A Generic Finite Automata Based Approach to Implementing Lymphocyte Repertoire Models

Johannes Textor
Theoretical Biology & Bioinformatics
Universiteit Utrecht
Padualaan 8
3584 CH Utrecht,
The Netherlands
johannes.textor@gmx.de

Katharina Dannenberg
Institute for Theoretical Computer Science
Universität zu Lübeck
Ratzeburger Allee 160
23538 Lübeck, Germany
k.dannenberg@hotmail.de

Maciej Liśkiewicz
Institute for Theoretical Computer Science
Universität zu Lübeck
Ratzeburger Allee 160
23538 Lübeck, Germany
liskiewi@tcs.uni-luebeck.de

ABSTRACT

Artificial immune systems (AIS) inspired by lymphocyte repertoires include negative and positive selection, clonal selection, and B cell algorithms. Such AISs are used in computer science for machine learning and optimization, and in biology for modeling of fundamental immunological processes. In both cases, the necessary size of repertoire models can be huge. Here, we show that when lymphocyte repertoire models based on string patterns can be compactly represented as finite automata (FA), this allows to efficiently perform negative selection, positive selection, insertion into, deletion from, uniform sampling from, and counting the repertoire. Specifically, for r-contiguous pattern matching, all these tasks can be performed in polynomial time. But even in NP-hard cases like Hamming distance matching, the FA representation can still lead to practically important efficiency gains. We demonstrate the feasibility and flexibility of this approach by implementing T cell positive selection simulations based on human genomic data using four different pattern rules. Hence, FA-based repertoire models generalize previous efficient negative selection algorithms to perform several related algorithmic tasks, are easy to implement and customize, and are applicable to real-world bioinformatic problems.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Pattern matching*; I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Artificial Immune Systems; Negative Selection; Immunology

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.

ACM 978-1-4503-2662-9/14/07 ...$15.00.
http://dx.doi.org/10.1145/2576768.2598331.

1. INTRODUCTION

Lymphocytes are the cognate elements of human immune systems. These cells carry individually different receptors that allow them to detect specific molecular patterns called *epitopes*, and trigger certain actions such as the initiation or suppression of an immune response. Molecular patterns examined by some B- and most T-lymphocytes are essentially short linear strings of amino acids – for instance, one famous epitope from HIV can be written as the string YNTVAT using the one-letter amino acid coding[1]. Lymphocyte-epitope interactions have inspired the development of a class of artificial immune systems (AIS) in which these interactions are modeled using string pattern matching [24]. In this paper, we call such AISs *repertoire models*. A particularly well-known repertoire model is the negative selection algorithm (NSA) [10], which is based on a process in the thymus where newborn T cells carrying random receptors are exposed to a population of peptides from the host organism. All cells recognizing such peptides are deleted, ensuring that cells leaving the thymus recognize many different foreign patterns but tolerate self. Other AISs based on string models of lymphocyte-epitope interactions include clonal selection (based on the principle of B cell affinity maturation) and the B cell algorithm (based on contiguous somatic hypermutation), which are used for optimization tasks.

Repertoire models have been applied in both computer science, particularly computer security [10], and in computational immunology [23]. The NSA in particular triggered a flurry of research after some initial enthusiasm [9], which however was soon replaced by skepticism about its computational cost [15, 28] and generalization capabilities, leading to its demise in the field of AIS. On the other hand, and largely unappreciated by the AIS community, repertoire models including NSA have been quite successfully applied in immunology, where their use as models of the real immune system continues to provide insight into fundamental principles of immunity and infection until recently [5, 6, 3], most notably in the field of HIV infection modeling [17, 18].

An issue pertaining to both application fields of repertoire models is the huge size of the repertoire sets that are typically required. For realistic immune system modeling, one

[1]Actually this epitope is usually written as SLYNTVATL, but only positions 3 through 8 from this string are in contact with the T cell receptor whereas the others are hidden in a carrier molecule called MHC [22].

aims to attain comparable size and diversity as in the modeled organism – for a mouse, this means on the order of 10^8 cells, and for a human, even 10^{10}. But also for uses in classification, empirical studies have often found the detector sets needed to achieve acceptable classification results to be prohibitively large (e.g. [15]). This problem can be solved by algorithms that generate polynomial sized compressed representations of repertoires [8] for the cases where the so-called r-contiguous [23] and r-chunk [1] matching rules are used to model T cell–peptide interactions. Here, we adopt this idea of repertoire compression and take it further by presenting a generic approach that uses deterministic finite automata (DFA) as a compressed representation of the lymphocyte repertoire. This representation allows us to perform six common tasks of repertoire modeling (defined below) in a rather flexible manner: The user only needs to supply code that generates a DFA that accepts strings encoding all detectors which match a given *single* sample string (Fig. 1). All other steps are then identical regardless the detector type.

Specifically, we show how the following tasks (used in e.g. [26, 5, 3]) can be performed using the DFA representation. Repertoire modeling often involves generating a set of patterns that do not match any string in an input dataset too strongly (*negative selection*) and/or weakly match at least one string from the same dataset (*positive selection*). Having obtained the initial repertoire, one usually examines a set of "antigens" (i.e., elements to classify), for which we search one or all matching detectors (*classification*). By *counting* the number of detectors from the repertoire which match an antigen string, we obtain a measure of dissimilarity between the antigen and the input set of self strings. But we may also wish to *sample* detectors explicitly from the whole repertoire or from the subset that also matches a given antigen in order to examine their properties or to perform further modifications on them (e.g., run a genetic algorithm). Lastly, we may wish to *manipulate* the repertoire, e.g. by deleting detectors that perform poorly or inserting mutated clones of those that perform exceptionally well.

