employed for the analysis of window-based pattern matching algorithms. We cover the field of alignment seed statistics in Section 8. Then, in Section 9, we apply PAAs to mass statistics of fragments resulting from enzymatic digestion. The optimization of read lengths in 454- and IonTorrent sequencing is discussed in Section 10.

All methods and applications are implemented in Java as part of the MoSDi software (Appendix J), available under the GNU General Public License (GPL) from http://mosdi.googlecode.com. MoSDi provides an application programming interface (API) to the generic, application-independent algorithms. All experimental results were produced with MoSDi on an Intel Core i7-2600 with 3.4 GHz and 16 GB RAM.

The relations between this paper and preliminary versions published in conference proceedings are as follows: Sections 2, 3 and parts of Section 6 are based on [1]. Section 5 is based on [2]. Section 6.4 contains material

from [3]. Sections 7, 8, and 9 are based on [2], [4], and [5], respectively. An extended version of [2] appeared as [6].

1.2 Related Work

The idea of combining pattern-recognizing automata with a random model, especially a Markov chain, is not a new one and has been thoroughly explored over the last decade. Early work on discrete Markov Additive Processes (MAPs) [7] does not make an explicit connection to pattern matching; more recent works by Lladser et al. [8] and Nuel [9] call the process *Markov chain embedding*. Most of this work can also be rephrased with weighted finite-state transducers [10], for which a thoroughly developed algebraic theory exists. Finite-memory random text models are also called probability transducers in [11]. The algorithmic chain proposed by Nicodème et al. [12] uses symbolic generating functions on automata, which can be either analyzed asymptotically or evaluated with Taylor expansions to compute concrete values. Mak and Benson [13] symbolically compute parameterized probabilities for a concrete application (see Section 8).

Thus, the PAA framework is only one way of formalizing computations with random values on random sequences, yet in our view an intuitive and easy one. The applications in this paper (except Section 10) have been previously discussed in the literature, but never been identified as instances of the same abstract scheme. We give further references in each section.

1.3 Notation and Conventions

Natural numbers without (with) zero are denoted $\mathbb{N}$ ($\mathbb{N}_0$). As usual, Σ is a finite alphabet and Σ^* the corresponding set of finite strings. Indices are zero-based, i.e., $s = s[0] \dots s[|s|-1]$ for $s \in \Sigma^*$. Substrings, prefixes, and suffixes are written $s[i \dots j] := s[i] \dots s[j]$, $s[..i] := s[0] \dots s[i]$, and $s[i..] := s[i] \dots s[|s|-1]$, respectively. All stochastic processes are discrete. Therefore, appropriate probability spaces can always be constructed; we do not clutter notation by stating them explicitly. By $\mathbb{P}$, we refer to a probability measure; $\mathcal{L}(X)$ denotes the distribution ("law") of the random variable X. Iverson brackets are written $[\![\cdot]\!]$, i.e., $[\![A]\!] = 1$ if the statement A is true and $[\![A]\!] = 0$ otherwise.

2 Probabilistic Arithmetic Automata

We define *probabilistic arithmetic automata* in order to formalize chains of operations with probabilistic operands.

Definition 2.1 (Probabilistic Arithmetic Automaton). *A probabilistic arithmetic automaton $\mathcal{P}$ is an 8-tuple*

$$\mathcal{P} = \left(\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu = (\mu_q)_{q\in\mathcal{Q}}, \theta = (\theta_q)_{q\in\mathcal{Q}}\right),$$

where

- *$\mathcal{Q}$ is a finite set of states,*
- *$q_0 \in \mathcal{Q}$ is called* start state,
- *$T : \mathcal{Q} \times \mathcal{Q} \to [0,1]$ is a transition function with $\sum_{q'\in\mathcal{Q}} T(q,q') = 1$ for all $q \in \mathcal{Q}$, i.e., $(T(q,q'))_{q,q'\in\mathcal{Q}}$ is a stochastic matrix,*
- *$\mathcal{V}$ is a set called* value set,
- *$v_0 \in \mathcal{V}$ is called* start value,
- *$\mathcal{E}$ is a finite set called* emission set,
- *each $\mu_q : \mathcal{E} \to [0,1]$ is an emission distribution associated with state q,*
- *each $\theta_q : \mathcal{V} \times \mathcal{E} \to \mathcal{V}$ is an operation associated with state q.*

We attach the following semantics: At first, the automaton is in its start state q_0, as for a classical deterministic finite automaton (DFA). In a DFA, the transitions are triggered by input symbols. In a PAA, the transitions are purely probabilistic; $T(q,q')$ gives the chance of moving from state q to state q'. Note that the tuple $(\mathcal{Q}, T, \delta_{q_0})$ defines a Markov chain on state set $\mathcal{Q}$ with transition matrix T, where the initial distribution δ_{q_0} is the Dirac distribution assigning probability 1 to $\{q_0\}$.

While moving from state to state, a PAA performs a chain of calculations on the value set $\mathcal{V}$. It starts with value v_0. Whenever a state transition is made, the entered state, say state q, generates an emission from $\mathcal{E}$ according to the distribution μ_q. The current value and this emission are then subject to the operation θ_q, resulting in the next value from the value set $\mathcal{V}$. Notice that the Markov chain $(\mathcal{Q}, T, \delta_{q_0})$, together with the emission set $\mathcal{E}$ and the distributions $\mu = (\mu_q)_{q\in\mathcal{Q}}$, defines a hidden Markov model (HMM). With HMMs, however, we usually focus on the sequence of emissions, whereas here we are interested in the value resulting from a chain of operations on these emissions.

By introducing PAAs, we emphasize that many applications can naturally be modeled as a chain of operations whose result is of interest. When compared to Markov chains, PAAs do not offer an increase in expressive power. In fact, from a theoretical point of view, every PAA might be seen as a Markov chain on the state space $\mathcal{Q} \times \mathcal{V}$. Thus, we advocate PAAs not because of their expressive power but for their merits as a modeling technique. The framework lends itself to many applications and often allows simple and intuitive problem formulations. The following definition formalizes the introduced semantics.

Definition 2.2 (Stochastic processes induced by a PAA). *For a given PAA $\mathcal{P} = (\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu, \theta)$, we denote its state process by $(Q_t^{\mathcal{P}})_{t\in\mathbb{N}_0}$. It is defined to be a Markov chain with $Q_0^{\mathcal{P}} \equiv q_0$ and*

$$\begin{aligned}&\mathbb{P}\left(Q_{t+1}^{\mathcal{P}} = q_{t+1} \mid Q_t^{\mathcal{P}} = q_t, \dots, Q_0^{\mathcal{P}} = q_0\right)\\ &\quad = \mathbb{P}\left(Q_{t+1}^{\mathcal{P}} = q_{t+1} \mid Q_t^{\mathcal{P}} = q_t\right) := T(q_t, q_{t+1}),\end{aligned} \tag{1}$$

for all $q_0, \dots q_{t+1} \in \mathcal{Q}$. Further, we define the emission process $(E_t^{\mathcal{P}})_{t\in\mathbb{N}_0}$ through

$$\begin{aligned}&\mathbb{P}\left(E_t^{\mathcal{P}} = e \mid Q_0^{\mathcal{P}} = q_0, \dots, Q_t^{\mathcal{P}} = q_t, E_0^{\mathcal{P}} = e_0, \dots, E_{t-1}^{\mathcal{P}} = e_{t-1}\right)\\ &\quad = \mathbb{P}\left(E_t^{\mathcal{P}} = e \mid Q_t^{\mathcal{P}} = q\right) := \mu_q(e),\end{aligned} \tag{2}$$

i.e., the current emission depends solely on the current state. We use $(Q_t^{\mathcal{P}})_{t\in\mathbb{N}_0}$ and $(E_t^{\mathcal{P}})_{t\in\mathbb{N}_0}$ to define the process of values $(V_t^{\mathcal{P}})_{t\in\mathbb{N}_0}$ resulting from the performed operations:

$$V_0^{\mathcal{P}} :\equiv v_0 \quad \text{and} \quad V_t^{\mathcal{P}} := \theta_{Q_t^{\mathcal{P}}}\left(V_{t-1}^{\mathcal{P}}, E_t^{\mathcal{P}}\right). \tag{3}$$

If the considered PAA is clear from the context, we omit the superscript $\mathcal{P}$ and write $(Q_t)_{t\in\mathbb{N}_0}$, $(V_t)_{t\in\mathbb{N}_0}$, and $(E_t)_{t\in\mathbb{N}_0}$, respectively.

3 State-Value Distribution of PAAs

We describe two algorithms to compute the distribution of resulting values. In other words, we seek to calculate the distribution $\mathcal{L}(V_n)$ of the random variable V_n for a given n. The idea is to compute the joint distribution $\mathcal{L}(Q_n, V_n)$ and then to derive the sought distribution by marginalization:

$$\mathbb{P}(V_n = v) = \sum_{q \in \mathcal{Q}} \mathbb{P}(Q_n = q, V_n = v)\ . \tag{4}$$

For the sake of a shorter notation, we define $f_t(q, v) := \mathbb{P}(Q_t = q, V_t = v)$ for $t \in \mathbb{N}_0$, $q \in \mathcal{Q}$, $v \in \mathcal{V}$.

A slight complication arises when $\mathcal{V}$ is infinite. However, for each t, the range of V_t is finite, as it is a function of the states and emissions up to time t, and these are finite sets. We define $\mathcal{V}_t := \operatorname{range} V_t$ and $\vartheta_n := \max_{0 \le t \le n} |\mathcal{V}_t|$. Clearly, $\vartheta_n \le (|\mathcal{Q}| \cdot |\mathcal{E}|)^n$. Therefore all actual computations are on finite sets. In many applications, ϑ_n grows only polynomially (even linearly) with n. In the following, we shall understand $\mathcal{V}$ as the appropriate union of $\mathcal{V}_t$ sets. Running times of algorithms are given in terms of ϑ_n.

3.1 Basic Recurrence

A basic recurrence to compute the distribution $f_n = \mathcal{L}(Q_n, V_n)$ follows from Definitions 2.1 and 2.2.

Lemma 3.1 (State-Value Recurrence). *For a given PAA, the state-value distribution can be computed by*

$$f_0(q, v) = \begin{cases} 1 & \text{if } q = q_0 \text{ and } v = v_0\ , \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

and

$$f_{t+1}(q, v) = \sum_{q' \in \mathcal{Q}} \sum_{(v', e) \in \theta_{q'}^{-1}(v)} f_t(q', v') \cdot T(q', q) \cdot \mu_q(e), \tag{6}$$

where $\theta_q^{-1}(v)$ denotes the inverse image set of v under θ_q.

Proof. See Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2012.109. □

We start with the distribution f_0 and calculate the subsequent distributions by applying (6) until we obtain the desired f_n.

