Radboud University Nijmegen

Faculty of Science

Finite-State Automata as T-Cell Receptor Repertoires

Weighted FSAs for TCR Sequences constructed through VDJ Recombination

Thesis MSc Computing Science

Supervisor:
F.E. Buytenhuijs

Supervisor:
G. Schröder

1st reader:
J.C. Textor

2nd reader:
D.A. van Leeuwen

Author:
F.S. Slijkhuis

2-11-2022

1 Introduction

Vertebrates rely on the adaptive immune system when encountering foreign agents able to cause diseases, which are called pathogens [2]. The adaptive immune system uses designated cells to recognize pathogens: immune cells. B- and T-cells, which are types of immune cells, bind to these intruders using receptor proteins on their surfaces, in order to eliminate them. In B-cells, these receptors are called immunoglobulins, and for T-cells, they are simply referred to as T-Cell Receptors (TCRs) [17]. A single receptor can only bind to a very small variety of pathogens, so in order for the adaptive immune system to recognize as many pathogens as possible, a highly diverse repertoire of receptors is essential [13].

The high diversity in the receptor repertoire is a result of a process called "V(D)J recombination". In V(D)J recombination, a receptor of an immature B- or T-cell is created by rearranging gene segments in a seemingly random fashion. V(D)J recombination creates a unique sequence consisting of nucleotides, the genetic 'building blocks' of DNA and RNA, which encodes the immune cell receptor. Because of this process, a highly diverse repertoire of receptors can be generated from a relatively small amount of DNA [13].

Through recent advances in repertoire sequencing, more specifically high-throughput immune repertoire sequencing (RepSeq), it has been made possible to analyse the diversity of entire T-cell receptor repertoires by observing their nucleotide-sequences [15, 20]. This has led to entire datasets consisting of T-cell receptor data, which gives us statistical insight into the diversity of TCR repertoires [12]. From these datasets, tools and algorithms have emerged for analysis of immune repertoires and artificial generation of sequences using the statistical properties of existing repertoires, such as [4, 5, 8, 9, 12, 16, 19, 21].

Many of the tools and algorithms presented in previous work make use of inferred statistical properties of receptor datasets in order to build stochastic models of V(D)J recombination, in the form of a joint distribution or 'dependency structure'. This model can then be used to find the recombination probability of a specific sequence. Such 'dependency structure-dependent models', of which IGoR [12] and OLGA [19] are examples, focus on improving the method for finding the generation probability, but in most cases, the same stochastic model is used.

The use of a joint distribution or dependency structure for finding the recombination probability of a sequence is in itself limited, as it becomes difficult to combine the models with other probabilistic processes found in the adaptive immune system, such as thymic selection [11]. This makes dependency structure-dependent models unsuitable for use in calculations with other models. In this work, we present a weighted Finite-State Automaton (FSA) for immune sequence analysis and sequence generation, which functions as a stochastic model of V(D)J recombination of T-cell receptors. The weighted FSA is able to calculate the probability of a given T-cell receptor sequence. The goal of the resulting FSA is to be used in large-scale models of the adaptive immune system, where calculations with other models are necessary. One such example of a large-scale model is the "Artificial Immunological Intelligence"-project [1], for which this work is carried out.

2 Background

In this section, a background is provided on the topics related to this work. First, we shall provide an overview of V(D)J recombination, which is the process being modeled by the weighted FSA. Next, we provide a formal definition of the weighted FSA used in this work. Lastly, we provide an introduction to dependency structures, which will be used to create our FSAs.

2.1 V(D)J recombination

V(D)J recombination works by combining gene segments, which are sequences of nucleotides. Nucleotides are organic molecules, functioning as the basic structural units of DNA and RNA. Four different nucleotides exist, and they are characterised by the letters A, C, G and T.

In V(D)J recombination, a specific set of Variable (V), Joining (J), and Diversity (D) gene segments is used. Here, a single V-gene and a single J-gene, and, in some cases, a D-gene, are combined in order to create a unique receptor sequence [13].