This paper is divided into a theoretical and an empirical part. After some basic definitions, we first consider the DFA approach generally, i.e., independent of the detector type. We show how several tasks of lymphocyte repertoire modeling relate to each other and how they can be solved using DFAs. Then we give a general sufficient condition which implies efficient solutions for all of these tasks. Afterwards, we delineate the theoretical limitations of our approach by showing an example for which no efficient DFA based repertoire representation exists, even though the example is easily solved by other means. In the empirical part, we compare the space and time cost of the DFA based approach to the dominant existing approach of generating detectors one by one explicitly [13]. We perform this approach on "typical" data of immune system modeling, i.e., short strings from large proteomes [4]. We find that while space savings are indeed relatively modest for "difficult" detector types, time savings of 4-5 orders of magnitude are nevertheless achievable.

2. DEFINITIONS

Let Σ be an alphabet and let $S \subseteq \Sigma^L$ for a fixed positive integer L. A *detector* of type D is a string d of length L over an auxiliary alphabet Σ_D with an associated matching rule $m_D : (\Sigma_D)^L \times \Sigma^L \to \{+1, -1\}$. We define $D[S^-]$ as the set of all detectors that classify *all* elements of S *negatively*, i.e., for all $d \in D[S^-]$, $m_D(d, s) = -1$ for all $s \in S$, whereas $D[S^+]$ refers to the set of detectors that classify *one or more* element of S *positively*. Hence, $D[S^-]$ is the complement of $D[S^+]$ in $(\Sigma_D)^L$. By convention, $D[\emptyset^+] = \emptyset$ and hence $D[\emptyset^-] = (\Sigma_D)^L$.

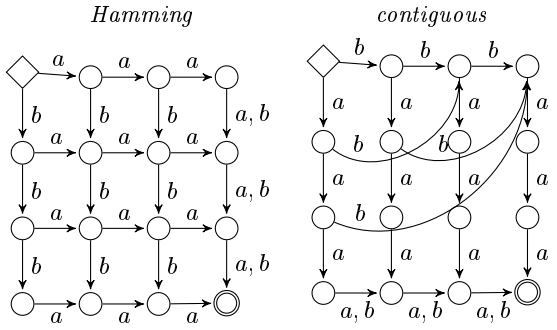

Figure 1: Finite automata encoding the detectors that match the string a^6 ($\Sigma = \{a, b\}$) for 3-Hamming and 3-contiguous matching (Definition 2). The left automaton accepts all strings $d \in \Sigma^6$ of Hamming distance ≤ 3 to the string a^6. The right automaton accepts all $d \in \Sigma^6$ such that d and a^6 are identical in ≥ 3 contiguous positions. The incomplete versions of the automata (i.e., without a sink state) are shown.

For formal definitions of DFAs and their accepted languages, see e.g. [12]. We say that a DFA is *complete* if for any state q and any symbol a there exists an edge starting in q and labeled with a. Any DFA can be made complete by adding one "sink" state and linking all missing transitions to this state. A DFA is called *minimal* (or *minimum-state* DFA) if no other DFA with fewer states accepts the same language.

The languages used in repertoire modeling are finite, and therefore are describable by DFAs that are acyclic, i.e. the transition graph of which is loop free. To minimize an acyclic automaton M we transform M to the complete automaton by adding the sink state and missing transitions, thus obtaining an automaton that is acyclic apart from a loop on the sink. An *m-level* DFA is a DFA in which all states (except the sink if the DFA is complete) can be partitioned into $m + 1$ levels with the following properties: (1) there is exactly one state at level 0 and it is the start state, (2) there is exactly one state at level m and it is the accept state, (3) every transition is from a state in level i to either a state in level $i + 1$ or to the sink state (if it exists), and (4) for any state, q, the accept state is reachable from q and q is reachable from the start state.

3. DFA-BASED REPERTOIRE MODELS

We now introduce the DFA-based approach to Repertoire modeling and prove some positive and negative results that characterize the promises and limitations of this approach.

3.1 Tasks of repertoire modeling

To handle the basic tasks of AIS and computational immunology efficiently, one needs an appropriate model for encoding detectors sets. Our approach here is to use DFAs

for this aim. This modeling allows succinct representation of, typically huge, detector sets $D[S^+]$ and/or $D[S^-]$ and efficient manipulations of these sets. With this DFA representation, below we specify the tasks to be solved.

Definition 1. For given detector types D and D' we define the following tasks related to lymphocyte repertoire modeling. We are given two sample sets $S, S' \subseteq \Sigma^L$, two sequences of detector sets $T_1, T_2, T_3, \ldots \subseteq (\Sigma_D)^L$ and $T'_1, T'_2, T'_3, \ldots \subseteq (\Sigma_D)^L$, and a number n.

1. *Positive selection.* Output a DFA M s.t. $L(M) = D[S^+]$.
2. *Negative selection.* Output a DFA M s.t. $L(M) = D'[S'^-]$.
3. *Positive and negative selection.* Output a DFA M s.t. $L(M) = D[S^+] \cap D'[S'^-]$.
   Note that we allow the detector types to be different (e.g., we may use different matching radii for positive and negative selection).
4. *Repertoire manipulation.* First perform task (3) computing M_0 with $L(M_0) = D[S^+] \cap D[S'^-]$ and next for $i = 1, 2, \ldots$ compute iteratively DFAs M_i s.t.
$$L(M_i) = (L(M_{i-1}) \setminus T_i) \cup T'_i.$$
5. *Counting the repertoire.* First perform any task of (1-4), then for the resulting DFA M output the cardinality of $L(M)$.
6. *Sampling from the repertoire.* Let M be the output of any task in (1-4). Output n strings from $L(M)$ uniformly at random (drawn with replacement).

