Regexpcount, a symbolic package for counting problems on regular expressions and words

Pierre Nicodème
Laboratoire Statistiques et Génome
ESA 8071 CNRS - La génopôle, Evry, France
and
Algorithm Project, INRIA-Rocquencourt
Domaine de Voluceau, BP 105, 78153 Le Chesnay Cedex France

Abstract. In previous work (Nicodème et al., 1999), we considered algorithms related to the statistics of word occurrences and regular expression occurrences in texts generated by Bernoulli or Markov sources. In this work these algorithms are extended for two purposes: to determine the statistics of simultaneous counting of different motifs, and to compute the waiting time for the first match with a motif in a model which may be constrained. This extension also handles matches with errors. The package is fully implemented and gives access to high and low level commands. We also consider an example corresponding to a practical biological problem: getting the statistics for the number of matches of words of size 8 in a genome (a Markovian sequence), knowing that an (overrepresented DNA protecting) Chi pattern occurs a given number of times.

1 Introduction

An important class of computational biology problems is related to counting. When considering words, one is interested in the probability of match in a sequence, the statistics of occurrences in a genome, and the selection of words with unusual counts. These problems have been studied by several authors using combinatorics or probabilistic (Poisson approximation) methods, in the Bernoulli and Markov case (Pevzner et al., 1989; Prum et al., 1995; Régnier, 1998; Régnier & Szpankowski, 1998; Reinert & Schbath, 1998). When considering regular expressions or motifs such as Prosite motifs, one is also interested in probability of match in a sequence (Sewell & Durbin, 1995), or, once again, in the statistics of number of matches (Nicodème et al., 1999). Here, we consider regular expressions, with words and patterns (finite set of words) as a subclass of these, and texts generated by either a (uniform or non-uniform) Bernoulli source or by a Markovian source. In (Nicodème et al., 1999), we addressed the counting problem for matches of one
motif in these texts. Our method was based on construction of automata and the analysis of their generating functions (with one variable and one parameter). We extend this approach to handle simultaneous counting of several motifs, which corresponds to generating functions with one variable and several parameters and gives access to covariance statistics. We also consider a new statistic, the waiting time for the first match with a regular expression \textit{RE2}; this last problem may be constrained by knowing that a match just occurred with a regular expression \textit{RE2}, either with rematch with \textit{RE2} allowed, or forbidden. Another new feature of the package is the possibility of handling statistics for matches with errors, for all the problems considered. In Section 2 we give some examples of our computations. In Section 3 we provide the necessary definitions, and in Sections 4 and 5 we describe our methods and algorithms. We give in Section 6 a semi-automatic application to a biological problem.

2 Examples

Table 1 contains some examples of the possible computations, with the following notation:

<table>
<thead>
<tr>
<th>pat.</th>
<th>stat.</th>
<th>md.</th>
<th>(\mu(n))</th>
<th>(\sigma^2(n))</th>
<th>(\mu(1000))</th>
<th>(\sigma(1000))</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{abab}</td>
<td>n.m.o.</td>
<td>\textit{B}_0</td>
<td>(\frac{1}{10}n - \frac{3}{10} + O(\alpha^n))</td>
<td>(\frac{17}{250}n - \frac{35}{250} + O(\alpha^n))</td>
<td>62.31</td>
<td>8.13</td>
</tr>
<tr>
<td>\textit{aaab}</td>
<td>n.m.o.</td>
<td>\textit{B}_0</td>
<td>(\frac{1}{10}n - \frac{3}{10} + O(\alpha^n))</td>
<td>(\frac{9}{250}n - \frac{15}{250} + O(\alpha^n))</td>
<td>62.31</td>
<td>5.92</td>
</tr>
<tr>
<td>\textit{abab}</td>
<td>n.m.r.</td>
<td>\textit{B}_0</td>
<td>(\frac{1}{10}n - \frac{13}{100} + O(\alpha^n))</td>
<td>(\frac{69}{2000}n - \frac{139}{2000} + O(\alpha^n))</td>
<td>49.87</td>
<td>5.86</td>
</tr>
<tr>
<td>\textit{abab}</td>
<td>n.m.o.</td>
<td>\textit{B}_1</td>
<td>(\frac{9}{10}n - \frac{27}{100} + O(\alpha^n))</td>
<td>(\frac{2601}{2000}n - \frac{5253}{2000} + O(\alpha^n))</td>
<td>35.05</td>
<td>6.28</td>
</tr>
<tr>
<td>\textit{abab}</td>
<td>n.m.o.</td>
<td>\textit{M}_1</td>
<td>(\frac{9}{10}n - \frac{297}{1000} + O(\alpha^n))</td>
<td>(\frac{2627}{2000}n - \frac{5263}{2000} + O(\alpha^n))</td>
<td>112.12</td>
<td>10.62</td>
</tr>
<tr>
<td>(\Delta(\textit{abab}, 1))</td>
<td>n.m.o.</td>
<td>\textit{B}_0</td>
<td>(\frac{5}{8}n - \frac{11}{8} + O(\alpha^n))</td>
<td>(\frac{19}{64}n - \frac{23}{64} + O(\alpha^n))</td>
<td>623.62</td>
<td>17.21</td>
</tr>
<tr>
<td>(\Delta(\textit{abab}, 1))</td>
<td>n.m.o.</td>
<td>\textit{B}_1</td>
<td>(\frac{63}{128}n - \frac{141}{128} + O(\alpha^n))</td>
<td>(\frac{6957}{1024}n - \frac{12663}{1024} + O(\alpha^n))</td>
<td>491.08</td>
<td>20.58</td>
</tr>
<tr>
<td>(\Delta(\textit{abab}, 1))</td>
<td>n.m.o.</td>
<td>\textit{M}_1</td>
<td>(\frac{57}{80}n - \frac{1311}{8000} + O(\alpha^n))</td>
<td>(\frac{7323}{3200}n - \frac{17601}{3200} + O(\alpha^n))</td>
<td>710.86</td>
<td>15.12</td>
</tr>
</tbody>
</table>