Our focus is only on human T-cell receptors. Most T-cell receptors consist of two chains, the alpha (α) chain (TCRa) and the beta (β) chain (TCRb) [6]. The alpha chain is created using VJ recombination, in which a V-gene is joined with a J-gene. These two genes are randomly selected. Next, the connection between the two selected genes is opened, which allows for the deletion and insertion of nucleotides. In nucleotide deletion, an existing nucleotide at the end of the V-gene or the beginning of the J-gene might be deleted. Another possible operation is the insertion of P-nucleotides, which are inverted repetitions of the complement of the gene. For example, when a gene ends in GCT, the complement of that gene with length 3 is CGA. This complement is inverted, resulting in the nucleotide sequence AGC. These P-nucleotides can only exist in between the V- and J-gene, which, in more formal terms, are the 3'-end of the V-gene, the end of the gene segment, and the 5'-end of the J-gene, the beginning of the gene segment. In most dependency structure-dependent models, where P-nucleotide insertions and deletions are considered to be separate events, nucleotide deletion can only occur when there are no P-nucleotide insertions. Because of this, we often refer to P-nucleotide insertions as 'negative deletions'. In between the two gene segments, nucleotide insertions can also occur, which causes a nucleotide to be inserted. The result of all these operations is a highly variable alpha-chain of the T-cell receptor [18].

In many ways, the beta chain is similar to the alpha chain of the TCR, except that it uses VDJ recombination instead of VJ recombination. In VDJ recombination, a D-gene is first joined with a J-gene, after which the result is joined with a V-gene. P-nucleotides, insertions and deletions can occur between the 3'-end of the V-gene and the 5'-end of the D-gene, and the 3'-end of the D-gene and the 5'-end of the J-gene. In the end, this results in an even greater variability in the resulting gene [14, 18].

2.2 Weighted Finite-State Automata

We create a Weighted Finite-State Automaton [3, 7], which can be defined as follows:

A weighted FSA $F = (\Sigma, Q, R, A, F)$ over a set of weights W consists of

1. A finite set of input symbols, the alphabet Σ.
2. A finite set of states, Q.
3. A set of initial states, $R \subseteq Q$.
4. The state-transition relation A, which contains all transitions between states in Q characterized by a source state and destination state $q, q \in Q$, a weight $w \in W$ and symbol(s) from Σ; $A \subseteq Q^2 \times (\Sigma \cup \{\epsilon\}) \times W$, where ϵ is the empty string.
5. A set of final states, $F \subseteq Q$.

For better readability and for our FSAs to be easier to implement, we make use of an adapted convention for representing weighted finite-state automata. In this work, we refer to the FSAs represented in this manner as 'TCR-FSAs'. In our convention, we label the states in Q using symbols from Σ, and, rather than using symbols from Σ in the state-transition function, we use them to label the destination state. Additionally, we use a single starting state and a single final state which are not labeled using symbols from Σ. Our alphabet Σ consists of nucleotides $= (A, C, G, T)$ and the empty string, ϵ. Therefore, we end up with an FSA which has nucleotides as states, and weights on the transitions between states.