Lemma 3.2. *Given a PAA $(\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu, \theta)$, the distribution of values $\mathcal{L}(V_n)$ can be computed in $\mathcal{O}(n \cdot |\mathcal{Q}|^2 \cdot \vartheta_n \cdot |\mathcal{E}|)$ time and $\mathcal{O}(|\mathcal{Q}| \cdot \vartheta_n)$ space.*

Proof. See Appendix, available in the online supplemental material. □

3.2 Doubling Technique

If n is large, executing the basic recurrence may be slow. We present an alternative that may be favorable for large n. We consider the conditional probability

$$U^{(t)}(q_1, q_2, v_1, v_2) := \mathbb{P}\big(Q_{t_0+t} = q_2, V_{t_0+t} = v_2 \bigm| Q_{t_0} = q_1, V_{t_0} = v_1\big).$$

Note that $U^{(t)}$ does not depend on t_0, because transition as well as emission probabilities do not change over "time" (a property called *homogeneity*). Once $U^{(n)}$ is known, we can simply read off the desired distribution $\mathcal{L}(Q_n, V_n)$:

$$\mathbb{P}(Q_n = q, V_n = v) = U^{(n)}(q_0, q, v_0, v).$$

The following lemma shows how $U^{(t)}$ is computed.

Lemma 3.3. *Let $(\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu, \theta)$ be a PAA and $(Q_t)_{t \in \mathbb{N}_0}$ and $(V_t)_{t \in \mathbb{N}_0}$ its state and value process. Then,*

$$U^{(1)}(q_1, q_2, v_1, v_2) = T(q_1, q_2) \cdot \sum_{\substack{e \in \mathcal{E}: \\ \theta_{q_2}(v_1, e) = v_2}} \mu_{q_2}(e), \tag{7}$$

and, for all $t_1 \in \mathbb{N}_0$ and $t_2 \in \mathbb{N}_0$,

$$U^{(t_1+t_2)}(q_1, q_2, v_1, v_2) = \sum_{q' \in \mathcal{Q}} \sum_{v' \in \mathcal{V}} U^{(t_1)}(q_1, q', v_1, v') \cdot U^{(t_2)}(q', q_2, v', v_2). \tag{8}$$

Thus, the distribution of values $\mathcal{L}(V_n)$ can be computed in $\mathcal{O}(\log n \cdot |\mathcal{Q}|^3 \cdot \vartheta_n^3)$ time and $\mathcal{O}(|\mathcal{Q}|^2 \cdot \vartheta_n^2)$ space.

Proof. See Appendix, available in the online supplemental material. □

We note that the doubling technique is asymptotically faster if ϑ_n is $o(\sqrt{n/\log n})$ and $\mathcal{Q}$ and $\mathcal{E}$ are fixed.

4 Waiting Times

Besides calculating the distribution of values after a fixed number of steps, we can ask for the distribution of the number of steps needed to reach a certain value or a certain state. Such *waiting time* problems play an important role in many applications. We discuss applications in Sections 6 and 10. A classical treatment of waiting time problems is given in [14]. Applications to occurrence problems in texts are reviewed in [15].

Definition 4.1 (Waiting Time for a Value). *The waiting time for a set of target values $\mathcal{T} \subset \mathcal{V}$ is a random variable defined as $W_{\mathcal{T}} := \min\{t \in \mathbb{N}_0 \mid V_t \in \mathcal{T}\}$ if this set is not empty, and defined as infinity otherwise.*

While $\mathbb{P}(W_{\mathcal{T}} \ge t)$ may be nonzero for all $t \in \mathbb{N}$, we are frequently only interested in the distribution up to a fixed time n. Then, of course, the exact values of $\mathcal{L}(W_{\mathcal{T}})(t) = \mathbb{P}(W_{\mathcal{T}} = t)$ are unknown for $t > n$, but their total probability $\mathbb{P}(W_{\mathcal{T}} > n)$ is known, and n is typically chosen such that this total probability remains below a desired threshold.

Lemma 4.2. *Let $(\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu, \theta)$ be a PAA and $\mathcal{T} \subset \mathcal{V}$. Then, the probabilities $\mathcal{L}(W_{\mathcal{T}})(0), \ldots, \mathcal{L}(W_{\mathcal{T}})(n)$ can be computed in $\mathcal{O}(n \cdot |\mathcal{Q}|^2 \cdot |\mathcal{E}| \cdot (\vartheta_n - |\mathcal{T}|))$ time and $\mathcal{O}(|\mathcal{Q}| \cdot (\vartheta_n - |\mathcal{T}|))$ space. Alternatively, this can be done using $\mathcal{O}(\log n \cdot |\mathcal{Q}|^3 \cdot (\vartheta_n - |\mathcal{T}|)^3)$ time and $\mathcal{O}(|\mathcal{Q}|^2 \cdot (\vartheta_n - |\mathcal{T}|)^2)$ space.*

Proof. We construct a modified PAA by defining a new value set $\mathcal{V}' := (\mathcal{V} \setminus \mathcal{T}) \cup \{\bullet, \circ\}$, assuming (without loss of generality) that $\bullet, \circ \notin \mathcal{V}$, and new operations

$$\theta'_q(v, e) := \begin{cases} \theta_q(v, e) & \text{if } v \notin \{\bullet, \circ\} \text{ and } \theta_q(v, e) \notin \mathcal{T}, \\ \bullet & \text{if } v \notin \{\bullet, \circ\} \text{ and } \theta_q(v, e) \in \mathcal{T}, \\ \circ & \text{if } v \in \{\bullet, \circ\}, \end{cases}$$

for all $q \in \mathcal{Q}$. Let V'_t be the modified value process. Using the modified PAA, the probability of waiting time t can be expressed as

$$\mathbb{P}(W_{\mathcal{T}} = t) = \mathbb{P}(V'_t = \bullet).$$

Time and space follow from Lemmas 3.2 and 3.3. □

Besides waiting for a set of values, we may also wait for a set of states. This problem solely concerns the Markov chain $(\mathcal{Q}, T, \boldsymbol{\delta}_{q_0})$ and is well studied in the literature [15], [16]. We briefly restate the construction.

Definition 4.3 (Waiting Time for a State). *The waiting time for a set of target states $\mathcal{S} \subset \mathcal{Q}$ is a random variable defined as $W_{\mathcal{S}} := \min\{t \in \mathbb{N}_0 \mid Q_t \in \mathcal{S}\}$ if this set is not empty and defined as infinity otherwise.*

Lemma 4.4. *Let $(\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu, \theta)$ be a PAA, $\alpha : \mathcal{Q} \to [0,1]$ be a probability distribution on $\mathcal{Q}$, and $\mathcal{S} \subset \mathcal{Q}$ be a set of target states. Consider the Markov chain $(\mathcal{Q}, T, \alpha)$, let $(Q'_t)_{t \in \mathbb{N}_0}$ be its state process, and let $W'_{\mathcal{S}} := \min\{t \in \mathbb{N}_0 \mid Q'_t \in \mathcal{S}\}$ be the waiting time for states $\mathcal{S}$. Then, $\mathcal{L}(W'_{\mathcal{S}})(0), \ldots, \mathcal{L}(W'_{\mathcal{S}})(n)$ can be computed in $\mathcal{O}(n \cdot |\mathcal{Q}|^2)$ time and $\mathcal{O}(|\mathcal{Q}|)$ space, or in $\mathcal{O}(\log n \cdot |\mathcal{Q}|^3)$ time and $\mathcal{O}(|\mathcal{Q}|^2)$ space using the doubling technique. If $\alpha = \boldsymbol{\delta}_{q_0}$, then $W_{\mathcal{S}} = W'_{\mathcal{S}}$.*

Proof. See Appendix, available in the online supplemental material. □

When $\alpha = \boldsymbol{\delta}_{q_0}$, the lemma yields the waiting time for the first event of reaching one of the states in $\mathcal{S}$. We now consider the waiting time of a return event

$$W^{t_0}_{\mathcal{S}} := \min\{t \in \mathbb{N} \mid Q_{t_0+t} \in \mathcal{S}\}.$$

If the Markov chain is aperiodic and irreducible, it has a unique stationary state distribution, against which the PAA state distribution converges exponentially fast. We can then use Lemma 4.4 to compute

$$\lim_{t \to \infty} \mathbb{P}\big(W^t_{\mathcal{S}} = t' \;\big|\; Q_t \in \mathcal{S}\big) \quad \text{for each } t' \in \mathbb{N},$$

by choosing α in Lemma 4.4 as the stationary distribution restricted to $\mathcal{S}$.

5 PAAs Based on Random Sequences

We now construct PAAs modeling the *deterministic* processing of *random* sequences, i.e., we ask for the distribution of resulting values when a deterministic computation is applied to random strings. An example is given by any pattern matching algorithm that computes the number of matches in a random sequence. Therefore, we first define *text models* and *deterministic arithmetic automata* to represent random texts and deterministic computations, respectively, and then combine both into a PAA.

5.1 Random Text Models

Given an alphabet Σ, a random text is a stochastic process $(S_t)_{t \in \mathbb{N}_0}$, where each S_t takes values in Σ. A text model $\mathbb{P}$ is a probability measure assigning probabilities to (sets of) strings. It is given by (consistently) specifying the probabilities $\mathbb{P}(S_0 \ldots S_{|s|-1} = s)$ for all $s \in \Sigma^*$. We only consider finite-memory models, which are formalized in the following definition.

Definition 5.1 (Finite-memory text model). *A finite-memory text model is a tuple $(\mathcal{C}, c_0, \Sigma, \varphi)$, where $\mathcal{C}$ is a finite state space (called* context space*), $c_0 \in \mathcal{C}$ a start context, Σ an alphabet, and $\varphi : \mathcal{C} \times \Sigma \times \mathcal{C} \to [0,1]$ a transition function with $\sum_{\sigma \in \Sigma, c' \in \mathcal{C}} \varphi\ (c, \sigma, c') = 1$ for all $c \in \mathcal{C}$. The random variable giving the text model state after t steps is denoted C_t with $C_0 :\equiv c_0$. A probability measure is now induced by stipulating*

$$\mathbb{P}(S_0 \ldots S_{n-1} = s, C_1 = c_1, \ldots, C_n = c_n) := \prod_{i=0}^{n-1} \varphi(c_i, s[i], c_{i+1}),$$

for all $n \in \mathbb{N}_0$, $s \in \Sigma^n$, and $(c_1, \ldots, c_n) \in \mathcal{C}^n$.

The model $(\mathcal{C}, c_0, \Sigma, \varphi)$ generates a random text by moving from context to context and emitting a character at each transition, where $\varphi(c, \sigma, c')$ is the probability of moving from context c to context c' and thereby generating the letter σ.

Note that the probability $\mathbb{P}(S_0 \ldots S_{|s|-1} = s)$ is obtained by marginalization over all context sequences that generate s. This can be efficiently done, using the decomposition of the following lemma.

Lemma 5.2. *Let $(\mathcal{C}, c_0, \Sigma, \varphi)$ be a finite-memory text model. Then,*

$$\mathbb{P}(S_0 \ldots S_n = s\sigma, C_{n+1} = c) = \sum_{c' \in \mathcal{C}} \mathbb{P}(S_0 \ldots S_{n-1} = s, C_n = c') \cdot \varphi(c', \sigma, c),$$

for all $n \in \mathbb{N}_0$, $s \in \Sigma^n$, $\sigma \in \Sigma$ and $c \in \mathcal{C}$.