Table 1: Examples of computations (see notation in Section 2)
pat.: pattern. $\Delta (regexp, k)$: language of words at edit distance (substitution, insertion, deletion) $\leq k$ from $regexp$. In the case of the wait statistics, two patterns are given (see stat.).

stat.: considered statistics; (1) n.m.o : number of matches in the overlapping case (overlapping matches are accepted); (2) n.m.r. : number of matches in the renewal case (restart after each match, or non-overlapping case); (3) wait (pattern $p_1, p_2$): wait time for the first match with pattern $p_2$, knowing that pattern $p_1$ has been found (rematch with $p_1$ is allowed during the wait). If $p_1 = \epsilon$, wait time for the first match with $p_2$ from the beginning of the text.

md.: model; (1) $B_0$: Bernoulli uniform; (2) $B_1$: Bernoulli non-uniform (probability of letters $\pi_a = 1/4, \pi_b = 3/4$); (3) $M_1$: Markov model of order 1; (probability of pairing of letters: $\pi_{v,a} = 1/4, \pi_{v,b} = 3/4, \pi_{a,a} = 1/4, \pi_{a,b} = 3/4, \pi_{b,a} = 1/2, \pi_{b,b} = 1/2$, where $\pi_{v,x}$ is the probability that the start letter is $x$ and $\pi_x \neq v, y$ is the probability that the letter $x$ is followed by the letter $y$ in the text.

$\mu(n), \sigma^2(n)$: asymptotic value of the expectation and variance for texts of size $n$.

$\mu(1000), \sigma(1000)$: numerical evaluation of the expectation and standard deviation for texts of size 1000.

$\mu, \sigma$ (wait statistics): numerical evaluation of the expectation and variance for the wait time.

See the appendix for a Maple session which computes lines $\Delta (abab, 1)$, n.m.o, $B_1$ and $M_1$ of Table 1.

3 Definitions

We recall in this section classical definitions for languages, generating functions and finite automata.

Languages. An alphabet denoted $\Sigma$ is a set of letters. A word is a concatenation of letters of the alphabet. A language is a subset of the set $\Sigma^*$ of all words over the alphabet. Concatenation of languages is denoted by a product ($A_1 A_2 = \{ w_1 w_2, w_1 \in A_1 , w_2 \in A_2 \}$). Union of languages is the classical set union. The empty word is denoted by $\epsilon$ and the Kleene star operator “*” is understood as $A^* = \epsilon + A + A^2 + A^3 + \ldots$, where $A^2 = AA$ and so on. A regular language is built by recursively applying Union, Concatenation and Kleene-Star operators to elementary regular expressions restricted to one letter of $\{\Sigma, \epsilon\}$. A regular expression is a short-hand description of a regular language (most classically using symbols “+, *” and brackets).

