HAL open science

Sparse approaches for the exact distribution of patterns in long multi-states sequences generated by a Markov source

Grégory Nuel, Jean-Guillaume Dumas

To cite this version:

Grégory Nuel, Jean-Guillaume Dumas. Sparse approaches for the exact distribution of patterns in long multi-states sequences generated by a Markov source. 2010. hal-00492738v1

HAL Id: hal-00492738

https://hal.science/hal-00492738v1

Preprint submitted on 16 Jun 2010 (v1), last revised 5 Jun 2012 (v4)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Sparse approaches for the exact distribution of patterns in long multi-states sequences generated by a Markov source*

Gregory Nuel† Jean-Guillaume Dumas ‡

June 16, 2010

Abstract

We present two novel approaches for the computation of the exact distribution of a pattern in a long sequence. Both approaches take into account the sparse structure of the problem. The first approach relies on a partial recursion computing the largest eigenvalue of the the transition matrix of a Markov chain embedding. The second approach uses fast Taylor expansions of an exact bivariate rational reconstruction of the distribution.

We illustrate the interest of both approaches on a simple toy-example and two biological applications: the transcription factors of the Human Chromosome 5 and the PROSITE signatures of functional motifs in proteins. On these examples our methods demonstrate their complementarity and their hability to extend the domain of feasibility for exact computations in pattern problems to a new level.

1 Introduction

The distribution of patterns in multi-states random sequences generated by a Markov sources have many applications in a wide range of fields including (but not limited to) reliability, insurance, communication systems, pattern matching, or bioinformatics. In the latter field in particular, the detection of statistically exceptional DNA or protein patterns (PROSITE signatures, CHI motifs, regulation patterns, binding sites, etc.) have been very successful allowing both to confirm biological known signals as well as new discoveries. Here follows a short selection of such work: Karlin et al. (1992); van Helden et al. (1998); Brazma

*Part of this work was done while the second author was visiting the Claude Shannon Institute of the University College Dublin, Ireland, under a CNRS grant.

†MAP5, UMR CNRS 8145, Department of Applied Mathematics, Paris Descartes University, France, Gregory.Nuel@parisdescartes.fr.

‡Laboratoire J. Kuntzmann, Université de Grenoble. 51, rue des Mathématiques, umr CNRS 5224, bp 53X, F38041 Grenoble, France, Jean-Guillaume.Dumas@imag.fr.

et al. (1998); El Karoui et al. (1999); Beaudoing et al. (2000); Frith et al. (2002); Hampson et al. (2002); Leonardo Mariño-Ramírez and Landsman (2004).

From the technical point of view, obtaining the distribution of a pattern count in a multi-state random sequence is a difficult problem which have drawn a considerable interest from the probabilist, combinatorics and computer science community over the last fifty years. Many concurrent approaches have been suggested, all of them having their own strengths and weakness, and we give here only a short selection of the corresponding references (see Reignier, 2000; Lothaire, 2005; Nuel, 2006b, for more comprehensive reviews).

Exact methods are based on a wide range of techniques like Markov chain embedding, moment generating functions, combinatorial methods, or exponential families Fu (1996); Stefanov and Pakes (1997); Antzoulakos (2001); Chang (2005); Boeva et al. (2005); Nuel (2006a); Stefanov and Szpankowski (2007); Boeva et al. (2007). There is also a wide range of asymptotic approximations, the most popular among them being: Gaussian approximations Pevzner et al. (1989); Cowan (1991); Kleffe and Borodovski (1997); Prum et al. (1995), Poisson approximations Godbole (1991); Geske et al. (1995); Reinert and Schbath (1999); Erhardsson (2000) and Large deviations approximations Denise et al. (2001); Nuel (2004).

More recently, several authors (Nicodème et al., 2002; Crochemore and Stefanov, 2003; Lladser, 2007; Nuel, 2008; Ribeca and Raineri, 2008) pointed out the strong connexion of the problem to the theory of pattern matching by providing the optimal Markov chain embedding of any pattern problem through minimal Deterministic Finite state Automata (DFA). However, this approach remains difficult to apply in practice when considering high complexity patterns and/or long sequences.

In this paper, we want to address this problem by suggesting two efficient ways to obtain the distribution of any pattern of interest in a (possibly long) homogeneous multi-states Markov sequence.

The paper is organized as follow. In a first part, we recall the principles of optimal Markov chain embedding through DFA, as well as the associated moment-generating function (mgf) formulas. We then present a new algorithm using partial recursion formulas. The next part take advantage of state-of-the-art results in exact computation to suggest a very efficient way to obtain the bivariate mgf of the problem through rational reconstruction and then to perform the necessary Taylor expansions using high-order liftings. We then apply our new algorithms successively to a simple toy-example, a selection of DNA (Transcription Factors) patterns, and to protein motifs (PROSITE signature). In all cases, the relative performance of the two algorithms are presented and discussed, highlighting their strengths and weakness. We conclude with some perspectives of this work.

2 DFA and optimal Markov chain embedding

2.1 Automata and languages

In this part we recall some classical definitions and results of the well known theory of languages and automata (Hopcroft et al., 2001).

We consider $\mathcal{A}$ a *finite alphabet* which elements are called *letters*. A *word* (or *sequence*) over $\mathcal{A}$ is a sequence of letters and a *language* over $\mathcal{A}$ is a set of words (finite or not). We denote by ε the *empty word*. For example $\mathtt{ABBABA}$ is a word over the binary alphabet $\mathcal{A} = \{\mathtt{A}, \mathtt{B}\}$ and $\mathcal{L} = \{\mathtt{AB}, \mathtt{ABBABA}, \mathtt{BBBBB}\}$ is a (finite) language over $\mathcal{A}$.

The *product* $\mathcal{L}_1 \cdot \mathcal{L}_2$ (the dot could be omitted) of two languages is the language $\{w_1 w_2, w_1 \in \mathcal{L}_1, w_2 \in \mathcal{L}_2\}$ where $w_1 w_2$ is the concatenation – or product – of w_1 and w_2. If $\mathcal{L}$ is a language, $\mathcal{L}^n = \{w_1 \dots w_n \text{ with } w_1, \dots w_n \in \mathcal{L}\}$ and the *star closure* of $\mathcal{L}$ is defined by $\mathcal{L}^* = \cup_{n \geqslant 0} \mathcal{L}^n$. The language $\mathcal{A}^*$ is hence the set of all possible words over $\mathcal{A}$. For example we have $\{\mathtt{AB}\} \cdot \{\mathtt{ABBABA}, \mathtt{BBBBB}\} = \{\mathtt{ABABBABA}, \mathtt{ABBBBBB}\}$; $\{\mathtt{AB}\}^3 = \{\mathtt{ABABAB}\}$ and $\{\mathtt{AB}\}^* = \{\varepsilon, \mathtt{AB}, \mathtt{ABAB}, \mathtt{ABABAB}, \mathtt{ABABABAB} \dots\}$.

A *regular language* is either the empty word, or a single letter, or obtained by a finite number of *regular operations* (namely: union, product and star closure). A finite sequence of regular operations describing a regular language is called a *regular expression* which size is defined as its number of operations. $\mathcal{A}^*$ is regular. Any finite language is regular.

Definition 1. *If $\mathcal{A}$ a finite alphabet, $\mathcal{Q}$ a finite set of states, $\sigma \in \mathcal{Q}$ a starting state, $\mathcal{F} \subset \mathcal{Q}$ a subset of final states and $\delta : \mathcal{Q} \times \mathcal{A} \to \mathcal{Q}$ a transition function then $(\mathcal{A}, \mathcal{Q}, \sigma, \mathcal{F}, \delta)$ is a* Deterministic Finite Automaton *(DFA). For all $a_1^d = a_1 \dots a_{d-1} a_d \in \mathcal{A}^d$ ($d \geqslant 2$) and $q \in \mathcal{Q}$ we recursively define $\delta(q, a_1^d) = \delta(\delta(q, a_1^{d-1}), a_d)$. A word $w \in \mathcal{A}^h$ is is* accepted *(or* recognized*) by the DFA if $\delta(s, w) \in \mathcal{F}$. The set of all words accepted by a DFA is called its language. See in Figure 1 a graphical representation of a DFA.*

We can now give the most important result of this part which is a simple application of the classical Kleene and Rabin & Scott theorems (Hopcroft et al., 2001):

Theorem 2. *For any regular language $\mathcal{L}$ there exists a unique (up to a unique isomorphism) smallest DFA which language is $\mathcal{L}$. If we denote by E the size of the regular expression corresponding to $\mathcal{L}$, then the size R of the smallest DFA is bounded by 2^E.*

For certain specific patterns (ex: $\mathcal{A}^* w$ where w is a simple word), a minimal DFA can be built directly using ad hoc construction or well-known algorithms (ex: Aho-Corasik algorithm). In general however, building a minimal DFA from a regular expression usually requires three steps: 1) turning the regular expression into Nondeterministic Finite Automaton – NFA – (Thompson's algorithm); 2) producing a DFA from the NFA (determinization); 3) minimizing the DFA

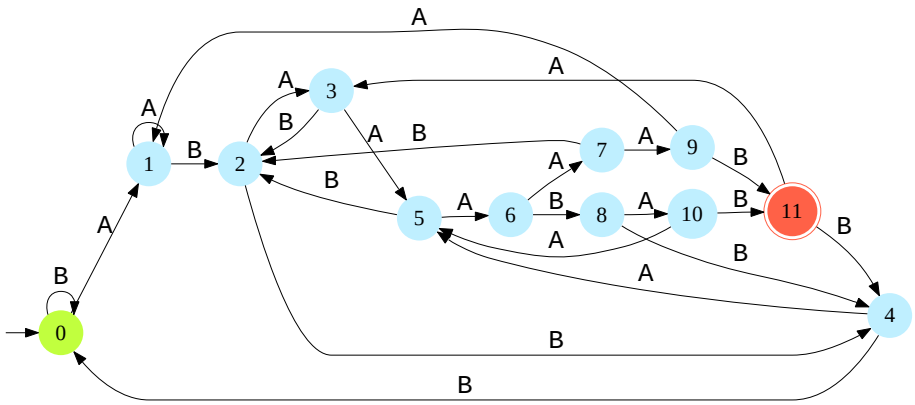

Figure 1: Graphical representation of the DFA $(\mathcal{A}, \mathcal{Q}, \sigma, \mathcal{F}, \delta)$ with $\mathcal{A} = \{\mathtt{A}, \mathtt{B}\}$, $\mathcal{Q} = \{0, 1, 2, \ldots, 10, 11\}$, $s = 0$, $\mathcal{F} = \{11\}$ and $\delta(0, \mathtt{A}) = 1$, $\delta(0, \mathtt{B}) = 0$, $\delta(1, \mathtt{A}) = 1$, $\delta(1, \mathtt{B}) = 2$, $\delta(2, \mathtt{A}) = 3$, $\delta(2, \mathtt{B}) = 4$, $\delta(3, \mathtt{A}) = 5$, $\delta(3, \mathtt{B}) = 1$, $\delta(4, \mathtt{A}) = 5$, $\delta(4, \mathtt{B}) = 0$, $\delta(5, \mathtt{A}) = 6$, $\delta(5, \mathtt{B}) = 2$, $\delta(6, \mathtt{A}) = 7$, $\delta(6, \mathtt{B}) = 8$, $\delta(7, \mathtt{A}) = 9$, $\delta(7, \mathtt{B}) = 2$, $\delta(8, \mathtt{A}) = 10$, $\delta(8, \mathtt{B}) = 4$, $\delta(9, \mathtt{A}) = 1$, $\delta(9, \mathtt{B}) = 11$, $\delta(10, \mathtt{A}) = 5$, $\delta(10, \mathtt{B}) = 11$, $\delta(11, \mathtt{A}) = 3$ and $\delta(11, \mathtt{B}) = 4$. This DFA is the smallest one that recognize the language $\mathcal{L} = \mathcal{A}^*\mathcal{W}_1$ with $\mathcal{A} = \{\mathtt{A}, \mathtt{B}\}$, $\mathcal{W}_1 = \mathtt{ab}\mathcal{A}^1\mathtt{AA}\mathcal{A}^1\mathtt{AB}$ and hence $|\mathcal{W}_1| = 4$.

k	1	2	3	4	5	6	7	8	9	10
$\lvert\mathcal{W}_k\rvert$	4	16	64	256	1 024	4 096	16 384	65 536	262 144	1 048 576
2^E	512	2 048	8 192	32 768	1.3×10^5	5.2×10^5	2.1×10^6	8.4×10^6	3.4×10^7	1.3×10^8
R	12	27	57	122	262	562	1 207	2 592	5 567	11 957
F	1	3	6	13	28	60	129	277	595	1 278

Table 1: Characteristics of the smallest DFA that recognizes the language $\mathcal{L} = \mathcal{A}^*\mathcal{W}_k$ with $\mathcal{A} = \{\mathtt{A}, \mathtt{B}\}$ and $\mathcal{W}_k = \mathtt{AB}\mathcal{A}^k\mathtt{AA}\mathcal{A}^k\mathtt{AB}$. The pattern cardinality is $|\mathcal{W}_k| = 2^k \times 2^k = 4^k$, R is the total number of states, F the number of final states, and $2^E = 2^{7+2k}$ is the theoretical upper bound of L.

by removing unnecessary states (minimization). See Hopcroft et al. (2001) for more details.

As stated in Theorem 2, the smallest DFA may have a total of 2^E states in the worse case. However, this upper bound is seldom reached in practice. This may be observed in Table 1 where the practical value of R is far below the upper bound.