Proof. See Appendix, available in the online supplemental material. □

Similar text models are used in [11], where they are called probability transducers. We call a finite-memory text model $(\mathcal{C}, c_0, \Sigma, \varphi)$ simply *text model*, as we do not consider other text models in this paper.

An i.i.d. model is obtained by setting $\mathcal{C} := \{\varepsilon\}$ and $\varphi(\varepsilon, \sigma, \varepsilon) := p_\sigma$ for each $\sigma \in \Sigma$, where p_σ is the occurrence probability of letter σ (and ε may be interpreted as an empty context). For a Markovian text model of order r, the distribution of the next character depends only on the r preceding characters (fewer at the beginning); thus we set $\mathcal{C} := \bigcup_{i=0}^{r} \Sigma^i$. The conditional follow-up probabilities are given by

$$\mathbb{P}(S_i = s[i] \mid S_{i-1} = s[i-1], \ldots, S_0 = s[0]) = \begin{cases} \varphi(s[..i-1], s[i], s[..i]) & \text{if } i < r\,, \\ \varphi(s[i-r \ldots i-1], s[i], s[i-r+1 \ldots i]) & \text{if } i \geq r. \end{cases}$$

This notion of text models also covers variable order Markov chains [17], which can be converted into equivalent models of fixed order. Text models as defined above have the same expressive power as character-emitting HMMs, that means, they allow to construct the same probability distributions (see Appendix, available in the online supplemental material).

5.2 Deterministic Arithmetic Automata (DAAs)

To model deterministic calculations on sequences, we define a deterministic counter-part to PAAs.

Definition 5.3 (DAA). *A* deterministic arithmetic automaton *is a tuple*

$$\mathcal{D} = \big(\mathcal{Q}, q_0, \Sigma, \delta, \mathcal{V}, v_0, \mathcal{E}, (\eta_q)_{q\in\mathcal{Q}}, (\theta_q)_{q\in\mathcal{Q}}\big),$$

where $\mathcal{Q}$ is a finite set of states, $q_0 \in \mathcal{Q}$ is the start state, Σ is a finite alphabet, $\delta : \mathcal{Q} \times \Sigma \to \mathcal{Q}$ is called transition function, $\mathcal{V}$ is a set of values, $v_0 \in \mathcal{V}$ is called the start value, $\mathcal{E}$ is a finite set of emissions, $\eta_q \in \mathcal{E}$ is the emission associated to state q, and $\theta_q : \mathcal{V} \times \mathcal{E} \to \mathcal{V}$ is a binary operation associated with state q. Further, we define the associated joint transition function

$$\bar{\delta} : (\mathcal{Q} \times \mathcal{V}) \times \Sigma \to (\mathcal{Q} \times \mathcal{V}),$$
$$\bar{\delta}\big((q, v), \sigma\big) := \big(\delta(q, \sigma)\ ,\ \theta_{\delta(q,\sigma)}(v, \eta_{\delta(q,\sigma)})\big).$$

We extend the definitions of δ and $\bar{\delta}$ inductively from Σ to Σ^* in their second argument. When $\bar{\delta}((q_0, v_0), s) = (q, v)$ for some $q \in \mathcal{Q}$ and $s \in \Sigma^*$, we say that $\mathcal{D}$ computes value v for input s and define $value_{\mathcal{D}}(s) := v$.

Informally, a DAA starts with the state-value pair (q_0, v_0) and reads a sequence of symbols from Σ. Being in state q with value v, upon reading $\sigma \in \Sigma$, the DAA performs a state transition to $q' := \delta(q, \sigma)$ and updates the value to $v' := \theta_{q'}(v, \eta_{q'})$ using the operation and emission of the new state q'.

For each state q, the emission η_q is fixed and could be dropped from the definition of DAAs. In fact, one could also dispense with values and operations entirely and define a DFA over state space $\mathcal{Q} \times \mathcal{V}$, performing the same operations as a DAA. However, we intentionally include values, operations, and emissions to emphasize the connection to PAAs.

5.3 PAAs from DAAs and Text Models

We now formally state how to convert a DAA into a (restricted) PAA, where each emission distribution is deterministic (assigning probability 1 to a particular value), given a text model.

Lemma 5.4 (DAA + Text Model → PAA). *Let $(\mathcal{C}, c_0, \Sigma, \varphi)$ be a text model and $\mathcal{D} = (\mathcal{Q}^{\mathcal{D}}, q_0^{\mathcal{D}}, \Sigma, \delta, \mathcal{V}, v_0, \mathcal{E}, (\eta_q)_{q\in\mathcal{Q}^{\mathcal{D}}}, (\theta_q^{\mathcal{D}})_{q\in\mathcal{Q}^{\mathcal{D}}})$ be a DAA. Then, define*

- *a state space $\mathcal{Q} := \mathcal{Q}^{\mathcal{D}} \times \mathcal{C}$,*
- *a start state $q_0 := (q_0^{\mathcal{D}}, c_0)$,*
- *transition probabilities*

$$T\big((q, c), (q', c')\big) := \sum_{\sigma\in\Sigma:\ \delta(q,\sigma)=q'} \varphi(c, \sigma, c'). \qquad (9)$$

- *(deterministic) emission probability vectors*

$$\mu_{(q,c)}(e) := \begin{cases} 1 & \text{if } e = \eta_q\ , \\ 0 & \text{otherwise}\ , \end{cases}$$

  for all $(q, c) \in \mathcal{Q}$.
- *operations $\theta_{(q,c)}(v, e) := \theta_q^{\mathcal{D}}(v, e)$ for all $(q, c) \in \mathcal{Q}$.*

Then, $\mathcal{P} = (\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu = (\mu_q)_{q\in\mathcal{Q}}, \theta = (\theta_q)_{q\in\mathcal{Q}})$ is a PAA with

$$\mathcal{L}(V_t) = \mathcal{L}\big(value_{\mathcal{D}}(S_0 \dots S_{t-1})\big),$$

for all $t \in \mathbb{N}_0$, where S is a random text according to the text model $(\mathcal{C}, c_0, \Sigma, \varphi)$.

Proof. See Appendix, available in the online supplemental material. □

Remark 5.5. In the above lemma, states having zero probability of being reached from q_0 may be omitted from $\mathcal{Q}$ and T.

Lemma 5.6.

1. *For a PAA constructed according to Lemma 5.4, the value distribution $\mathcal{L}(V_n)$, or the joint state-value distribution, can be computed with $\mathcal{O}(n \cdot |\mathcal{Q}^{\mathcal{D}}| \cdot |\Sigma| \cdot |\mathcal{C}|^2 \cdot \vartheta_n)$ operations using $\mathcal{O}(|\mathcal{Q}^{\mathcal{D}}| \cdot |\mathcal{C}| \cdot \vartheta_n)$ space. The same statement holds for computing the waiting time distribution up to time n.*
2. *If for all $c \in \mathcal{C}$ and $\sigma \in \Sigma$, there exists at most one $c' \in \mathcal{C}$ such that $\varphi(c, \sigma, c') > 0$, then the time is bounded by $\mathcal{O}(n \cdot |\mathcal{Q}^{\mathcal{D}}| \cdot |\Sigma| \cdot |\mathcal{C}| \cdot \vartheta_n)$.*
3. *Using the doubling technique, the distributions can be computed in $\mathcal{O}(\log n \cdot |\mathcal{Q}^{\mathcal{D}}|^3 \cdot |\mathcal{C}|^3 \cdot \vartheta_n^3)$ time and $\mathcal{O}(|\mathcal{Q}^{\mathcal{D}}|^2 \cdot |\mathcal{C}|^2 \cdot \vartheta_n^2)$ space.*

Proof. See Appendix, available in the online supplemental material. □

This concludes the development of the PAA framework. The following sections are devoted to several applications.

6 Pattern Matching Statistics

An algorithm that counts the number of times a pattern occurs in a text computes a function $\Sigma^* \to \mathbb{N}$, i.e., it deterministically processes a string to compute a value. We ask for the distribution of the number of occurrences of a given pattern in a random text.

There are many relevant pattern types in computational biology, such as single strings, sets of strings, Prosite patterns [18], consensus strings together with a distance measure and a distance threshold, abelian patterns, position weight matrices in connection with a threshold, etc. All these pattern types are ways to concisely describe finite sets of strings and can be expressed as DFAs that recognize the respective string set. As our method is based on this DFA representation, it is very general and flexible regarding the pattern type.

Besides specifying a pattern, one has to decide how overlaps are to be handled. We refer to the used strategy as *counting scheme*. The *nonoverlapping count* is the maximal number of nonoverlapping matches. The *overlapping count* is the number of substrings that match the pattern. For a set of strings without restrictions, this scheme makes counting more complicated as some words may be substrings of others. Many authors avoid the problem—at least partly—by counting the positions where at least one pattern ends, which we refer to as *match position count*.

6.1 Related Work

The topic of statistics of words on random texts has been studied extensively. An overview is provided in the book by Lothaire [19], Chapter 6 ("Statistics on Words with

Applications to Biological Sequences"), which is based on the review by Reinert et al. [15].

Typically, a generating function is derived for the sought quantity, which allows asymptotic analysis. Using symbolic Taylor expansion, the concrete values can be computed; see, e.g., Régnier [20], who gives formulas for mean, variance, and higher statistical moments of the exact occurrence count distribution. Her framework admits Markovian sources as well as finite sets of patterns in the overlapping as well as in the nonoverlapping case. Closely related is the algorithmic chain of Nicodème et al. [12] to compute the distribution of the match position count for regular expressions. Lladser et al. [8] coined the terms of *Markov chain embedding* for combining finite automata with Markov chains. Related approaches to compute the exact p-values based on automata are developed in [9], [21]. Another dynamic programming approach for p-values of position weight matrices describing transcription factor binding sites (TFBSs) was presented in [22].

PAAs provide a unifying framework for the efficient computation of pattern matching statistics. In contrast to existing approaches, overlaps are handled correctly and arbitrary finite-memory text models (including HMMs) can be used.

6.2 Constructing DAAs from DFAs

As usual, a DFA is a tuple $(\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{F})$, where $\mathcal{Q}$ is a finite state space, Σ is a finite alphabet, $\delta : \mathcal{Q} \times \Sigma \to \mathcal{Q}$ is a transition function, q_0 is a start state, and $\mathcal{F} \subset \mathcal{Q}$ is a set of accepting states. Again, we extend δ to Σ^* in its second argument, as for DAAs in Definition 5.3. Suppose a pattern is given in the form of a DFA, that means, a string is an instance of the pattern if it is accepted by the DFA. We calculate the distribution of the number of occurrences of this pattern.