Generating functions. We give the definitions for an alphabet of 2 letters $\Sigma = \{a, b\}$ and consider generating functions for a language $L$. They generalise immediately to alphabets of higher cardinality. Define a counting generating function as $F(a, b) = \sum c_{i,j} a^i b^j$ where $c_{i,j}$ is the number of words of $L$ with $i$ letters $a$ and $j$ letters $b$; this is equivalent to $F(a, b) = \sum_{w \in L} \text{com}(w)$, where the operator “com”, applied to a word $w$, produces the monomial obtained by letting the letters of $w$ commute. For
example, if \( L = \{aab, aba\} \) then \( F(a, b) = 2a^2b \). Remark that if \( d_n \) is defined by \( F(z, z) = \sum d_n z^n \) then \( d_n \) counts the number of words of size \( n \) in the language. Define a probability generating function as \( P(z) = \sum \omega_n z^n \), where \( \omega_n \) is the probability that a word of size \( n \) belongs to \( L \). In the Bernoulli models \( B_0 \) and \( B_1 \) this is \( F(z/2, z/2) \) and \( F(\pi_a z, \pi_b z) \) where \( \pi_a, \pi_b \) are the probabilities of occurrences of letters \( a \) and \( b \) respectively. For example with \( L = \{aab, aba\} \), we have \( P(z) = z^3/4 \) in the model \( B_0 \). In the Markov model the probability of each word is computed using the probability of occurrences of the first letters and the transitions probabilities (according to the order of the Markovian source).

**Translation rules.** If unions are disjoint and concatenations are unambiguous, unions of languages translate to sums of generating functions and concatenations translate to products of generating functions; with the same conditions, the counting and probability generating functions of \( L^* \) are the quasi-inverses \( F_L(a, b) = 1/(1 - F_L(a, b)) \) and \( P_L(z) = 1/(1 - P_L(z)) \).

**Automata.** An Nondeterministic Finite Automaton (or NFA) is a 5-tuple \((\Sigma, Q, s, F, \delta)\) such that: \( \Sigma \) is the input alphabet; \( Q \) is a finite collection of states; \( s \in \Sigma \) is the start state; \( F \subseteq Q \) is the collection of final states; \( \delta \) is a (possibly partial) transition function from \( Q \times \Sigma \) to \( 2^Q \) the set of subsets of \( Q \).

There exists a transition from state \( q_i \) to state \( q_j \) if there is a letter \( \ell \in \Sigma \) such that \( q_j \in \delta(q_i, \ell) \). A word \( w = w_1 w_2 \cdots w_n \in \Sigma^* \) is accepted or recognised by an NFA \( A = (\Sigma, Q, s, F, \delta) \) if there exists a sequence of states \( q_{i_0}, q_{i_1}, q_{i_2}, \ldots, q_{i_n} \) such that \( q_{i_0} = s, q_{i_j} \in \delta(q_{i_{j-1}}, w_j) \) and \( q_{i_n} \in F \).

Kleene’s theorem states that a language is regular if and only if it is recognised by an NFA. Given an input regular language there are several algorithms to construct an NFA that recognises it. See (Kelley, 1995) among numerous text books for the construction with \( \epsilon \)-transitions or (Berry & Sethi, 1986) for the Berry-Sethy algorithm. Deterministic finite automata (or DFAs) are special cases of NFAs where the images of the transition function are singletons. By a classical theorem of Rabin & Scott, NFAs are equivalent to DFAs in the sense that they recognise the same class of languages.

An important theorem states that the generating function of a regular language is rational (Chomsky & Schützenberger, 1963); the proof uses a DFA recognising the language as an intermediate step.

## 4 Methods

### 4.1 Number of matches

The process is best illustrated with an example. Consider the pattern \( aba \) and counting the number of matches of this pattern in random texts over the alphabet \( \Sigma = \{a, b\} \). This has been done by considering marked texts where a mark \( m \) not belonging to \( \Sigma \) is inserted in the texts after each match. If we consider the text \( t = aababaabaab \), we get mark\((t) = aabambamaabamb \) if overlapping matches are allowed and mark\((t) = aabambaabamb \) otherwise.
A marked automaton is an automaton \( \{\Sigma, Q, s, F, \delta, M_1, \ldots, M_k\} \) where \( M_1, \ldots, M_k \) are subsets of the set \( Q \) of states that are distinguished. For instance, consider the DFA \( A = \{\Sigma, Q, s, F, \delta\} \) recognising the regular expression \( \Sigma^*aba \). When running this automaton against a text, each state reached after reading an \( aba \) is a final state. From there, define a marked automaton \( A_m \) with marked subset \( M_1 \) as \( A_m = (\Sigma, Q, s, Q, \delta, F) \). We consider now a DFA \( A' = (\{a, am, b, am\}, Q, s, Q, \delta_m) \) deduced from \( A_m \) by transforming \( \delta \) to a marked transition function \( \delta_m \) as follows: \( \forall s \in Q, \forall l \in \Sigma, \) if \( \delta(s, l) \not\in M_1 \) then \( \delta_m(s,l) = \delta(s,l) \), else \( \delta_m(s,lm) = \delta(s,l) \) and \( \delta_m(s,l) \) is not defined. Then \( A' \) recognises all the marked texts where the mark \( m \) has been inserted after each match. This method has been developed in (Nicodème et al., 1999) and we extend it to the case where \( r \) patterns are considered. In the latter case, we define \( r \) marked sets \( M_1, \ldots, M_r \) corresponding to the matches with each pattern, and a suitable rule of marking the transitions is chosen.

