Ann Inst Stat Math (2017) 69:231–248
DOI 10.1007/s10463-015-0540-y

Faster exact distributions of pattern statistics through sequential elimination of states

Donald E. K. Martin[1] · **Laurent Noé**[2]

Received: 18 April 2014 / Revised: 1 July 2015 / Published online: 18 September 2015

Abstract When using an auxiliary Markov chain (AMC) to compute sampling distributions, the computational complexity is directly related to the number of Markov chain states. For certain complex pattern statistics, minimal deterministic finite automata (DFA) have been used to facilitate efficient computation by reducing the number of AMC states. For example, when statistics of overlapping pattern occurrences are counted differently than non-overlapping occurrences, a DFA consisting of prefixes of patterns extended to overlapping occurrences has been generated and then minimized to form an AMC. However, there are situations where forming such a DFA is computationally expensive, e.g., with computing the distribution of spaced seed coverage. In dealing with this problem, we develop a method to obtain a small set of states during the state generation process without forming a DFA, and show that a huge reduction in the size of the AMC can be attained.

Keywords Active proper suffix · Auxiliary Markov chain · Computational efficiency · Extended seed patterns · Minimal deterministic finite automaton · Overlapping pattern occurrences · Seeded alignments · Spaced seed coverage

✉ Donald E. K. Martin
donald_martin@ncsu.edu

Laurent Noé
laurent.noe@univ-lille1.fr

[1] Department of Statistics, North Carolina State University, 5116 SAS Hall, Raleigh, NC 27695, USA

[2] CRIStAL (UMR 9189 Lille University/CNRS), INRIA Lille Nord-Europe, Villeneuve d'Ascq, France

1 Introduction

A statistic that has an exceptionally large or small value can signal the presence of important phenomena. As examples, in DNA the chi motif has an exceptionally high frequency due to its association with cellular recombination, and palindromes, which are associated with restriction enzyme sites, are rare (Robin et al. 2005, pp. 6–9). To determine the statistical significance of a statistic's value requires distributional properties. Due to the numerous applications of pattern statistics, techniques for computing their distribution have received much attention in mathematical statistics, computer science, and application areas. We refer the reader to three books as well as a review article that give good surveys of much of the work in this area (Balakrishnan and Koutras 2002; Fu and Lou 2003; Robin et al. 2005; Lladser et al. 2008).

One method that is frequently used to compute distributions associated with patterns and more general statistics is to associate the statistic's distribution with an auxiliary Markov chain (AMC). Once this is done, the Chapman–Kolmogorov equations (see, e.g., Parzen 1962, pp. 194–195) give a way to obtain probabilities of the auxiliary chain lying in its various states at any particular time, leading to methods to compute the desired distribution. Fu and Koutras (1994) forwarded this approach with their finite Markov chain imbedding (FMCI) method, and AMCs have been used in many papers since then (e.g., Koutras and Alexandrou 1995; Fu 1996; Ebneshahrashoob et al. 2005; Aston and Martin 2007; Martin 2008; Martin and Aston 2008). Though extremely useful, forming an AMC for computation has the drawback that for certain complex patterns, the state space of the AMC can be prohibitively large.

In recent years, several authors (e.g., Nuel 2008; Kucherov et al. 2007; Lladser et al. 2008; Marshall and Rahmann 2008; Ribeca and Raineri 2008; Martin and Coleman 2011; Martin and Aston 2013) have used theory from computer science on minimizing a deterministic finite automaton (DFA) to turn an AMC into a smaller version, thereby obtaining an "optimal Markov chain embedding" in the sense that the AMC has the smallest number of states (for a given order of Markovian dependence) that is sufficient for obtaining the desired probabilistic results. However, for certain complex patterns, forming a DFA before applying a minimization algorithm is computationally prohibitive, and can be infeasible for current computer memory. In this paper, we bypass this problem by using clues from DFA minimization algorithms to delete states in the process of forming the original AMC. We show that although the AMC resulting from our implementation is not necessarily minimal, it can lead to a vastly reduced number of states and thus more efficient computation.

The organization of the paper is as follows. The next section discusses background information on computing a distribution through an AMC, and how minimizing an associated DFA can be helpful in this regard. Section 3 then presents the problem motivating this work, which is computing the distribution of the number of sequence positions covered by spaced seed hits. After observing the intricacy of the problem, which can render the forming of an associated DFA before applying a minimization algorithm as computationally prohibitive, we show how to form a small set of states in a sequential fashion that bypasses the need to first obtain the DFA. The process of obtaining the states is integrated with obtaining state transitions and transition probabilities, thus streamlining the process. The algorithm that is developed is applied to relatively small spaced seeds to help make the presentation clearer, and then to a long seed that has been used in the Patternhunter algorithm for similarity searches over biosequences (Ma et al. 2002). The reduction in the size of the state space and computation time is highlighted. The final section is a summary.

2 Auxiliary Markov chains and minimal DFA

A *pattern* u (also called a *word* or *string*) is a sequence of symbols from a *state space*, or *alphabet* Σ. The length of a pattern u is denoted by $|u|$ ($|B|$ is also used to denote the number of elements in a set B; the meaning should be clear from the context). The concatenation of pattern v to the right of pattern u is denoted by $u \cdot v$. For $u = u_1, \ldots, u_{|u|}$, and $d \leq |u|$, $(u)_d \equiv u_{|u|-d+1}, \ldots, u_{|u|}$ and ${}_d(u) \equiv u_1, \ldots, u_d$ are, respectively, the *suffix* and *prefix* of length d of pattern u. If $d < |u|$, $(u)_d$ and ${}_d(u)$ are, respectively, a *proper suffix* and *proper prefix*.

Consider now an mth-order Markovian sequence $\mathbf{X} = X_1, \ldots, X_n$ with realization $\boldsymbol{x} = x_1, \ldots, x_n$, where each x_i lies in Σ. Denote $x_a, x_{a+1}, \ldots, x_b$ by $x_a : x_b$ for $1 \leq a < b \leq n$ and let $\tilde{x}_j \equiv x_{j-m+1} : x_j$ for $j = m, m+1, \ldots, n$. Let $\tilde{\pi}$ be the initial probability distribution over m-tuples $\tilde{x}_m$ and $p(x_{j+1}|\tilde{x}_j)$ the transition probabilities of the sequence that are stored in a matrix T. When $\mathbf{X}$ is stationary, we can obtain $\tilde{\pi}$ through the equation $\tilde{\pi} T = \tilde{\pi}$, with the entries of $\tilde{\pi}$ summing to 1. Otherwise, $\tilde{\pi}$ must be specified.

2.1 Computing distributions through auxiliary Markov chains

Fu and Koutras (1994) called a discrete random variable Z "finite Markov chain imbeddable" if there exists a finite Markov chain A_t, $t = 0, 1, \ldots, n$ with state space Q, initial row probability vector π_0, and transition probability matrices Ω_t such that for z in the range Υ_Z of Z, $P(Z = z) = P(A_n \in Q_z|\pi_0)$, where $Q_z, z \in \Upsilon_Z$ is a partition of the states of Q. Based on the Chapman–Kolmogorov equations,

$$P(Z = z) = \pi_0 \left(\prod_{t=1}^{n} \Omega_t \right) V'(Q_z), \tag{1}$$

where $V(Q_z)$ is a row vector with 1's in positions corresponding to the states of Q_z, and 0's elsewhere.