First, we note that some further common tasks are special cases of the tasks above. Particularly, *classification*: given an S and a test string m determine if there is a detector in $D[S^-]$ which matches m, and *response simulation*: given an S and an antigen x choose randomly detectors in $D[S^-]$ which match x. The first task can be modeled as follows: compute a DFA M s.t. $L(M) = D[S^-] \cap D[\{m\}^+]$ and output m as "positive" if $L(M) \neq \emptyset$; otherwise output m as "negative". Similarly, in the second task we compute a DFA M s.t. $L(M) = D[S^-] \cap D[\{x\}^+]$ and next output strings from $L(M)$ uniformly at random.

3.2 Algorithms and complexity: good news

We start with the following obvious observation: The DFAs for tasks (1-3) listed in Definition 1 are acyclic. Moreover they can be transformed to complete L-level DFAs. Next we give a sufficient condition for implementing repertoire models efficiently.

LEMMA 1. *Let D, D' be detector types for which there is a polynomial time algorithm for positive selection. Then any task listed in Definition 1 can be solved by an algorithm whose runtime is polynomial in L, n, and the cardinalities of input sets.*

PROOF. With positive selection solved, we can solve negative selection by simply inverting the accepted states. Task 3 can then be solved by computing the intersection of the automata resulting from positive and negative selection. For task 4 any iteration can be done in polynomial time according to the output size of the previous iteration. This can be solved by removing or appending the strings in T_i and T'_i from the language accepted by M_{i-1} one-by-one, starting with M_0 given as output by task (3). Each string addition or removal can be done by adding at most L states and $L \cdot |\Sigma_D|$ transitions to M_{i-1}. A standard depth first search counting of accepting paths solves (5) because any M output by (1-4) must be deterministic and acyclic. Lastly, for random sampling from $L(M)$ (task 6), one can use e.g. the linear algorithm proposed by Bernardi and Giménez [2]. □

3.3 Algorithms and complexity: bad news

Below we analyze the computational complexity of tasks (1-4) in more details. For a given sample set $S \subseteq \Sigma^L$, let M_s, with $s \in S$, denote a minimum-state DFA s.t. $L(M_s) = D[\{s\}^+]$. Then, from the computational complexity point of view, task 1 can be equivalently formulated as: output a DFA M such that $L(M) = \bigcup_{s \in S} L(M_s)$. Similarly, if automata M'_s are minimum-state DFAs s.t. $L(M'_s) = D[\{s\}^-]$ then task 2 can be formulated as: output a DFA M' such that $L(M') = \bigcap_{s \in S} L(M'_s)$. We show that in general, these problems are computationally difficult – more strongly, even only deciding if $L(M) = \Sigma^*$, or equivalently if $L(M') = \emptyset$, is difficult.

We start with a very general setting and consider the problem called *the finite automaton intersection problem*, formulated as follows: Let $M_1, \ldots, M_k$ be k DFAs with a common alphabet Σ. The problem is to determine whether $\bigcap_{i=1}^k L(M_i)$ is nonempty. This problem is PSPACE-complete for DFAs with cycles [19]. For acyclic DFAs the complexity interestingly drops to NP but remains NP-complete, as we now show. First, it is easy to see that the problem for acyclic DFAs is in NP: simply "guess" a string $w \in \Sigma^*$ of length less or equal to the maximum number of states over all DFAs $M_1, \ldots, M_k$ and then check deterministically if $w \in L(M_i)$, for all $i = 1, \ldots, k$. To prove NP-hardness, one can modify the proof of Kozen [19]. However, in the context of our approach to repertoire representation we can prove an even stronger NP-completeness statement.

LEMMA 2. *There exist detector types D for which, if M'_s denotes a minimum-state DFA s.t. $L(M'_s) = D[\{s\}^-]$ then (1) for any $s \in \Sigma^L$ the number of states of M'_s is at most L^2, and (2) the problem to decide if for a given $S \subseteq \Sigma^L$ set $\bigcap_{s \in S} L(M'_s)$ is nonempty is* NP*-complete.*

PROOF. The lemma holds, e.g. for r-Hamming detectors D (see Definition 2), even for binary alphabets $\Sigma = \{a, b\}$. For this detector type it is easy to construct for any $s \in \Sigma^L$ a DFA M'_s for $D[\{s\}^-]$ having at most L^2 states (see Fig. 1 for an example automaton accepting $D[\{s\}^+]$, with $s = a^6$, which can be easily modified to obtain the required automaton M'_s). Property (2) follows directly from Theorem 2 in [21] that says that the problem to decide if for a given set $S \subseteq \Sigma^L$ the detector set $D[S^-]$ is nonempty, is NP-complete. Hardness for binary alphabets follows from the fact that the well-known farthest string problem is NP-complete even for binary strings [20]. □

In the next section we will provide a generic algorithm that allows to solve tasks (1-4) efficiently if the resulting automaton M, with $L(M) = \bigcup_{s \in S} L(M_s)$, has small size. However, we first give an example where the minimal automaton is huge: For an arbitrary alphabet $\Sigma = \Sigma_D$, consider

the matching function $m(d,s) = +1$ iff (a) $d,s \in \{uu : \text{for } u \in \Sigma^{L/2}\}$ if L is even and (b) $d,s \in \{uau : \text{for } u \in \Sigma^{L/2} \text{ and } a \in \Sigma\}$ if L is odd.