2.2 Connexion with patterns

We call *pattern* (or *motif*) over the finite alphabet $\mathcal{A}$ any regular language (finite or not) over the same alphabet. For any pattern $\mathcal{W}$ any DFA that recognizes the regular language $\mathcal{A}^*\mathcal{W}$ is said to be *associated* with $\mathcal{W}$. According to Theorem 2, there exists a unique (up to unique isomorphism) smallest DFA associated with a given pattern. For example, if we work with the binary alphabet $\mathcal{A} = \{\mathtt{A}, \mathtt{B}\}$ then the smallest DFA associated with Pattern $\mathcal{W}_1 = \mathtt{AB}\mathcal{A}^1\mathtt{AA}\mathcal{A}^1\mathtt{AB}$ has $R = 12$ states and $F = 1$ final state (see Figure 1), and the smallest DFA associated with Pattern $\mathcal{W}_1 = \mathtt{AB}\mathcal{A}^1\mathtt{AA}\mathcal{A}^1\mathtt{AB}$ has $R = 27$ states and $F = 3$ final states (see Figure 2).

It is well known from the pattern matching theory (Cormen et al., 1990; Crochemore and Hancart, 1997) that such a DFA provides a simple way to find all occurrences of the corresponding pattern in a sequence.

Proposition 3. *Let $\mathcal{W}$ be a pattern over the finite alphabet $\mathcal{A}$ and $(\mathcal{A}, \mathcal{Q}, \sigma, \mathcal{F}, \delta)$ a DFA that is associated to $\mathcal{W}$. For all deterministic sequence $x_1^\ell = x_1x_2\ldots x_\ell$ over $\mathcal{A}$, we recursively define the sequence $y_0^\ell = y_0y_1\ldots y_\ell$ over $\mathcal{Q}$ by $y_0 = \sigma$ and $y_i = \delta(y_{i-1}, x_i)$. For all $1 \leqslant i \leqslant \ell$ we then have then the following property* [1]*: $x_1^i \in \mathcal{A}^*\mathcal{W} \iff y_i \in \mathcal{F}$.*

Proof. Since the DFA is associated to $\mathcal{W}$, $x_1^i \in \mathcal{A}^*\mathcal{W}$ is equivalent to $\delta(\sigma, x_1^i) \in \mathcal{F}$. One can then easily show by induction that $\delta(\sigma, x_1^i) = y_i$ and the proof is achieved. □

Example 4. *Let us consider the pattern $\mathcal{W}_1 = \mathtt{ab}\mathcal{A}^1\mathtt{aa}\mathcal{A}^1\mathtt{ab}$ over the binary alphabet $\mathcal{A} = \{\mathtt{A}, \mathtt{B}\}$. Its smallest associated DFA is represented in Figure 1. If*

[1] $x_1^i \in \mathcal{A}^*\mathcal{W}$ means that an occurrence of $\mathcal{W}$ ends in position i in x_1^ℓ.

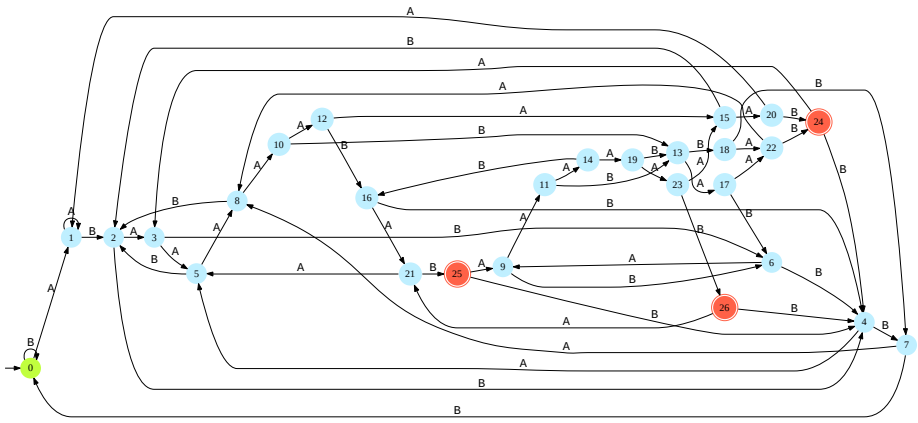

Figure 2: Graphical representation of the smallest DFA associated with $\mathcal{W}_2 = \mathtt{AB}\mathcal{A}^2\mathtt{AA}\mathcal{A}^2\mathtt{AB}$ with $\mathcal{A} = \{\mathtt{A}, \mathtt{B}\}$. This DFA has $L = 27$ states including $F = 3$ final states.

$x_1^{20} = \mathtt{ABAAABBAAAABBAABABAB}$ *is a binary sequence, we build the sequence* y_0^{20} *according to Proposition 3 and we get:*

$x_1^{20} =$	–	A	B	A	A	A	B	B	A	A	A	A	B	B	A	A	B	A	B	A	B
$y_0^{20} =$	0	1	2	3	5	6	8	4	5	6	7	9	**11**	4	5	6	8	10	**11**	3	2

where final states are in bold. We hence see two occurrences of $\mathcal{W}_1$*: one ending in position 12 (*ABBAAAAB*) and one in position 18 (*ABBAABAB*, which overlaps the previous occurrence).*

If this approach may be useful to localize pattern occurrences in deterministic sequences, one should note that NFA (Nondeterministic Finite Automata) are usually preferred over DFA for such a task since they are far more memory efficient and can achieve similar speed thanks to lazy construction algorithm. The DFA-based approach however have a great advantage when we work with random sequences.

2.3 Markov chain embedding

Let $(\mathcal{A}, \mathcal{Q}, \sigma, \mathcal{F}, \delta)$ be a *minimal* DFA associated to $\mathcal{W}$. We additionally assume that this automaton is non m-ambiguous (a DFA having this property is also called a m-th order DFA in Lladser, 2007) which means that for all $q \in \mathcal{Q}$, $\delta^{-m}(p) \stackrel{\text{def}}{=} \{a_1^m \in \mathcal{A}_1^m, \exists p \in \mathcal{Q}, \delta(p, a_1^m) = q\}$ is either a singleton, or the empty set. When the notation is not ambiguous, $\delta^{-m}(p)$ may also denotes its unique element (singleton case). We then have the following result:

Theorem 5. *We consider the random sequence over $\mathcal{Q}$ defined by $Y_0 \stackrel{def}{=} \sigma$ and $Y_i \stackrel{def}{=} \delta(Y_{i-1}, X_i)$, $\forall i, 1 \leqslant i \leqslant \ell$. Then $(Y_i)_{i \geqslant m}$ is a heterogeneous order 1 Markov chain over $\mathcal{Q}' \stackrel{def}{=} \delta(s, \mathcal{A}^m \mathcal{A}^*)$ such as, for all $p, q \in \mathcal{Q}'$ and $1 \leqslant i \leqslant \ell - m$, the starting distribution $\mu_m(p) \stackrel{def}{=} \mathbb{P}(Y_m = p)$ and the transition matrix $T_{i+m}(p, q) \stackrel{def}{=} \mathbb{P}(Y_{i+m} = q | Y_{i+m-1} = p)$ are given by:*

$$\mu_m(p) = \begin{cases} \nu(\delta^{-m}(p)) & \text{if } \delta^{-m}(p) \neq \emptyset \\ 0 & \text{otherwise} \end{cases} ; \tag{1}$$

$$T_{i+m}(p, q) = \begin{cases} \pi_{i+m}(\delta^{-m}(p), b) & \text{if } \exists b \in \mathcal{A}, \delta(p, b) = q \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

and we have the following property:

$$X_1^i \in \mathcal{A}^* \mathcal{W} \iff Y_i \in \mathcal{F}. \tag{3}$$

Proof. The result is immediate considering the properties of the DFA and Proposition 3. See Lladser (2007) or Nuel (2008) for more details. □

Corollary 6. *With the same hypothesis and notations than in Theorem 5, the moment generating function of N_ℓ is then explicitly given by:*

$$G_\ell(y) = \sum_{n \geqslant 0} \mathbb{P}(N_\ell = n) y^n = \mathbf{u}(\mathbf{P} + y\mathbf{Q})^{\ell - m} \mathbf{v} \tag{4}$$

and we also have:

$$G(y, z) = \sum_{\ell \geqslant m} \sum_{n \geqslant 0} \mathbb{P}(N_\ell = n) = \sum_{\ell \geqslant m} G_\ell(y) z^\ell = \mathbf{u}(\mathbf{I} - z\mathbf{P} - yz\mathbf{Q})^{-1} \mathbf{v} z^m. \tag{5}$$

Proof. It is clear that $\mathbf{u}\mathbf{T}^\ell$ give the marginal distribution of Y_ℓ. If we now introduce the dummy variable y to keep track of the number of pattern occurrences, then $\mathbf{u}(\mathbf{P} + y\mathbf{Q})^\ell$ gives the joint distribution of (Y_ℓ, N_ℓ). Marginalizing over Y_ℓ through the product with $\mathbf{v}$ hence achieves the proof of Equation (4). Equation (5) is then obtained simply by exploiting the relation $\sum_{k \geqslant 0} \mathbf{M}^k = (\mathbf{I} - \mathbf{M})^{-1}$ with $\mathbf{M} = z(\mathbf{P} + y\mathbf{Q})$. □

Example 7. *Considering the same pattern and associated DFA than in Example 4, one can directly apply Theorem 5 to get the expression of $\mathbf{T}$, the transition*

matrix of Y_0^ℓ *over* $\mathcal{Q} = \{0,1,2,3,4,5,6,7,8,9,10,11\}$*:*

$$\mathbf{T} = \begin{pmatrix}
\pi_{\mathtt{B,B}} & \pi_{\mathtt{B,A}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \pi_{\mathtt{A,A}} & \pi_{\mathtt{A,B}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \pi_{\mathtt{B,A}} & \pi_{\mathtt{B,B}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \pi_{\mathtt{A,B}} & 0 & \pi_{\mathtt{A,A}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\pi_{\mathtt{B,B}} & 0 & 0 & 0 & 0 & \pi_{\mathtt{B,A}} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \pi_{\mathtt{A,B}} & 0 & 0 & 0 & \pi_{\mathtt{A,A}} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \pi_{\mathtt{A,A}} & \pi_{\mathtt{A,B}} & 0 & 0 & 0 \\
0 & 0 & \pi_{\mathtt{A,B}} & 0 & 0 & 0 & 0 & 0 & 0 & \pi_{\mathtt{A,A}} & 0 & 0 \\
0 & 0 & 0 & 0 & \pi_{\mathtt{A,B}} & 0 & 0 & 0 & 0 & 0 & \pi_{\mathtt{A,A}} & 0 \\
0 & \pi_{\mathtt{A,A}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \pi_{\mathtt{A,B}} \\
0 & 0 & 0 & 0 & 0 & \pi_{\mathtt{A,A}} & 0 & 0 & 0 & 0 & 0 & \pi_{\mathtt{A,B}} \\
0 & 0 & 0 & \pi_{\mathtt{B,A}} & \pi_{\mathtt{B,B}} & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

where $\pi_{a,b} = \mathbb{P}(X_2 = b | X_1 = a)$ *for all* $a, b \in \{\mathtt{A}, \mathtt{B}\}$.

3 Partial recursion

We want here to focus directly on the expression of $G_\ell(y)$ is Equation (4) rather than exploiting the bivariate expression $G(y,z)$ of Equation (5). A straightforward approach consists in computing recursively $\mathbf{u}(\mathbf{P}+y\mathbf{Q})^i$ (or conversely $(\mathbf{P}+y\mathbf{Q})^i\mathbf{v}$) up to $i = \ell - m$ taking full advantage of the sparse structure of matrices $\mathbf{P}$ and $\mathbf{Q}$ at each step. This is typically what is suggested in Nuel (2006a). The resulting complexity to compute $\mathbb{P}(N_\ell = n)$ is then $O(\Omega \times n \times \ell)$ in time and $O(\Omega \times n)$ in space, where $\Omega = R \times |\mathcal{A}|$ is the number of non-zero terms in $\mathbf{P} + \mathbf{Q}$. This straightforward (but effective) approach is easy to implement and is from now referred as the "full recursion" Algorithm.

Another appealing idea is two compute directly the matrix $(\mathbf{P} + y\mathbf{Q})^{\ell-m}$ using a classical dyadic decomposition of $\ell - m$. This is typically the approach suggested by Ribeca and Raineri (2008). The resulting complexity to obtain $\mathbb{P}(N_\ell = n)$ is $O(R^3 \times n^2 \times \log \ell)$ in time and $O(R^2 \times n \times \log \ell)$ in space. The algorithm can be further refined by using FFT-polynomial products (and a very careful implementation) in order two replace the quadratic complexity in n by $O(n \log n)$. The resulting algorithm however suffers from numerical instabilities when considering the tail distribution events and is therefore not suitable for computing extreme p-values. If this approach might offer interesting performance for relatively small values of R and n, its main drawback is that it totally ignore the sparse structure of the matrices and therefore fail to deal with highly complex patterns (large R).

We want here to introduce another approach that fully take advantage of the sparsity of the problem to display a linear complexity with R (like with the full recursion) but also dramatically reduce the complexity with the sequence length ℓ in order to be suitable for large scale problems.