Depending on the problem at hand, there are variations in how to set up the AMC $\{A_t\}$ and compute $P(Z = z)$. Whereas Fu (1996) defined the Markov chain $\{A_t\}$ operating on ω as $A_t(\omega) = (u, v)$, where u is the number of pattern occurrences in the first t trials and v is the pattern prefix (called an "ending block" in that paper), Koutras and Alexandrou (1995) cleverly formulated the transition probability matrix of the AMC only for pattern prefixes, which are repeated for each possible pattern occurrence. This approach reduces the order of the transition probability matrix from $l|\Lambda|$ to $|\Lambda|$, where $|\Lambda|$ is the number of pattern prefixes used, and l is the maximum value of the statistic that is possible in $\mathbf{X}$.

As for the computation of Eq. (1), when $\Omega_t = \Omega$, a fixed matrix, one could compute $\pi_n = \pi_0 \Omega^n$ sequentially through $\pi_t = \pi_{t-1}\Omega$, $t = 1, \ldots, n$, retaining the sparcity of the multiplier Ω. In that case, each of the n vector-matrix multiplications requires one to deal with the $|\Sigma| \times |\Lambda|$ non-zero elements of Ω, where $|\Sigma|$ is the size of the state space. Alternatively, Ω^n can be obtained by matrix doubling (see, e.g., Martin and Coleman 2011), thus requiring $O(\log_2 n)$ multiplications as opposed to $O(n)$ (but losing the sparcity of Ω; each matrix–matrix multiplication then using $O|\Lambda|^3$ multiplications/additions), or using the fast Fourier transform (which must be carefully implemented due to numerical instability; see Ribeca and Raineri 2008). The choice between sequentially multiplying by Ω or multiplying by Ω^n can then be made based on comparing $n|\Sigma||\Lambda|$ and $(\log_2 n)|\Lambda|^3$.

In either case, it is obvious that for computational efficiency, keeping the number of states $|\Lambda|$ small is of utmost importance. DFA and minimal DFA, which can help with this task, will be discussed next.

2.2 AMC state minimization through DFA

A DFA D is a 5-tuple $D = (Q, \Sigma, \delta, q_0, F)$, where Q is a set of states, Σ is the alphabet, δ the transition function for states $q \in Q$, i.e., $\delta : Q \times \Sigma \to Q$, $q_0 \in Q$ is a start state, and $F \subset Q$ is a set of “final” or “accepting” states. For an input sequence $(x_1, x_2, \ldots, x_n)$ $(x_i \in \Sigma)$, the DFA begins in state q_0 and transitions according to δ as symbols x_i are fed in. If $q_j \in F$ for some $j \in (1, \ldots, n)$, then the string $x_1 : x_j$ is “accepted,” otherwise it is “rejected.” The *language* of a DFA is the set of strings that it accepts. When a state of F is entered, an action is taken. For example, for a statistic of a collection of patterns, F consists of the patterns, and the statistic is incremented when a state of F is entered.

States q and $q*$ are *equivalent* if beginning in each and entering an arbitrary string (including the empty string), the result is either a final state in both cases or a non-final state in both cases. A well-known result from computer science that is useful in our framework is that beginning with any DFA, one can find an equivalent DFA that recognizes the same language and has a minimal number of states (see, e.g., Hopcroft 1971; Hopcroft et al. 2001).

Minimizing a DFA is a special case of the multi-function “coarsest partition” problem (Tewari et al. 2002) where, given an initial partition $H_1, \ldots, H_r$ of a set Q and functions $f_1, \ldots f_\gamma$ over the states, one finds the partition $G_1, \ldots, G_s$, $s \geq r$ with the smallest number of equivalence classes s such that (i) each G_i is a subset of some class H_j, and (ii) q and $q*$ in G_h imply that $f_\alpha(q)$ and $f_\beta(q*)$ are both in some class G_k for all α, β. For minimizing a DFA, $r = 2$, the initial partition is $(H_1, H_2) = (F, Q/F)$, and the resulting partition is the equivalence classes of the DFA, which are the states of the minimal DFA. The minimal DFA has the smallest number of states for any DFA that recognizes the same language, and is also unique, up to a renaming of the states (see, e.g., Hopcroft 1971; Hopcroft et al. 2001, pp. 154–162).

Whereas in this work, we consider AMC state reduction in the context of pattern statistics, minimal DFA can be helpful for more general statistics as well. In the more general case, given any AMC, a DFA minimization algorithm can be applied to its states/transition function to obtain a minimal version. If no states are eliminated the AMC states are already in minimal form. This is the case, for example, with the state space for computing p values for scan statistics (Martin 2014).

Also note that a DFA with states corresponding to pattern prefixes (an Aho–Corasick automaton; see Aho and Corasick 1975) can be modified to obtain an AMC suitable to compute distributions of pattern statistics over mth-order Markovian $\mathbf{X}$. To form the state space, add to the pattern prefixes all m-tuples that are not pattern prefixes (m-tuples are required due to the Markovian assumption), and then delete all strings of length less than m, as the computation can be initialized at time m.

On symbol x_i, pattern prefix q transitions to the longest suffix of $q \cdot x_i$ that is in Q (Aho and Corasick 1975). To compute probabilities, define an initial probability distribution over the m-tuples Σ^m, along with transition probabilities $\Pr(q_{j-1} \to q_j) = \Pr(\tilde{x}_{j-1} \to \tilde{x}_j)$, where $\tilde{x}_r = (q_r)_m$, $r = j-1, j$. The states of the resulting Markov chain can be minimized using an analog of the Hopcroft (1971) algorithm (with the additional restriction that all states in an equivalence class must have the same m-tuple as their suffix), to give an AMC with a minimal number of states for that particular model order. The next two paragraphs give examples of AMC formation through DFA minimization.

First, consider computing the distribution of the number of overlapping occurrences of the chi motif of *H. influenza* $F = W_8 = \{GATGGTGG, GCTGGTGG, GGTGGTGG, GTTGGTGG\}$ (Ledent and Robin 2005) in a first-order Markovian sequence $\mathbf{X}$, with $\Sigma = \{A, C, G, T\}$. In the Aho–Corasick automaton, Q consists of prefixes of the patterns of F (see Fig. 1a). The modified DFA with 1-tuples added and the empty string ε deleted is shown in Fig. 1b. Final states F and Q/F form the initial state partition. States $W_7 = \{GATGGTG, GCTGGTG, GGTGGTG, GTTGGTG\}$ can enter F on symbol G whereas other states of Q/F cannot, and thus W_7 is split off as a separate class in the minimization process (these states transition the same on other symbols as well, and thus they remain a single class throughout the minimization process.) Then, states W_6 that enter W_7 on symbol G can be split from those that do not, forming another equivalence class. In the end, states of length longer than two are equivalent if they differ only in their second symbol, and thus four states can be combined in each of the equivalence classes $W_3, \ldots, W_8$ (for length two, states $W_{2a} = GG$ and $W_{2b} = \{GA, GC, GT\}$ are distinguishable because they transition differently on symbols A, C, or G). The final partition for the chi motif is shown in Figure 1c, with numbered states j representing W_j. Fig. 1d shows the result of deleting the final state from Fig. 1c, while re-mapping the entering transition. Notice that the number of AMC states has been reduced from 30 to 9.