LEMMA 3. *For the detector defined above, for any $s \in \Sigma^L$ the number of states of the minimum DFA M_s, with $L(M_s) = D[\{s\}^+]$, is at least $|\Sigma|^{L/2+1}$.*

PROOF. Assume $s = a^L$ for some $a \in \Sigma$, and let $S = \{s\}$. Then $D[\{s\}^+] = \{uu : \text{for } u \in \Sigma^{L/2}\}$ if L is even and otherwise $D[\{s\}^+] = \{uau : \text{for } u \in \Sigma^{\lfloor L/2 \rfloor}\}$. But in both cases, any state-minimal DFA for this language has at least $|\Sigma|^{L/2+1}$ states. □

However, note that tasks (1-4) are easily solvable in polynomial time for this detector type. Therefore, our example illustrates that not every repertoire model that is easy to implement has also an efficient DFA-based implementation.

3.4 A generic algorithm for positive selection

Lemma 1 shows that once positive selection is solved (or negative selection, for that matter), all other tasks can then be performed applying basic DFA-related algorithms on the resulting automaton. But also positive selection itself can be solved by combining a much simpler task with standard DFA manipulation, as we now show. Recall that our task is to compute for a given S a DFA M such that $L(M) = D[S^+]$. The following algorithm accomplishes this.

1. $M \leftarrow$ "DFA recognizing $\emptyset$".
2. For all strings $s \in S$ do
3. compute M_s such that $L(M_s) = D[\{s\}^+]$,
4. compute N such that $L(N) = L(M) \cup L(M_s)$,
5. minimize N,
6. set $M \leftarrow N$.
7. Return M.

Note that the only part of this algorithm that depends on the detector type is the construction of M_s in line 3. Therefore, we have now broken down all tasks of repertoire modeling to the construction of an automaton that accepts the language of all detectors matching a single input string (Fig. 1). Computing the DFA N in line 4 with $L(N) = L(M) \cup L(M_s)$ can be done in time $\mathcal{O}(L \cdot n \cdot n_s \cdot |\Sigma|)$, where n denotes the width of M (i.e. maximum, over all $i = 1, \ldots, L$, of number of states on level i) and n_s is the width of M_s. The minimization step in line 5 is optional, but can be important when the product automaton computed in line 4 is highly redundant. Using the linear time minimization algorithm for acyclic DFAs by Revuz [25] we can implement step 5 in time $\mathcal{O}(L \cdot n \cdot n_s \cdot |\Sigma|)$.

LEMMA 4. *If for any $S \subseteq \Sigma^L$ the number of states of minimum-state DFA M, s.t. $L(M) = D[S^-]$, is polynomial with respect to $|S|$, $|\Sigma|$, and L then the negative selection task can be solved in polynomial time.*

Since the union operation on languages is associative and commutative our generic algorithm can be implemented in other ways than presented here. Particularly, one can compute the resulting DFA M using the "divide and conquer" strategy, i.e. divide S into two disjoint S_1 and S_2 of almost the same size, compute M_1 for $D[S_1^+]$ and M_2 for $D[S_2^+]$ and M for $L(M_1) \cup L(M_2)$. This algorithm can be also implemented using a "dynamic programming" approach such that first automata $M_1, M_2, \ldots$ for $L(M_{s_1}) \cup L(M_{s_2})$, $L(M_{s_3}) \cup L(M_{s_4}), \ldots$ are computed, then for $L(M_1) \cup L(M_2)$, $L(M_3) \cup L(M_4), \ldots$ and so on.

4. DETECTOR-SPECIFIC RESULTS

The framework outlined in the last section is not only of practical interest, but it also provides a generic proof system for obtaining positive or negative (hardness) results on the computational complexity of specific detector types. More formal versions of the below definitions can be found in [29].

Definition 2. Throughout, d is a detector and s a string.

1. *r-contiguous detectors:* $d \in \Sigma^L$ matches $s \in \Sigma^L$ iff d and s are identical in $\geq r$ contiguous positions.

2. *r-pattern detectors:* $d \in (\Sigma \cup \{\#\})^L$ matches $s \in \Sigma^L$ at every non-# position and contains at most r "don't-care-symbols" #.

3. *r-Hamming detectors:* $d \in \Sigma^L$ matches $s \in \Sigma^L$ iff d and s differ in at most r positions.

4. *r-PMBEC detectors:* We use this additional detector type as a biologically motivated variation of the Hamming distance. Let $\Delta : \Sigma \times \Sigma \to \mathbb{Z}$ be a similarity function between the symbols in Σ. Then d matches s iff (1) d and s differ in at most r positions and (2) for all positions i where they do differ, $\Delta(d[i], s[i]) \geq \vartheta$. In our empirical evaluation, we use the PMBEC amino acid similarity matrix that has been designed for immunoinformatic purposes [16] and has recently been applied to T cell epitope modeling [4]. The recommended cutoff threshold for this matrix is $\vartheta = 5$ [16], allowing 17 letter pairs to substitute, e.g. `S` and `T`.

LEMMA 5. *Assume $S \subseteq \Sigma^L$ and let M be a minimum-state complete DFA, s.t. $L(M) = D[S^+]$, for r-contiguous detectors. Then DFA M has $\mathcal{O}(L \cdot r \cdot \min\{|S|, |\Sigma|^r\})$ states.*

PROOF. (Sketch). One can prove this by considering the equivalence relation over Σ^* defined as

$$u \sim v \quad \text{iff} \quad \forall z \in \Sigma^* (uz \in D[S^+] \;\Leftrightarrow\; vz \in D[S^+]).$$

This relation has $\mathcal{O}(L \cdot r \cdot \min\{|S|, |\Sigma|^r\})$ equivalence classes. By the Myhill-Nerode theorem, this matches the minimum number of states of a DFA recognizing $D[S^+]$. For details, see the full version of our paper. □

From Lemmas 5 and 4 together with Lemma 1, we now immediately obtain the following generalization from negative selection to repertoire models.