To find all instances of a pattern in a (long) text, we construct an automaton that accepts all strings that have a suffix matching the pattern (see [23]). For a pattern given in the form of a DFA or nondeterministic finite automaton (NFA), it is always possible to construct a DFA for this task as follows: First, we add a self-transition labeled with the whole alphabet Σ to the automaton's start state to ensure that the start state remains active all the time. The result is an NFA, which can be made deterministic again by employing the classical subset construction (e.g., [23]), but there are often simpler and more direct ways to build the DFA (see Sections 6.2.1 and 6.2.3).

When a DFA reads a text, it is in an accepting state whenever (at least) one instance of the pattern ends. The number of these events equals the *match position count* as defined above. To count the number of times a DFA $(\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{F})$ is in an accepting state, we define emissions

$$\eta_q := \begin{cases} 1 & \text{if } q \in \mathcal{F}, \\ 0 & \text{otherwise.} \end{cases}$$

We call a tuple $(\mathcal{Q}, \Sigma, \delta, q_0, (\eta_q)_{q\in\mathcal{Q}})$ a *counting DFA*.

To compute the distribution of the *overlapping count* instead of the match position count, we take into account that more than one match can end at a text position. Thus, we define emissions $\eta = (\eta_q)_{q\in\mathcal{Q}}$, where $\eta_q \in \mathbb{N}_0$ gives the number of matches to be counted upon entering state q. This is well defined if the pattern represents a finite set of strings. (In the case of an infinite set, such as a regular expression containing a star operator, there may be states in $\mathcal{F}$ for which the number is unbounded. In this case, one is restricted to the match position count.)

To obtain the *nonoverlapping count*, the automaton can be modified accordingly: We change the outgoing transitions of each accepting state of the match position count automaton to act as if they originate from the start state q_0. Therefore, we define a modified transition function δ' by $\delta'(q, \sigma) := \delta(q, \sigma)$ if $q \notin \mathcal{F}$, and $\delta'(q, \sigma) := \delta(q_0, \sigma)$ otherwise.

The main idea is now to construct a DAA from the counting DFA by adding the emissions generated in each state, and turn it into a PAA using Lemma 5.4. In practice, it suffices to truncate the values at a constant $M \in \mathbb{N}$. (The nonoverlapping and the match position count are always bounded by the length of the processed text, so $\vartheta_n = \Theta(n)$.)

Theorem 6.1. *Let a counting DFA $D = (\mathcal{Q}, \Sigma, \delta, q_0, (\eta_q)_{q\in\mathcal{Q}})$ and a text model $(\mathcal{C}, c_0, \Sigma, \varphi)$ be given, and let $(S_t)_{t\in\mathbb{N}_0}$ be a random text distributed according to that model. Then, the (truncated) distribution of accumulated counts*

$$\mathcal{L}\left(\min\left\{M\,,\ \sum_{i=0}^{n-1} \eta_{\delta(q_0, S_0 \ldots S_i)}\right\}\right), \tag{10}$$

can be computed in $\mathcal{O}(n \cdot |\mathcal{Q}| \cdot |\mathcal{C}|^2 \cdot |\Sigma| \cdot M)$ time and $\mathcal{O}(|\mathcal{Q}| \cdot |\mathcal{C}| \cdot M)$ space. If for all $c \in \mathcal{C}$ and $\sigma \in \Sigma$, there exists at most one $c' \in \mathcal{C}$ such that $\varphi(c, \sigma, c') > 0$, then the runtime is bounded by $\mathcal{O}(n \cdot |\mathcal{Q}| \cdot |\mathcal{C}| \cdot |\Sigma| \cdot M)$. Alternatively, (10) can be computed in $\mathcal{O}(\log n \cdot |\mathcal{Q}|^3 \cdot |\mathcal{C}|^3 \cdot M^2)$ time and $\mathcal{O}(|\mathcal{Q}|^2 \cdot |\mathcal{C}|^2 \cdot M)$ space.

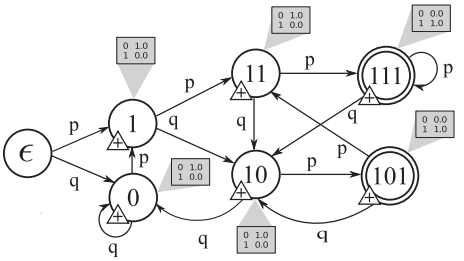
Proof. The proof in the Appendix, available in the online supplemental material, contains details of the DFA $\to$ DAA $\to$ PAA construction. □

Before turning the DFA into a PAA, one may wish to minimize it. Using an algorithm by Hopcroft [24], a classical DFA can be minimized in $\mathcal{O}(|\mathcal{Q}| \log |\mathcal{Q}|)$ time for an alphabet of constant size, where $\mathcal{Q}$ is the set of states. Refer to Knuutila [25] for a tutorial-like introduction and a variant that runs in $\mathcal{O}(|\Sigma| \cdot |\mathcal{Q}| \log |\mathcal{Q}|)$ time when the alphabet size is not considered to be a constant. Hopcroft's algorithm can be adapted to minimize counting DFAs by using the partition induced by the different emissions as an initial partition, i.e., states with the same emission are grouped together.

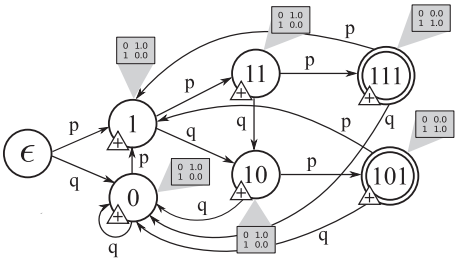
We now review several pattern classes important in practice, particularly in computational biology.

6.2.1 Finite Sets of Strings

If the pattern is given as a finite set of strings, an Aho-Corasick automaton, which essentially is a DFA, can be constructed in linear time, either by using the original algorithm [26], or via a recent elegant algorithm using the suffix tree of the reverse strings [27]. The emissions η_q (number of matches) can be read off the Aho-Corasick automaton's output function for each state. Fig. 1 shows an example of the resulting PAA.



(a) PAA for overlapping hits



(b) PAA for non-overlapping hits

Fig. 1. PAAs for the distribution of matches of the pattern set $\{101, 111\}$, assuming an i.i.d. text model over alphabet $\Sigma = \{0, 1\}$ with probability p for character 1, and $q := 1 - p$. The start state is denoted ϵ. Each state is associated with the operation "+" and a Dirac emission distribution, shown in the gray boxes.

6.2.2 Finite Sets of Generalized Strings

Generalized strings are finite sequences of sets of characters over an alphabet Σ, for example, `[abc][ac][ab]` (which matches `aaa`, `ccb` but not `aba`). Rather than enumerating all strings matching a generalized string, we can directly construct an NFA that recognizes all strings ending with an instance of it. The NFA for one generalized string is just a linear chain of states; a start state plus one state for each position, where the start state is additionally equipped with a self-transition. The NFA for the set of generalized strings is constructed by merging all individual start states into one common start state. The next step is to build a DFA from the NFA. The classical subset construction results, in the worst case, in an exponential increase in the number of states; however it is feasible in many practical cases. The construction can be modified such that it always results in the minimal DFA [6]. By the subset construction, a set B_q of NFA states corresponds to each DFA state q. The number of final NFA states in B_q equals the number of matching generalized strings that end when DFA state q is entered, giving us the number of matches η_q to be emitted by q.

6.2.3 Prosite Patterns

Prosite is a database of biologically meaningful amino acid motifs [18]. Prosite patterns are generalized strings with the extension that, for each position, a "multiplicity range" can be specified. In the pattern `A-x(2,3)-C`, for example, an `A` is followed by either two or three arbitrary characters followed by a `C`. We translate every Prosite pattern into a set of generalized strings. The above example would result in the two patterns `A-x-x-C` and `A-x-x-x-C`. This set can then be dealt with as explained above.

Release 20.81 of Prosite contains 1,308 patterns, 20 of which we ignored because they refer to the start or ending of a sequence or contain nonstandard amino acids. For the remaining 1,288 patterns we constructed sets of generalized strings, NFAs, DFAs, and finally PAAs as described above. The computation was aborted if either the number of generalized strings obtained from the Prosite pattern exceeded 1,000 or the number of nodes in the constructed DFA exceeded one million. This was the case for 44 patterns. For the 1,244 successfully processed patterns, we computed the distribution of the occurrence count with parameters $M = 50$ (maximum number of occurrences of interest) and $n = 1{,}000$ (text length) for an i.i.d. text model. The constructed PAAs had a median size of 49 states and the median runtime was 0.02s for the whole workflow. Detailed histograms can be found in the Appendix, available in the online supplemental material.

6.3 Waiting Time for Pattern Occurrences

The waiting time for the first occurrence of a pattern equals the waiting time for a state that emits a match. Therefore, its distribution can be computed with Lemma 4.4. The waiting time for a subsequent occurrence can be computed by choosing α in Lemma 4.4 as equilibrium distribution restricted to all match states.

6.4 Clump Size Distribution

We quantify a pattern's tendency to overlap itself by its *clump size distribution* Ψ, which is, besides its theoretical value, useful for the construction of compound Poisson approximations [3], which are accurate approximations to pattern occurrence count distributions and therefore widely used [28], [29], [30].

Definition 6.2. *Given a sequence $s \in \Sigma^*$ and a pattern p, a* clump *is a maximal set of overlapping occurrences of p in s.*

For example, let $s :=$ `GACACATTACAAA` and $p :=$ `ACA`. Then, s contains three occurrences of p in two clumps (underlined). By definition, a clump consists of at least one match. We call the position of a match's last character *match position* and consider the first match position in a clump. Further, we call the distribution of PAA states at such positions *clump start distribution* and denote it γ; i.e., given that j is the first match position in a clump, then $\mathbb{P}(Q_j = q) =: \gamma(q)$, which is asymptotically independent of j under certain assumptions. For now, we assume γ to be known. Its calculation is straightforward; technical details can be found in the Appendix, available in the online supplemental material.