As a second example, consider computing the distribution of *clump count and coverage*. These statistics are incremented differently for overlapping and non-overlapping occurrences of a collection A_W of patterns, so that they must be distinguished. Bassino et al. (2008) and Martin and Coleman (2011) used a DFA with states that are prefixes of the set A_{ext}, where A_{ext} itself is defined by:

$$A_{ext} = \{u | \exists q, w \in A_W \text{ with } u = \alpha_q \cdot v_{qw} \cdot \beta_w, \alpha_q \cdot v_{qw} = q, v_{qw} \cdot \beta_w = w, \text{and } \alpha_q, v_{qw}, \beta_w \text{ are nonempty}\}. \tag{2}$$

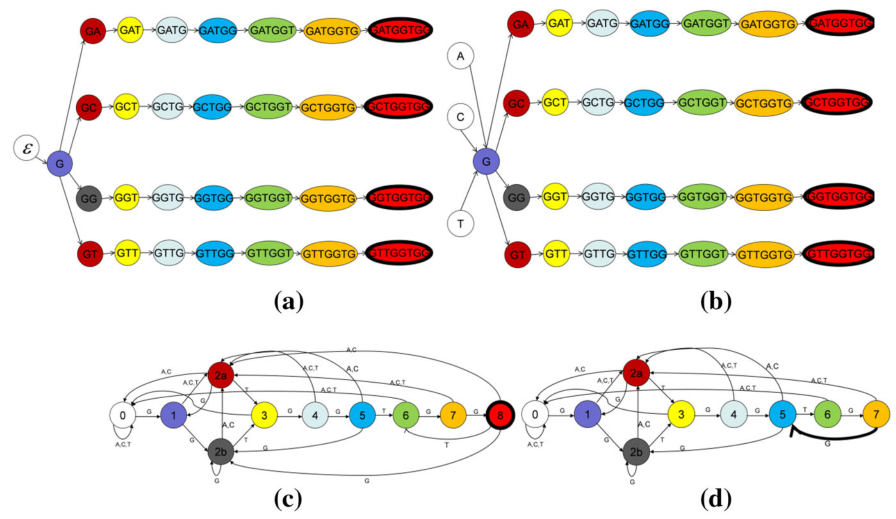

Fig. 1 **a** Aho–Corasick DFA for the Chi motif $\{GATGGTGG, GCTGGTGG, GGTGGTGG, GTTGGTGG\}$ (note: in plots (**a**) and (**b**), some transitions are not shown for clarity); **b** modified version for Markov chain ($m = 1$), with m-tuples replacing ε; **c** minimized version; **d** minimal version with state representing chi motif eliminated and its transition (marked with a *bold line*) re-mapped. In plots **a**, **b**, and **c**, final states are marked with a *bold outline*

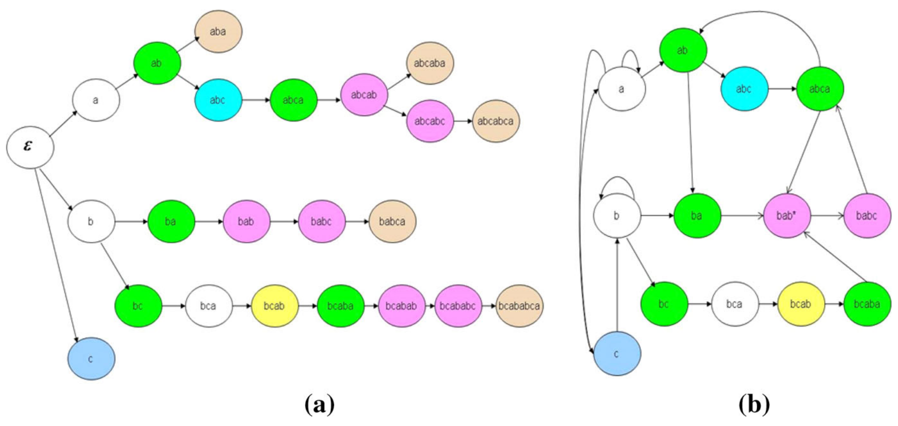

Fig. 2 Transition systems for computing statistics of clumps of pattern $W = \{ab, bc, ba, abca, bcaba\}$: **a** Prefixes of $A_{ext} \cup \Sigma^m$; **b** AMC states after minimization. Color code: Q_{pre} (*white*), $Q_{\text{pre,new}}$ (*yellow*), $Q_{\text{pre,ext}}$ (*turquoise*), Q_W (*green*), $Q_{\text{path,ext}}$ (*pink*), Q_{ext} (*light brown*), Q_m (*light blue*) (see Martin and Coleman 2011 for definitions of the classes of states). For the sake of clarity, not all transitions are shown, and edge labels are omitted. The state $bab*$ in **b** represents $\{bab/abcab/bcabab\}$

In the latter reference, the minimization of the AMC used an initial partition into multiple classes that were based on roles relative to updating the clump statistics, and the Hopcroft algorithm was then applied. (See the initial and ending AMC states for clumps of the collection of patterns $W = \{ab, bc, ba, abca, bcaba\}$ in Fig. 2a, b, where the various classes of the initial partition are color coded).

Whereas using prefixes of an extended set A_{ext} can be useful for computing pattern statistics that are incremented differently for overlapping and non-overlapping pattern occurrences, the current authors encountered a situation (computing the distribution of spaced seed coverage) where forming an AMC/DFA before applying a minimization algorithm is computationally expensive. After describing the spaced seed problem, we give an approach to form an efficient AMC state space without first setting up a full DFA, by which we mean, an automaton consisting of prefixes of patterns of the extended set A_{ext}.

3 Distribution of spaced seed coverage

Heuristic methods are typically used to locate similar segments in DNA sequences. The standard heuristic method is to initially search for relatively short matching (or nearly matching) segments, and then look for alignments around a match with similarity scores that are significantly high. *Seeds* give the shape of the matching segments. Short seeds can occur many times even in sequences with non-similar structure, but searching for long exact matches can result in missing segments whose underlying structure is similar. Thus, a trade-off is beneficial.

In recent years, *spaced seeds* (Ma et al. 2002; Keich et al. 2004; Buhler et al. 2005) have been used to provide a way to increase the probability of observing at least one seed hit (matching segment) without simultaneously increasing the number of hits that occur at random. A spaced seed is a pattern $S = s_1, \ldots, s_k$ from $\{1, *\}$, with the restriction that $s_1 = s_k = 1$. A "1" indicates a position where sequence symbols must match, and "*" indicates a position where a match is not required. When the number of stars r equals zero, the seed is termed a *contiguous* seed (or a success run).

Let $\mathbf{X} = x_1, \ldots, x_n$ be the binary sequence formed by aligning two DNA segments of length n and assigning a value $x_j = 1$ if the jth position of the segments match, and $x_j = 0$ for a mismatch. Spaced seed S *hits* or *occurs* in $\mathbf{X}$ at position υ if for $j = 1, \ldots, k$, $x_{\upsilon-k+j} = 1$ whenever $s_j = 1$. A "1" at position $\upsilon - k + j$ of $\mathbf{X}$ is *covered* by a seed hit if there exists $\upsilon \in (k, \ldots, n)$ and $j \in (1, \ldots, k)$ such that S occurs at position υ and for that occurrence, $s_j = 1$. For example, for spaced seed $S = 11 * 1$ and sequence segment

$$X = \underset{\bullet}{1}\,\underset{\bullet}{1}\,1\,\underset{\bullet}{1}\,0\,0\,1\,\underset{\bullet}{1}\,\underset{\bullet}{1}\,0\,\underset{\bullet}{1}\,\underset{\bullet}{1}\,\underset{\bullet}{1}\,\underset{\bullet}{1}\,\underset{\bullet}{1}$$

of length $n = 15$, there are seed hits at sequence positions 4, 11, 14 and 15, and the ten 1's with "$\bullet$" underneath are covered.

One solution to the trade-off between the use of short and long seeds is to use short seeds, but to require multiple seed hits clustered close together to trigger an alignment. This approach was used in Benson's algorithm for detecting tandem repeats (Benson 1999). The criterion used in that algorithm was based on the number of successes in success runs of length at least k (coverage of a contiguous seed), with the null hypothesis of similarity (with the associated clumping of matches) only rejected for small values of the statistic. The critical value was determined using a normal approximation to the statistic's distribution. Lou (2003), Fu et al. (2002) and Martin

(2006) gave the exact distribution of the statistic for independent trials, first-order Markovian sequences and Markovian sequences of a general order, respectively.