COROLLARY 1. *For r-contiguous detectors, all tasks listed in Definition 1 are solvable in polynomial time.*

These results illustrate how we can apply the DFA framework to obtain polynomial time algorithms for repertoire modeling. Unfortunately, for all other detector types defined above, we can only obtain hardness results. The corollary below follows immediately from the intractability results for r-Hamming detectors presented in [21].

COROLLARY 2. *For r-Hamming, r-pattern and r-PMBEC detectors, tasks (1-5) in Definition 1 are not solvable in polynomial time unless* P = NP.

For r-Hamming, even only deciding if $D[S^-] \neq \emptyset$ is already NP-hard [21]; therefore, also sampling (task 6) is not efficiently solvable in that case. For r-pattern, this decision problem is trivial to answer [21], but solving positive selection in polynomial time would allow an efficient solution of the more general counting problem (task 5), which is #P-hard. Therefore tasks (1-5), which all contain positive selection as a special case, cannot be solved efficiently unless P = NP. Note that the exclusion of task 6 in Corollary 2 is because there exist detector types [31] for which sampling can be efficiently solved although the other tasks cannot (assuming P $\neq$ NP).

5. EMPIRICAL EVALUATION

Computational immunology has been a fruitful application domain of repertoire models in general and the most fruitful one of negative selection algorithms in particular [5, 6, 17, 18, 3]. Therefore, we evaluate the practical performance of the DFA approach using typical data from that domain. The input dataset sizes were set similarly as in studies by Košmrlj and others [17, 18, 3]. In those studies, computational T cell repertoire models were used for predictive modeling of the real human immune system, which provided important insight into the dynamics of HIV infection [18]. Typically, those models use thousands to tens of thousands of input strings. As string length we use $L = 6$ (which is the number of amino acids that determine CD8 T cell recognition [22]), even though Košmrlj et al. [18] used $L = 5$ instead, perhaps for computational reasons.

As in the mentioned studies [17, 18, 3] we generated the input datasets S by applying so-called MHC-binding prediction algorithms to the human proteome (i.e., the set of protein sequences from all human proteins), which was downloaded from the UniProt database [7]. Potential T cell epitopes were predicted using the NetMHCPan algorithm [11][2]. This resulted in a candidate set of 180,000 potential epitopes (strings of length $L = 6$) that can potentially be used for negative and positive selection in the thymus. From these candidate epitope sets, we randomly generated subsets with cardinalities ranging from 4,000 to 32,000 strings (similar as in [18]) and used these sets as input S. We implemented custom algorithms for union and intersection of acyclic L-level DFAs that take advantage of some of the properties of these automata to achieve better runtime than more generic algorithms. For instance, our algorithms can assume that all DFAs have only a single accepting state, and can build the product automaton level by level. For other DFA manipulation tasks, such as minimization, we used the AT&T Finite State Machine library[3].

5.1 Space efficiency

Our DFA based approach can only be expected to perform better than simple exhaustive algorithms when the DFA representation is significantly more compact than the set of represented detectors. To empirically investigate the space efficiency of the DFA representation, we first need to introduce a quantitative measure of space efficiency. For this purpose, we use the compression ratio. We first define the *size* of an (incomplete) DFA M as $\text{size}(M) = (n+m)\lceil \log_2(n) \rceil$, where n is the number of states and m is the number of transitions –i.e., $\text{size}(M)$ is the number of bits needed to store an adjacency list representation of M. Furthermore, we define the size of a string set $S \subseteq \Sigma^L$ as $\text{size}(S) = |\Sigma| \lceil \log(|\Sigma|) \rceil$ –i.e., $\text{size}(S)$ is the number of bits needed to store S. Now we can define the compression ratio of an automaton M as $\text{comp}(M) = \text{size}(L(M))/\text{size}(M)$, e.g., $\text{comp}(M) = 10$ would mean that storing $L(M)$ explicitly would require 10 times more bits than storing M.

We addressed the compression ratios by applying positive selection to the predicted T cell epitopes as input data as described above. We do not present results for negative selection because these are expected to be very similar[4]. The results of our analysis are shown in Fig. 2. The resulting DFA sizes ranged from 10^1 to 10^6 states. In general, the matching parameter r turned out to be the most important determinant of the DFA size, and the largest DFAs were found around $r = 5$ for r-contiguous matching and $r = 3$ for r-Hamming and r-PMBEC matching. As expected from our theoretical understanding of the problem, we found the highest compression ratios with r-contiguous matching, where they ranged up to the order of $10,000 : 1$, and of course even higher in the trivial cases where $L(M)$ equals or almost equals Σ^L. Except for such cases, we obtained comparatively lower but still substantial compression ratios of around $10 : 1$ for the r-Hamming and r-PMBEC distances. In general, the DFA size was always well below the size of the string set, except for some of the degenerate instances $r = 6$ (for r-contiguous) and $r = 0$ (for the others), where $L(M) = S$.

As indicated above, the compression ratio can be a misleading measure for very large detector sets, because a "naive" implementation could also just store the much smaller complement instead. Therefore, we focused our further evaluation on those instances where $|D[S^+]| < 20^6/2$, and investigated the determinants of the compression ratio. From Fig. 2 it is apparent that the compression ratio correlates with r, but r itself is in turn correlated with $|D[S^+]|$. Hence, we used a linear regression analysis to investigate which of the two parameters more strongly determines $\text{comp}(M)$.