From now on, let us assume that $\mathbf{P}$ is an irreducible and aperiodic matrix

Input: The matrices $\mathbf{P}$ and $\mathbf{Q}$, the vectors $\mathbf{u}$ and $\mathbf{v}$, $K+1$ column-vectors $\boldsymbol{\Delta}_0, \boldsymbol{\Delta}_1, \ldots, \boldsymbol{\Delta}_K$ of dimension L, $K+1$ real numbers $R_0, R_1, \ldots, R_K$, and one real number C

// Compute λ through the power method
$\boldsymbol{\Delta}_0 \leftarrow \mathbf{v}$, $\lambda \leftarrow 1$
while λ has not converged with relative precision ε **do**
  $\lambda \leftarrow \mathbf{P}\boldsymbol{\Delta}_0/\boldsymbol{\Delta}_0$ (point-wise division) and $\boldsymbol{\Delta}_0 \leftarrow \mathbf{P}\boldsymbol{\Delta}_0$
// Normalize $\mathbf{P}$ and $\mathbf{Q}$
$\mathbf{P} \leftarrow \mathbf{P}/\lambda$ and $\mathbf{Q} \leftarrow \mathbf{Q}/\lambda$
// Compute α such as $\mathbf{C}_K = \mathbf{D}_K^K(\alpha)$
$\boldsymbol{\Delta}_0 \leftarrow \mathbf{v}$
for $i = 1 \ldots K$ **do**
  for $k = i \ldots 1$ **do**
    $\boldsymbol{\Delta}_k \leftarrow \mathbf{P}\boldsymbol{\Delta}_k + \mathbf{Q}\boldsymbol{\Delta}_{k-1} - \boldsymbol{\Delta}_k$ *// so that $\boldsymbol{\Delta}_k$ now contains $\mathbf{D}_k^i(i)$*
  $\boldsymbol{\Delta}_0 \leftarrow \mathbf{P}\boldsymbol{\Delta}_0 - \boldsymbol{\Delta}_0$ *// so that $\boldsymbol{\Delta}_0$ now contains $\mathbf{D}_0^i(i)$*
for $i = K+1 \ldots \ell - m$ **do**
  for $k = K \ldots 1$ **do**
    $\boldsymbol{\Delta}_k \leftarrow \mathbf{P}\boldsymbol{\Delta}_k + \mathbf{Q}\boldsymbol{\Delta}_{k-1}$ *// so that $\boldsymbol{\Delta}_k$ now contains $\mathbf{D}_k^K(i)$*
  $\boldsymbol{\Delta}_0 \leftarrow \mathbf{P}\boldsymbol{\Delta}_0$ *// so that $\boldsymbol{\Delta}_0$ now contains $\mathbf{D}_0^K(i)$*
  if $\boldsymbol{\Delta}_K$ as converged towards $\mathbf{C}_K$ with relative precision η **then**
    $\alpha \leftarrow i$ and **break**
// Compute all $\mathbf{D}_k^0(\alpha - K)$ for $0 \leqslant k \leqslant K$
$\boldsymbol{\Delta}_0 \leftarrow \mathbf{v}$
for $i = 1 \ldots \alpha - K$ **do**
  for $k = K \ldots 1$ **do**
    $\boldsymbol{\Delta}_k \leftarrow \mathbf{P}\boldsymbol{\Delta}_k + \mathbf{Q}\boldsymbol{\Delta}_{k-1}$ *// so that $\boldsymbol{\Delta}_k$ now contains $\mathbf{D}_k^0(i)$*
  $\boldsymbol{\Delta}_0 \leftarrow \mathbf{P}\boldsymbol{\Delta}_0$ *// so that $\boldsymbol{\Delta}_0$ now contains $\mathbf{D}_0^0(i)$*
// Compute $R_k \stackrel{def}{=} P_{k,\ell-m}(\alpha - K + k)$ for all $0 \leqslant k \leqslant K$
for $k = 0 \ldots K$ **do**
  $R_k \leftarrow \mathbf{u}\boldsymbol{\Delta}_k$
$C \leftarrow 1.0$
for $k = 1 \ldots K$ **do**
  for $j = K \ldots 1$ **do**
    $\boldsymbol{\Delta}_j \leftarrow \mathbf{P}\boldsymbol{\Delta}_j + \mathbf{Q}\boldsymbol{\Delta}_{j-1} - \mathbf{P}\boldsymbol{\Delta}_j$ *// so that $\boldsymbol{\Delta}_j$ now contains $\mathbf{D}_j^k(\alpha - K + k)$*
  $\boldsymbol{\Delta}_0 \leftarrow \mathbf{P}\boldsymbol{\Delta}_0 - \boldsymbol{\Delta}_0$ *// so that $\boldsymbol{\Delta}_0$ now contains $\mathbf{D}_0^k(\alpha - K + k)$*
  $C \leftarrow C \times (\ell - d - \alpha + K - k + 1)/k$
  for $j = k \ldots K$ **do**
    $R_k \leftarrow R_k + C\mathbf{u}\boldsymbol{\Delta}_k$

Output: return $\mathbb{P}(N_\ell = k) = R_k$ for all $0 \leqslant k \leqslant K$

Algorithm 1: Compute $\mathbb{P}(N_\ell = k)$ with relative error η for all $0 \leqslant k \leqslant K$ using floating point arithmetic with a relative error ε ($< \eta$). The workspace complexity is $O(K \times L)$ and since all matrix vector product exploit the sparse structure of the matrices, the time complexity is $O(\alpha \times K \times |\mathcal{A}| \times L)$ with $\alpha = O(K \times \log(\varepsilon - \nu))$.

and we denote by λ its largest eigenvalue (we know that $\lambda > 0$ thanks to Perron-Frobénius). Let us define $\widetilde{\mathbf{P}} \stackrel{\text{def}}{=} \mathbf{P}/\lambda$ and $\widetilde{\mathbf{Q}} \stackrel{\text{def}}{=} \mathbf{Q}/\lambda$.

For all $i \geqslant 0$ and all $k \geqslant 0$ we consider the vector $\mathbf{F}_k(i) \stackrel{\text{def}}{=} [y^k](\widetilde{\mathbf{P}} + y\widetilde{\mathbf{Q}})^i\mathbf{v}$. By convention, $\mathbf{F}_k(i) = \mathbf{0}$ if $i < 0$. It is then possible to derive from Equation (4) that $\mathbb{P}(N_\ell = k) = [y^k]F(y) = \lambda^{\ell-m}\mathbf{u}\mathbf{F}_k(\ell - m)$. Additionally, we recursively define the quantity $\mathbf{D}_j^k(i)$ for all $k, i, j \geqslant 0$ by $\mathbf{D}_k^0(i) \stackrel{\text{def}}{=} \mathbf{F}_k(i)$ and, if $i \geqslant 1$ and $j \geqslant 1$, $\mathbf{D}_k^j(i) \stackrel{\text{def}}{=} \mathbf{D}_k^{j-1}(i) - \mathbf{D}_k^{j-1}(i-1)$ so that $\mathbf{D}_k^j(i) = \sum_{\delta=0}^{j}(-1)^\delta\binom{j}{\delta}\mathbf{F}_k(i-\delta)$.

Lemma 8. *For all $j \geqslant 0$, $k \geqslant 1$, and $i \geqslant j$ we have:*

$$\mathbf{D}_0^j(i+1) = \widetilde{\mathbf{P}}\mathbf{D}_0^j(i) \quad \textit{and} \quad \mathbf{D}_k^j(i+1) = \widetilde{\mathbf{P}}\mathbf{D}_k^j(i) + \widetilde{\mathbf{Q}}\mathbf{D}_{k-1}^j(i). \tag{6}$$

Proof. The results for $j = 0$ come directly from the definition of $\mathbf{D}_k^0(i) = \mathbf{F}_k(i)$, the rest of the proof is achieved by induction. □

From now on, all asymptotic developments in the current section are meant to be valid for $i \to +\infty$.

Lemma 9. *For all $k \geqslant 0$, it exists $\mathbf{C}_k \in \mathbb{R}^L$ such as:*

$$\mathbf{D}_k^k(i) = \mathbf{C}_k + O(\nu^{i/(k+1)}) \tag{7}$$

where ν denotes the magnitude of the second eigenvalue of $\widetilde{\mathbf{P}}$ when we order them by decreasing magnitude.

Proof. It is clear that $\mathbf{D}_0^0(i) = \widetilde{\mathbf{P}}^i\mathbf{v}$, elementary linear algebra hence proves lemma for $k = 0$ with $\mathbf{C}_0 = \widetilde{\mathbf{P}}^\infty\mathbf{v}$. We assume now that Equation (7) is proved up to a fixed rank $k - 1 \geqslant 0$. A recursive application of Lemma 8 gives for all $\alpha \geqslant k$ and $i \geqslant 0$ that $\mathbf{D}_k^k(i+\alpha) = \widetilde{\mathbf{P}}^i\mathbf{D}_k^k(\alpha) + \sum_{j=1}^{i}\mathbf{P}^{i-j}\widetilde{\mathbf{Q}}\mathbf{D}_{k-1}^k(j-1+\alpha)$. Thanks to Equation (7) it is clear that the second term of this sum is a $O(\nu^{\alpha/k})$, and we then have $\mathbf{D}_k^k(i+\alpha) = \widetilde{\mathbf{P}}^\infty\mathbf{D}_k^k(\alpha) + O(\nu^i) + O(\nu^{\alpha/k})$. Therefore we have $\mathbf{D}_k^{k+1}(i+\alpha) = O(\nu^i) + O(\nu^{\alpha/k})$. For any $j \geqslant k+1$, the particular where $\alpha = j \times k/(k+1)$ hence gives $\mathbf{D}_k^{k+1}(j) = O(\nu^{j/(k+1)})$ which achieves the proof. □

Proposition 10. *For all $j \geqslant 0$, $\alpha \geqslant 0$ and $i \geqslant j + \alpha$ we have:*

$$\mathbf{D}_k^j(i) = \sum_{j'=j}^{k}\binom{i-\alpha-k+j'-j}{k-j}\mathbf{D}_k^{j'}(\alpha+j') + \binom{i-\alpha-j}{k-j}O\left(\nu^{\alpha/(k+1)}\right) \tag{8}$$

and in particular for $j = 0$ we have:

$$\mathbf{F}_k(i) = \sum_{j'=0}^{k}\binom{i-\alpha-k+j'}{k}\mathbf{D}_k^{j'}(\alpha+j') + \binom{i-\alpha}{k}O\left(\nu^{\alpha/(k+1)}\right). \tag{9}$$

Proof. This is proved by induction, using repetitively Lemma 9 and the fact that $\mathbf{D}_k^j(i) = \mathbf{D}_k^j(\alpha+j) + \mathbf{D}_k^{j+1}(\alpha+j+1) + \ldots + \mathbf{D}_k^{j+1}(i)$. □

Corollary 11. *For any $\alpha \geqslant 0$, for any $k \geqslant 0$ and $\ell - m \geqslant 0$ we have:*

$$\mathbb{P}(N_\ell = k) = \underbrace{\lambda^{\ell-m} \sum_{j'=0}^{k} \binom{\ell - m - \alpha - k + j'}{k} \mathbf{u}\mathbf{D}_k^{j'}(\alpha + j')}_{P_{k,\ell-m}(\alpha)} + \lambda^{\ell-m} \binom{\ell - m - \alpha}{k} O\left(\nu^{\alpha/(k+1)}\right)$$

and $P_{k,\ell-m}(\alpha)$ approximates $\mathbb{P}(N_\ell = k)$ with a relative error of:

$$\left| \frac{P_{k,\ell-m}(\alpha) - \mathbb{P}(N_\ell = k)}{\mathbb{P}(N_\ell = k)} \right| = \frac{\lambda^{\ell-m}}{\mathbb{P}(N_\ell = k)} \binom{\ell - m - \alpha}{k} O\left(\nu^{\alpha/(k+1)}\right).$$

4 High-order liftings

Another appealing idea consists in using Equation (5) to compute explicitly the rational function $G(y, z)$ and then performing (fast) Taylor expansions, first in z, then in y in order to get the appropriate values of $\mathbb{P}(N_\ell = n)$ (Nicodème et al., 2002; Lladser, 2007).

There, a really naive approach would solve the bivariate polynomial system over the rationals. In the solution, the polynomials would be of degree at most L in each variable and the rationals of size at most $L \log_2(L)$, thus yielding a binary cost of $O(L^9 \log_2^2(L))$ already for the computation of $G(y, z)$ and a further cost of the same magnitude for the extraction of the coefficient of degree ℓ in the development with e.g. linear system solving.

Since this resulting complexity is usually prohibitive, we first use modular methods and Chinese remaindering to compute with polynomials and rationals so that the cost of the arithmetic remains linear. Also to take advantage of the sparsity we do not inverse the matrix but rather compute directly a rational reconstruction of the solution from the first iterates of the series, Equation (4). We thus only use sparse matrix-vector products and do not fill the matrix. Note that this is equivalent to solving the system using iterative z-adic methods. Finally, we compute only small chunks of the development of Equation (5) using the "high-order" lifting of Storjohann (2002), or the method of Fiduccia (1985), *modulo* $y^{\nu+1}$. We mean to compute only all the coefficients $g_{\ell,n} = \mathbb{P}(N_\ell = n)$ for a given ℓ and a given $n \in [\mu, \nu]$, for an interval $[\mu, \nu]$ with a small ν, but where ℓ is potentially large. Thus we want to avoid computing the whole Taylor development of the fraction.

Indeed, let $G(y, z) = \frac{B(y,z)}{A(y,z)}$ with $B, A \in \mathbb{Q}[y, z]$ of degrees $d_{Az} = \deg(A, z)$ and $d_{Bz} = \deg(B, z) \leq d_{Az} - 1$. Overall let us denote by d a bound on d_{Az} and thus on d_{Bz}. We can assume that A and B are polynomials since we can always pre-multiply them both by the lowest common multiple of their denominators.

Thus $B = \sum_{i=0}^{d_{Bz}} b_i(y) z^i$ and if we denote by $[z^i]P(z)$, the i-th coefficient of

the polynomial P:

$$[z^\ell]G(y,z) = \sum_{i=0}^{d_{Bz}} b_i(y) \times [z^{\ell-i}]\left(\frac{1}{A(y,z)}\right)$$

Now, the idea is that for a given coefficient ℓ, the only coefficients of the development of $1/A$ that are needed are the coefficients of order $\ell - m$ to ℓ, denoted by $[A^{-1}]_{\ell-m}^{\ell}$. This is precisely what Storjohann's "High-order" lifting can compute fast.

We will use several Chinese remaindering and rational reconstructions. We therefore give first two examples of these notions.