Coverage of spaced seeds can be used as the "matching" criterion under the multiple seed hit paradigm, and also relates to the "group criterion" mentioned in Noé and Kucherov (2004). However, distributional results for spaced seed coverage are rare. The only known result is that of Benson and Mak (2009), who gave a method for computing the distribution in the i.i.d. case, and illustrated their computational method for relatively short seeds. We extend their work to higher-order Markovian sequences, with the efficiency of the method allowing computation for the longer seeds that are used in practice.

3.1 Computation procedure

For spaced seed $S = s_1, \ldots, s_k$ (k is assumed to be greater than the order of Markovian dependence m) with r stars, let A_S be the collection of the 2^r seed words that may be formed by replacing stars of the seed by either 0 or 1. Since a covered "1" can only be counted once, the AMC used to compute the coverage distribution needs to keep track of which positions have been covered previously. One option for the states is to use prefixes of strings of A_{ext} as in Martin and Coleman (2011) (see Eq. (2)), add in m-tuples that are not pattern prefixes, and then delete any strings of length less than m. Such a representation allows the determination of covered positions from the prefix string. However, the resulting state space (which we call Λ_{ext}) may contain many unnecessary strings, and is thus not efficient. An example for the small spaced seed $11 * 1$ is given in Fig. 3a (where covered positions are indicated by a coverage string c for convenience). For that example, a minimization algorithm would discard 7 of the 19 states, although that is not obvious from the representation. Minimizing Λ_{ext}, as in Martin and Coleman (2011), is thus a remedy when it is feasible, but for long seeds with several stars, setting up Λ_{ext} before applying a minimization algorithm is computationally expensive. The question is then "Can one set up a state space with a small number of states without forming a full automaton before state reduction?"

We begin with an alternate representation of the set of AMC states that will be used (this set of states will be denoted by Λ). States $\lambda \in \Lambda$ are represented as $\lambda = \begin{pmatrix} q \\ c \end{pmatrix}$,

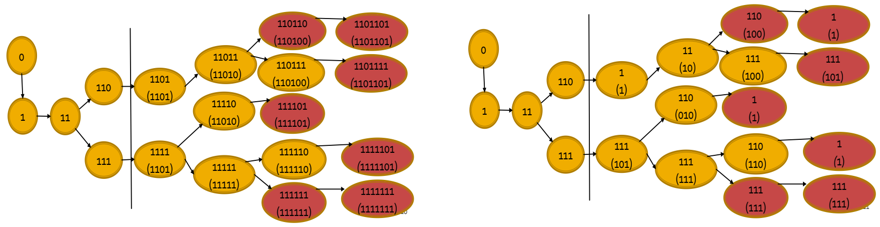

Fig. 3 **a** States of the AMC for computing the coverage distribution for spaced seed $11 * 1$ in a first-order Markovian sequence; **b** alternate representation with no q strings of length greater than $k - 1 = 3$. States colored red are those that would be eliminated by a minimization algorithm

with $q \in Q = \Sigma^m \cup \tilde{Q}$, where Σ^m are m-tuples and $\tilde{Q}$ are prefixes of words of A_S. Here, Σ^m may be partitioned as $\Sigma^m = \hat{Q}_{\text{nosuf}} \cup \hat{Q}_{\text{suf}} \cup \tilde{Q}_m$, where $\tilde{Q}_m$ are prefixes of words of A_S of length m, $\hat{Q}_{\text{suf}}$ contains the m-tuples that are not prefixes of words of A_S, but have a proper suffix that is, and $\hat{Q}_{\text{nosuf}} = \Sigma^m / (\hat{Q}_{\text{suf}} \cup \tilde{Q}_m)$. Also, $\tilde{Q} = \tilde{Q}_m \cup \tilde{Q}_{>m}$, where $\tilde{Q}_{>m}$ are strings that are prefixes of words of A_S of lengths $m + 1 \le |q| \le k - 1$). Thus, $Q = \Sigma^m \cup \tilde{Q}_{>m} = \hat{Q}_{\text{nosuf}} \cup \hat{Q}_{\text{suf}} \cup \tilde{Q}$.

The fact that no string $q \in Q$ has length greater than $k - 1$ reduces the storage requirement compared with strings of Λ_{ext}, while serving the same purpose. Also, the strings $q \in Q$ transition the same regardless of their associated coverage string, reducing the number of transitions and transition probabilities that need to be specified.

Strings $c \in C$ indicate previously covered positions (C is the set of possible coverage strings). A coverage string c has the same length as the corresponding string q. This representation is depicted in Fig. 3b for seed $11 * 1$. For that example, the representation makes it clear that six of the seven states that would be eliminated by a minimization algorithm are redundant. In what follows, using the latter representation along with properties of states that render them as equivalent, we effect sequential state elimination. Before proceeding, we give some definitions and also useful properties of a concept we call *active proper suffixes*.

Definition 1 On symbol x, state $q \in Q$ transitions to the longest suffix of $q \cdot x$ that is in Q.

Definition 2 A *direct hit* associated with $q \in \tilde{Q}$ is the occurrence of a word $\alpha \in A_S$ with $\alpha = q \cdot u$. A *future hit* associated with $q \in \tilde{Q}$ is the occurrence of a word $\beta \in A_S$, where $\beta = (q)_d \cdot v$ ($d < |q|$ so that $(q)_d$ is a proper suffix of q).

Informally, a direct hit must start from the beginning of a string $q \in \tilde{Q}$ (pattern prefixes), whereas future hits begin at the beginning of a proper suffix of string q (an overlapping pattern occurrence). A future hit requires that the direct hit has occurred previously. Note that a prefix of length less than m of a word of A_s is represented in Q by an m-tuple of $\hat{Q}_{\text{suf}}$.

Definition 3 The longest proper suffix of $q \in Q_{>m}$ that is in $\tilde{Q}$ is called its *active proper suffix*, and is denoted by $\text{aps}(q)$. If no such suffix exists, then $\text{aps}(q) = (q)_m$.

Note that $\text{aps}(q)$ is only defined for strings of length greater than m. An $\text{aps}(q) \in \tilde{Q}$ gives the maximal progress towards an overlapping (future) seed hit. If $\text{aps}(q) \in \hat{Q}_{\text{suf}}$, the progress toward an overlapping seed hit is of length less than m, and if $\text{aps}(q) \in \hat{Q}_{\text{nosuf}}$, there is no progress towards an overlapping seed hit.

To illustrate these concepts, if $S = 11 * 1$ (where $A_S = \{1101, 1111\}$) with $m = 2$, $\Sigma^m = \{00, 01, 10, 11\}$, $\hat{Q}_{\text{nosuf}} = \{00, 10\}$, $\hat{Q}_{\text{suf}} = \{01\}$, $\tilde{Q}_m = \{11\}$, and $\tilde{Q}_{>m} = \{110, 111\}$, so that $Q = \{00, 01, 10, 11, 110, 111\}$. If $x_1 : x_5 = 11111$ and $q = x_1 : x_3 = 111$, then $x_1 : x_4$ is a direct hit associated with q, and $x_2 : x_5$ is a future hit that begins with $aps(q) = 11 \in \tilde{Q}_m$. If $x_1 : x_3 = q = 110$, $aps(q) = 10 \in \hat{Q}_{\text{nosuf}}$, meaning that although q is a prefix of 1101, there is no current progress towards a future seed hit.