Since the marked automata we constructed in the last paragraph are deterministic, the Chomsky-Schützenberger method is available. It produces, in the Bernoulli case, a generating function \( F(a,b,m) \). We are interested in texts where the total number of letters \( a \) and \( b \) is \( n \). From \( F(a,b,m) \) we get the multivariate probability generating function \( P(z,u) = F(\pi_a z, \pi_b z, u) = \sum p_{n,k} u^k z^n \), where \( p_{n,k} \) is the probability that a text of size \( n \) contains \( k \) occurrences of the pattern.

### 4.2 Waiting times

Next consider the patterns \( p_1 = ab \) and \( p_2 = ba \). In this case, we are interested in the waiting time for a match with \( p_2 \), knowing that a match with \( p_1 \) occurred. We consider the text \( t = abbbba \). This text begins with a match with \( p_1 \), finishes with a match with \( p_2 \), and has no other matches with \( p_2 \). In this case, the mark text is \( \text{mark}(t) = abmbmbmam \); this text contains \( m \) marks, corresponding to the number of letters read after the match with \( p_1 \) until the match with \( p_2 \) occurred. We assume that we know how to construct a marked automaton \( B_{m,1} = (\Sigma, Q, s, F, \delta, M_1, M_2) \) verifying the following conditions: \( B_{m,1} \) recognises the texts that begin with a match with \( p_1 \), end with a match with \( p_2 \) and have no other match with \( p_2 \) (except possibly within the first match with \( p_1 \)); the DFA \( B_{p_1} = (\Sigma, Q, s, M_1, \delta) \) recognises \( p_1 \); if \( Q_{p_1} \) is the subset of states of \( Q \) in the DFA \( B_{p_1} \) crossed until \( p_1 \) is recognised, no path from a state of \( M_1 \) to a state of \( F \) intersects \( Q_{p_1} \); \( M_2 = Q - Q_{p_1} \). Said in different words, the constructed automaton splits into a first part where \( p_1 \) is recognised and into a second part where the first match with \( p_2 \) is recognised, and the marked set \( M_2 \) is the set of states of the second part. Forget \( M_1 \) in \( B_{m,1} \) to produce \( B_{m,2} = (\Sigma, Q, s, F, \delta, M_2) \). Transform \( B_{m,2} \) to an automaton \( B_{m,3} \) with marked transitions and translate again into a generating function \( F(a,b,m) \) which enumerates the marked texts. The substitution \( P(u) = F(\pi_a, \pi_b, u) = \sum s_n u^n \) provides in the Bernoulli model a univariate probability generating function where \( s_n \) is the probability that the waiting time is \( n \). From there, the expectation and second moment of the statistics are obtained immediately.
5 Algorithms

We consider a problem for one or several motifs $R_1, \ldots, R_i$ ($i \geq 1$) The algorithmic chain is as follows.

1. If the match(es) with the motif $R_i$ is (are) considered with a function of error $\Delta$, build first an automaton recognising $R_i$ and next an “error” automaton recognising $\Delta(R_i)$. Use classical $\epsilon$-transitions constructions to connect this automaton to the automata built during the following step.
2. Construct a deterministic Bernoulli automaton corresponding to the problem.
3. If the problem is Markov, transform the last automaton constructed to a Markov automaton.
4. Compute the (eventually multivariate) counting or probability generating function of the language recognised by the automaton.
5. Use computer algebra methods to extract the Taylor coefficients of order $n$ of the generating functions, which provide expectation, moment of order 2, or covariance of the statistical parameters under study. See (Nicodème et al., 1999) for details.

Remark about implementation: steps 1-3 are implemented in the package regexpcount; step 4 is performed either by use of the package combstruct (combinatorial structures), or by combination of functions of the packages regexpcount and combstruct and by the standard solver of Maple; step 5 is performed either by the package gfun (generating functions) when computing exact coefficients or by the packages equivalent (asymptotic expansions of coefficients of generating functions) and gdev (general asymptotic expansions) for asymptotic results.