Example 12. *We first give an example of the Chinese remaindering theorem with* 2 *prime numbers. Let* p *and* q *be prime numbers, and let* $0 \le u < p$ *and* $0 \le v < q$*. Then,* $x = u + p \times (p^{-1} \mod q) \times (v - u) \mod pq$ *satisfies both* $x \equiv u \mod p$ *and* $x \equiv v \mod q$*. With* $p = 5$*,* $q = 7$*,* $u = 2$*,* $v = 6$ *we have* $5 \times 3 \equiv 1 \mod 7$ *so that* $5^{-1} \mod 7 = 3$ *and thus* $2 + (6-2) \times 5 \times 3 = 62$ *is congruent to* 2 *modulo* 5 *and to* 6 *modulo* 7 *as is* $x = 27 \equiv 62 \mod 35$*.*

This generalizes easily to any set of coprime integer moduli, as well as with any set of coprime polynomial moduli, see e.g. Gathen and Gerhard (1999); Dumas et al. (2007) for more details.

Example 13. *Now we show a rational reconstruction: suppose we are given a congruence* $S \mod M$ *and we want to find* a *and* b*, of absolute value smaller than some bound* $\beta < M$*, satisfying* $S \equiv \frac{b}{a} \mod M$*. Then the successive steps of the extended Euclidean algorithm, see e.g. Gathen and Gerhard (1999), will produce some possible answers. For instance consider* $45 \mod 53$*, the sequence of remainders,* $\{45, 8, 5, 3, 2, 1\}$*, in the extended gcd computation of* 45 *and* 53*, provide a decreasing sequence of candidates for* b*. Simultaneously, the sequence of non zero cofactors of* 45*, namely* $\{1, -1, 6, -7, 13, -20\}$ *provides a sequence, increasing in absolute value, of candidates for* a*:*

$$\begin{array}{rcrcr}
0 \times 45 & + & 1 \times 53 & = & 53 \\
1 \times 45 & + & 0 \times 53 & = & 45 \\
-1 \times 45 & + & 1 \times 53 & = & 8 \\
6 \times 45 & + & -5 \times 53 & = & 5 \\
-7 \times 45 & + & 6 \times 53 & = & 3 \\
13 \times 45 & + & -11 \times 53 & = & 2 \\
-20 \times 45 & + & 17 \times 53 & = & 1
\end{array}$$

As soon as a remainder is below the bound β*, if on the one hand its associated cofactor is also of absolute value less than the bound, then the rational reconstruction problem has a solution. If on the other hand its associated cofactor does not satisfying the bound, then the problem has no solution. For instance here we have* $45 \equiv \frac{5}{6} \mod 53$ *with both* 5 *and* 6 *below* $7 = \lfloor\sqrt{53}\rfloor$*.*

Once again this generalizes to polynomials with bounds on the degrees, see e.g. Gathen and Gerhard (1999).

In our cases, we have a series $\sum^{\infty} g_i z^i$ which is actually equal to a fraction $\frac{B(z)}{A(z)}$ by definition. Therefore, provided that we consider sufficiently many of the first coefficients in the development, the rational reconstruction of the truncated series $\sum^{2d} g_i z^i$, even with rational polynomial in y as coefficients, will *eventually yield $A(z)$ and $B(z)$ as solutions.*

4.1 Rational reconstruction

The first step is thus to recover both polynomials B and A from the series development of Equation (5). Now, one could compute the whole rational reconstruction of a polynomial over the rationals domain, then using $d^2 \times d^2$ operations for the domain arithmetic, which would yield a d^6 complexity to compute all the d^2 coefficients. We rather use two modular projections, one for the rational coefficients, one for the polynomials in y, in order to reduce the cost of the arithmetic to only d^2. Then the overall cost will be dominated by the cost of the computation of only d coefficients of the series, as shown on proposition 14.

Our algorithm has then four main steps:

1) We take the series in z modulo some prime numbers (below 2^γ where γ is e.g. close to word size) and some evaluation points on the variable y;

2) We perform a univariate polynomial fraction reconstruction in z over each prime field and for each evaluation point;

3) We interpolate the polynomials in y over the prime field from their evaluations;

4) We finally Chinese remainder and rational reconstruct the rational coefficients from their modular values.

The details of the approach are given in Algorithm 2 whose complexity is given by the following proposition.

Proposition 14. *Let $d = \max\{d_A, d_B\}$ be the degree in z and ν be the largest degree in y of the coefficients of A and B. Let $\Omega = |\mathcal{A}|L$ be the total number of non-zero coefficients of the $L \times L$ matrices $\mathbf{P}$ and $\mathbf{Q}$. If the coefficients of the matrices $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{u}$ and $\mathbf{v}$, and $|\mathcal{A}|$ are constants, then the cost of the computation of B and A in Algorithm 2 is*

$$O\left(d^3 L \log(L)\right),$$

where the intermediate memory requirements are of the same order of magnitude.

Proof. Polynomial fraction reconstruction of degree d requires $2d$ coefficients. The computation of one coefficient of the evaluated series modulo cost one matrix-vector product, Ω word operations, and a dotproduct, $L \leq \Omega$ operations.

Input: The matrices $\mathbf{P}$, $\mathbf{Q}$ and row and column vectors $\mathbf{u}$, $\mathbf{v}$ defining $G(y,z)$.
Output: Polynomials $B(y,z)$ and $A(y,z)$ of degree $\leq d$ with $G(y,z) = \frac{B(y,z)}{A(y,z)}$.

Let $G(y,z) = \mathbf{uv}$, $v_0(y) = \mathbf{v}$;
for $n = 1$ **to** $2d$ **do**
  $v_n(y) = (\mathbf{P} + y\mathbf{Q})v_{n-1}(y)$;
  $G(y,z) = G(y,z) + \mathbf{u}v_n(y)z^n$;
Let $\overline{d} = (2d+2)\log_{2^\gamma}(||\mathbf{P},\mathbf{Q},\mathbf{u},\mathbf{v}||_\infty)$;
for $i = 0$ **to** $\overline{d}$ **do**
  Let $p_i \geq 2^\gamma > d$ be a prime, coprime with $p_0, \ldots, p_{i-1}$;
  $G_i(y,z) = G(y,z) \mod p_i$;
  for $j = 0$ **to** d **do**
    Let y_j be an element modulo p_i, distinct from $y_0, \ldots, y_{j-1}$;
    $G_{i,j}(z) = G_i(y_j, z) \mod p_i$;
    $\frac{B_{i,j}(z)}{A_{i,j}(z)}$ =**FractionReconstruction**$(G_{i,j}(z)) \mod p_i \mod (y - y_j)$;
    // $B_{i,j}(z) = \sum_{n=0}^{d} B_{i,j,n} z^n$ *and* $A_{i,j}(z) = \sum_{n=0}^{d} A_{i,j,n} z^n$
  for $n = 0$ **to** d **do**
    Interpolate $B_{i,n}(y) \mod p_i$ from $B_{i,j,n}$ for $j = 0..d$;
    Interpolate $A_{i,n}(y) \mod p_i$ from $A_{i,j,n}$ for $j = 0..d$;
    // $B_i(y,z) = \sum_{n=0}^{d} B_{i,n}(y) z^n$ *and* $A_i(y,z) = \sum_{n=0}^{d} A_{i,n}(y) z^n$
for $n = 0$ **to** d **do**
  for $l = 0$ **to** d **do**
    ChineseRemainder $[y^n z^l]B(y,z)$ from $[y^n z^l]B_i(y,z)$ for $i = 0..\overline{d}$;
    ChineseRemainder $[y^n z^l]A(y,z)$ from $[y^n z^l]A_i(y,z)$ for $i = 0..\overline{d}$;
    RationalReconstruct both obtained coefficients;

Algorithm 2: Modular Bivariate fraction reconstruction over the rationals.

By definition $\deg(g_j(y)) = \deg((\mathbf{P} + y\mathbf{Q})^j) \leq j$, thus $\nu \leq 2d$ and, similarly, the size of the rational coefficients is bounded by $(2d+2)\log(L||\mathbf{P}, \mathbf{Q}, \mathbf{u}, \mathbf{v}||_\infty^2)$.

Thus steps 3 and 4 in Algorithm 2 cost

$$\sum_{n=0}^{d} O(\Omega n^2 \log(L||\mathbf{P}, \mathbf{Q}, \mathbf{u}, \mathbf{v}||_\infty^2) = O(d^3 \Omega \log(L||\mathbf{P}, \mathbf{Q}, \mathbf{u}, \mathbf{v}||_\infty^2)$$

operations.

Then a fraction reconstruction of degree d costs less than $O(d^2)$ operations by Berlekamp-Massey or extended Euclidean algorithm and an interpolation of d points costs less than $O(d^2)$ operations so that, overall, steps 12, 14 and 15 costs less than $O(d^4)$ operations. Then Chinese remaindering and rational reconstruction of size d costs less than $O(d^2)$ for an overall cost of

$$O(d^4 \log^2(||\mathbf{P}, \mathbf{Q}, \mathbf{u}, \mathbf{v}||_\infty).$$

As $d \leq L \leq \Omega$, if $\log_{2^\gamma}^2(||\mathbf{P}, \mathbf{Q}, \mathbf{u}, \mathbf{v}||_\infty) = O(1)$ then this latter term is dominated by $O(d^3 \Omega \log(L))$. Finally, if $|\mathcal{A}|$ is constant, we have that $\Omega = O(L)$.

Now, the memory requirements are bounded by those of the series. The vector $v_n(y)$ is of size L, of degree n in y with rational coefficients of size $n \log(L||\mathbf{P}, \mathbf{Q}, \mathbf{u}, \mathbf{v}||_\infty^2)$. Thus $v_n(y)$ and the dot product $\mathbf{u}v_n(y)$ are of size $O(Ln^2 \log(L))$ so that $G(y,z) = \sum_{n=0}^{2d} \mathbf{u}v_n(y)$ is $O(L \log(L) d^3)$. □

Thus the dominant computation in Algorithm 2 is the computation of the first terms of the series $G(y,z)$.

4.2 Early termination strategy for the determination of the fraction degrees

There remains to determine the value of the degree d. As the series is the solution of a polynomial linear system of size L and degree 1, the determinant, and thus the denominator and numerator of the solution are of degree bounded by L. Now in practice, we will see that very often this degree is much smaller than L. As the complexity is cubic in d it is of major importance to determine as accurately as possible this d beforehand.

The strategy we propose is an early termination, probabilistic of Las Vegas type, i.e. it is always correct, but sometimes slow. We reconstruct the rational fraction only from a small number d_0 of iterations, and then again after $d_0 + 1$ iterations. If both fractions are identical with numerator and denominator of degrees strictly less than d_0 then there is a good chance that the recovered fraction is the actual one. This can be checked by just applying the matrix A to the obtained guess and verifying that it corresponds to the right-hand side.

If both fraction disagree or if the check fails, one can then try again after some more iterations.

In our bivariate case over the rationals, we have a very fast strategy which consists in finding first the degree at a single evaluation point modulo a single

prime and e.g. roughly doubling the number of iterations at each failure. This search thus requires less than $2 \times 2d$ iterations and has then a marginal cost of $O(d\Omega + d^2)$.

4.3 High-order lifting for polynomial fraction development

Once the bivariate fraction $\frac{B}{A}$ is recovered, the next step is to compute the coefficients of degree $\ell \in [\alpha, \beta]$ of its series development. The idea of the high-order lifting of Storjohann (2002) is to make use of some particular points in the development, that are computable independently. Then these points will enable fast computation of only high-order terms of the development and not of the whole series. In the following, we call these points *residues*.

We first need the fundamental Lemma 15.

Lemma 15. *Let $A, B \in R[z]$ be of respective degree d_A and $d_B \leq d_A - 1$. Then for all $\ell \in \mathbb{N}$, there exists $B_\ell \in R[z]$ of degree $d_{B_\ell} \leq d_A - 1$, and $(g_i)_{i=0..\ell-1} \in R^\ell$ such that*

$$\frac{B(z)}{A(z)} = \sum_{i=0}^{\ell-1} g_i z^i + \frac{B_\ell(z)}{A(z)} z^\ell$$

Proof. We use the same construction as Salvy (2009): the initial series rewrites as, $\frac{B}{A} = \sum_{i=0}^{\infty} g_i z^i = \sum_{i=0}^{\ell-1} g_i z^i + z^\ell \sum_{i=0}^{\infty} g_{i+\ell} z^i$. Then let $\overline{B_\ell} = B - A\left(\sum_{i=0}^{\ell-1} g_i z^i\right)$. By construction, $degree(\overline{B_\ell}) = d_A + \ell - 1$, but we also have that $\overline{B_\ell} = z^\ell \sum_{i=0}^{\infty} g_{i+\ell} z^i$. We thus let $B_\ell = \overline{B_\ell} z^{-\ell}$ which satisfies the hypothesis. □

The question is how to compute efficiently the ℓth residue B_ℓ defined in lemma 15. The idea is to use the high-order lifting of Storjohann (2002).

We follow the presentation of the lifting of Salvy (2009) but define a slightly different bracket notation for chunks of a polynomial:

$$[A(z)]_\alpha^\beta = a_\alpha z^\alpha + \ldots + a_\beta z^\beta \tag{10}$$

Roughly there are two main parts. The first one generalize the construction of lemma 15 using only $2d$ coefficients of A^{-1}. The second part builds small chunks of size $2d$ of A^{-1} at high-orders, each being close to a power of 2.

The efficient computation of residues given in Algorithm 3 takes simply advantage of the fact that a given residue has a small degree and depends only on a small part of the development of A^{-1}. We first give a version where the adequate part of A is given as input. We will later detail the way to efficiently compute these coefficients.

Lemma 16. *The arithmetic complexity of Algorithm 3 is $2d^2$ operations.*

Then, we define Γ_ℓ to be the high-order residue of the expansion of A^{-1}, using lemma 15 with $B = 1$:

$$\frac{1}{A(z)} = \sum_{i=0}^{\ell-1} g_i z^i + \frac{\Gamma_\ell(z)}{A(z)} z^\ell \tag{11}$$

Input: A, B, j and $V = [A^{-1}]_{j-2d+1}^{j-1}$.
Output: B_j defined by lemma 15.