5.2 Runtime efficiency

While we believe that the number of bits required to store M is a realistic measure for the space complexity of the DFA approach, there is no obvious machine-independent measure of practical (as opposed to asymptotic) runtime. Therefore, we compared the actual runtimes of a C++ implementation of naive positive selection and our C++ implementation of the DFA approach. To avoid skewing of our results due to potential paging between memory and hard-disk, we performed these experiments on an Intel(R) Xeon(R) X5680 server with 96GB of RAM. Although the machine had 12 processors with 6 cores each available, we did not parallelize our algorithms such that each experiment was run on a sin-

[2]More specifically, HLA-0201 MHC binders were predicted and the 6-mer from position 3 to 8, which are the T cell receptor contact residues for that MHC type, were extracted. For further details on binding prediction, see e.g. the textbook by Lund et al. [22].

[3]This library, written by Mehryar Mohri and Fernando C. N. Pereira and Michael D. Riley, can be accessed at www2.research.att.com/~fsmtools/fsm/ref.html

[4]We would only need to invert the DFA generated by positive selection. This is not completely trivial because the DFA is stored as an incomplete representation. However, it is easy to show that the size of an DFA M and its complement $\bar{M}$ can differ at most by factor $|\Sigma|$.

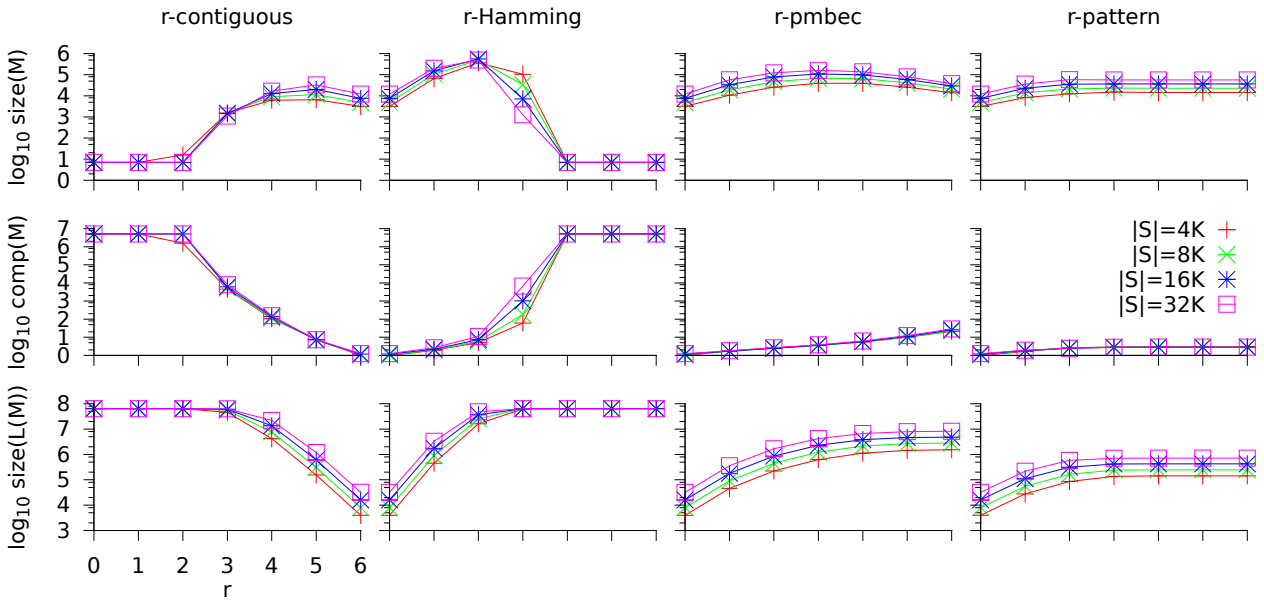

Figure 2: Empirical evaluation of the space efficiency of DFA-based representations of detector sets on proteomic data. The input sets S were generated using a human epitope prediction algorithm, i.e., $|\Sigma| = 20$ and $L = 6$, and positive selection was performed for each input dataset. Then the size $|D[S^+]| = |L(M)|$ of the generated repertoire was determined. Please note that the largest compression ratios achieved, i.e. for $\{0, 1, 2\}$-contiguous and $\{4, 5, 6\}$-Hamming are simply due to the fact that these parameters lead to the degenerate case $D[S^+] = (\Sigma_D)^L$ where all possible detectors match some input string.

gle core only. No other jobs were run on the machine at the same time such that cores were close to 100% occupied by the experiments.

The naive (i.e., exhaustive enumeration) positive selection algorithms were implemented according to the template pseudocode shown in Fig. 3. The only change for different detector types is the implementation of the matching function m_D. Although the outer loop that iterates over $(\Sigma_D)^L$ seems daunting, it is still quite feasible for our use case $|\Sigma| = 20, L = 6$ if S is not too large. For larger S, the runtime depends crucially on the matching probability between d and s (line 4). If this probability is for instance bounded from below by a constant p, then the inner loop terminates after $\mathcal{O}(1/p)$ iterations and the runtime becomes independent of $|S|$. In the worst case, if no detector can be found ($p = 0$), we have to iterate over all elements of S. This important role of the matching probability between d and s for naive implementations of repertoire models led to thorough theoretical investigations for the case of r-contiguous detectors [27].

Fig. 4 shows the results of our runtime analysis. For r-contiguous detectors, the runtime of the DFA approach is for most parameters superior to the naive approach, and even comes close to a Java implementation [31] of the fastest known dedicated algorithm for this case [8]. For r-Hamming, consistent with its theoretical difficulty, we find a less convincing speedup. Although it can still amount to some orders of magnitude for $r \leq 2$, it is not much better than exhaustive search for $r \geq 3$. However, the r-PMBEC version of the Hamming distance, which from a theoretical perspective is equally difficult, performed much better. Note that this difference between r-PMBEC and r-Hamming is *not* simply

1. $D \leftarrow \emptyset$
2. For all strings $d \in (\Sigma_D)^L$ do
3. For all strings $s \in S$ do
4. if $m_D(d, s) = +1$ then
5. insert d into D
6. continue with the next d
7. Output the cardinality of D.