NFA determinization uses the subset construction adapted to the case of marked automata.

Automaton minimisation is performed when necessary during the algorithmic chain.

The minimisation algorithm used is a slight generalisation of the Hopcroft $n \log(n)$ algorithm adapted for the case of marked automata.

Product automaton. An automaton is complete if for any state and any letter the transition function is defined. Given two complete DFAs $A_i = (\Sigma, Q_i, s_i, F_i, \delta_i), i = 1, 2$, the marked product automaton $B = A_1 \times A_2 = (\Sigma, Q_B, [s_1, s_2], F_B, \delta_B, M_1, M_2)$ (where $[.,.]$ denotes an ordered list) is constructed as follows: $Q_B$ is a subset of $\{[x, y], x \in Q_1, y \in Q_2\}$; if $[x, y] \in Q_B$, then $\delta_B([x, y], l) = [\delta_1(x, l), \delta_2(y, l)]$ belongs to $Q_B$. The start state $[s_1, s_2]$ belongs to $Q_B$. $F_B = \{[x, y], x \in F_1, y \in F_2\}$. The distinguished subsets are $M_1 = \{[x, y], x \in F_1\}, M_2 = \{[x, y], y \in F_2\}$. The construction is classical, but completed to handle the case of marked automata. This construction generalises immediately to the product of more than 2 automata.

Number of occurrences. The algorithm to construct an automaton adapted to count the number of occurrences of a pattern in a text has been described in (Nicodème et al., 1999) and in Section 4. To simultaneously consider the occurrences of 2 patterns $p_1$ and $p_2$, construct two automata $A_1$ and $A_2$ recognising the regular
expressions $\Sigma^*p_1$ and $\Sigma^*p_2$ respectively and compute the product $B = A_1 \times A_2 = \{\Sigma, Q, s, Q, M_1, M_2\}$ of the two automata, where the set of final states is made equal to $Q$. This is the required automaton. The corresponding probability generating function is of the form $P(z, u, v) = \sum p_{n,i,j}u^iv^jz^n$ where $p_{n,i,j}$ is the probability that a text of size $n$ has $i$ occurrences of $p_1$ and $j$ occurrences of $p_2$. The construction generalises immediately to the case where more than two patterns are simultaneously considered.

**Waiting time, first match** with a pattern $p_1$. Construct the automaton for $\Sigma^*p_1$ and erase all transitions from final states.

**Waiting time $p_1 \rightarrow p_2$** for a match with $p_2$ after a match with $p_1$. Construct the automata $A_1$ recognising $p_1$ and $A_2$ recognising $\Sigma^*p_2$. Construct the automaton $B = A_1 \times A_2 = (\Sigma, Q, s, F, M_1, M_2)$. Construct an automaton $C$ by duplicating the states of $B$ by an injective copy function $\psi$ of $Q$ on $\psi(Q)$ such that $Q \cap \psi(Q) = \emptyset$. Then $C = (\Sigma, Q \cup \psi(Q), s, \psi(M_2), \delta_C, \psi(Q))$ and $\delta_C$ is defined as follows: if $s \in Q - M_1$, then $\delta_C(s, l) = \delta(s, l)$; if $s \in M_1$, then $\delta_C(s, l) = \psi(\delta(s, l))$; and if $s \in \psi(Q)$, then $\delta_C(s, l) = \psi(\psi^{-1}(s, l)))$. The distinguished states are all the copied states.

**Error automaton.** We only consider the case of an error function $\Delta$ with 1 substitution allowed over a motif $R_k$. Insertions and deletions are similar. Let $A_i = (\Sigma, Q, s, F, \delta)$ be a complete DFA recognising $R_i$. We use a copy function $\psi$ as in the preceding paragraph. The NFA error automaton is $B_i = (\Sigma, Q + \psi(Q), s, F + \psi(F), \delta_C, \psi)$. Here, we have for $\delta_C$: $\forall s \in Q, \delta_C(\psi(s), l) = \psi(\delta(s, l))$, $\delta_C(s, l) = \delta(s, l) \cup \{\psi(\delta(s, l_i \in \Sigma - l))\}$. Use $B_i$ as input in the algorithm corresponding to the problem to produce an $\epsilon$-NFA. Determine and minimise it, and translate it to a generating function. In case of insertions or deletions, use $\epsilon$-transitions during the construction of $B_i$. When $k$ errors are allowed, repeat the construction $k - 1$ times.