1: Compute $U \stackrel{\text{def}}{=} [VB]_{j-d}^{j-1}$;
2: Return $B_j \stackrel{\text{def}}{=} z^{-j}[B - AU]_j^{j+d-1}$;

Algorithm 3: Residue(A,B,j,V).

The idea of the fast lifting is that when substituting A^{-1} in the right hand side of Equation 11 by this actual right hand side, one gets:

$$\sum_{i=0}^{\ell-1} g_i z^i + \Gamma_\ell(z) \left(\sum_{i=0}^{\ell-1} g_i z^i + \frac{\Gamma_\ell(z)}{A(z)} z^\ell \right) z^\ell = \sum_{i=0}^{2\ell-1} g_i z^i + \frac{\Gamma_{2\ell}(z)}{A(z)} z^{2\ell}$$

This shows that $\Gamma_{2\ell}$ depends only on Γ_ℓ and of chunks of A^{-1}, of size d, at 0 and around Γ_ℓ; more generally one gets the following formula:

$$\left[A^{-1}\right]_\alpha^\beta = z^\ell \left[\Gamma_\ell [A^{-1}]_{\alpha-\ell-d}^{\beta-\ell}\right]_{\alpha-\ell}^{\beta-\ell} \tag{12}$$

which states, from Equation (11), that the Taylor coefficients of order greater than ℓ, can also be recovered from the Taylor development of $\frac{\Gamma_\ell(z)}{A(z)}$.

Then the second part of the high-order lifting is thus Algorithm 4 which gets a small chunk of A^{-1} at a double order of what it is given as input, as shown in Figure 3.

Input: $S = [A^{-1}]_0^{d-1}$, $V_e = [A^{-1}]_{2^e-2d+1}^{2^e-1}$ and Γ_{2^e-d} defined by eq. (11).
Output: $\Gamma_{2^{e+1}-d}$ and $V_{e+1} = [A^{-1}]_{2^{e+1}-2d+1}^{2^{e+1}-1}$.

1: Compute $V_L \stackrel{\text{def}}{=} z^{2^e-d}[\Gamma_{2^e-d} V_e]_{2^e-d}^{2^e-1}$; // *eq. (12)*
2: Compute $\Gamma_{2^{e+1}-d} \stackrel{\text{def}}{=} z^{-2^{e+1}+d}[1 - AV_L]_{2^{e+1}-d}^{2^{e+1}-1}$; // *Residue*$(A, 1, 2^{e+1} - d)$
3: Compute $V_H \stackrel{\text{def}}{=} z^{2^{e+1}-d}[\Gamma_{2^{e+1}-d} S]_0^{d-1}$; // *eq. (12)*
4: Return $\Gamma_{2^{e+1}-d}$ and $V_{e+1} = [A^{-1}]_{2^{e+1}-2d+1}^{2^{e+1}-1} \stackrel{\text{def}}{=} [V_L]_{2^{e+1}-2d+1}^{2^{e+1}-d-1} + V_H$.

Algorithm 4: Double-Order(S, T, Γ, e).

Lemma 17. *The arithmetic complexity of Algorithm 4 is* $3d^2$ *operations.*

Proof. Below are the complexity of the successive steps of Algorithm 4

1) One truncated polynomial multiplication of degree $d-1$, complexity d^2;

2) One truncated polynomial multiplication of degree $d-1$, complexity d^2;

3) One truncated polynomial multiplication of degree $d-1$, complexity d^2.

4) No-op, $V^{(L)}$ and $V^{(H)}$ do not overlap.

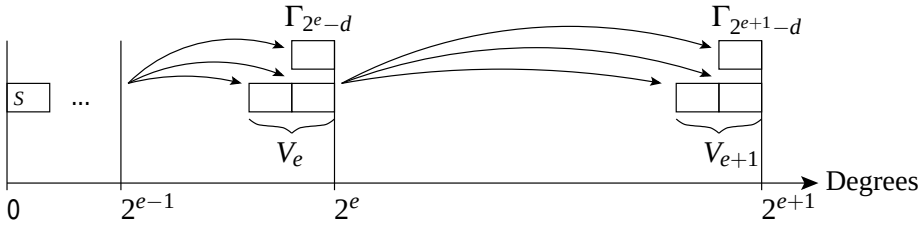

Figure 3: Computing chunks at double order.

□

Then Algorithm 5 gives the complete precomputation to get a sequence of doubling order Γ's which will enable fast computations of the chunks of the Taylor expansion.

Input: A polynomial $A(z)$ of degree d.
Input: A valuation α and a degree $\beta \geq d$.
Output: e_0 s.t. $2^{e_0-1} < 2d \leq 2^{e_0}$; e_β s.t. $2^{e_\beta} \leq \beta + d < 2^{e_\beta+1}$.
Output: The Taylor development of $\frac{1}{A}$ up to $\delta = \max\{2^{e_0} - 1; \beta - \alpha\}$.
Output: $(\Gamma_{2^{e_0}-d}, \ldots, \Gamma_{2^{e_\beta}-d})$.

1: Compute e_0 s.t. $2^{e_0-1} < 2d \leq 2^{e_0}$; e_β s.t. $2^{e_\beta} \leq \beta + d < 2^{e_\beta+1}$;
2: Let $\xi_0 = k_{e_0} \stackrel{\text{def}}{=} 2^{e_0} - d$ and $\delta \stackrel{\text{def}}{=} \max\{2^{e_0} - 1; \beta - \alpha\}$;
3: Compute $S \stackrel{\text{def}}{=} [A^{-1}]_0^\delta$, via Taylor expansion of A;
4: $U_0 \stackrel{\text{def}}{=} [A^{-1}]_{\xi_0-d}^{\xi_0-1} = [S]_{\xi_0-d}^{\xi_0-1}$;
5: Compute $\Gamma_{\xi_0} \stackrel{\text{def}}{=} z^{-\xi_0}[I - AU_0]_{\xi_0}^{\xi_0+d-1}$; // *Residue*$(A, 1, \xi_0)$
6: $V_{e_0} \stackrel{\text{def}}{=} [A^{-1}]_{\xi_0-d+1}^{\xi_0+d-1} = [S]_{2^{e_0}-2d+1}^{2^{e_0}-1}$;
7: **for** $i = e_0 + 1$ **to** e_β **do**
8: $\quad k_i \stackrel{\text{def}}{=} 2^i - d$;
9: $\quad (\Gamma_{k_i}; V_i) \stackrel{\text{def}}{=}$Double-Order$([A^{-1}]_0^{d-1}, V_{i-1}, \Gamma_{k_{i-1}}, i-1)$;

Algorithm 5: High-Order(A, α, β).

Figure 4 shows which high-order terms are recovered during this giant steps of precomputation, for a computation using 50 precomputed terms of the Taylor development with Algorithm 5 and $degree(A) = 6$.

Lemma 18. *The arithmetic complexity of Algorithm 5 is less than* $3\log_2\left(\frac{\beta+d}{2d}\right)d^2 + \max\{4d^2; d(\beta - \alpha + 2d)\}$ *operations*

Proof. Below are the complexity of the successive steps of Algorithm 5

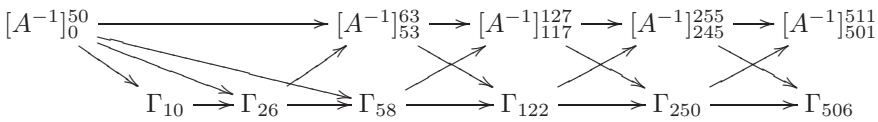

Figure 4: High-Order lifting to $2^{\lfloor \log_2(\beta) \rfloor} - 1$ computing Γ_{2^4-6}, Γ_{2^5-6}, Γ_{2^6-6}, Γ_{2^7-6}, Γ_{2^8-6} and Γ_{2^9-6}.

3. One Taylor expansion of an inverse of degree d up to the degree $\delta \leq \max\{2d; \beta - \alpha\}$, complexity $\sum_{i=1}^{d} 2i - 1 + \sum_{i=d+1}^{\delta} 2d - 1 \leq d(2\delta - d) \leq \max\{3d^2; d(\beta - \alpha + d)\}$.
5. One truncated polynomial multiplication of degree $d - 1$, complexity d^2.
9. $e_\beta - e_0 \leq \log_2\left(\frac{\beta+d}{2d}\right)$ calls to Algorithm 4, complexity bounded by $3\log_2\left(\frac{\beta+d}{2d}\right) d^2$.

□

Once the high-order terms are computed, one can get the development chunks of $[\frac{B}{A}]_\alpha^\beta$ as shown on Algorithm 6.

Input: $S = [A^{-1}]_0^\delta$, with $\beta - \alpha \leq \delta$.
Output: $[BA^{-1}]_\alpha^\beta$
if $\beta \leq \delta$ **then**
  Return $[BS]_\alpha^\beta$;
else
  $B_\alpha \stackrel{\text{def}}{=} \text{Residue}(A, B, (\Gamma_i), S, \alpha)$;
  Return $z^\alpha [B_\alpha S]_0^{\beta-\alpha}$; // *eq. (12)*

Algorithm 6: DevelChunk$(A, B, (\Gamma_i), S, \alpha, \beta)$.

Algorithm 6 uses a variant of Algorithm 3, which needs to compute on the fly the chunks of the inverse it requires. This variant is shown on Algorithm 7.

Output: B_α;
if $\alpha = 0$ **then**
  Return B;
else
  $V \stackrel{\text{def}}{=} \text{InverseChunk}(A, (\Gamma_i), S, \alpha - 2d + 1, \alpha - 1)$;
  $U \stackrel{\text{def}}{=} [VB]_{\alpha-d}^{\alpha-1}$;
  Return $z^{-\alpha}[B - AU]_\alpha^{\alpha+d-1}$. // *Residue*$(A, B, \alpha, V)$

Algorithm 7: Residue$(A, B, (\Gamma_i), S, \alpha)$.

The point is that those chunks of the development of the inverse are recovered just like the chunks of any fraction, but with some high-order residues already

computed. Algorithm 8 is thus a variant of Algorithm 6 with $B = 1$ and a special residue call.

Input: $S = [A^{-1}]_0^\delta$ with $\delta > \beta - \alpha$.
Output: $[A^{-1}]_\alpha^\beta$

1: **if** $\beta \leq \delta$ **then**
2: Return $[S]_\alpha^\beta$;
3: **else**
4: $\Gamma_\alpha \stackrel{\text{def}}{=} \text{Get}\Gamma(A, (\Gamma_i), S, \alpha)$; // *Residue*$(A, 1, \alpha)$;
5: Return $z^\alpha [\Gamma_\alpha S]_0^{\beta-\alpha}$; // *eq. (12)*

Algorithm 8: InverseChunk$(A, (\Gamma_i), S, \alpha, \beta)$.

Algorithm 9 is this special residue call, variant of Algorithm 7, which uses the precomputed high-order Γ_i to get the actual Γ_α it needs, in a logarithmic binary-like decomposition of α.

Output: Γ_α;

1: **if** $\alpha = 0$ **then**
2: Return 1.
3: **else if** $\alpha \leq \delta$ **then**
4: $U \stackrel{\text{def}}{=} [S]_0^{\alpha-1}$;
5: Return $z^{-\alpha}[1 - AU]_\alpha^{\alpha+d-1}$. // *Residue*$(A, 1, \alpha, S)$
6: **else**
7: Let ϵ s.t. $2^\epsilon - d \leq \alpha < 2^{\epsilon+1} - d$;
8: Return Residue$(A, \Gamma_\epsilon, (\Gamma_i), \alpha - 2^\epsilon + d)$;

Algorithm 9: Get$\Gamma(A, (\Gamma_i), S, \alpha)$.

We have shown in Figure 4 the high-order precomputation of the different Γ's required for the computation e.g. of $[BA^{-1}]_{950}^{1000} = [B_{950}A^{-1}]_0^{50}$, with A of degree 6. Then, Figure 5 shows the successive baby step calls of residue and inverse chunks needed to compute the 950th residue B_{950}

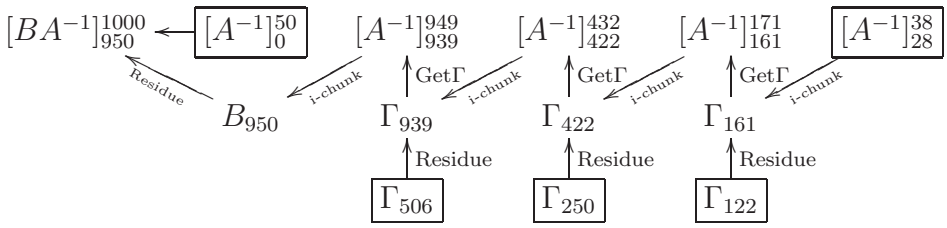

Figure 5: *DevelChunk* Expansion with boxed precomputed chunks.

Lemma 19. *The arithmetic complexity of computing the* $[\alpha, \beta]$ *chunk of the development of* $[BA^{-1}(z)]_\alpha^\beta$ *via Algorithm 6 is less than* $\log_2\left(\frac{\beta+d}{2d}\right)\left(3d^2 + 2(\beta-\alpha)^2\right)$ *operations.*

Proof. Except for the calls to other functions, DevelChunk has complexity $(\beta-\alpha)^2$, GetΓ has complexity d^2, InverseChunk has complexity $(\beta-\alpha)^2$ and Residue has complexity $2d^2$.

Now the logarithmic binary decomposition of β shows that GetΓ, InverseChunk and Residue are each called less than $\log_2\left(\frac{\beta+d}{2d}\right)$ times. □

4.4 Fiduccia's algorithm for linear recurring sequences

An alternative to the high-order lifting is to directly use Fiduccia's algorithm for linear recurring sequences (Fiduccia, 1985). Its complexity is slightly different and we show next that it can be interesting when $\beta = \alpha$.