Lemma 1 *If $q \in \tilde{Q}_{>m}$, no symbol of q preceding its proper suffix* $\text{aps}(q)$ *can be in a future seed hit.*

Proof Assume that a future seed hit relative to q begins to the left of $\text{aps}(q)$. This contradicts the definition of $\text{aps}(q)$ itself, and thus the result follows. □

For m-tuple $q \in \Sigma^m$, no symbol preceding its longest suffix that is a prefix of a word of A_S (which can be empty if $q \in \hat{Q}_{\text{nosuf}}$) can be in a future hit.

Lemma 2 *If $q, q* \in \tilde{Q}_{>m}$, states $\begin{pmatrix} q \\ c_1 \end{pmatrix}$ and $\begin{pmatrix} q* \\ c_2 \end{pmatrix}$ may be combined as a single state if $|q| = |q*|$, $c_1 = c_2$, and $\text{aps}(q) = \text{aps}(q*)$ $\Big($or if $\begin{pmatrix} \text{aps}(q) \\ 0 \end{pmatrix}$ and $\begin{pmatrix} \text{aps}(q*) \\ 0 \end{pmatrix}$ have been previously combined$\Big)$.*

Proof For strings $q, q* \in \tilde{Q}_{>m}$ of the same length and with the same coverage history, on a direct seed hit the coverage increments and the updated coverage strings will be the same. If, in addition, the strings q_1 and q_2 have the same active proper suffix (or if the active proper suffixes were previously combined), the resulting prefix strings on the direct hit will be equivalent. Strings with the same (or combined) active proper suffix behave the same on any future hits since the active proper suffix contains the only part of the string that can possibly be involved in a future hit. Such strings must have the same m-tuples as their suffix (since $|\text{aps}(q)| \geq m$ for all q), and therefore have the same transition probabilities on any symbol. The result follows. □

If $\begin{pmatrix} q \\ \mathbf{0} \end{pmatrix}$ and $\begin{pmatrix} q* \\ \mathbf{0} \end{pmatrix}$ are combined states, (or if $q = q*$), we write $q \sim q*$ to denote their equivalence.

Based on these results we can use the following basic sequential method for setting up Λ. First form the m-tuples. Then, sequentially over lengths $m + 1, \ldots, k - 1$, generate the proper prefixes of seed words of A_S, and note their active proper suffixes. Strings q and $q*$ satisfying the conditions of Lemma 2 are combined by discarding the one not already in Q. The strings of Q are associated with coverage strings $c = \mathbf{0}$ of length q to represent prefixes of non-overlapping occurrences of strings of A_S. The transitions $q \rightarrow q'$ with $|q| < k - 1$ (or if $|q| = k - 1$ and the symbol is "0") are defined according to Definition 1.

We then generate states $\lambda = \begin{pmatrix} q \\ c \end{pmatrix}$ with $q \in Q$ and non-zero coverage vectors c, as in Fig. 3b. If $q \in \tilde{Q}$ with $|q| = k - 1$ and the input symbol is "1", $q \rightarrow q' = aps(q \cdot 1)$, and the new coverage string is $c' = (c_{tem})_{|aps(q \cdot 1)|}$, where the coverage template $c_{tem} \equiv (c_{tem,1}, \ldots, c_{tem,k})$ has entries $c_{tem,j} = 1$ if $s_j = 1$, $c_{tem,j} = 0$ otherwise, $j = 1, \ldots, k$. The number of new states added in this stage, η, is recorded, and the total number of states is incremented. Also, we record the update to coverage $k - r$ for the transition $q \rightarrow q'$ in the appropriate entry of a $|\Lambda| \times |\Lambda|$ update matrix denoted by U.

Now for each of the η new states just defined and input symbol $x_i \in \{0, 1\}$ we obtain the transitions of their prefix string q and coverage string c, as well as the coverage increment on a seed hit (i.e., if $q \in \tilde{Q}$ with $|q| = k - 1$ and $x_i = 1$). The prefix q transitions exactly as when there is no coverage. To update c, if there is no seed hit, then $c \to c' = (c \cdot 0)_{|q'|}$ (concatenate a zero to the end of c and then take the suffix of the same length as the new prefix string q'). Otherwise, $c \to c'$, where c' is the suffix of length $|q'|$ of the string of length k that has a "1" in any position where at least one of $c \cdot 0$ or c_{tem} has a "1", and zeroes elsewhere. The increase of coverage for the transition $q \to q'$ is obtained by computing the number of newly covered positions with the seed hit. This coverage increment is stored in $U(i, j)$, where (i, j) are the indices for the transition $\lambda_i = \begin{pmatrix} q \\ c \end{pmatrix} \to \lambda_j = \begin{pmatrix} q' \\ c' \end{pmatrix}$.

If, during any stage of the state generation process, a destination state $\begin{pmatrix} q' \\ c' \end{pmatrix}$ already exists in Λ, the transition is mapped there. Otherwise a new state is generated to receive the transition. At the beginning of each stage, η is set to 0 and incremented by one as each new state is generated. The procedure for generating states on state transitions is repeated for the new strings at each stage while $\eta > 0$. In this manner, we end up with a sufficient number of states to carry out the computation, while keeping the number of states small. Transition probabilities for states of Λ are stored in the transition probability matrix Ω.

This basic algorithm is augmented with further (possible) combining/elimination of states that will be described below. To illustrate the basic setup of Λ given above as well as the additional reductions that are possible, consider first the seed 1*11*1 for which $A_S = \{101101, 101111, 111101, 111111\}$, and assume that the Markov order is $m = 1$. First generate the m-tuples 0 and 1 along with prefixes of seed words of lengths 2 to 4 (all these strings are needed in Λ because for each of those lengths, the active proper suffixes are unique). For length 5, according to Lemma 2, strings $\{10110, 11110\}$ may be combined as a single state (they both have 10 as their active proper suffix). Then, the set $Q = \{0, 1, 10, 11, 101, 111, 1011, 1111, 10110, 10111, 11111\}$, with 10110 also representing 11110, which is discarded. Obtaining transitions of 10110, 10111, and 11111 on symbol 1, we generate the respective new states of the form $\begin{pmatrix} q \\ c \end{pmatrix}$: $\begin{pmatrix} 101 \\ 101 \end{pmatrix}$, $\begin{pmatrix} 1111 \\ 1101 \end{pmatrix}$, and $\begin{pmatrix} 11111 \\ 01101 \end{pmatrix}$.

We then obtain transitions of these states and generate new states as needed, repeating the process for the new states at each stage. The resulting state space, of size $|\Lambda| = 27$, is depicted in Fig. 4. For this seed, $|\Lambda_{ext}| = 84$, and thus the reduction is substantial.

In Fig. 4, notice the transition of $\begin{pmatrix} 1111 \\ 1101 \end{pmatrix}$ on symbol "0". Concatenating "0" to the end of 1111 gives 11110, however, by Lemma 2 this prefix string is represented by 10110. Using the rules for obtaining coverage strings given above, the corresponding coverage string would be 11010, which means that the second position of the prefix string 10110 is a 0, whereas the second position of the coverage string 11010 is a 1. There is no need to indicate that a position is covered when the position itself is a "0", and thus we make the adjustment to the basic algorithm that a coverage value is set to zero if the corresponding position of q is zero. Whereas this gives no reduction of states for this example, in general it can lead to a large reduction, because the algorithm then "sees" more common coverage strings.