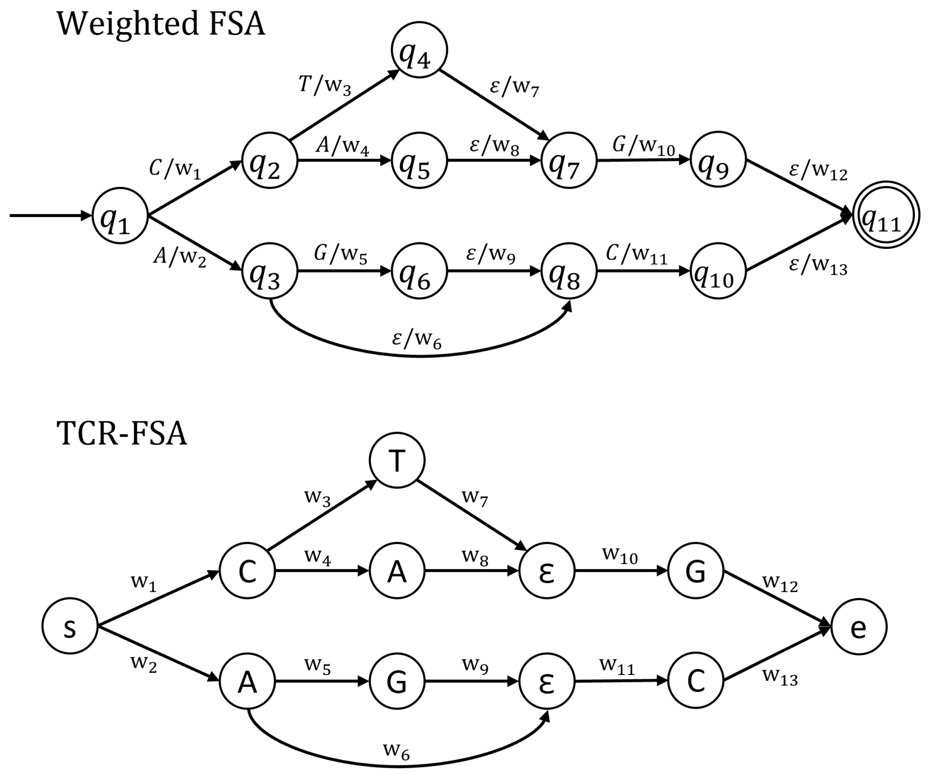


Figure 1: Set of nucleotide sequences represented through a regular weighted FSA (**top**), and the same set of nucleotide sequences represented using the convention used throughout this work (**bottom**). For both FSAs, the probability of a sequence is the product of all probabilities on the edges of its path from start to finish. Valid sequences with their respective recombination probabilities are C, T, G ($P_{C,T,G} = w_1w_3w_7w_{10}w_{12}$); C, A, G ($P_{C,A,G} = w_1w_4w_8w_{10}w_{12}$); A, C ($P_{A,C} = w_2w_6w_{11}w_{13}$) and A, G, C ($P_{A,G,C} = w_2w_5w_9w_{11}w_{13}$).

In the TCR-FSA, any valid nucleotide sequence corresponding to a T-cell receptor is represented by all the nodes in a path such that s is the first state in the path and e is the final state in the path. The probability of this nucleotide sequence is then P_{recomb}, which is the product of all weights alongside all of the transitions within the path. In the case of non-determinism, where the same sequences might be represented by multiple paths, a probability of a given nucleotide sequence is the sum of all paths which represent this nucleotide sequence. Fig. 1 shows an example of a stochastic model of nucleotide sequences represented through a weighted FSA according to the previously provided definition (top) and the same stochastic model represented through our convention, the

TCR-FSA (bottom). Both models contain the same probabilities and represent the same set of nucleotide sequences. The TCR-FSA has two empty states, which we use to represent edges containing an empty string.

2.3 Dependency structures

Our goal is to build a weighted FSA which assigns the same recombination probabilities to sequences as IGoR, such that we can use it as a replacement of the existing stochastic models of V(D)J recombination. For this to work, we require the same dependencies to be present in the FSA as in these models. Therefore, we translate dependency structures for both the TCRa and TCRb from previous work into a weighted TCR-FSA.

Let us first look at the simpler stochastic model of the two, the dependency structure for the alpha-chain of the T-cell receptor [12]:

$$\begin{aligned} P^{\alpha}_{\text{recomb}} =& P(\mathrm{V},\mathrm{J}) \times P(\mathrm{delV}|\mathrm{V}) \times P(\mathrm{delJ}|\mathrm{J}) \\ & \times P(\mathrm{insVJ} = n) \times \prod_{i=1}^{n} P(n_i|n_{i-1}) \end{aligned} \tag{1}$$