4.4.1 Single coefficient recovery

With the same notations as before, one wants to solve $B = A \times T$ for B and A polynomials of degree bounded by d and T is the series development. We want to obtain the coefficients of T only between degrees α and β. The algorithm is as follows: solve directly for the first d terms of T and afterwards, if $A = \sum_{i=0}^{d} a_i Z^i$, we obtain a recurring linear sequence for the coefficients of $T = \sum^{\infty} t_i Z^i$:

$$a_0 t_\ell = -\sum_{i=1}^{d} a_i t_{\ell-i} \tag{13}$$

If $a_0 \neq 0$, let us define the characteristic polynomial $P(Z) = \mathrm{rev}(A)/a_0 = A(1/Z)Z^d/a_0$. This induces the following linear system involving the companion matrix of P:

$$\begin{pmatrix} t_{\ell-d+1} \\ \vdots \\ t_\ell \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & & \dots & 0 & 1 \\ -\frac{a_d}{a_0} & & \dots & & -\frac{a_1}{a_0} \end{pmatrix} \times \begin{pmatrix} t_{\ell-d} \\ \vdots \\ t_{\ell-1} \end{pmatrix} = \mathrm{Companion}(P)^T \times \begin{pmatrix} t_{\ell-d} \\ \vdots \\ t_{\ell-1} \end{pmatrix}$$

Denote by $C = \mathrm{Companion}(P(Z))$, then by developing the above system $\ell - d$ times, we obtain one coefficient of T with the simple dotproduct

$$t_\ell = [0, \dots, 0, 1] \times \left(C^T\right)^{\ell-d} \times [t_1, \dots, t_d]^T = [t_1, \dots, t_d] \times C^{\ell-d} \times [0, \dots, 0, 1]^T \tag{14}$$

The idea of Fiduccia is then to use the Cayley-Hamilton theorem, $P(C) = 0$, and identify polynomials to a vector representation of their coefficients, in order to obtain that $C^{\ell-d} \times [0, \dots, 0, 1]^T = Z^{\ell-d} \times Z^{d-1} = Z^{\ell-1} \mod P(Z)$. Now the modular exponentiation is obtained by binary recursive squaring in $\log_2(\ell-1)$ steps, each one involving 1 or 2 multiplications and 1 or 2 divisions

of degree d, depending on the bit pattern of $\ell - 1$. Then, the complexity of the modular exponentiation is bounded by $\log_2(\ell)(8d^2)$ with an average value of $\log_2(\ell)(6d^2)$, the exact same constant factor of the high-order lifting when $\beta = \alpha$. But then the additional operations are just a dotproduct of the obtained polynomial with $[t_1, \ldots, t_d]$ and the initial direct Taylor recovery of the latter coefficients. Thus yielding the overall complexity for a single coefficient of the series of $\log_2(\ell)(6d^2) + d^2$ arithmetic operations.

4.4.2 Cluster of coefficients recovery

In the more generic case of several clustered coefficients, $\ell \in [\alpha, \beta]$, one needs to modify the algorithm, in order to avoid computing $\beta - \alpha$ products by $[0, \ldots, 0, 1, 0, \ldots, 0]^T$. Instead one will recover d coefficients at a time, in $\frac{\beta-\alpha}{d}$ steps.

First the binary recursive powering algorithm is used to get $\frac{\beta-\alpha}{d}$ expressions of $C^{\alpha+(j-1)d} = \sum_{i=0}^{d-1} c_i^{(j)} C^i$, at an average global cost of $\left(\frac{\beta-\alpha}{d}\log_2(d) + \log_2(\alpha)\right)\left(6d^2\right)$. Then for $v = [t_1, \ldots, t_d]^T$, the sequence $v, Cv, C^2v, \ldots, C^{d-1}v$ is computed once, iteratively. Finally this sequence is combined with the coefficients $c_i^{(j)}$ to obtain the $\beta - \alpha$ coefficients at an overall cost of $\left(\frac{\beta-\alpha}{d}\log_2(d) + \log_2(\alpha)\right)\left(6d^2\right) + 4d^2 + \max\{d^2; d(2(\beta-\alpha) - d\}$.

4.4.3 High-Order and Fiduccia algorithm comparison

We compare in table 2 the arithmetic complexity bound of Storjohann's high-order lifting and the average complexity bound for Fiduccia's algorithm.

Algorithm	$\ell = \beta = \alpha$	$\ell \in [\alpha, \beta]$
High-Order	$6d^2 \log_2(\frac{\beta}{2d}) + 4d^2$	$\left(6d^2 + 2(\beta-\alpha)^2\right)\log_2(\frac{\beta}{2d}) + d(\beta - \alpha + 2d)$
Recurring	$6d^2 \log_2(\beta) + d^2$	$\left(6d^2\right)\left(\frac{\beta-\alpha}{d}\log_2(d) + \log_2(\alpha)\right) + d(2(\beta-\alpha) + 3d)\}$

Table 2: Complexities, for $\beta > 2d$, of Storjohann's high-order lifting and Fiduccia's algorithm, the latter on average.

From this table we see that Storjohann's algorithm should be preferred when $\beta \neq \alpha$ and that Fiduccia's algorithm should be preferred when both conditions $\beta = \alpha$ and "d is small" are satisfied. In practice, on the bivariate examples of section 5, with $\beta = \alpha$, the differences remained within 20% and where dominated anyway by the fraction reconstruction. Therefore, in the following, we use preferably Storjohann's high-order lifting.

4.5 Bivariate lifting

We come back to the bivariate case of Equation (5). B, A and all the g_l are polynomials in y (not fractions). Therefore one can compute the lifting on z using the arithmetic modulo y^{n+1} for the coefficients. Operations in this latter

domain thus costs no more than $O(n^2)$ operations over $\mathbb{Q}$. In the following we use the formalism of the high-order lifting, but the algorithm of Fiduccia can be used as a replacement if needed.

Finally, for faster computations, one can also convert the rational coefficients to their floating point approximations in order to get Algorithm 10.

Input: $\frac{B(y,z)}{A(y,z)} \in \mathbb{Q}[y](z)$.
Output: A floating point $\left[[BA^{-1}]_\alpha^\beta\right]_0^n$.
1: $B_f(y,z)$ and $A_f(y,z)$ are the conversion of B and A in floating points.
2: (Γ_i) =**High-Order**(A_f, α, β) modulo y^{n+1};
3: $(g_j(y))_{j=\alpha,\ldots,\beta}$ =**DevelChunk**$(A_f, B_f, (\Gamma_i), \alpha, \beta)$ modulo y^{n+1};
4: Return $([g_j(y)]_0^n)_{j=\alpha,\ldots,\beta}$.

Algorithm 10: Bivariate floating point lifting.

Then floating point arithmetic modulo y^{n+1}, together with lemmata 19 and 18, yield the following complexity for the computation of chunks of the Taylor development of a bivariate polynomial fraction:

Proposition 20. *Let $G(y,z) = \frac{B(y,z)}{A(y,z)}$ be a rational fraction with B and A polynomials of degrees less than d with floating point coefficients. Suppose now that $\beta >> d$, and that $\beta - \alpha = O(d)$. Then the overall complexity to compute $\left[[BA^{-1}]_\alpha^\beta\right]_0^n$ with Algorithm 10 and classical polynomial arithmetic is*

$$O(\log(\beta)d^2n^2)$$

rational operations.

This improves e.g. on the algorithm of Knuth (1997) used in Nicodème et al. (2002, Algorithm 8), which has complexity $O(\log(\beta)d^3n^2)$.

Note that, with fast floating point polynomial arithmetic (for instance using FFT), our complexity could reduce to

$$O(\log(\beta)d\log(d)n\log(n)).$$

4.6 Overall complexity

Another improvement can be made on Algorithm 2 when the desired degree n in y of the development is smaller than the degree d of the obtained bivariate fraction: compute the series and the fraction reconstruction also modulo y^{n+1}. Recall that we consider computing $\mathbb{P}(N_\ell = n)$ where the transition matrix is of dimension L, with $O(L)$ non zero coefficients, and the rational fraction is of degree $d \leq L$. Therefore, the overall complexity of Algorithms 2-10, with fast arithmetic, computing the latter, is bounded by:

$$O\left(\min\{n,d\}d^2L + \log(\ell)d^{1+\epsilon}n^{1+\epsilon}\right) \tag{15}$$

POSIX regex	L	F	$1-\lambda$	t_1	frac. deg.	t_2
AD(A|D){0}AD	5	1	3.7×10^{-3}	0.03	2/4	0.00
AD(A|D){2}AD	12	2	9.5×10^{-4}	0.11	6/8	0.01
AD(A|D){5}AD	50	8	1.2×10^{-4}	0.49	28/30	0.20
AD(A|D){10}AD	555	89	3.7×10^{-5}	6.14	321/323	13.90
AD(A|D){15}AD	6155	987	1.2×10^{-7}	73.46	3570/3572	MT

Table 3: Toy-example motifs over the alphabet $\mathcal{A} = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}\}$. L (resp. F) is the number of states (resp. final states) of the minimal order 0 DFA associated to the regular expression. λ is the largest eigenvalue of $\mathbf{P}$, and t_1 the time to compute λ using the power method. "frac. deg." corresponds to the fractional degrees of $G(y, z)$, and t_2 is the time to compute $G(y, z)$ using Algorithm 2. All running times are expressed in seconds, MT stands for Memory Thrashing.

5 Applications

5.1 Toy-examples

We consider here independent and identically distributed sequence of letters that are uniformly distributed over the four letters alphabet $\mathcal{A} = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}\}$. Partial recursion is performed with a floating point-arithmetic precision of $\varepsilon = 1/2^{1024} \simeq 10^{-710}$ (implementation using the `mpf` class from the GMP [2]), and relative error $\eta = 10^{-15}$. The bivariate polynomial fraction reconstruction has been implemented using LinBox [3] and Givaro [4] and the high-order lifting using Givaro and Mpfr [5] with the mpreal [6] C++ wrapper.

In Table 3 we consider 5 example motifs of increasing complexities. For the partial recursion approach, the eigenvalue λ is reported along with the corresponding computational time. One should note that this part of the algorithm uses in fact a very inefficient approach (the power method) while more effective approaches are available (ex: Lanczos iterations). But the performances remain acceptable overall. We also report in this table the fractional degrees of $G(y, z)$ computed through the rational reconstruction. For that matter, we can see that the limiting factor of the series computation is memory. For example, for Motif AD(A|D){15}AD, only storing the first $2d = 3570 + 3572 = 7142$ bivariate terms over the rationals of the series would require the order of $4d^3 L \log_{2^8}(L) \approx 1.7 \times 10^6$ Gigabytes to complete, using the estimates of Proposition 14. Note that for this motif, the degrees in z of the numerator and denominator of the fraction are only probabilistic since they where computed modulo a random word-size prime number at a random evaluation in y.

In Table 4 we perform the computation of $\mathbb{P}(N_\ell = k)$ for the considered

[2] GNU Multi-Precision library `http://gmplib.org/`
[3] `http://linalg.org`
[4] `http://ljk.imag.fr/CASYS/LOGICIELS/givaro`
[5] `http://www.mpfr.org`
[6] `http://www.holoborodko.com/pavel/?page_id=12`

POSIX regex	k	α	ℓ	$\mathbb{P}(N_\ell = k)$	e.r.	t_0	t_3	$+t_1$	t_4	$+t_2$
AD(A|D){0}AD	10	90	2,000	9.12559×10^{-2}	0.234	0.50	0.04	0.07	0.01	0.01
			20,000	4.37982×10^{-21}	0.168	5.00	0.04	0.07	0.01	0.01
			200,000	3.82435×10^{-302}	0.063	49.92	0.04	0.07	0.01	0.01
	100	666	2,000	9.06698×10^{-59}	0.025	4.47	2.53	2.56	0.01	0.01
			20,000	2.95125×10^{-3}	0.586	46.04	2.53	2.56	0.01	0.01
			200,000	1.07460×10^{-196}	1.495	461.61	2.53	2.56	0.01	0.01
AD(A|D){2}AD	10	128	2,000	6.06131×10^{-5}	0.025	1.12	0.13	0.24	0.01	0.02
			20,000	8.13580×10^{-3}	0.114	11.38	0.13	0.24	0.01	0.02
			200,000	2.54950×10^{-67}	0.158	113.13	0.13	0.24	0.01	0.02
	100	971	2,000	4.58582×10^{-94}	0.027	10.44	8.97	9.08	0.01	0.02
			20,000	1.14066×10^{-34}	0.260	107.05	8.97	9.08	0.01	0.03
			200,000	5.92396×10^{-14}	0.232	1075.90	8.97	9.08	0.01	0.03
AD(A|D){5}AD	2	158	2,000	2.59931×10^{-2}	0.031	1.23	0.07	0.56	0.00	0.20
			20,000	2.55206×10^{-1}	0.040	12.80	0.07	0.56	0.01	0.20
			200,000	1.35276×10^{-8}	0.041	124.76	0.07	0.56	0.01	0.21
	20	278	2,000	1.59351×10^{-22}	0.055	8.76	2.18	2.67	0.02	0.64
			20,000	3.79239×10^{-11}	0.126	88.19	2.18	2.67	0.03	0.65
			200,000	5.79753×10^{-2}	0.044	912.11	2.18	2.67	0.04	0.66
AD(A|D){10}AD	2	75	2,000	2.38948×10^{-4}	0.017	14.38	1.05	7.19	0.20	72.33
			20,000	4.4012×10^{-3}	0.093	148.49	1.05	7.19	0.49	72.42
			200,000	1.33166×10^{-1}	NA	NA	1.05	7.19	0.73	73.79
	20	380	2,000	1.24717×10^{-27}	0.000	100.45	34.41	40.55	0.89	790.21
			20,000	1.25298×10^{-25}	NA	NA	34.41	40.55	2.23	792.76
			200,000	6.25326×10^{-18}	NA	NA	34.41	40.55	3.19	798.07
AD(A|D){15}AD	2	87	2,000	6.74582×10^{-6}	0.001	153.54	12.95	86.41	-	MT
			20,000	7.02066×10^{-5}	NA	NA	12.95	86.41	-	MT
			200,000	9.09232×10^{-4}	NA	NA	12.95	86.41	-	MT
	20	491	2,000	5.72720×10^{-30}	NA	NA	477.05	550.51	-	MT
			20,000	6.39056×10^{-29}	NA	NA	477.05	550.51	-	MT
			200,000	1.42666×10^{-27}	NA	NA	477.05	550.51	-	MT

Table 4: $\mathbb{P}(N_\ell = k)$ for the toy-example motifs over the alphabet $\mathcal{A} = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}\}$ using a i.i.d. and uniformly distributed background model. α is the rank of the partial recursion (depends only on k), "e.r." is the ratio of the relative error of the computation divided by the targeted relative error $\eta = 10^{-15}$, t_0 is the running time to perform the computation using the full recursion, t_3 is the running time to perform the computation using the partial recursion ("$+t_1$" gives the total running time $t_1 + t_3$), t_4 is the running time to perform the computation using the high-order lifting ("$+t_2$" give the total running time $t_2 + t_4$). All running times are expressed in seconds, MT stands for Memory Trashing.

motifs for a various range of values for ℓ and k. For validation purpose, the results of the partial recursion are compared to those of the slower full recursion. The relative error between the two approaches is compared to expected relative error η: in all cases but one the resulting error ratio (e.r.) is below 1.0 thus proving that both results are quite consistent. In the remaining case, e.r. is only slightly larger than 1.0 (1.495) which remains acceptable. In terms of computational time however, the partial recursion approach is dramatically faster that the full recursion. This is especially sensitive for the more complex motifs for which full recursion was not even performed in some cases.