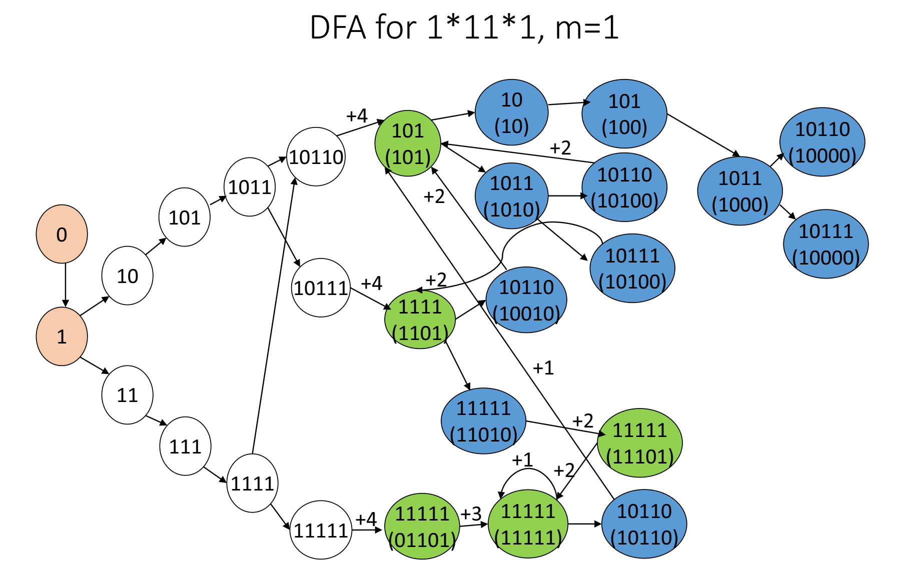

Fig. 4 **a** States of the AMC for seed $1 * 11 * 1$ with $m = 1$. Coverage for strings is shown in *parentheses*. For clarity, not all transitions are shown. On the symbol 1, state 10110 (10000) transitions to 101 (101) with coverage increment 3, and state 10111 (10000) transitions to 1111 (1101), also with coverage increment 3. State 11110 is merged with 10110, both with and without coverage. The m-tuples are shaded in *peach*, prefixes with no coverage in *white*, counting states in *green*, and the other states with non-zero coverage in *blue*. Increments to the coverage count are indicated for transitions into counting states

One more adjustment is made to the basic algorithm. For the seed $11 * 1$ with $m = 1$ (see Fig. 3b), the state $\begin{pmatrix} 110 \\ 100 \end{pmatrix}$ is obtained in the state generation process, and based on the rules above it would be added to the automaton. However, the state $\begin{pmatrix} 110 \\ 010 \end{pmatrix}$ is already in the automaton at that point. With the direct hit, the update to coverage would be the same for these two states. Also, since the active proper suffix of 110 is 0, the “1” in the second position of $q = 110$ cannot possibly be in a future seed hit. Thus, probabilities and coverage updates for the two states will be exactly the same, and a minimization algorithm would combine them. We add a rule, which is given in the next paragraph, to combine such states as well. Whereas for this small seed, combining these states leads to a reduction of only one state, for longer, more complicated seeds, large reductions can be obtained through a cascading of combined states.

In general, when attempting to enter a state $\begin{pmatrix} q' \\ c' \end{pmatrix}$ into Λ and searching for a possible matching state, if there is a state $\begin{pmatrix} q* \\ c* \end{pmatrix}$ with $q^* \sim q'$ but $c^* \neq c'$, the algorithm checks to see if $aps(q') \in \hat{Q}_{nosuf}$. In that case, none of the 1’s of q' and $q*$ can possibly be involved in a future seed hit. Then, if the updates to coverage on the direct hits of q' and q^* are the same, the two states may be combined, since coverage updates and probabilities are guaranteed to be the same for any input string.

We point out here that it is possible to have states with $q^* \sim q'$ and $c^* \neq c'$, and $\text{aps}(q') \notin \hat{Q}_{\text{nosuf}}$, yet have exactly the same coverage updates for all possible input strings. Such strings would be combined by a minimization algorithm. In our sequential algorithm, we do not search for such situations, believing that such a search may be more computationally costly than the advantage in possibly finding a smaller state space, and thus we do not obtain the minimal state space in all cases. (This conjecture will be examined in future work). However, we do obtain the minimal state space in many cases, and in all cases get a vastly reduced state space that facilitates fast computation. If the minimal state space is sought, it can be obtained by applying a minimization procedure to Λ, while avoiding the need to ever form a full automaton with states Λ_{ext}. In the next subsection, we compare the size of the state spaces Λ_{ext} and Λ to show the vast reduction in states that is attained, and also compare $|\Lambda|$ to minimal automata either with or without q values representing seed words.

3.2 Reduction in the number of states

Table 1 presents the number of states $|\Lambda_{\text{ext}}|$ in Λ_{ext} (formed from prefixes of A_{ext}; see Eq. (2)), and the number of states $|\Lambda|$ in the reduced AMC based on the algorithm above. For comparison purposes, we also applied DFA minimization algorithms to determine the number of states in a minimized version of Λ [we call this number $|\Lambda_{\text{Mealy}}|$ in the case where seed words are not included in Q and coverage updates are incorporated on the appropriate state transitions (Mealy 1955), and $|\Lambda_{\text{Moore}}|$ in the case where seed words are included in Q and coverage updates take place when strings of A_S are reached (Moore 1956)].

The reduction in the number of states in Λ when compared with Λ_{ext} is apparent. In the most extreme case of the table, for seed $1 * 1 * 1 * 1$, $|\Lambda_{\text{ext}}| = 286$ and $|\Lambda| = 25$. In many cases, Λ achieves the minimal number of states $|\Lambda_{\text{Mealy}}|$, and for every case, Λ has less states than Λ_{Moore}. Thus, we attain a small state space without ever forming a full automaton with state space Λ_{ext}, and one that compares well with minimized

Table 1 Size of state spaces Λ_{ext}(extended Aho–Corasick), Λ_{Moore} and Λ_{Mealy} (minimal versions using updates on entering states and on transitions), and Λ (the sequentially reduced state space forwarded in this paper)

Seed	$\lvert\Lambda_{\text{ext}}\rvert$	$\lvert\Lambda_{\text{Moore}}\rvert$	$\lvert\Lambda_{\text{Mealy}}\rvert$	$\lvert\Lambda\rvert$	Seed	$\lvert\Lambda_{\text{ext}}\rvert$	$\lvert\Lambda_{\text{Moore}}\rvert$	$\lvert\Lambda_{\text{Mealy}}\rvert$	$\lvert\Lambda\rvert$
1*111	26	17	15	15	1*1111*1	112	45	36	36
11*11	29	20	16	16	1*111*11	132	42	35	38
111*1	24	20	15	15	1*11*111	144	50	44	44
1*1*1	66	20	16	16	1*1*1111	142	39	35	35
1*1*1*1	286	30	25	25	1**11111	132	56	51	51
11*11*1	99	36	28	29	1**1*111	444	85	76	76
1**11	72	37	31	31	1**11*11	512	61	53	53
11**1	67	48	36	36	1**111*1	356	79	68	70
11*1*1	85	50	37	38	1*1*11*1	362	69	56	58

versions in terms of the number of states. In Sect. 3.4, we illustrate how the reduction in states is important in an application to a long spaced seed.

3.3 Computing the distribution

After setting up the state space Λ and its transition probability matrix Ω, we use an adaptation of a computation method used in Aston and Martin (2007) to obtain the distribution of spaced seed coverage for a seed of length k with r stars. The computation follows the basic principle of the efficient FMCI computation technique of Koutras and Alexandrou (1995) in that the transition probability matrix Ω of the AMC has order equal to $|\Lambda|$, and not $|\Lambda|$ multiplied by the number of possible coverage counts, as would be the case with the original FMCI formulation.