If $m \geq 2$ is the length of the given motif, then a clump ends if $m - 1$ consecutively visited states do not emit a match. We thus need to keep track of 1) the number of nonmatch states consecutively visited and 2) the number of matches the clump contains so far. We modify the PAAs described in the previous sections. We define a new value set $\mathcal{V}' := \mathbb{N}_0 \times \{0, \ldots, m-1, \bullet\}$ with the start value $v_0' := (1, 0)$ and attach the following semantics: If we are in state q and the current value is (h, x), we have seen h matches in the current clump and the last of these matches occurred x steps in the past; i.e., if $x = 0$, a match has been emitted from the current state. The special value $x = \bullet$ indicates that the clump has ended. We define the operations accordingly:

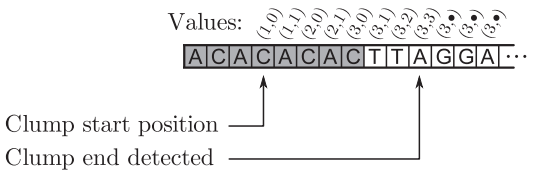


Fig. 2. A clump (shaded gray) of three occurrences of the pattern ACAC. By definition, the clump starts at the last character of the first occurrence. The values computed by the clump size DAA are shown above the string. As soon as the second counter reaches $m-1=3$, the clump has ended.

$$\theta'_q:\ \big((h,x),e\big) \mapsto \begin{cases} (h+e,0) & \text{if } e>0 \text{ and } x\in\{0,\dots,m-2\}\ , \\ (h,x+1) & \text{if } e=0 \text{ and } x\in\{0,\dots,m-2\}\ , \\ (h,\bullet) & \text{if } x\in\{m-1,\bullet\}\ . \end{cases}$$

In other words, if a match has been found ($e>0$), we increase the number of matches h by e and reset the distance to the last match to 0. Otherwise ($e=0$, no match occurred), h remains unmodified, but the number of steps x since the last match is increased by one. See Fig. 2 for an example. To incorporate the clump start distribution γ, we use one additional state q'_0 that becomes the new start state; consequently, we set $\mathcal{Q}' := \{q'_0\} \cup \mathcal{Q}$ and define the new transition function to be

$$T' : (q,q') \mapsto \begin{cases} \gamma(q') & \text{if } q = q'_0\ , \\ T(q,q') & \text{otherwise}\ . \end{cases} \tag{11}$$

The set $\mathcal{V}'$ is infinite. As discussed in Section 3, this does not pose a problem as the range of each V_t is finite. Furthermore, for many applications it is sufficient to truncate the clump size distribution and use the value set $\mathcal{V}'' := \{1,\dots,M\} \times \{0,\dots,m-1,\bullet\}$ along with adapted operations θ''_q. Employing one of the algorithms shown in Sections 3.1 and 3.2, respectively, we can then calculate the joint state-value distributions $\rho_t(q,h,x) := \mathbb{P}(Q_t = q, V_t = (h,x))$. A clump ends if no new match has occurred $m-1$ steps after the previous match. The clump size distribution Ψ is thus given by

$$\Psi(h) = \sum_{t=0}^{\infty} \sum_{q\in\mathcal{Q}} \rho_t(q,h,m-1)\ . \tag{12}$$

To actually compute Ψ, we start with the initial table ρ_0 and iteratively calculate the tables ρ_t for larger t. Each ρ_t contributes to the sought distribution through the inner sum from (12) and we can successively add the contributions to an intermediate clump size distribution. Observe that the difference between the intermediate clump size distribution after iteration t and the exact one is bounded by

$$1 - \sum_{q\in\mathcal{Q}} \sum_{h=0}^{M} \rho_t(q,h,\bullet)\ .$$

Thus, we iterate until this quantity drops under an accuracy threshold. The number of necessary steps, however, is bounded by $\mathcal{O}(M\cdot m)$, because a clump containing M matches can have a length of at most $\mathcal{O}(M\cdot m)$. In total, we need $\mathcal{O}(|\Sigma|\cdot|\mathcal{C}|\cdot|\mathcal{Q}|\cdot M^2\cdot m^3)$ time to compute the exact clump size distribution. Again, a factor of $|\mathcal{C}|$ can be saved if for all $c\in\mathcal{C}$ and $\sigma\in\Sigma$, there exists at most one $c'\in\mathcal{C}$ such that $\varphi(c,\sigma,c')>0$.

7 Analysis of Window-Based Pattern Matching Algorithms

The pattern matching problem is to (efficiently) find all occurrences of a *pattern* in a (long) *text*. Let n be the text length and m be the pattern length. The Knuth-Morris-Pratt algorithm [31] reads each text character exactly once from left to right after preprocessing the pattern in $\Theta(m)$ time and needs n character accesses. In contrast, the Boyer-Moore [32], Horspool [33], Backward DAWG Matching (BDM, [34]), and Backward Oracle Matching (BOM, [35]) algorithms move a length-m search window across the text and first compare its *last* character to the last character of the pattern. This often allows to move the search window by more than one position, for a best case of $\Theta(n/m)$ character accesses after $\Theta(m)$ preprocessing time.

In general, a window-based algorithm that searches for a pattern p is characterized by

- its window size m,
- a cost function $\xi^p : \Sigma^m \to \mathbb{N}_0$ giving the cost caused by a window,
- a shift function $shift^{\,p} : \Sigma^m \to \{1,\dots,m\}$ giving the number of positions the window is shifted by.

That is, the algorithm starts by considering the size-m window w at the beginning of the text, processes it incurring a cost given by $\xi^p(w)$, shifts the window according to $shift^{\,p}(w)$, and iterates these two steps until the end of the text is reached. The total cost of processing a text $s\in\Sigma^n$ is denoted $\xi^p(s)$.

We employ the framework of DAAs and PAAs to calculate the exact distribution of $\xi^p(S_0\dots S_{n-1})$ when S is a random text and pattern p is fixed. It is based on our earlier paper [36], where we elaborate on many details omitted here. Prior to that, [37], [38] analyzed the expected number of character accesses for Horspool's algorithm. In [39] it is shown that the number of character accesses is asymptotically normally distributed for i.i.d. texts, and [40] extends this result to Markovian text models. Mean and variance of these distributions can be computed using methods introduced in [41].

The difficulty in constructing a DAA that computes $\xi^p(s)$ lies in that DAAs process one character at a time while the algorithm might move the window by more than one character. To overcome this problem, we define the DAA state space as $\mathcal{Q} := \Sigma^m \times \{0,\dots,m\}$, where being in state (w,x) means that the last m read characters spell w and that x more characters need to be read to reach the end of the current window. All states of the form $(w,0)$ emit the cost $\xi^p(w)$ while all states (w,x) with $x>0$ emit 0. Transitions outgoing from states of the former type set x to $shift^{\,p}(w)-1$, while those outgoing from states of the latter type decrease x by one. Refer to the Appendix, available in the online supplemental material, for a formal definition of this DAA.

By virtue of Lemma 5.4, the DAA can be combined with any finite-memory text model to obtain a PAA. The PAA, in turn, allows us to compute the distribution of running time

costs under that text model by means of the generic dynamic programming algorithms described in Section 3.

A key obstacle for these computations is the exponential growth of the state space in m. However, its size can be reduced considerably in practice by minimizing the DAA along the lines of classical results on DFA minimization [24]; refer to [36] for details. To demonstrate the practicability, we constructed minimal DAAs for Horspool's algorithm for all patterns of length eight over the DNA alphabet $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ and computed the distribution of character access counts for text length 100 and an i.i.d. text model. The average number of DAA states amounted to 37.0 and the average total runtime to 0.2 seconds (for DAA construction, minimization, PAA construction, and computation of the target distribution). Histograms of state counts and running times can be found in the Appendix, available in the online supplemental material. Detailed case studies comparing Horspool, BDM, and BOM are presented in [36].

8 Alignment Seed Sensitivity

Methods to compute the pattern occurrence count distribution (Section 6) are applicable to the problem of assessing the quality of *alignment seeds*.

In homology search, a sequence database is searched for a query sequence in order to find potential homologs, i.e., evolutionarily related and therefore usually similar sequences. Each database sequence is compared with the query, but exhaustive local alignment algorithms, such as Smith-Waterman, are too slow in practice. Thus, fast homology search algorithms are based on a two-phase filtration technique [42], [43], [44]. First, candidate sequences are selected that share a common pattern ("seed") of matching characters with the query. These candidates (or "hits") are further investigated by an exact method.

8.1 Unifying Previous Approaches

The initial BLAST [43] implementation used perfectly matching DNA 11-mers as seeds or, expressed differently, used the seed `11111111111`. The latter notation refers to the *representative string* $\mathcal{A}$ of an alignment. For ungapped alignments, $\mathcal{A}$ is a string over $\Sigma = \{0, 1\}$ containing one character for each alignment column, `1` for a match and `0` for a mismatch:

Query :	G	C	G	A	A	T	G	C	C	T
Database :	G	C	C	A	A	C	G	C	T	T
$\mathcal{A}$:	1	1	0	1	1	0	1	1	0	1

PatternHunter (PH) by Ma et al. [45] was the first tool to systematically advocate and investigate *spaced seeds*: PH looks for 18-mers with at least 11 matching positions distributed as `111*1**1*1**11*111`, where `*` denotes a don't care position (match or mismatch). A good seed exhibits high sensitivity for alignments of related biosequences, and low sensitivity for alignments of unrelated sequences. Where the former ensures that true hits are unlikely to be missed, the latter ensures that only promising candidates pass the filtration phase. The PH approach led to an increase in both sensitivity and filtration efficiency, compared to seeds of contiguous matches. The advantages of spaced seeds over consecutive seeds have been subsequently evaluated by many authors [46], [47], [48].

To assess seeds, a random *homology model* over the alignment alphabet $\Sigma = \{0, 1\}$ is assumed. In the simplest case, it is an i.i.d. model, where the only parameter p is the probability of a matching character and thus the expected fraction of identical characters. Now a seed's sensitivity is defined to be the probability of hitting a random representative string of length ℓ with respect to the homology model. In the literature, ℓ is often (somewhat arbitrarily) chosen as $\ell = 64$. As a spaced seed is a generalized string over Σ, its sensitivity can be computed by the methods discussed in Section 6.2.2.

Over time, various other seed models have been proposed in the literature, including consecutive seeds [42], [43], spaced seeds [46], [49], [50], subset seeds [11], vector seeds [51], indel seeds [52], and multiple seeds [53], [54], [55]. These extensions vary in how the alignment alphabet is defined, how seeds are defined, and in the used homology model. In any case, the seed can be expressed as a finite set of strings over the alignment alphabet and the homology model is (some special case of) a finite-memory text model.

Indel seeds [52], for instance, use the alignment alphabet $\Sigma = \{0, 1, 2, 3\}$, where `2` and `3` indicate an insertion and a deletion, respectively. An indel seed is a string over the alphabet $\Xi = \{1, *, ?\}$, where `1` and `*` are as above, and `?` stands for zero or one character from the $\Sigma = \{0, 1, 2, 3\}$. Two consecutive `?` symbols represent any character pair except "`23`" or "`32`." By means of this interpretation, the model explicitly allows for indels of variable size. The sensitivity is evaluated with respect to a first order Markov model over Σ. This example shows that, although significantly extended compared to [45], this approach stays well within the capabilities of the PAA framework. The same holds true for the other cited approaches. The PAA framework hence provides a unifying method for computing seed sensitivity for different alignment models and seed models that were so far developed in an ad hoc fashion in the literature.

8.2 The PatternHunter Seed Revisited

To showcase one application of our framework, we compute the distribution of hit counts of the PH seed and compare it to the respective distribution for the classical BLAST seed. We used our MoSDi implementation to calculate the probability of exactly k seed hits for $k = 0, \ldots, 3$ for both seeds. As a homology model, we employed the classical ungapped i.i.d. model with parameter p giving the match probability. The calculations were carried out from $p = 0.3$ up to $p = 0.95$ in increments of 0.05. That is, we span the whole range from unrelated sequences, where a low sensitivity is desirable, to highly similar sequences, where a high sensitivity is desirable. The results for $p = 0.3$ and $p = 0.95$ are shown in Table 1. Consistently with previous results, the PH seed is indeed much more sensitive than the contiguous seed for $p = 0.95$.