With the high-order lifting approach (Algorithms 2-10) we see that whenever the degree of the bivariate fraction remains small, the overall performance is very good. Moreover, one could compute the fraction once and then use the very fast high-order lifting to recover any coefficient at a negligible cost. Now when the degrees and the size of the involved matrices grows, memory becomes the limiting factor, just to store the series, prior to any computation on it.

In terms of empirical complexity, the full recursion increases at the expected $O(k \times \ell)$ rate. On the other hand, the partial recursion running time is consistent with the $O(\alpha \times k)$ with α increasing at a rough linear rate with k.

5.2 Transcription factors in Human Chromosome 5

In this section, we consider the complete DNA ($\mathcal{A} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$) sequence [7] of the Human Chromosome 5. In order to take into account the codon (DNA words of size 3) structure of the sequence, we adjust a homogeneous order Markov model on the data. Sequence length is $\ell = 131,624,728$, and the sequence starts with the two letters `GA`. The maximum likelihood estimate (MLE) of the transition matrix of the model is directly obtained from the observed counts of all DNA words of size 3. For example, since $N(\mathtt{TAC}) = 1451956$, $N(\mathtt{TAG}) = 1655432$, $N(\mathtt{TAT}) = 2565811$, and $N(\mathtt{TCA}) = 2572660$, we get the MLE:

$$\widehat{\mathbb{P}}(X_i = \mathtt{C} | X_{i-2}X_{i-1} = \mathtt{TA}) = \frac{1451956}{2632852 + 1451956 + 1655432 + 2565811}.$$

One should note that our Markov parameter are then all rationals.

In Table 5 we consider a selection of various Transcription Factors (TFs) motifs. These TFs are typically involved in the regulation of gene expression. The selected motifs range from the simple patterns (ex: `CGCACCC`) to highly complex ones (ex: `GCGCN{15}GCGC`). For each motif, the precomputations necessary for the partial recursion (computation of λ) and the high-order lifting approach (computation of $G(y, z)$) are indicated. Like in Table 3 we see that the running time increases with the motif complexity, eventually resulting in a memory trashing (MT) for the computation the rational reconstruction. One should note that time t_2 is larger for these motifs than for the toy examples of previous section even when the fractional degrees are similar. This is explained by the more complex nature of the model parameters (ex: 1451956/8306051 for the TFs *vs* 1/4 for the toy-example).

[7] `http://www.pseudogenes.org/data/human/build36/genome/chr5.fa`

Transcription Factor	L	F	$1-\lambda$	t_1	frac. deg.	t_2
CGCACCC	21	1	6.10426×10^{-5}	0.13	18/19	3.24
TCCGTGGA	22	1	1.52604×10^{-5}	0.13	19/20	3.62
ACAACAAC	23	1	1.50220×10^{-5}	0.15	21/22	5.71
(A|C)TAAA(C|T)AA	25	2	6.08161×10^{-5}	0.18	20/20	4.36
(A|T){3}TTTGCTC(A|G)	30	2	3.81483×10^{-6}	0.20	23/23	5.50
A{24}	38	1	2.66454×10^{-15}	0.25	36/37	19.58
TA(A|T){4}TAG(A|C)	54	2	3.05260×10^{-5}	0.45	21/22	6.96
(C|T)CCN(C|T)TN(A|G){2}CCGN	66	4	1.52614×10^{-5}	0.63	24/25	9.57
GCGCN{6}GCGC	228	8	1.51359×10^{-5}	3.84	54/55	66.52
CGGN{8}CGG	419	13	2.40786×10^{-4}	10.12	81/82	283.61
TTGACAN{17}TATAAT	2068	34	5.96047×10^{-8}	34.91	173/173	MT
TTGACAN{16, 18}ATATAAT	2904	55	4.47035×10^{-8}	49.18	253/253	MT
GCGCN{15}GCGC	6158	225	1.51359×10^{-5}	202.48	1079/1080	MT

Table 5: Regular expression of Transcription Factors (TFs) defined over the DNA alphabet $\mathcal{A} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ using the IUPAC notation $\mathtt{N} = (\mathtt{A}|\mathtt{C}|\mathtt{G}|\mathtt{T})$. L (resp. F) is the number of states (resp. final states) of the minimal order 2 DFA associated to the TFs. λ is the largest eigenvalue of $\mathbf{P}$, and t_1 the time to compute λ using the power method. "frac. deg." corresponds to the fractional degrees of $G(y, z)$, and t_2 is the time to compute $G(y, z)$ using Algorithm 2. All running times are expressed in seconds, MT stands for Memory Trashing.

Transcription Factor	k	α	$\mathbb{P}(N=k)$	t_3	$+t_1$	t_4	$+t_2$
CGCACCC	10	117	3.64365×10^{-571}	0.19	0.32	0.41	3.65
	20	204	1.27159×10^{-551}	0.60	0.73	1.05	4.32
	40	373	2.07574×10^{-518}	2.10	2.23	3.17	6.44
TCCGTGGA	10	131	1.33747×10^{-268}	0.20	0.33	0.01	3.63
	20	225	3.46367×10^{-252}	0.63	0.76	0.03	3.65
	40	409	3.11336×10^{-225}	2.17	2.30	0.05	3.70
AACAACAAC	10	142	3.86490×10^{-170}	0.25	0.40	0.02	5.73
	20	258	1.22856×10^{-155}	0.88	1.03	0.03	5.76
	40	492	1.69964×10^{-132}	3.24	3.39	0.06	5.79
(A|C)TAAA(C|T)AA	10	136	$6.76399 \times 10^{-8067}$	0.26	0.44	0.53	4.89
	20	240	$4.79070 \times 10^{-8036}$	0.87	1.05	1.35	5.71
	40	449	$3.22178 \times 10^{-7980}$	3.14	3.32	4.07	8.44
(A|T){3}TTTGCTC(A|G)	10	150	6.03263×10^{-579}	0.30	0.50	0.63	6.13
	20	267	2.40165×10^{-559}	0.99	1.19	1.59	7.12
	40	500	5.10153×10^{-526}	3.58	3.78	4.87	10.42
A{24}	5	171	1.16314×10^{-4}	0.31	0.56	0.03	19.61
	10	310	1.09217×10^{-6}	1.01	1.26	0.05	40.32
	20	589	9.62071×10^{-11}	3.68	3.93	0.11	40.27
TA(A|T){4}TAG(A|C)	5	93	$1.60427 \times 10^{-3914}$	0.21	0.66	0.27	7.23
	10	148	$3.23597 \times 10^{-3899}$	0.54	0.99	0.57	6.45
	20	256	$1.79579 \times 10^{-3871}$	1.69	2.14	1.46	7.40
(C|T)CCN(C|T)TN(A|G){2}CCGN	5	150	1.94195×10^{-173}	0.43	1.06	0.02	9.59
	10	215	8.71218×10^{-165}	1.00	1.63	0.02	8.60
	20	342	2.39167×10^{-150}	3.01	3.64	0.05	8.63
GCGCN{6}GCGC	1	65	4.73516×10^{-19}	0.24	4.08	0.62	67.09
	2	92	1.08880×10^{-17}	0.50	4.34	0.87	98.62
	4	138	1.91912×10^{-15}	1.22	5.06	1.49	155.99
CGGN{8}CGG	1	82	5.21188×10^{-467}	0.59	10.71	1.47	284.93
	2	114	2.80818×10^{-464}	1.17	11.29	1.79	403.75
	4	169	2.71751×10^{-459}	2.74	12.86	3.12	651.20
TTGACAN{17}TATAAT	1	92	6.97988×10^{-7}	3.06	37.97	-	MT
	2	137	5.93598×10^{-6}	6.72	41.63	-	MT
	4	199	1.43106×10^{-4}	15.23	50.14	-	MT
TTGACAN{16, 18}ATATAAT	1	96	2.28201×10^{-6}	4.86	54.04	-	MT
	2	129	1.79676×10^{-5}	9.38	58.56	-	MT
	4	202	3.71288×10^{-4}	23.16	72.34	-	MT
GCGCN15GCGC	1	119	4.71467×10^{-19}	12.62	215.10	-	MT
	2	173	1.08420×10^{-17}	27.15	229.63	-	MT
	4	255	1.91136×10^{-15}	63.45	265.93	-	MT

Table 6: $\mathbb{P}(N_\ell = k)$ for the TFs motifs over the alphabet $\mathcal{A} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ using a order 2 homogeneous Markov model. α is the rank of the partial recursion (depends only on k), t_3 is the running time to perform the computation using the partial recursion ("$+t_1$" gives the total running time $t_1 + t_3$), t_4 is the running time to perform the computation using the high-order lifting ("$+t_2$" give the total running time $t_2 + t_4$). All running times are expressed in seconds, MT stands for Memory Trashing.

In Table 6, we can see the computed values of $\mathbb{P}(N_\ell = k)$ for our TFs motifs and for various values of k. Due to the large value of ℓ, the full recursion was not tractable anymore and there is hence no reference value for the probability of interest. However, the results of both approaches are always the same (up to the requested relative precision). For low complexity TFs, the high-lifting is always much faster than the partial recursion when considering only the core computations. However, we get the opposite results when we consider as well the pre-computation time (i.e.: obtaining $G(y, z)$ for the high-order lifting, or computing λ for the partial recursion). Like for the toy-examples, we see that the high-order lifting approach cannot cope with high complexity patterns since the fractional reconstruction is not feasible for them.

5.3 Protein signatures

We consider here the complete human proteome build as the concatenation of all human protein sequences over the 20 aminoacids alphabet (from the Uniprot database [8]) resulting in a unique sequence of length $\ell = 9,884,385$. We fit a order 0 Markov model on these data from the observed counts of all aminoacids:

$$\begin{array}{lllll} N(\mathtt{A}) = 691113, & N(\mathtt{R}) = 555875, & N(\mathtt{N}) = 357955, & N(\mathtt{D}) = 472303, & N(\mathtt{C}) = 227722, \\ N(\mathtt{E}) = 698841, & N(\mathtt{Q}) = 469260, & N(\mathtt{G}) = 649800, & N(\mathtt{H}) = 258779, & N(\mathtt{I}) = 432849, \\ N(\mathtt{L}) = 981769, & N(\mathtt{K}) = 567289, & N(\mathtt{M}) = 212203, & N(\mathtt{F}) = 363883, & N(\mathtt{P}) = 617242, \\ N(\mathtt{S}) = 816977, & N(\mathtt{T}) = 529157, & N(\mathtt{W}) = 119979, & N(\mathtt{Y}) = 267663, & N(\mathtt{V}) = 593726. \end{array}$$

As a consequence, our MLE parameters hence are expressed as rational. For example: $\widehat{\mathbb{P}}(X_i = \mathtt{W}) = 119979/9884385$.

We also consider a selection of 10 PROSITE [9] signatures which corresponds to known functional motifs in proteins. In Table 7, the complexity of the considered motifs are studied along with the computational time to obtain λ (time t_1) or to obtain $G(y, z)$ (time t_2). Motifs are sorted by increasing complexities, from Signature `PILI_CHAPERONE` (whose minimal DFA has $L = 46$ states including $F = 1$ final state) to Signature `SUGAR_TRANSPORT_2` (whose minimal DFA has $L = 1152$ states including $F = 40$ final states). For both approaches, the running time for these precomputations is similar but, like for previous applications, we observe a steeper increase for the fractional reconstruction when considering high complexity motifs.

In Table 8 we compute $\mathbb{P}(N_\ell = k)$ for all considered PROSITE signatures and a range of values for k. For each combination, both the partial recursion and the high-order lifting are performed and the two methods agree perfectly in their results. Like for the TFs, the fast Taylor expansion (time t_4) is much faster that the recursion part (time t_3) but the precomputation of $G(y, z)$ (time t_4) has a high cost, especially for the signatures of high complexity which is consistent with previous observations.