Probability matrices $\Psi_t, t = m, m+1, \ldots, n$ hold probabilities for the AMC lying in the various states of Λ, where the subscript t indicates the time point. The matrices each have $n - k + r + 2$ rows that correspond to all the reachable coverage values $0, k - r, k - r + 1, \ldots, n$ (Koutras and Alexandrou 1995 use probability vectors for each of the possible values of the statistic of interest, so using a probability matrix is analogous.) The $|\Lambda|$ columns of the matrices hold probabilities of the AMC lying in the various states of Λ for each particular coverage value. The sum of the elements in each matrix is one, with the sum over row i of Ψ_t giving the probability of coverage i at time t, so that the coverage distribution may be obtained as $\Psi_n 1$, with 1 being a column vector of $1's$ of length $|\Lambda|$.

We initialize the computation at time $t = m$, with the non-zero elements of the first row of Ψ_m holding initial probabilities for the m-tuples (these initial probabilities are input to the algorithm in row vector $\tilde{\pi}$ or computed using the input transition probability matrix T if stationarity is assumed). The remaining rows of Ψ_m contain all zeroes. To update the system from time t to $t + 1$, two steps are carried out: (i) multiplication by Ω to update state probabilities, and (ii) moving the probabilities for transitions corresponding to seed hits to the correct row. More details are given in the next paragraph.

After multiplying Ψ_t by Ω, a column j that corresponds to a non-zero entry of $U(i, j)$ for transitions $\lambda_i \rightarrow \lambda_j$ will hold probabilities for a seed hit at time $t + 1$. However, since the coverage count increases when λ_j is entered, if $h > 1$, the probability $e_{i,j} = \Psi_t(h, i) \times \Omega(i, j)$ for entering λ_j from λ_i with current coverage $h+k-r-2$ is added to $\Psi_{t+1}(h + U(i, j), j)$, and subtracted from $\Psi_{t+1}(h, j)$. When $h = 1$ (corresponding to no coverage), $e_{i,j}$ is added to $\Psi_{t+1}(2, j)$ (which corresponds to coverage $k - r$) and subtracted from $\Psi_{t+1}(1, j)$. Steps (i) and (ii) are repeated to obtain $\Psi_{m+1}, \ldots, \Psi_n$, and the probability of coverage i is obtained as the sum of the elements of row i of Ψ_n.

3.4 Application to Patternhunter seed

Version 1 of the Patternhunter software (Ma et al. 2002) used the spaced seed 111*1**1*1**11*111 that is optimal in the sense that it has the highest single hit probability for Bernoulli trials with match probability $p = 0.7$ on alignment

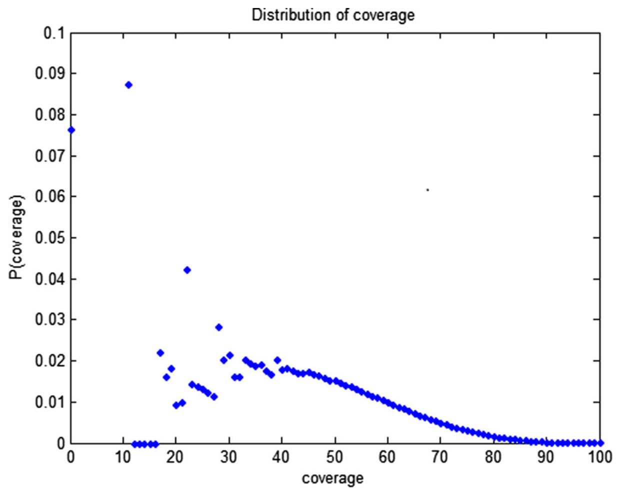

Fig. 5 Distribution of coverage of the Patternhunter seed for a first-order Markovian sequence **X** of length $n = 100$, with transition probabilities $P(X_t = 1|X_{t-1} = 0) = 0.45$; $P(X_t = 1|X_{t-1} = 1) = 0.85$, and stationary distribution $\tilde{\pi}_0 = 1/4$, $\tilde{\pi}_1 = 3/4$

length $n = 64$ (Ma et al. 2002). The algorithm of the last section was programmed in MATLAB (version R2010b), and applied to compute the distribution of coverage for seed 111*1**1*1**11*111. The underlying sequence was assumed to be a stationary Markovian sequence of order $m = 1$, with transition probabilities defined by $P(X_t = 1|X_{t-1} = 0) = 0.45$; $P(X_t = 1|X_{t-1} = 1) = 0.85$ and $P(X_t = 0|X_{t-1} = x) = 1 - P(X_t = 1|X_{t-1} = x)$, $x \in \{0, 1\}$ (the corresponding stationary distribution is $\tilde{\pi}_0 = 1/4$, $\tilde{\pi}_1 = 3/4$). Figure 5 displays the computed distribution for $n = 100$.

The size of the state space used was $|\Lambda| = 4215$, compared with $|\Lambda_{\text{Moore}}| = 4260$, $|\Lambda_{\text{Mealy}}| = 3782$ and $|\Lambda_{\text{ext}}| = 321596$. Note that Λ_{ext} is the state space of the AMC that would be obtained (before minimization) using the Markov chain embedding procedure of Martin (2013), and thus a substantial improvement is obtained using the current procedure. The computation took 91 s (33.7 s for setting up Λ and 57.3 s to compute the distribution) using a Dell PC with an Intel Core i7 CPU 873 with 2.93 GHz and 8 GB RAM. We point out here that a program to set up the state space Λ_{ext} (and not compute the distribution) took over 26 h to run on the same computer. Thus, we see the importance of avoiding setting up Λ_{ext}. Note also that to compute probabilities for single or multiple hits requires only 322 states when seed words are included, and 278 when they are not (these numbers were derived using a DFA minimization algorithm), highlighting the difficulty of computing the "coverage" distribution. The feasibility of the algorithm for computing distributions for the long seeds that are used in bioinformatics is apparent from the results of this section.

4 Summary

This paper deals with computing distributions of statistics using an auxiliary Markov chain. As stated previously, the computation itself is similar in nature to the efficient

FMCI formulation of Koutras and Alexandrou (1995), in that the AMC states of Λ are not repeated for each of the possible coverage counts. A difference in the procedures is that whereas the latter reference dealt with statistics that either stay the same or increase by one (called Markov chain imbeddable of the binomial type in that work), we deal with statistics whose value can increase by an arbitrary amount on a pattern occurrence. The main difference, however, is in the nature of the problem on which the current paper focuses, as the focus of this paper is on reducing the number of AMC states $|\Lambda|$ in problems where $|\Lambda_{\text{ext}}|$ can be very large, so large, in fact, that setting up a DFA before minimizing it is prohibitive. A situation where this can occur is where overlapping pattern occurrences are reckoned differently than non-overlapping occurrences, rendering a need for using prefixes Λ_{ext} of patterns extended to overlapping occurrences.

In the example of Sect. 3.4, searching for hits of the Patternhunter spaced seed required 278 AMC states (in minimal form), whereas a minimal AMC to determine spaced seed coverage required 3782 states (compared with 321,596 states in an AMC set up using a reasonable Markov chain embedding procedure). The computation of the distribution of spaced seed coverage is especially difficult because one must keep track of the overlapping structure of pattern occurrences since a "1" of the sequence can only be counted once in coverage. As was shown in that example, the reduction of states (and computation time) afforded by our algorithm can be considerable. This is the main contribution of the present work. Thus, whereas our procedure would obtain no reductions of $|\Lambda|$ for situations such as those handled in Koutras and Alexandrou (1995), we do add to the Markov chain embedding toolkit for situations as described above.