Figure 3: Pseudocode for the "naive" exhaustive enumeration approach to positive selection used in our runtime analysis.

a consequence of the sizes of the resulting automata, which are in fact quite similar (Fig. 2). Instead, it is due to the sizes of the intermediate automata computed "on the way" to the final one. Lastly, the r-pattern detectors performed similar as in the r-PMBEC case, as already in the space complexity analysis.

Hence, while our empirical results partly reflect the theoretical hardness of the corresponding decision problems, especially for the Hamming case, it also emphasizes the obvious point that there can be enormous practical performance difference within these "hard" cases, rendering some practically tractable and putting others out of reach. Still, in general, the DFA implementation does seem preferable to the "classical" approach of generating detectors by exhaustive search or by random sampling [13]. For instance, we note that generating the "naive approach" data shown in Fig. 4 required about one week (171.7h) of CPU time on our Xeon server, whereas the DFA based data required only 1.5h.

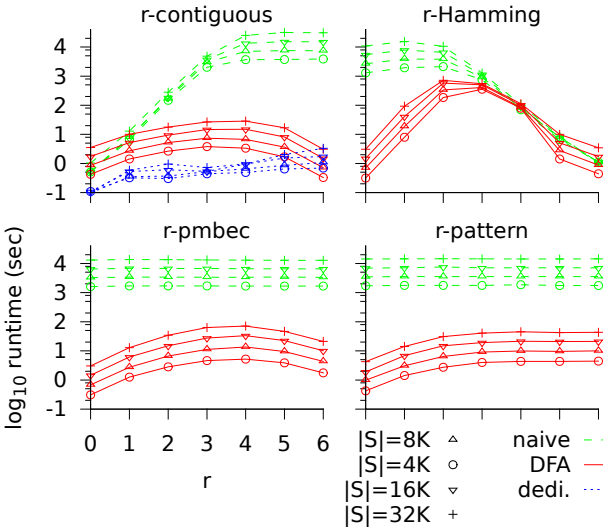

Figure 4: Empirical runtime analysis for repertoire counting after positive selection. We compare a naive implementation and the general DFA-based approach for $|S| \in \{4000, 8000, 16000, 32000\}$, $|\Sigma| = 20$ and $L = 6$. For r-contiguous detectors, we also run the dedicated *negative* selection implementation (dedi) by Textor [30] and convert the output to the positive selection count using the identity $|D[S^+]| = |\Sigma|^L - |D[S^-]|$.

6. DISCUSSION

Here we proposed a DFA based approach to implement lymphocyte repertoire models based on string patterns (detectors). This approach can be used to implement negative selection, positive selection, repertoire counting and sampling, classification, and allows on-line repertoire manipulation. To implement all these tasks for a certain detector type, the user only has to provide code that outputs a DFA describing the detectors that match a given string. We have explored the algorithmic and complexity theoretic issues of the DFA approach and implemented and tested it for four specific detector types using typical data from computational immunology.

DFAs are not the only conceivable formalism to support repertoire modeling – one could e.g. also use grammars or regular expressions. It is also not necessarily the best formalism – indeed, we have given an example where polynomial runtime is achievable but the DFA approach needs exponential time (Lemma 3). Nevertheless, DFAs subsume some other well-known string compression tools like tries or prefix DAGs and can therefore be expected to perform well when the string sets to be represented have at least some degree of regular structure (like for r-contiguous detectors). Because DFAs are a fundamental subject in computer science, there are also many highly efficient algorithms and software libraries for DFA processing available. This led us to choose this representation type over other options like regular expressions or grammars. An interesting perspective for future research would be to investigate whether nondeterministic FAs can be used to solve some repertoire modeling tasks approximatively that still require too many resources to be solved exactly. For instance, for all detector types considered here it is very easy to construct a polynomial size nondeterministic finite automaton (NFA) describing the language $D[S^+]$. Since a polynomial time algorithm exists to sample approximately at uniform from the language described by a given NFA [14], this could be an alternative approach for the repertoire sampling tasks.

The DFA based approach can of course not provide polynomial time repertoire modeling algorithms for Hamming distance and related detectors because these detector types lead to NP-hard decision problems [21]. Despite this, we still found substantial performance improvements compared to explicit detector generation at least for r-pattern and r-PMBEC detectors. It would now be very interesting to apply these detector types for anomaly detection tasks and compare their classification performance to r-contiguous and the related r-chunk detectors, which were so far the only detector types for which a thorough empirical comparison to other anomaly detection algorithms could be performed [30]. For space reasons, this could not be added to the present paper anymore. However, because the string lengths (6-10) [5], sample sizes (thousands) and alphabet sizes (<100) used in that study were similar to ours, we do expect this to be feasible for r-pattern detectors. The study also contained a protein anomaly detection problem for which the r-PMBEC detectors could be applied.

DFA-based string compression might be useful in other areas than AIS and computational immunology. For instance, many learning classifier systems use r-pattern-like matching rules, and may thus benefit from compressed rule representations when large rule sets are needed. Moreover, one could explore whether the DFA approach would allow evolutionary optimization algorithms to efficiently maintain and manipulate very large populations of candidate solutions, and whether that could improve optimization performance.