**Markov automaton.** The construction from a Bernoulli automaton into a Markov automaton of order 1 has been given in (Nicodème et al., 1999). The extension to a Markov automaton of order $k$ is immediate: Let $s$ be a state of the Bernoulli automaton, and let $W = \{w_i, w_i$ is a path of length $k$ ending in $s\}$. Create states $s_{w_1}, w_i \in W$ in the Markov automaton and add the necessary transitions.

Remark that, in the case of a Markov of order $k$, if letter $\rho$ is used for the transitions, a transition $\rho_{x_1, \ldots, x_k, x_{k+1}}$ from a state $s$ means that the state has been entered by a transition of the form $\rho_{y, x_1, \ldots, x_k}$ and that the letter currently read is $x_{k+1}$. It translates to $\pi_{x_1, \ldots, x_k, x_{k+1}} = \pi_{x_1, \ldots, x_{k+1}}$ when computing generating functions.

## 6 Occurrences of words under constraint

Although the full power of the package `regexpcount` is best demonstrated on regular expressions, we give an application of our computations to a biological problem over words. This application could be fully automatised. We consider the $\chi$ (Chi) sequence $gxtggtgg$ of the 1830140 bp. long genome of $H. Influenzae$ read in the transcription direction. We consider the counts $c_{i,j}$, with $i, j \in a, c, g, t$ where the base $i$
is followed by the base $j$ in the genome. From this, we deduce the Markov probability $\pi_{i,j} = c_{i,j}/(c_{i,a}+c_{i,c}+c_{i,g}+c_{i,t})$. With this Markov model of order 1, the expectation and standard deviation of number of occurrences of $\chi$ in the genome are 56.26 and 7.59. However, the $\chi$ sequence is highly overrepresented in the genome, and found 223 times. When looking for exceptional words in H. Influenzae, we must condition the statistics by the effective number of occurrences of the $\chi$ sequence. We compute the constrained statistics of the word $w = tggtgggc$ as follows. Construct an automaton $A_1$ for $\Sigma^*xgtggg$ and an automaton $A_2$ for $\Sigma^*tggtgggc$ ($\Sigma = \{a, c, g, t\}$). Compute the product $B = Markov(A_1 \times A_2) = \{\Sigma, Q, s, F, \delta, M_1, M_2\}$, where $M_1$ and $M_2$ are the set of states corresponding to matches with $\chi$ and with $w$. Set $F = Q$ in $B$ to recognise all the marked texts $mark_{m_1,m_2}(\Sigma^*)$, where $m_1$ and $m_2$ are respectively inserted after a match with $\chi$ and with $w$. Compute the generating function $F(z,u,v) = \sum p_{n,i,j}u^iv^jz^n$ where $p_{n,i,j}$ is the probability that a text of size $n$ distributed as H. Influenzae contains $i$ occurrences of $\chi$ and $j$ occurrences of $w$.

We use a mean-shifting method to get the constrained probability. Consider

$$\phi(n, \alpha) = \frac{[z^n] \frac{\partial F(z,\alpha,1)}{\partial \alpha}}{[z^n]F(z, \alpha, 1)} \quad \text{and} \quad \mu_c = \frac{[z^n] \frac{\partial F(z,\alpha_0,v)}{\partial v}}{[z^n]F(z, \alpha_0, 1)}.$$ (1)

Remark that $\phi(n, \alpha)$ is the shifted mean of occurrences of $\chi$ for parameter $\alpha$. Solving $\phi(1830140, \alpha) = 223$ numerically provides $\alpha_0 = 3.715$. The expectation $\mu_c$ of the constrained statistics is given in Eq 1. The moment of order 2 is computed similarly, and from there, the standard deviation $\sigma_c$ follows. We get after a 35 seconds computation the numerical values $\mu_c = 23.52$ and $\sigma_c = 4.85$, to be compared to the unconditioned parameters $\mu = 15.21$ and $\sigma = 3.9$ (computed in 3 seconds). Theoretical and numerical considerations indicate that $\mu \approx \sigma^2$ and $\mu_c \approx \sigma_c^2$; computation of $\mu_c$ alone that requires only 12 see therefore provides a good estimation of $\sigma_c$. As a comparison, the package R'MES at http://www-bia.inra.fr/J/AB genomf/ computes the unconstrained statistics for all the words of size 8 in the Markov case in a few seconds, but fails to compute the constrained case.