The first term in the dependency structure, $P(\mathrm{V},\mathrm{J})$ is the probability of the choice of a V-gene and a J-gene in the alpha-chain of the T-cell receptor. The next terms, $P(\mathrm{delV}|\mathrm{V})$ and $P(\mathrm{delJ}|\mathrm{J})$, are the probabilities for the number of deletions in the V-gene and the J-gene, respectively. This number can either be positive, negative, or zero. In the case of a negative number of deletions, this corresponds to P-nucleotide insertions, insertions of the reverse complement of the gene-segment. A zero number of deletions means that no nucleotides are deleted from the gene segment. As we can see, the number of V-gene deletions is dependent on the choice of V-gene, and J-gene deletion is dependent on the choice of J-gene. The next term, $P(\mathrm{insVJ})$, corresponds to the probability of the number of insertions between the V- and J-gene. This is a non-negative integer, where the number 0 means that we have no nucleotide insertions. When the number of insertions is greater than 0, we have another term, $\prod_{i=1}^{n} P(n_i|n_{i-1})$, describing the probabilities of the nucleotide choice used for insertions. The choice of nucleotide is dependent on the previous nucleotide in the sequence. The total probability is equal to the product of all probabilities of the nucleotide choices for insertions. For n_1, the previous nucleotide is n_0, which is the last nucleotide of the sequence before the insertion. In the case of VJ-insertion, this would be the nucleotide on the 3'-side of the V-gene. As an example of VJ-insertion, let us assume we have 3 insertions (insVJ$= 3$), that the last nucleotide in between V and J is A (after deletion), and that we insert the nucleotides C, A and T. The terms corresponding to nucleotide insertions then become $P(\mathrm{insVJ} = 3) \times P(C|A) \times P(A|C) \times P(T|A)$.

The dependency structure for the beta-chain is very similar to that of the alpha-chain. It is given by the following equation [12, 14]:

$$\begin{aligned} P^{\beta}_{\text{recomb}} =& P(\mathrm{V},\mathrm{D},\mathrm{J}) \times P(\mathrm{delV}|\mathrm{V}) \\ & \times P(\mathrm{insVD} = n) \times P(\mathrm{delDl},\mathrm{delDr}|\mathrm{D}) \\ & \times P(\mathrm{insDJ} = m) \times P(\mathrm{delJ}|\mathrm{J}) \\ & \times \prod_{i=1}^{n} P(n_i|n_{i-1}) \times \prod_{i=1}^{m} P(m_i|m_{i-1}) \end{aligned} \tag{2}$$

We shall only cover the terms which are different from those in the dependency structure for the alpha-chain. The first term, $P(\mathrm{V}, \mathrm{D}, \mathrm{J})$[1], corresponds to the probability of the choice of V-, D- and J-gene. For the beta-chain, we have two insertions, one between V and D, and one between D and J. D-gene deletion, $P(\mathrm{delDl}, \mathrm{delDr}|\mathrm{D})$, is a bit different from the other deletions, as we can have deletions on both sides of the D-gene. In this case, delDl corresponds to deletions to the 5'-side of the D-gene, or the 'left side', and delDr corresponds to deletions to the 3'-side of the D-gene, the 'right side'. Again, we can have both negative, positive, and zero deletions, corresponding to P-nucleotide insertions, nucleotide deletions, and no deletions at all, respectively. delDl and delDr are jointly distributed, and are both dependent on the choice of D-gene.

Having provided the dependency structures, our goal is now to translate the dependencies into the FSA structure, which we will do in the following section.

3 Methods

Our goal is to construct a stochastic model of V(D)J recombination using weighted TCR-FSAs. We will do this for the T-cell receptor, which has an alpha- (TCRa) and a beta-chain (TCRb).