Instead of filtering out sequences with less than one seed hit, a homology search method could also filter out regions with less than h hits for $h > 1$, trading sensitivity for filter efficiency as h grows. As we compute the whole hit count

TABLE 1
The Probabilities for Exactly k (Possibly Overlapping) Hits in a Target Region of Length 64 with respect to an i.i.d. Homology Model Are Shown for the classic BLAST seed 11111111111 and the PatternHunter (PH) seed 111*1**1*1**11*111, with $p = 0.95$ (Top) and $p = 0.3$ (Bottom) being the Match Probability

$p=0.95$	$k=0$	$k=1$	$k=2$	$k=3$
BLAST	$4.128 \cdot 10^{-4}$	$5.400 \cdot 10^{-4}$	$8.447 \cdot 10^{-4}$	0.001
PH	$6.733 \cdot 10^{-6}$	$4.498 \cdot 10^{-5}$	$1.612 \cdot 10^{-4}$	$4.167 \cdot 10^{-4}$
$p=0.3$	$k=0$	$k=1$	$k=2$	$k=3$
BLAST	0.999932	$4.761 \cdot 10^{-5}$	$1.402 \cdot 10^{-5}$	$4.129 \cdot 10^{-6}$
PH	0.999917	$8.278 \cdot 10^{-5}$	$2.243 \cdot 10^{-7}$	$8.995 \cdot 10^{-9}$

distributions, we can precisely analyze this tradeoff. Changing h from one to two, for instance, decreases the sensitivity (probability of h or more hits for $p = 0.95$) of the PH seed from 0.999993 to 0.99994, while the filter efficiency (probability of less than h random hits for $p = 0.3$) increases from 0.9999170 to 0.9999998.

The Appendix, available in the online supplemental material, contains figures showing the dependency of the sensitivity for $h = 1, 2, 3$ on p: Different values of h can be the tradeoff of choice between filter efficiency and sensitivity. When the sequence length is $\ell = 256$, for instance, the standard 1-hit PH seed has an undesirably high sensitivity of 0.10 for low-similarity alignments with $p = 0.5$. By requiring $h = 3$ hits, it drops to 0.002, resulting in an increased filter efficiency. For alignments with a higher similarity of $p = 0.85$, the 3-hit seed achieves a sensitivity of $1 - 1.4 \cdot 10^{-6}$, which may be an acceptable loss compared to $1 - 2.1 \cdot 10^{-8}$ for the 1-hit seed.

To measure runtime, we computed the described distributions for all seeds of length 18 with 7 don't care positions for all p from 0.3 to 0.95 in increments of 0.05. Doing this for $\ell = 64$ and $\ell = 256$ took 7.2 and 20.3 minutes, respectively.

8.3 Outlook

For a simple homology model with few parameters (say, the i.i.d. homology model with a single parameter p), we can evaluate the basic recurrence from Lemma 3.1 symbolically, i.e., by representing the entries of the transition matrix $T(q', q)$ and the probabilities of the state-value distribution $f_t(q, v)$ as polynomials in the parameters. The resulting polynomial only needs to be computed once; then one can assess the sensitivity of a seed under different parameter values. This was previously presented by Mak and Benson [13], without using the PAA approach.

The PAA framework can be applied to design efficient sets of seeds. Similar to [55], we can successively find seeds that locally maximize the conditional sensitivity, given that the seeds already present in the set do not hit an alignment. Such a conditional sensitivity can be computed by making the existing seeds' accepting states absorbing without counting a match, so only instances of the current seed candidate are counted. Evaluating each seed candidate and picking the best one yields the seed to be added to the set.

9 Protein Fragment Mass Statistics

Peptide mass fingerprinting (PMF) is a technique for protein identification based on mass spectrometry (MS) and database search. The protein of interest is enzymatically cleaved into smaller peptides whose masses are determined by MS. The set of peptide masses, the so-called *peptide mass fingerprint*, is then used to query a database of known proteins. Each database sequence is processed *in silico* to obtain a theoretical fingerprint, and experimental and theoretical fingerprints are subsequently compared and scored. To assess a hit's significance, many PMF software packages compare the measured fragment masses to empirical frequencies of fragments in large protein databases. As this is error prone in case of rare (or yet unobserved) fragment masses, Kaltenbach [56] introduced database-independent scores based on the probability of observing a given fragment mass in a random protein, using an i.i.d. text model estimated from the UniprotKB database [57] as a background model. We show how such questions can be addressed in the PAA framework and thereby generalize the computations to arbitrary finite-memory text models.

First, we construct a DAA encoding the enzymatic cleavage reaction. Second, from this DAA and a model for random sequences of amino acids, a PAA is constructed by means of Lemma 5.4. The resulting PAA provides peptide statistics, in particular, the length distribution and the joint length-mass distribution of such fragments. Furthermore, we compute the probability that the peptide mass fingerprint of a random protein contains at least one peptide of a certain mass or in a certain mass range. Moreover, our framework allows incorporating the influence of isotopic distributions, incomplete cleavage, and posttranslational modifications by appropriate modifications of the PAA, as we discuss in the Appendix, available in the online supplemental material.

9.1 DAA for Fragment Masses

Cleavage enzymes cut proteins at well-defined places which are often determined by one or two adjacent amino acids (see [56], [58]). The most widely used cleavage agent Trypsin cleaves after lysine (K) and arginine (R) unless the next amino acid is proline (P). Such cleavage sites can be described by $\Gamma\bar{\Pi}$ with *cleavage characters* $\Gamma \subset \Sigma$ and *prohibition characters* $\Pi \subset \Sigma$, where $\bar{\Pi}$ denotes the complement of Π in the amino acid alphabet $\Sigma = \{\mathrm{A}, \ldots, \mathrm{Z}\} \setminus \{\mathrm{B}, \mathrm{J}, \mathrm{O}, \mathrm{U}, \mathrm{X}, \mathrm{Z}\}$.

We distinguish the first fragment F_1 from following fragments F_+: While the first fragment may start with a prohibition character, following fragments may not. The final fragment does not necessarily end with a cleavage character because of the finiteness of protein sequences. We write $F_\star$ to denote an unspecified fragment. That is, $F_\star$ is either a first, a following, or a final fragment.

A DAA $\mathcal{D} = (\mathcal{Q}, q_0, \Sigma, \delta, \mathcal{V}, v_0, \mathcal{E}, (\eta_q)_{q\in\mathcal{Q}}, (\theta_q)_{q\in\mathcal{Q}})$ is constructed that sums amino acid masses (for now, we ignore isotopes) until a cleavage point is encountered. We set $\mathcal{Q} := \Sigma \cup \{q_0, \bullet, \circ\}$, where q_0 is the start state. For each state $q \in \Sigma$, the emission η_q gives the mass of amino acid q; the set of possible values is $\mathcal{V} := \mathbb{R}^+$, where $v_0 := 0$ is the start value. Although $\mathcal{V}$ is infinite, the number of values reachable in n steps (ϑ_n) is finite for all n (see Section 3). For states $q' \in \{q_0, \bullet, \circ\}$ no mass is emitted, that means, $\eta_{q'} = 0$. To sum amino acid masses, we define operations by

$\theta_q : (v, e) \mapsto v + e$ for all $q \in \mathcal{Q}$. The transition function $\delta : \mathcal{Q} \times \Sigma \to \mathcal{Q}$ reflects the cleavage rules:

$$\delta(q, \sigma) = \begin{cases} \sigma & \text{if } q \in (\Sigma \setminus \Gamma) \cup \{q_0\} , \\ \sigma & \text{if } q \in \Gamma, \sigma \in \Pi , \\ \bullet & \text{if } q \in \Gamma, \sigma \notin \Pi , \\ \circ & \text{if } q \in \{\bullet, \circ\} . \end{cases}$$

This DAA has the property that $\delta(q_0, s[..i+1]) = \bullet$ if and only if the first fragment in s has length i. Then, $value_{\mathcal{D}}(s[..i+1]) = m$, where m is the mass of the first fragment.

From the DAA and a text model, a PAA is obtained by using Lemma 5.4. Computations up to an instrument-dependent mass accuracy suffice in practice. We assume that masses have been scaled appropriately and then rounded to integers.

The difference between the first fragment F_1 and the following fragments $F_2, F_3, \ldots$ lies solely in the initial state distribution. The transition probabilities from the start state to other states need to be adjusted in such a way that 1) prohibition characters are forbidden at the beginning of a following fragment and 2) the transitions reflect the distribution of text model states at the end of the previous fragment. This distribution can be obtained from the PAA for the previous fragment. Statistics for fragments F_k with $k > 1$ are i.i.d. if the underlying text model is i.i.d., assuming an infinite text length [56], [58].

9.2 Mass and Length Distributions

By means of the constructed PAAs, we compute fragment statistics as the length distribution and the joint length-mass distribution. We denote the length of a fragment by $L(F_\star)$ and the (integer) mass of a fragment by $M(F_\star)$. The length of a random fragment generated by the PAA corresponds to the waiting time for reaching state $\bullet$ minus one (as the last processed character does not belong to the fragment).

When only the distribution of fragment lengths (and not the mass distribution) is required, we apply Lemma 4.4. In general, the joint state-value distribution $f_n(q, v) = \mathbb{P}(Q_n = q, V_n = v)$ from Section 3 yields the joint length-mass distribution $\nu_\star$ of $F_\star$:

$$\begin{aligned} \nu_\star(n, m) &:= \mathbb{P}\big(L(F_\star) = n,\ M(F_\star) = m\big) \\ &= f_{n+1}(\bullet, m) \ = \ \mathbb{P}(Q_{n+1} = \bullet, V_{n+1} = m). \end{aligned}$$

The computation of f_{n+1} takes $\mathcal{O}(n \cdot |\mathcal{Q}|^2 \cdot \vartheta_n \cdot |\mathcal{E}|)$ time and $\mathcal{O}(|\mathcal{Q}| \cdot \vartheta_n)$ space (see Lemma 3.2). Since both $|\mathcal{Q}|$ and $|\mathcal{E}|$ are constants, and the range of possible masses ϑ_n grows linearly with both n and the mass scaling factor (or mass precision) λ, we need $\mathcal{O}(\lambda n^2)$ time and $\mathcal{O}(\lambda n)$ space.