7 Conclusion

We provide here a general purpose symbolic package for statistical properties of occurrences of words and regular expressions. Our method goes through the construction of automata and translations to generating function. Although determinization could lead to an explosion of the size of the automata constructed, previous work shows that this is not the case when considering exact matches for a biological application such as calibrating Prosite motifs. The package should be able to cope with DNA or RNA motifs with errors in reasonable time. Semi-combinatorial methods could use the output of the waiting statistics to compute generating functions for $r$-scans of motifs. Complete biological applications and a push-button interface are part of future work.
Availability. The regexpcount package is written in Maple. It is fully documented and available at http://algo.inria.fr/libraries/software.html. The packages combstruct, gfun, equivalent and gdev are also at this address.

Acknowledgements. The regexpcount package has been developed in close collaboration with Bruno Salvy. Philippe Flajolet indicated to me the mean-shifting method.

References


A Appendix: a Maple session

We consider here the number of occurrences with overlap and one possible error (insertion, substitution, deletion) of the pattern \( abab \) in the non-uniform Bernoulli and Markov models. The asymptotic expectation and variance for these statistics are computed (see lines \( \Delta(abab,1) \), \( n.m.o \), \( B_1 \) and \( M_1 \) of Table 1).

\[ \text{Bw} := [[1/4,a],[3/4,b]]: \quad \text{# Bernoulli weights} \]

\[ \text{Mw} := [[1/4,\text{rho}[a]],[3/4,\text{rho}[b]]], \quad \text{# Markov weights} \]

\[ \text{G} := \{ \text{abab}=\text{Prod}(a,b,a,b), \quad a=\text{Atom}, \quad b=\text{Atom} \}: \quad \text{# grammar for word abab} \]

\[ \text{t1} := \text{time}(); \]

\[ \text{marked_automat} := \text{regexpcount}[\text{regexptomatchesgram}](\text{G}, \text{ABAB}, \]

\[ [[\text{abab}, \text{m}, 'overlap', 'error' \quad [1, 'subj', 'ins', 'del']]]); \]

\[ \text{time}() - \text{t1}; \]

\[ \text{marked_automat} := \begin{align*}
    &\{ a = \text{Atom}, b = \text{Atom}, \\
    &  w6 = \text{Union}(E, \text{Prod}(b, \text{w5}), \text{Prod}(a, \text{w1})), \\
    &  w1 = \text{Union}(E, \text{Prod}(a, \text{w1}), \text{Prod}(b, \text{m}, \text{w2})), m = E, \\
    &  w5 = \text{Union}(E, \text{Prod}(b, \text{w5}), \text{Prod}(a, \text{w1})), \\
    &  w7 = \text{Union}(E, \text{Prod}(b, \text{w3}), \text{Prod}(a, \text{w1})), \\
    &  w3 = \text{Union}(E, \text{Prod}(a, \text{m}, \text{w8}), \text{Prod}(b, \text{m}, \text{w4})), \\
    &  ABAB = \text{Union}(E, \text{Prod}(a, \text{w7}), \text{Prod}(b, \text{w5})), \\
    &  w2 = \text{Union}(E, \text{Prod}(a, \text{m}, \text{w8}), \text{Prod}(b, \text{m}, \text{w4})), \\
    &  w4 = \text{Union}(E, \text{Prod}(b, \text{m}, \text{w6}), \text{Prod}(a, \text{w1})), \\
    &  w8 = \text{Union}(E, \text{Prod}(a, \text{m}, \text{w9}), \text{Prod}(b, \text{m}, \text{w2})), \\
    &  w9 = \text{Union}(E, \text{Prod}(a, \text{w1}), \text{Prod}(b, \text{m}, \text{w2})) \} \]
\]

\[ .167 \]

\[ \text{getvals} := \text{proc}(\text{auto}, \text{init}, \text{weight}) \]

\[ \text{local} \ \text{gfz}1, \ \text{gfz}2, \ \text{gfzu}, \ \text{gfz}, \ \text{var}, \ \text{eq}; \]

\[ \text{wauto} := \text{regexpcount}[\text{gramweight}](\text{auto}) \]

\[ \text{gfz} := \text{comstruct}[\text{gfz}] \text{wauto}, \text{unlabelled}, z, \text{weight}; \]

\[ \text{gfzu} := \text{subs}(\text{solve}([\text{op}(\text{gfz}1), \text{seq}(\text{op}(1,i), i=\text{gfz}2)]), \text{init}(z, u); \]

\[ \text{printf} \left( '\text{gfz}=\%a\text{\^{}n}', \text{gfzu} \right) ; \]

\[ \text{gfz}[1] := \text{subs}(u=1, \text{diff}(\text{gfzu}, u)) \quad \text{# g.f for expect} \]