3.1 External sources

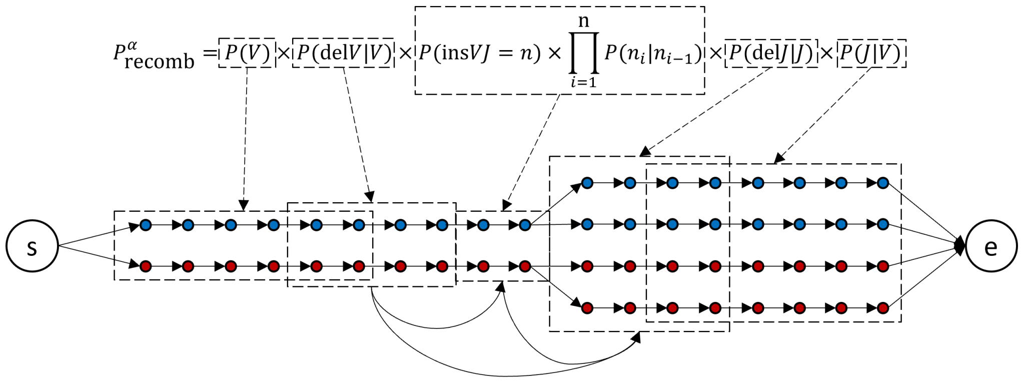
Two T-cell receptor FSAs will be created following the previously provided definition: one for the alpha-chain of the T-cell receptor and one for the beta-chain of the T-cell receptor, TCRa and TCRb. To do this, we make use of Python and an open-source package for creating graphs and networks; networkX [10]. NetworkX is suitable for our purpose, since it allows us to create directed graphs with labels for both nodes and edges (states and transitions). It also provides functions useful for graph analysis, which will be useful in sequence analysis and sequence generation using our TCR-FSA. When building the FSAs using networkX, we use the same convention as provided earlier in this document, with nucleotides as state labels instead of labels along transitions. The probabilities within the TCR-FSAs come from existing human TCRa and TCRb-datasets, which are included in IGoR's software.

3.2 FSA creation

Let us consider the dependency structures from section 2.3. In order to translate the dependency structures into FSAs, we try to convert each term of the dependency structure into a 'layer' of the FSA. A layer will then contain all of the probabilities relevant to the respective term in the dependency structure. Apart from the relevant probabilities, the layer will also contain all of the nucleotides corresponding to this part of the nucleotide sequence. This way, we can preserve structure and consistency across the TCR-FSA more easily. Let us illustrate this using the alpha-chain of the TCR as an example, with the dependency structure from eq. 1. In order to allow for a layered FSA-structure, we first factor the joint probability of the V- and J-gene choice: $P(\mathrm{V}, \mathrm{J}) = P(\mathrm{V})P(\mathrm{J}|\mathrm{V})$. We then reorder the dependency structure, such that the order of terms corresponds to the reading order of the nucleotide sequence (simply from left to right, starting with the V-gene with V-gene deletion, then VJ-insertion, and finally J-gene selection with J-gene deletion). Fig. 2 illustrates how we can then translate the dependency structure into a TCR-FSA structure for the alpha-chain.

Fig. 2 shows how the resulting FSA can 'branch' when dealing with dependencies (represented by different colours). There is no notion of memory in the FSA, so each

[1]Note that in the IGoR-paper, V is assumed to be independent of D and J for the TCRb, but the data used in IGoR uses the joint probability V, D, J, which is why we assume a joint probability.



$P^{\alpha}_{\text{recomb}} = P(V) \times P(\text{del}V|V) \times P(\text{ins}VJ = n) \times \prod_{i=1}^{n} P(n_i|n_{i-1}) \times P(\text{del}J|J) \times P(J|V)$
s
e

Figure 2: Translation of the dependency structure of the TCRa to a TCR-FSA structure, simplified. From the left to right, the FSA represents all possible TCR-sequences, with nucleotides as nodes. Keep in mind that deletions can either delete parts of a gene, but can also add nucleotides to a gene, whenever the number of deletions is negative (P-nucleotide insertion). The arrows on the bottom represent ways of skipping nucleotide nodes, which can be caused by non-negative deletions of parts of the V- and/or J-gene, or because of the absence of insertions.

choice made in the FSA which is relevant later on must be 'remembered' by adding a separate branch. In the illustration, we have two V-genes, a blue one and a red one. As we can see from the eq. 1, the choice of J-gene is dependent on the choice of V-gene. Therefore, we must remember the choice of V-gene, which we do by branching the FSA. There are two J-genes in this simplified representation, and because of the dependency on V, we end up with four branches when going into the J-gene layer.