9.3 Mass Occurrence Probabilities

For the interpretation of mass spectra, an important quantity is the occurrence probability of a measured mass m, i.e., the probability that the fragmentation of a random protein contains at least one fragment of mass m. To account for inaccurate measurements, we compute the probability that a fragment in a certain mass range $[m - \Delta, m + \Delta]$ occurs. We modify the above PAA such that it does not move into the absorbing state $\circ$ once the first fragment has ended. We therefore remove state $\circ$ and replace $\bullet$ by $|\Sigma|$ states named $\bullet_\sigma$ for each $\sigma \in \Sigma$; so $\mathcal{Q} := \{q_0\} \cup \Sigma \cup \{\bullet_\sigma : \sigma \in \Sigma\}$. Being in state $\bullet_\sigma$ means that a fragment has ended and the first character of the next fragment is σ. Therefore, the emission distribution of $\bullet_\sigma$ is the same as for state σ. In terms of the transition function δ, each state $\bullet_\sigma$ acts like state σ. Formally,

$$\delta(q, \sigma) := \begin{cases} \sigma & \text{if } q \in (\Sigma \setminus \Gamma) \cup \{q_0\} , \\ \sigma & \text{if } q \in \Gamma, \sigma \in \Pi , \\ \bullet_\sigma & \text{if } q \in \Gamma, \sigma \notin \Pi , \\ \delta(\sigma', \sigma) & \text{if } q = \bullet_{\sigma'} . \end{cases}$$

To keep track of whether a fragment in the given mass range has already been observed, we introduce an absorbing value $\diamond$; that means, all operations are adapted such that once the value $\diamond$ has been attained, it is not changed. The new states $\bullet_\sigma$ get the following operations:

$$\theta_{\bullet_\sigma}(v, e) := \begin{cases} \diamond & \text{if } v \in [m - \Delta, m + \Delta] , \\ e & \text{otherwise} . \end{cases}$$

The probability that a protein sequence of length n contains a segment in the mass range $[m - \Delta, m + \Delta]$ is given by $\mathbb{P}(V_n = \diamond) + \mathbb{P}(V_n \in [m - \Delta, m + \Delta])$, where the second summand accounts for the probability that the last fragment has the sought mass.

9.4 Experiments

We computed the probability that a random protein contains at least one fragment with a mass in $[m - \Delta, m + \Delta]$ for $\Delta = 0.5$ Da, all m in $\{100, 101, \ldots, 1000\}$, and all m in $\{1100, 1200, \ldots, 15000\}$. We used discretized masses with accuracy $\lambda = 0.1$ Da, the empirical protein length distribution (truncated at 3,000 Da) given by the UniprotKB database (release 2012_04), and a first order Markovian text model estimated from this database. Results (see Appendix, available in the online supplemental material) show that, for masses up to 1,000 Da, the occurrence probabilities vary over more than four orders of magnitude. Due to the combinatorial nature of the underlying process, even similar masses often have drastically different occurrence probabilities. For masses greater than 1,000 Da, the curve is smoother and resembles the expected geometric distribution. The time to compute the probability for one mass range $[m - \Delta, m + \Delta]$ grows linearly with m, from 18s for $m = 100$ to 1 hour and 35 minutes for $m = 15{,}000$. In practical applications, the probabilities for large masses can be approximated accurately by a geometric distribution, while those for all small masses can be precomputed.

10 High-Throughput Sequencing

In several recent DNA sequencing technologies, such as 454 or IonTorrent, the four different nucleotides are cyclically flowed over many different DNA templates in parallel. The machine observes, either optically or electrically, whether the currently flowed nucleotide was incorporated into each template. The particular flow order of nucleotides is called the *dispensation order* and repeated for a fixed number of cycles (200 for current 454 instruments, which use the fixed dispensation order TACG). In case of a homopolymer run (the same type of nucleotide appearing several times

consecutively), the corresponding templates are extended by the corresponding number of nucleotides at once, and a more intense signal is measured [59].

Kong [60] obtained generating functions for the resulting distribution of read lengths, but the dispensation order was restricted to a permutation of the nucleotides, and the text model was restricted to an i.i.d. model. Here, we use PAAs to compute the exact length distribution of sequencing reads for a fixed number f of nucleotide flows, a given dispensation order, and an arbitrary finite-memory text model.

We first construct a DAA that computes the number of nucleotide flows needed to read an input sequence using a given dispensation order $d = d[0] \dots d[\ell - 1]$. In each step, the DAA reads one nucleotide to be sequenced and the emitted value gives the number of necessary flows needed to reach this nucleotide starting from the previously sequenced nucleotide. Computing the sought length distribution is then equivalent to a waiting time problem on the PAA resulting from combining this DAA with a text model. To be more precise, it corresponds to computing the distribution of the number of steps needed to reach a value indicating that f or more flows have been used. This waiting time problem is covered by Lemma 4.2.

We define a DAA with state set $\mathcal{Q}^{\mathcal{D}} := (\{0, \dots, \ell - 1\} \times \{0, \dots, \ell - 1\}) \cup \bigcup_{i=-1}^{\ell-1} \{(-1, i)\}$ with start state $q_0 := (-1\,, -1)$ and attach the following semantics: Being in state (i, j) means that the last two sequenced nucleotides were appended in flows with indices i and j, respectively. Therefore, the last two nucleotide were $d[i]$ and $d[j]$. These semantics immediately imply a transition function. Each state (i, j) deterministically emits the number of flows needed to get from flow index i to flow index j in the cyclically interpreted dispensation order and thus equals $j - i \bmod \ell$. The operation in each state adds the emitted number of flows, truncating at $f + 1$, where f is the number of nucleotide flows specified by the sequencing protocol (typically $f = 800$ for 454 sequencing). Refer to the Appendix, available in the online supplemental material, for a formal specification of the described DAA.

Although the size of the state space is $O(\ell^2)$, there exist at most $O(|\Sigma|\ell)$ reachable states. This follows from the fact that the transition target of each state (i, j) only depends on j and the read character.

Invoking Lemma 5.4 yields the corresponding PAA for a general finite-memory text model. We are interested in the sequence length distribution up to a given length n. To this end, we define the target set $\mathcal{T} := \{f + 1\}$ and obtain the read length as one less than the waiting time

$$W_{\mathcal{T}} = \min\{t \in \mathbb{N}_0 \mid V_t = f + 1\}.$$

Applying Lemma 5.6 with $\Sigma = \mathcal{O}(1), |\mathcal{C}| = \mathcal{O}(1), |\mathcal{Q}^{\mathcal{D}}| = \mathcal{O}(\ell)$, and $\vartheta_n = \mathcal{O}(\ell n)$ results in a running time of $\mathcal{O}(n^2\,\ell^2)$ and a space requirement of $\mathcal{O}(n\ell^2)$.

As an application, we estimated a second-order Markov model from the genome of the GC-rich bacteria *S. meliloti* and computed the expected read length for all dispensation orders of length 4. The default dispensation order TACG yields an expected read length of 513.9 bp, while the optimal order TCGA yields 559.3 bp, an increase of approximately 8.9 percent. These calculations took a total of 8.6 s.

11 Conclusion

We have presented the concept of probabilistic arithmetic automata, which can be seen as Markov chains over a larger state space. Their benefit lies in their utility as a modeling technique, specifically by separating states and values in a semantically meaningful way. Many applications, especially those involving the deterministic processing of random sequences, can conveniently be approached using the PAA framework. In these cases, a PAA can be constructed from a deterministic arithmetic automaton and a finite-memory text model, a general class of text models that covers simple i.i.d. models as well as arbitrary-order Markov models and character-emitting HMMs. The PAA framework along with the generic implementations in the MoSDi software package significantly speed up the development of new applications.

Acknowledgments

The authors thank Jens Stoye, Sebastian Böcker, and Timo Stöcker. IH was supported by the NRW Graduate School in Bioinformatics and Genome Research.

References

[1] T. Marschall and S. Rahmann, "Probabilistic Arithmetic Automata and Their Application to Pattern Matching Statistics," *Proc. 19th Ann. Symp. Combinatorial Pattern Matching (CPM)*, pp. 95-106, 2008.

[2] T. Marschall and S. Rahmann, "Exact Analysis of Horspool's and Sunday's Pattern Matching Algorithms with Probabilistic Arithmetic Automata," *Proc. Fourth Int'l Conf. Language and Automata Theory and Applications (LATA)*, pp. 439-450, 2010.

[3] T. Marschall and S. Rahmann, "Efficient Exact Motif Discovery," *Bioinformatics*, vol. 25, no. 12, pp. i356-i364, June 2009.

[4] I. Herms and S. Rahmann, "Computing Alignment Seed Sensitivity with Probabilistic Arithmetic Automata," *Proc. Eighth Int'l Workshop Algorithms in Bioinformatics (WABI)*, pp. 318-329, 2008.

[5] H.-M. Kaltenbach, S. Böcker, and S. Rahmann, "Markov Additive Chains and Applications to Fragment Statistics for Peptide Mass Fingerprinting," *Proc. Joint Satellite Conf. Systems Biology and Computational Proteomics*, pp. 29-41, 2006.

[6] T. Marschall, "Construction of Minimal Deterministic Finite Automata from Biological Motifs," *Theoretical Computer Science*, vol. 412, pp. 922-930, 2011.

[7] E. Cinlar, "Markov Additive Processes I," *Z. Wahrscheinl. Verw. Geb.*, vol. 24, pp. 85-93, 1972.

[8] M. Lladser, M.D. Betterton, and R. Knight, "Multiple Pattern Matching: A Markov Chain Approach," *J. Math. Biology*, vol. 56, nos. 1/2, pp. 51-92, 2008.

[9] G. Nuel, "Pattern Markov Chains: Optimal Markov Chain Embedding through Deterministic Finite Automata," *J. Applied Probability*, vol. 45, pp. 226-243, 2008.

[10] M. Mohri, "Weighted Automata Algorithms," *Handbook of Weighted Automata*, M. Droste, W. Kuich, and H. Vogler, eds., pp. 213-254, Springer, 2009.

[11] G. Kucherov, L. Noé, and M. Roytberg, "A Unifying Framework for Seed Sensitivity and Its Application to Subset Seeds," *J. Bioinformatics Computational Biology*, vol. 4, no. 2, pp. 553-569, 2006.

[12] P. Nicodème, B. Salvy, and P. Flajolet, "Motif Statistics," *Theoretical Computer Science*, vol. 287, pp. 593-617, 2002.

[13] D.Y.F. Mak and G. Benson, "All Hits All the Time: Parameter Free Calculation of Seed Sensitivity," *Proc. Fifth Asia Pacific Bioinformatics Conf. (APBC)*, pp. 327-340, 2007.