In the context of spaced seed coverage, active proper suffixes are used to assist in producing a reduced state space sequentially without ever generating a full deterministic finite automaton with states Λ_{ext}. Transition probabilities for AMC states and updates of the statistic's values for certain transitions are determined during the state generation process. In many cases, a minimal state space is attained, and even in those when it is not, the size of the state space compares well with the minimized version. If one seeks a minimal state space, then applying a minimization algorithm to our state space is much more efficient than minimizing a full automaton with states Λ_{ext}.

In future work, the authors would like to explore the efficacy of obtaining minimal state spaces using a sequential procedure, and answer the question of whether such a pursuit is worthwhile, or whether or not we are better off stopping with a vastly reduced state space, as in the example of Sect. 3.4. The authors have already obtained preliminary results applying spaced seed coverage to string kernels and alignment-free distances (Noé and Martin 2014) and are working on extending the algorithm of this paper to coverage of multiple spaced seeds for applications in similarity searches over biosequences. The need to keep the state space small without ever forming a full DFA is even more pronounced when dealing with that problem.

Acknowledgements The authors greatly appreciate the comments of the associate editor and the referees, which proved to be extremely useful in improving the manuscript. D. E. K. Martin was supported in this research by the National Science Foundation under Grant DMS-1107084. Laurent Noé was supported by a CNRS Mastodons grant, and benefited from a half-time course buyout from the French Institute for Research in Computer Science and Automation (Inria).

References

Aho, A. V., Corasick, M. J. (1975). Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, *18*(6), 333–340.

Aston, J. A. D., Martin, D. E. K. (2007). Distributions associated with general runs and patterns in hidden Markov models. *Annals of Applied Statistics*, *1*(2), 585–611.

Balakrishnan, N., Koutras, M. V. (2002). *Runs and scans with applications*. New York: Wiley.

Bassino, F., Clement, J., Fayolle, J., Nicodème, P. (2008). Constructions for clumps statistics. *Discrete Mathematics and Theoretical Computer Science* (DMTCS), AI, 179–194.

Benson, G. (1999). Tandem repeats finder: a program to analyze DNA sequences. *Nucleic Acids Research*, *27*, 573–580.

Benson, G., Mak, D. Y. F. (2009). Exact distribution of a spaced seed statistic for DNA homology detection. *String processing and information retrieval, Lecture Notes in Computer Science*, *5280*, 282–293.

Buhler, J., Keich, U., Sun, Y. (2005). Designing seeds for similarity search in genomic DNA. *Journal of Computer and System Sciences*, *70*, 342–363.

Ebneshahrashoob, M., Gao, T., Wu, M. (2005). An efficient algorithm for exact distribution of discrete scan statistic. *Methodology and Computing in Applied Probability*, *7*(4), 459–471.

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

Fu, J. C., Koutras, M. V. (1994). Distribution theory of runs: a Markov chain approach. *Journal of the American Statistical Association*, *89*, 1050–1058.

Fu, J. C., Lou, W. Y. (2003). *Distribution theory of runs and patterns and its applications*. Singapore: World Scientific Publishing Co.

Fu, J. C., Lou, W. Y. W., Bai, Z.-D., Li, G. (2002). The exact and limiting distributions of the number of successes in success runs within a sequence of Markov-dependent two-state trials. *Annals of the Institute of Statistical Mathematics*, *54*(4), 719–730.

Hopcroft, J. E. (1971). An $n \log n$ algorithm for minimizing states in a finite automaton. In Z. Kohavi & A. Paz (Eds.), *Theory of Machines and Computations* (pp. 189–196). New York: Academic Press.

Hopcroft, J. E., Motwani, R., Ullman, J. D. (2001). *Introduction to automata theory, languages, and computation*. New York: Addison-Wesley.

Keich, U., Li, M., Ma, B., Tromp, J. (2004). On spaced seeds for similarity search. *Discrete Applied Mathematics*, *138*(3), 253–263.

Koutras, M. V., Alexandrou, V. A. (1995). Runs, scans and urn models: A unified Markov chain approach. *Annals of the Institute of Statistical Mathematics*, *47*, 743–766.

Kucherov G., Noé, L., Roytberg, M. (2007). Subset seed automaton. In *Implementation and application of automata, Lecture Notes in Computer Science*, *Volume 4783* (pp. 180–191).

Ledent, S., Robin, S. (2005). Checking homogeneity of motifs' distribution in heterogenous sequences. *Journal of Computational Biology*, *12*, 672–685.

Lladser, M., Betterton, M. D., Knight, R. (2008). Multiple pattern matching: a Markov chain approach. *Journal of Mathematical Biology*, *56*(1–2), 51–92.

Lou, W. Y. W. (2003). The exact distribution of the k-tuple statistic for sequence homology. *Statistics and Probability Letters*, *61*, 51–59.

Ma, B., Tromp, J., Li, M. (2002). Patternhunter-faster and more sensitive homology search. *Bioinformatics*, *18*(3), 440–445.

Marshall, T. and Rahmann, S. (2008). Probabilistic arithmetic automata and their application to pattern matching statistics. *Lecture Notes In Computer Science*; Vol. 5029, *Proceedings of the 19th Annual Symposium on Combinatorial Pattern Matching* (pp. 95–106).

Martin, D. E. K. (2006). The exact joint distribution of the sum of heads and apparent size statistics of a "tandem repeats finder" algorithm. *Bulletin of Mathematical Biology*, *68*, 2353–2364.

Martin, D. E. K. (2008). Application of auxiliary Markov chains to start-up demonstration tests. *European Journal of Operational Research*, *184*(2), 574–583.

Martin, D. E. K. (2013). Coverage of spaced seeds as a measure of clumping. In American Statistical (Ed.), *Association 2013 Proceedings of the Section on Computational Statistics*. Alexandria, VA: American Statistical Association.

Martin, D. E. K. (2014). P-values for the discrete scan statistic through slack variables. *Communications in Statistics, Simulation and Computation*. doi:10.1080/03610918.2013.777457.

Martin, D. E. K., Aston, J. A. D. (2008). Waiting time distribution of generalized later patterns. *Computational Statistics and Data Analysis*, *52*, 4879–4890.

Martin, D. E. K., Aston, J. A. D. (2013). Distributions of statistics of hidden state sequences through the sum-product algorithm. *Methodology and Computing in Applied Probability*, *15*(4), 897–918.

Martin, D. E. K., Coleman, D. A. (2011). Distributions of clump statistics for a collection of words. *Journal of Applied Probability*, *48*, 1049–1059.

Mealy, G. H. (1955). A method for synthesizing sequential circuits. *Bell System Technical Journal*, *34*(5), 1045–1079.

Moore E.F. (1956) Gedanken-experiments on sequential machines. *Automata Studies: Annals of Mathematical Studies*, 34, 129–153. Princeton, N.J.: Princeton University Press.

Noé, L., Kucherov, G. (2004). Improved hit criteria for DNA local alignment. *BMC Bioinformatics*, *5*, 149.

Noé, L., Martin, D. E. K. (2014). A coverage criterion for spaced seeds and its applications to SVM string kernels and k-mer distances. *Journal of Computational Biology*, *21*(12), 947–963.

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

Parzen, E. (1962). *Stochastic Processes*. San Francisco: Holden-Day Inc.

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

Robin, S., Rodolphe, F., Schbath, S. (2005). *DNA, words and models*. United Kingdom: Cambridge University Press.

Tewari, A., Srivastava, U., Gupta, P. (2002). A parallel DFA minimization algorithm. In High performance computing Hi PC, *Lecture Notes in Computer Science* (Vol. 2552, pp. 34–40).