[8] `http://www.uniprot.org`
[9] `http://expasy.org/prosite/`

PROSITE signature	AC	L	F	$1-\lambda$	t_1	degrees	t_2
PILI_CHAPERONE	PS00635	46	1	1.7×10^{-10}	0.63	15/18	0.27
EFACTOR_GTP	PS00301	52	4	1.0×10^{-8}	0.74	14/16	0.41
ALDEHYDE_DEHYDR_CYS	PS00070	67	17	1.1×10^{-6}	0.91	11/12	0.38
SIGMA54_INTERACT_2	PS00676	85	1	8.8×10^{-10}	1.08	0/16	0.40
ADH_ZINC	PS00059	87	8	2.2×10^{-7}	1.40	37/41	4.46
SUGAR_TRANSPORT_1	PS00216	188	54	6.7×10^{-7}	3.48	17/18	1.05
THIOLASE_1	PS00098	254	6	2.6×10^{-15}	5.21	37/38	2.16
FGGY_KINASES_2	PS00445	463	6	2.2×10^{-7}	11.52	26/30	4.10
PTS_EIIA_TYPE_2_HIS	PS00372	756	46	1.3×10^{-9}	17.47	77/80	30.44
SUGAR_TRANSPORT_2	PS00217	1152	40	8.6×10^{-7}	36.95	149/151	110.10

Table 7: Characteristics of some PROSITE signatures defined over the aminoacids alphabet. AC is the accession number in the PROSITE database, L (resp. F) is the number of states (resp. final states) of the minimal order 2 DFA associated to the signatures. λ is the largest eigenvalue of $\mathbf{P}$, and t_1 the time to compute λ using the power method. "frac. deg." corresponds to the fractional degrees of $G(y,z)$, and t_2 is the time to compute $G(y,z)$ using Algorithm 2. All running times are expressed in seconds.

6 Conclusion

We developed two efficient approaches to obtain the exact distribution of a pattern in a long sequence.

The first is approach uses a partial recursion and is suitable even for high complexity patterns. Unfortunately, its quadratic complexity with the observed number of occurrence k make it not recommended for high values of k.

The second approach has two steps: first obtaining $G(y,z)$ through fraction reconstruction of the series $\sum G_\ell(y)z^\ell$ and then performing fast Taylor expansion using high-order lifting. On one hand, in all examples, just computing the series appears to be the bottleneck, especially for high complexity patterns. On the other hand, the fast Taylor expansion is usually very fast, even if the running time increases with the fractional degrees.

Overall, the high-order lifting approach is very efficient for low or med complexity motifs, but cannot deal efficiently with the highly complex motifs. In our examples, we dealt with two real applications (TFs in Human Chromosome 5, and PROSITE signatures in Human complete proteome) which demonstrate the practical interest of our approaches.

Further work include improvement of the precomputation of λ, using e.g. Lanczos-like approaches; the precomputation of $G(y,z)$ can also be improved, for instance reconstructing the rational fraction from approximated evaluations of the series using e.g. Kaltofen and Yang (2007) could yield significant improvements both in terms of memory and computational time.

PROSITE signature	k	α	$\mathbb{P}(N = k)$	t_3	$+t_1$	t_4	$+t_2$
PILI_CHAPERONE	5	125	1.20635×10^{-16}	0.98	1.16	0.01	0.39
	10	221	5.78491×10^{-35}	3.17	3.80	0.01	0.28
	20	413	1.81452×10^{-74}	11.20	11.83	0.02	0.29
EFACTOR_GTP	5	114	7.25588×10^{-8}	1.07	0.84	0.01	0.42
	10	196	2.30705×10^{-17}	3.41	0.95	0.01	0.31
	20	364	3.18090×10^{-39}	12.23	1.28	0.01	0.31
ALDEHYDE_DEHYDR_CYS	5	88	1.85592×10^{-2}	1.13	2.04	0.07	0.44
	10	151	1.15181×10^{-1}	3.59	4.50	0.00	0.33
	20	270	6.05053×10^{-3}	12.27	13.18	0.01	0.34
SIGMA54_INTERACT_2	5	106	4.09432×10^{-13}	1.45	2.53	0.00	0.40
	10	189	6.70971×10^{-28}	4.68	5.76	0.01	0.37
	20	350	2.45724×10^{-60}	16.17	17.25	0.01	0.39
ADH_ZINC	5	116	4.51132×10^{-2}	1.91	3.31	0.04	4.50
	10	196	6.99469×10^{-5}	5.98	7.38	0.05	6.43
	20	352	2.29397×10^{-13}	20.52	21.92	0.09	6.39
SUGAR_TRANSPORT_1	1	60	6.06925×10^{-3}	0.65	4.13	0.05	1.10
	2	75	1.64759×10^{-2}	1.29	4.77	0.08	1.51
	4	110	5.41084×10^{-2}	3.30	6.78	0.12	2.29
THIOLASE_1	1	85	2.54343×10^{-8}	1.23	6.44	0.02	2.18
	2	127	1.61364×10^{-15}	2.85	8.06	0.02	2.79
	4	207	6.25151×10^{-30}	7.96	13.17	0.02	3.54
FGGY_KINASES_2	1	73	2.43018×10^{-1}	2.08	13.60	0.02	4.12
	2	97	2.68005×10^{-1}	4.21	15.73	0.02	5.62
	4	142	1.08650×10^{-1}	10.27	19.48	0.02	7.62
PTS_EIIA_TYPE_2_HIS	1	76	1.23843×10^{-2}	3.41	20.88	0.89	31.33
	2	105	7.76535×10^{-5}	7.32	24.79	1.19	44.27
	4	163	1.0177×10^{-9}	19.18	36.65	2.12	72.01
SUGAR_TRANSPORT_2	1	96	1.71124×10^{-3}	6.78	43.73	3.18	113.28
	2	124	7.28305×10^{-3}	13.51	50.46	4.26	164.82
	4	177	4.39742×10^{-2}	32.81	69.76	7.99	275.25

Table 8: $\mathbb{P}(N_\ell = k)$ for the PROSITE signatures over the aminoacids alphabet a order 0 homogeneous Markov model. α is the rank of the partial recursion (depends only on k), t_3 is the running time to perform the computation using the partial recursion ("$+t_1$" gives the total running time $t_1 + t_3$), t_4 is the running time to perform the computation using the high-order lifting ("$+t_2$" give the total running time $t_2 + t_4$). All running times are expressed in seconds.

Then, dealing with exact computation for frequent (large k) high complexity (large R) motifs yet remains an open problem.

References

D. L. Antzoulakos. Waiting times for patterns in a sequence of multistate trials. *J. Appl. Prob.*, 38:508–518, 2001.

E. Beaudoing, S. Freier, J.R. Wyatt, J.-M. Claverie, and D. Gautheret. Patterns of variant polyadenylation signal usage in human genes. *Genome Res.*, 10(7): 1001–1010, 2000.

V. Boeva, J. Clément, M. Régnier, and M. Vandenbogaert. Assessing the significance of sets of words. In *Combinatorial Pattern Matching 05, Lecture Notes in Computer Science, vol. 3537*, pages 358–370. Springer-Verlag, 2005.

V. Boeva, J. Clement, M. Regnier, 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*, 2:13, 2007.

A. Brazma, I. Jonassen, J. Vilo, and E Ukkonen. Predicting gene regulatory elements in silico on a genomic scale. *Genome Res.*, 8(11):1202–1215, 1998.

Y.-M. Chang. Distribution of waiting time until the rth occurrence of a compound pattern. *Statistics and Probability Letters*, 75(1):29–38, 2005.

T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*, chapter 34, pages 853–885. MIT Press, 1990.

Cowan. Expected frequencies of dna patterns using whittle's formula. *J. Appl. Prob.*, 28:886–892, 1991.

M. Crochemore and C. Hancart. *Handbook of Formal Languages, Volume 2, Linear Modeling: Background and Application*, chapter Automata for Matching Patterns, pages 399–462. Springer-Verlag, Berlin, 1997.

M. Crochemore and V. Stefanov. Waiting time and complexity for matching patterns with automata. *Info. Proc. Letters*, 87(3):119–125, 2003.

A. Denise, M. Régnier, and M. Vandenbogaert. Assessing the statistical significance of overrepresented oligonucleotides. *Lecture Notes in Computer Science*, 2149:85–97, 2001.

Jean-Guillaume Dumas, Jean-Louis Roch, Eric Tannier, and Sébastien Varrette. *Théorie des codes : Compression, Cryptage, Correction.* Dunod, 2007.

M. El Karoui, V. Biaudet, S. Schbath, and A. Gruss. Characteristics of chi distribution on different bacterial genomes. *Res. Microbiol.*, 150:579–587, 1999.

T. Erhardsson. Compound Poisson approximation for counts of rare patterns in Markov chains and extreme sojourns in birth-death chains. *Ann. Appl. Probab.*, 10(2):573–591, 2000.

Charles M. Fiduccia. An efficient formula for linear recurrences. *SIAM Journal on Computing*, 14(1):106–112, February 1985.

M. C. Frith, J. L. Spouge, U. Hansen, and Z. Weng. Statistical significance of clusters of motifs represented by position specific scoring matrices in nucleotide sequences. *Nucl. Acids. Res.*, 30(14):3214–3224, 2002.

J. C. Fu. Distribution theory of runs and patterns associated with a sequence of multi-state trials. *Statistica Sinica*, 6(4):957–974, 1996.

Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 1999. ISBN 0-521-64176-4.

M. X. Geske, A. P. Godbole, A. A. Schaffner, A. M Skrolnick, and G. L. Wallstrom. Compound poisson approximations for word patterns under markovian hypotheses. *J. Appl. Probab.*, 32:877–892, 1995.

A. P. Godbole. Poissons approximations for runs and patterns of rare events. *Adv. Appl. Prob.*, 23, 1991.

S. Hampson, D. Kibler, and P. Baldi. Distribution patterns of over-represented k-mers in non-coding yeast DNA. *Bioinformatics*, 18(4):513–528, 2002.

J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction the automata theory, languages, and computation, 2d ed.* ACM Press, New York, 2001.

Erich Kaltofen and Zhengfeng Yang. On exact and approximate interpolation of sparse rational functions. In Christopher W. Brown, editor, *Proceedings of the 2007 ACM International Symposium on Symbolic and Algebraic Computation, Waterloo, Canada*, pages 203–210. ACM Press, New York, July 29 – August 1 2007.

S. Karlin, C. Burge, and A.M. Campbell. Statistical analyses of counts and distributions of restriction sites in DNA sequences. *Nucl. Acids. Res.*, 20(6): 1363–1370, 1992.

J. Kleffe and M. Borodovski. First and second moment of counts of words in random texts generated by markov chains. *Bioinformatics*, 8(5):433–441, 1997.

Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, 2nd edition, 1997. ISBN 0-201-89684-2.

Gavin C. Kanga Leonardo Mariño-Ramírez, John L. Spouge and David Landsman. Statistical analysis of over-represented words in human promoter sequences. *Nuc. Acids Res.*, 32(3):949–958, 2004.

M. E. Lladser. Mininal markov chain embeddings of pattern problems. In *Information Theory and Applications Workshop*, pages 251–255, 2007.

M. Lothaire, editor. *Applied Combinatorics on Words*. Cambridge University Press, Cambridge, 2005.

Pierre Nicodème, Bruno Salvy, and Philippe Flajolet. Motif statistics. *Theoretical Computer Science*, 287(2):593–617, September 2002. ISSN 0304-3975.

G. Nuel. Ld-spatt: Large deviations statistics for patterns on markov chains. *J. Comp. Biol.*, 11(6):1023–1033, 2004.

G. Nuel. Effective p-value computations using Finite Markov Chain Imbedding (FMCI): application to local score and to pattern statistics. *Algorithms for Molecular Biology*, 1(1):5, 2006a.

G. Nuel. Numerical solutions for patterns statistics on markov chains. *Stat. App. in Genet. and Mol. Biol.*, 5(1):26, 2006b.

G. Nuel. Pattern Markov chains: optimal Markov chain embedding through deterministic finite automata. *J. of Applied Prob.*, 45(1):226–243, 2008.

P.A. Pevzner, M.Y. Borodovski, and A.A. Mironov. Linguistic of nucleotide sequences: The significance of deviation from mean statistical characteristics and prediction of frequencies of occurrence of words. *J. Biomol. Struct. Dyn.*, 6:1013–1026, 1989.

B. Prum, F. Rodolphe, and E. de Turckheim. Finding words with unexpected frequencies in dna sequences. *J. R. Statist. Soc. B*, 11:190–192, 1995.

M. Reignier. A unified approach to word occurrences probabilities. *Discrete Applied Mathematics*, 104(1):259–280, 2000.

G. Reinert and S. Schbath. Compound poisson and poisson process approximations for occurrences of multiple words in markov chains. *J. of Comp. Biol.*, 5:223–254, 1999.

P. Ribeca and E. Raineri. Faster exact Markovian probability functions for motif occurrences: a DFA-only approach. *Bioinformatics*, 24(24):2839–2848, 2008.

Bruno Salvy. Solutions rationnelles de systèmes linéaires à coefficients polynomiaux. In *Algorithmes en Calcul Formel et en Automatique*, chapter 7. 2009. http://algo.inria.fr/chyzak/mpri/poly-20091201.pdf.

V. Stefanov and A. G. Pakes. Explicit distributional results in pattern formation. *Ann. Appl. Probab.*, 7:666–678, 1997.

V. T. Stefanov and W. Szpankowski. Waiting Time Distributions for Pattern Occurrence in a Constrained Sequence. *Discrete Mathematics and Theoretical Computer Science*, 9(1):305–320, 2007.

A. Storjohann. High-order lifting. In Teo Mora, editor, *Proceedings of the 2002 ACM International Symposium on Symbolic and Algebraic Computation, Lille, France*, pages 246–254. ACM Press, New York, July 2002.

J. van Helden, B. André, and J. Collado-Vides. Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *J. Mol. Biol.*, 281(5):827–842, 1998.