[14] W. Feller, *An Introduction to Probability Theory and its Applications.* John Wiley & Sons, 1968.
[15] G. Reinert, S. Schbath, and M.S. Waterman, "Probabilistic and Statistical Properties of Words: An Overview," *J. Computational Biology,* vol. 7, nos. 1/2, pp. 1-46, 2000.
[16] P. Brémaud, *Markov Chains, Gibbs fields, Monte Carlo Simulation, and Queues.* Springer, 1999.
[17] M. Schulz, D. Weese, T. Rausch, A. Döring, K. Reinert, and M. Vingron, "Fast and Adaptive Variable Order Markov Chain Construction," *Proc. Eighth Int'l Workshop Algorithms in Bioinformatics (WABI),* pp. 306-317, 2008.
[18] N. Hulo, A. Bairoch, V. Bulliard, L. Cerutti, E. De Castro, P.S. Langendijk-Genevaux, M. Pagni, and C.J.A. Sigrist, "The PROSITE Database," *Nucleic Acids Research,* vol. 34, no. S1, pp. D227-D230, 2006.
[19] M. Lothaire, *Applied Combinatorics on Words (Encyclopedia of Mathematics and Its Applications).* Cambridge Univ. Press, 2005.
[20] M. Régnier, "A Unified Approach to Word Occurrence Probabilities," *Discrete Applied Math.,* vol. 104, pp. 259-280, 2000.
[21] V. Boeva, J. Clément, M. Régnier, M.A. Roytberg, and V.J. Makeev, "Exact P-Value Calculation for Heterotypic Clusters of Regulatory Motifs and Its Application in Computational Annotation of Cis-Regulatory Modules," *Algorithms for Molecular Biology,* vol. 2, article 13, Oct. 2007.
[22] J. Zhang, B. Jiang, M. Li, J. Tromp, X. Zhang, and M.Q. Zhang, "Computing Exact P-Values for DNA Motifs," *Bioinformatics,* vol. 23, no. 5, pp. 531-537, 2007.
[23] G. Navarro and M. Raffinot, *Flexible Pattern Matching in Strings.* Cambridge Univ. Press, 2002.
[24] J. Hopcroft, "An $n \log n$ Algorithm for Minimizing the States in a Finite Automaton," *The Theory of Machines and Computations,* Z. Kohavi and A. Paz, eds., pp. 189-196, Academic Press, 1971.
[25] T. Knuutila, "Re-Describing An Algorithm by Hopcroft," *Theoretical Computer Science,* vol. 250, pp. 333-363, 2001.
[26] A.V. Aho and M.J. Corasick, "Efficient String Matching: An Aid to Bibliographic Search," *Comm. ACM,* vol. 18, no. 6, pp. 333-340, 1975.
[27] S. Dori and G.M. Landau, "Construction of Aho Corasick Automaton in Linear Time for Integer Alphabets," *Information Processing Letters,* vol. 98, no. 2, pp. 66-72, 2006.
[28] S. Schbath, "Compound Poisson Approximation of Word Counts in DNA Sequences," *ESAIM: Probability and Statistics,* vol. 1, pp. 1-16, 1995.
[29] M.S. Waterman, *Introduction to Computational Biology: Maps, Sequences and Genomes.* Chapman & Hall/CRC, 1995.
[30] E. Roquain and S. Schbath, "Improved Compound Poisson Approximation for the Number of Occurrences of Multiple Words in a Stationary Markov Chain," *Advances in Applied Probability,* vol. 39, no. 1, pp. 128-140, 2007.
[31] D.E. Knuth, J. Morris, and V.R. Pratt, "Fast Pattern Matching in Strings," *SIAM J. Computing,* vol. 6, no. 2, pp. 323-350, 1977.
[32] R.S. Boyer and J.S. Moore, "A Fast String Searching Algorithm," *Comm. ACM,* vol. 20, no. 10, pp. 762-772, 1977.
[33] R.N. Horspool, "Practical Fast Searching in Strings," *Software-Practice and Experience,* vol. 10, pp. 501-506, 1980.
[34] M. Crochemore, A. Czumaj, L. Gasieniec, S. Jarominek, T. Lecroq, W. Plandowski, and W. Rytter, "Speeding Up Two String-Matching Algorithms," *Algorithmica,* vol. 12, nos. 4/5, pp. 247-267, 1994.
[35] C. Allauzen, M. Crochemore, and M. Raffinot, "Efficient Experimental String Matching by Weak Factor Recognition," *Proc. 12th Ann. Symp. Combinatorial Pattern Matching (CPM),* pp. 51-72, 2001.
[36] T. Marschall and S. Rahmann, "An Algorithm to Compute the Character Access Count Distribution for Pattern Matching Algorithms," *Algorithms,* vol. 4, pp. 285-306, 2011.
[37] R.A. Baeza-Yates, G.H. Gonnet, and M. Régnier, "Analysis of Boyer-Moore-Type String Searching Algorithms," *Proc. First Ann. ACM-SIAM Symp. Discrete Algorithms (SODA),* pp. 328-343, 1990.
[38] R.A. Baeza-Yates and M. Régnier, "Average Running Time of the Boyer-Moore-Horspool Algorithm," *Theoretical Computer Science,* vol. 92, no. 1, pp. 19-31, 1992.
[39] H.M. Mahmoud, R.T. Smythe, and M. Régnier, "Analysis of Boyer-Moore-Horspool String-Matching Heuristic," *Random Structures and Algorithms,* vol. 10, nos. 1/2, pp. 169-186, 1997.
[40] R.T. Smythe, "The Boyer-Moore-Horspool Heuristic with Markovian Input," *Random Structures and Algorithms,* vol. 18, no. 2, pp. 153-163, 2001.
[41] T.-H. Tsai, "Average Case Analysis of the Boyer-Moore Algorithm," *Random Structures and Algorithms,* vol. 28, no. 4, pp. 481-498, 2006.
[42] W. Pearson and D. Lipman, "Improved Tools for Biological Sequence Comparison," *Proc. Nat'l Academy of Sciences USA,* vol. 85, pp. 2444-2448, 1988.
[43] S.F. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic Local Alignment Search Tool," *J. Molecular Biology,* vol. 215, pp. 403-410, 1990.
[44] W.J. Kent, "BLAT-the BLAST-Like Alignment Tool," *Genome Research,* vol. 12, no. 4, pp. 656-664, 2002.
[45] B. Ma, J. Tromp, and M. Li, "PatternHunter - Faster and More Sensitive Homology Search," *Bioinformatics,* vol. 18, pp. 440-445, 2002.
[46] J. Buhler, U. Keich, and Y. Sun, "Designing Seeds for Similarity Search in Genomic DNA," *Proc. Seventh Ann. Int'l Conf. Research in Computational Molecular Biology (RECOMB),* pp. 67-75, 2003.
[47] K.P. Choi and L. Zhang, "Sensitivity Analysis and Efficient Method for Identifying Optimal Spaced Seeds," *J. Computer System Sciences,* vol. 68, pp. 22-40, 2004.
[48] M. Li, B. Ma, and L. Zhang, "Superiority and Complexity of the Spaced Seeds," *Proc. 17th Ann. ACM-SIAM Symp. Discrete Algorithm (SODA),* pp. 444-453, 2006.
[49] B. Brejová, D.G. Brown, and T. Vinar, "Optimal Spaced Seeds for Homologous Coding Regions," *J. Bioinformatics Computational Biology,* vol. 1, no. 4, pp. 595-610, 2004.
[50] K.P. Choi, F. Zeng, and L. Zhang, "Good Spaced Seeds for Homology Search," *Bioinformatics,* vol. 20, no. 7, pp. 1053-1059, 2004.
[51] B. Brejová, D.G. Brown, and T. Vinar, "Vector Seeds: An Extension to Spaced Seeds," *J. Computer System Sciences,* vol. 70, no. 3, pp. 364-380, 2005.
[52] D. Mak, Y. Gelfand, and G. Benson, "Indel Seeds for Homology Search," *Bioinformatics,* vol. 22, no. 14, pp. e341-e349, 2006.
[53] M. Li, B. Ma, D. Kisman, and J. Tromp, "PatternHunter II: Highly Sensitive and Fast Homology Search," *J. Bioinformatics Computational Biology,* vol. 2, no. 3, pp. 417-439, 2004.
[54] G. Kucherov, L. Noé, and M. Roytberg, "Multiseed Lossless Filtration," *IEEE/ACM Trans. Computational Biology Bioinformatics,* vol. 2, no. 1, pp. 51-61, Jan.-Mar. 2005.
[55] Y. Sun and J. Buhler, "Designing Multiple Simultaneous Seeds for DNA Similarity Search," *J. Computational Biology,* vol. 12, no. 6, pp. 847-861, 2005.
[56] H.-M. Kaltenbach, "Statistics and Algorithms for Peptide Mass Fingerprinting," PhD dissertation, Bielefeld Univ., 2007.
[57] T.U. Consortium, "Reorganizing the Protein Space at the Universal Protein Resource (UniProt)," *Nucleic Acids Research,* vol. 40, no. D1, pp. D71-D75, 2012.
[58] I.-J. Wang, C.P. Diehl, and F.J. Pineda, "A Statistical Model of Proteolytic Digestion," *Proc. IEEE CS Conf. Bioinformatics (CSB),* pp. 506-508, 2003.
[59] S. Rahmann, "Subsequence Combinatorics and Applications to Microarray Production, DNA Sequencing and Chaining Algorithms," *Proc. 17th Ann. Conf. Combinatorial Pattern Matching (CPM),* pp. 153-164, 2006.
[60] Y. Kong, "Statistical Distributions of Pyrosequencing," *J. Computational Biology,* vol. 16, no. 1, pp. 31-42, 2009.

Tobias Marschall studied computer science and mathematics at Bielefeld University and graduated in 2007. He received the PhD degree in computer science from TU Dortmund in 2011 for the thesis on algorithms and statistical methods for exact motif discovery. He is currently a postdoctoral researcher at Centrum Wiskunde & Informatica (CWI) in Amsterdam, The Netherlands. His research interests include statistical modeling in computational biology, sequence analysis, combinatorial optimization, and in particular the analysis of next-generation sequencing data.

Inke Herms studied biomathematics at Greifswald University from 2000 until 2005. From 2005 to 2009, she was working toward the PhD degree at the Graduate School for Bioinformatics and Genome Research, Bielefeld University, working on stochastic models in sequence analysis, where she graduated in October 2009. She then worked as a postdoctoral researcher in the Genome Informatics group at Bielefeld University focusing on statistical analysis of metagenomic samples. She is now a software developer in accounting and business intelligence.

Hans-Michael Kaltenbach studied mathematics and computer science with a minor in biological medicine at Hannover University and Medical University Hannover, Germany, where he received the diploma in mathematics in 2003. He receied the PhD degree in bioinformatics at the International Graduate School in Bioinformatics and Genome Research at Bielefeld University, Germany, in 2006 on algorithms and statistics for mass spectrometry. He then completed a year of postdoc training at the Institut Pasteur, Paris, France in 2007/2008 before joining the Computational Systems Biology group of Jörg Stelling at ETH Zurich in 2008. Since 2011, he is a senior assistant and lecturer in the same group, working on mathematics and algorithms for analysis of dynamic signaling network models arising in systems biology.

Sven Rahmann studied mathematics and computer science with a focus on statistical methods in bioinformatics at the universities of Göttingen, University of California at Santa Cruz, and Heidelberg. He wrote the doctoral thesis on oligonucleotide design for microarrays in the Computational Molecular Biology group at the Max Planck Institute for Molecular Genetics in Berlin. He was an independent Junior Research Group leader at Bielefeld University, spent four months at HHMI Janelia Farm Research Campus as a visiting scientist, and was a professor for bioinformatics for high-throughput technologies at the Chair of Algorithm Engineering, Computer Science Department, TU Dortmund. Presently, he holds the chair of Genome Informatics at the Faculty of Medicine at the University of Duisburg-Essen. He also leads a project on resource-constrained analysis of spectrometry data within the DFG collaborative research center (Sonderforschungsbereich) 876: "Providing Information by resource-constrained data analysis."