7. REFERENCES

[1] J. Balthrop, F. Esponda, S. Forrest, and M. R. Glickman. Coverage and generalization in an artificial immune system. In *Proc. of GECCO 2002*, pages 1045–1050, 2002.

[2] O. Bernardi and O. Giménez. A linear algorithm for the random sampling from regular languages. *Algorithmica*, 62(1-2):130–145, 2012.

[3] T. C. Butler, M. Kardar, and A. K. Chakraborty. Quorum sensing allows T cells to discriminate between self and nonself. *PNAS*, 110(29):11833–11838, 2013.

[4] J. J. A. Calis, R. J. D. Boer, and C. Kesmir. Degenerate T-cell recognition of peptides on MHC molecules creates large holes in the T-cell repertoire. *PLoS Comp Biol*, 8(3), 2012.

[5] D. L. Chao, M. P. Davenport, S. Forrest, and A. S. Perelson. A stochastic model of cytotoxic T cell responses. *J. Theor. Biol.*, 228:227–240, 2004.

[6] D. L. Chao, M. P. Davenport, S. Forrest, and A. S. Perelson. The effects of thymic selection on the range of T cell cross-reactivity. *Eur. J. Immunol.*, 35:3452–3459, 2005.

[5] More precisely, the actual sequence lengths were much larger and varied, but the anomaly detection was done using a sliding window of small size.

[7] T. U. Consortium. Ongoing and future developments at the universal protein resource. *Nucleic Acids Res.*, 39:D214–D219, 2011.

[8] M. Elberfeld and J. Textor. Negative selection algorithms on strings with efficient training and linear-time classification. *Theor. Comput. Sci.*, 412:534–542, 2011.

[9] S. Forrest, S. A. Hofmeyr, and A. Somayaji. Computer immunology. *Commun. ACM*, 40:88–96, 1997.

[10] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In *Proc. of the IEEE Symposium on Research in Security and Privacy*, pages 202–212. IEEE Computer Society Press, 1994.

[11] I. Hoof et al. NetMHCpan, a method for MHC class I binding prediction beyond humans. *Immunogenetics*, 61:1–13, 2009.

[12] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to automata theory, languages, and computation*. Pearson/Addison Wesley, 2007.

[13] Z. Ji and D. Dasgupta. Revisiting negative selection algorithms. *Evol. Comput.*, 15(2):223–251, 2007.

[14] S. Kannan, Z. Sweedyk, and S. Mahaney. Counting and random generation of strings in regular languages. In *Proc. of SODA 1995*, pages 551–557, 1995.

[15] J. Kim and P. J. Bentley. An evaluation of negative selection in an artificial immune system for network intrusion detection. In *Proc. of GECCO 2001*, pages 1330–1337. Morgan Kaufmann, 2001.

[16] Y. Kim, J. Sidney, C. Pinilla, A. Sette, and B. Peters. Derivation of an amino acid similarity matrix for peptide: MHC binding and its application as a Bayesian prior. *BMC Bioinformatics*, 10:394, 2009.

[17] A. Košmrlj, A. K. Jha, E. S. Huseby, M. Kardar, and A. K. Chakraborty. How the thymus designs antigen-specific and self-tolerant T cell receptor sequences. *PNAS*, 105(43):16671–16676, 2008.

[18] A. Košmrlj et al. Effects of thymic selection of the T-cell repertoire on HLA class I–associated control of HIV infection. *Nature*, 465:350–354, 2010.

[19] D. Kozen. Lower bounds for natural proof systems. In *Proc. of FOCS 1977*, pages 254–266, 1977.

[20] J. K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. *Information and Computation*, 185:41–55, 2003.

[21] M. Liśkiewicz and J. Textor. Negative selection algorithms without generating detectors. In *Proc. of GECCO 2010*, pages 1047–1054. ACM, 2010.

[22] O. Lund, C. Keşmir, M. Nielsen, C. Lundegaard, and S. Brunak. *Immunological Bioinformatics*. MIT Press, 2005.

[23] J. K. Percus, O. E. Percus, and A. S. Perelson. Predicting the size of the T-cell receptor and antibody combining region from consideration of efficient self-nonself discrimination. *PNAS*, 90(5):1691–1695, 1993.

[24] A. S. Perelson and G. F. Oster. Theoretical studies of clonal selection: minimal antibody repertoire size and reliability of self-non-self discrimination. *J. Theor. Biol.*, 81(4):645–670, 1979.

[25] D. Revuz. Minimisation of acyclic deterministic automata in linear time. *Theor. Comput. Sci.*, 92(1):181–189, 1992.

[26] D. J. Smith, S. Forrest, D. H. Ackley, and A. S. Perelson. Using lazy evaluation to simulate realistic-size repertoires in models of the immune system. *Bull. Math. Biol.*, 60:647–658, 1998.

[27] T. Stibor. Foundations of r-contiguous matching in negative selection for anomaly detection. *Nat. Comput.*, 8:613–641, 2009.

[28] T. Stibor, P. Mohr, J. Timmis, and C. Eckert. Is negative selection appropriate for anomaly detection? In *Proc. of GECCO 2005*, pages 321–328. ACM, 2005.

[29] J. Textor. *Search and Learning in the Immune System: Models of Immune Surveillance and Negative Selection*. PhD thesis, University of Lübeck, Germany, 2011.

[30] J. Textor. A comparative study of negative selection based anomaly detection in sequence data. In *Proc. of ICARIS 2012*, volume 7597 of *LNCS*, pages 28–41. Springer, 2012.

[31] J. Textor. Efficient negative selection algorithms by sampling and approximate counting. In *Proc. of PPSN 2012*, volume 7492 of *LNCS*, pages 32–41. Springer, 2012.