\[ \text{gfz}[2] := \text{subs}(u=1, \text{diff}(u \text{diff}(\text{gfzu}, u), u)) \quad \text{# g.f. for second moment} \]

\[ \text{for} \ i \ \text{to} \ 2 \ \text{do} \ \text{eq}[i] := \text{equivalent} \left( \text{gfz}[i], z, n, 6 \right); \ \od \]

\[ \text{print}(\text{"expectation",eq[1]}); \]

\[ \text{var} := \text{eq}[2] - \text{eq}[1]^2; \]

\[ \text{print}(\text{"variance"}, \text{gdev}(\text{var}, n=\text{infinity}, 10)); \]

\[ \text{end}; \]
\( t2 := \text{time(): getvals(marked_automat,ABAB,[op(Bwg),[u,m]]); time()-t2; } \)

\[
gfz = \frac{-1024 - 120z^4u^3 + 144z^3u^2 - 48z^3u^2 - 192z^2u^2 + 276z^4u^2 + 192z^2u^2 + 9z^5u^3 - 18z^5u^2 - 192z^2u^2 + 384z^3u + 36z + 9z^5u}{8(128z - 128 + 9z^5u^3 - 6z^3u^2 - 6z^3u^2 + 24z^2u^2 - 9z^4u^2 - 24z^2u^2)}
\]

"expectation", \[
\frac{63}{128} n - \frac{141}{128} + O(n^{(-\infty)})
\]

"variance", \[
\frac{6957}{16384} n - \frac{12663}{16384} + 0.519
\]

\( t3 := \text{time(): markov_marked_automat:= regexpcount[grammarkov](marked_automat,ABAB,1,rho,[m]); time()-t3; } \)

\[
\text{markov_marked_automat := } \{ \\
  w7_a = \text{Union(Prod}(\rho_a, a, w1_a), E, \text{Prod}(\rho_{a,b}, w3_b)), \\
  ABAB = \text{Union}(E, \text{Prod}(\rho_a, w7_a), \text{Prod}(\rho_b, w5_b)), \\
  w1_a = \text{Union(Prod}(\rho_a, a, w1_a), \text{Prod}(\rho_{a,b}, m, w2_b), E), \\
  m = E, \\
  w9_a = \text{Union(Prod}(\rho_a, a, w1_a), \text{Prod}(\rho_{a,b}, m, w2_b), E), \\
  w4_b = \text{Union}(E, \text{Prod}(\rho_b, m, w6_b), \text{Prod}(\rho_{b,a}, w1_a)), \\
  w6_b = \text{Union}(E, \text{Prod}(\rho_{b,a}, m, w6_b), \text{Prod}(\rho_{b,a}, w1_a)), \\
  w2_b = \text{Union}(E, \text{Prod}(\rho_{b,a}, m, w8_a), \text{Prod}(\rho_{b,b}, m, w4_b)), \\
  w3_b = \text{Union}(E, \text{Prod}(\rho_{b,b}, m, w8_a), \text{Prod}(\rho_{b,b}, m, w4_b)), \\
  w8_a = \text{Union(Prod}(\rho_{a,b}, m, w2_b), E, \text{Prod}(\rho_{a,a}, m, w9_a)), \\
  w5_b = \text{Union}(E, \text{Prod}(\rho_{b,b}, w5_b), \text{Prod}(\rho_{b,a}, w1_a)), \\
  \rho_a = \text{Atom}, \rho_b = \text{Atom, } \rho_{a,b} = \text{Atom, } \rho_{b,a} = \text{Atom, } \rho_{b,b} = \text{Atom, } \rho_{a,a} = \text{Atom} \\
\}
\]

\( t4 := \text{time(): getvals(markov_marked_automat,ABAB,[op(Mwg),[u,m]]); time()-t4; } \)

\[
gfz = \frac{-32z - 48z^2 + 12z^3u^3 + 18z^4u^2 - 128 + 36z^3u^2 + 128z^4u^3 - 80z^2u^3 - 30z^4u^2 - 66z^3u + 18z^2u^2 + 15z^4u - 3z^4}{2(48z - 8z^2 + 6z^3u^3 + 3z^4u^3 - 6z^3u^2 - 64 - 6z^3u^2 + 24z^2u^2 - 3z^4u^2)}
\]

"expectation", \[
\frac{57}{80} n - \frac{1311}{800} - \frac{27}{25} (-4)^{(-n)} + O\left(\frac{4^{(-n)}}{n^8}\right)
\]

"variance", \[
\frac{7323}{32000} n - \frac{17601}{128000} + O\left(n \cdot 4^{(-n)}\right)
\]

.785