Let us now go into the FSA structure in more detail. We will consider the alpha- and beta-chains separately, which makes it easier to explain the design choices for each of them. However, they both follow the same general structure, as explained before. We will go over the FSA structures layer by layer. For readability purposes, we will refer to dependencies on specific gene segment choices as simply the name of the gene, both in the text and in the illustrations, i.e. $P(J = J_a|V = V_a)$ becomes $P(J = J_a|V_a)$.

3.2.1 Alpha-chain

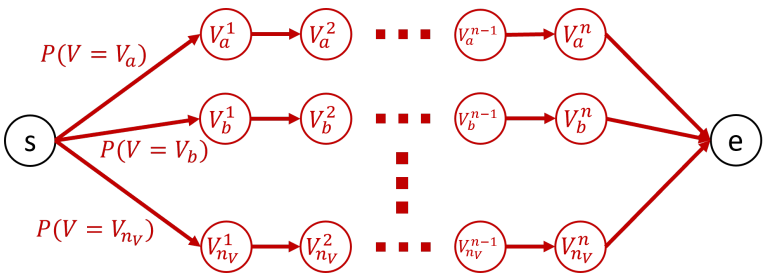
When building the TCR-FSA for the alpha-chain, we do so layer by layer. Therefore, this section also explains the structure by covering each layer separately. We have provided simplified illustrations for each part of the TCR-FSA. We consider eq. 1, but, just like in the example provided before, we rewrite it such that the order of terms corresponds to the reading order of the nucleotide sequence. We also factor the joint distributions. The dependency structure then becomes:

$$\begin{aligned} P^{\alpha}_{\text{recomb}} =& P(\text{V}) \times P(\text{delV}|\text{V}) \\ & \times P(\text{insVJ} = n) \times \prod_{i=1}^{n} P(n_i|n_{i-1}) \\ & \times P(\text{J}|\text{V}) \times P(\text{delJ}|\text{J}) \end{aligned} \tag{3}$$

Starting state and ending state As per our definition of the TCR-FSA (see sec. 2.2), we require the FSA to have a starting state and ending state, s and e. Every valid TCR-sequence is represented by a path from the beginning of the FSA to the end of the FSA.

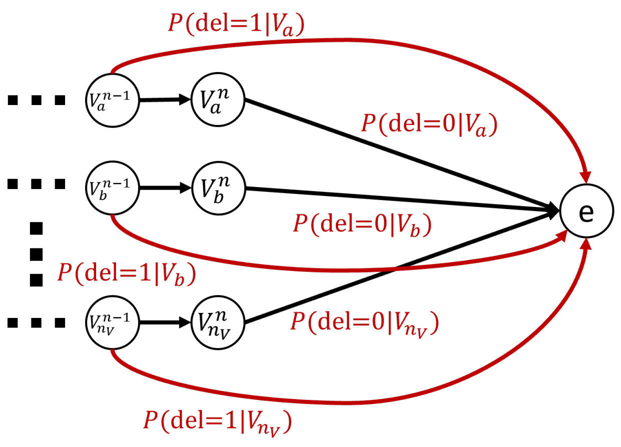
Therefore, we require these states to be present. We have decided to draw both states already, and then build up the FSA structure in between them. This allows each of the partial TCR-FSAs we draw here to actually be valid according to our definition. In our illustrations, each transition (edge) has a probability of 1, unless stated otherwise:

V-gene selection Let us again consider the restructured dependency structure from eq. 3. The first term is $P(\mathrm{V})$, which corresponds to the choice of V-gene. Therefore, our first layer will contain a branch for each V-gene, containing all of the nucleotides in the respective V-gene, read from left to right (5'-side to 3'-side). Going into one of the branches in this layer from the starting state then requires a probability of $P(\mathrm{V})$, depending on the V-gene. Let us now illustrate the V-gene selection layer, which we shall draw in between the starting and the ending state. Let us suppose we have the V-genes $V_a, V_b, ..., V_{n_V}$, each with nucleotides $1, 2, ..., n-1, n$. Each V-gene has its own branch, with probability $P(\mathrm{V})$ to enter the branch. The result is shown in the following illustration:

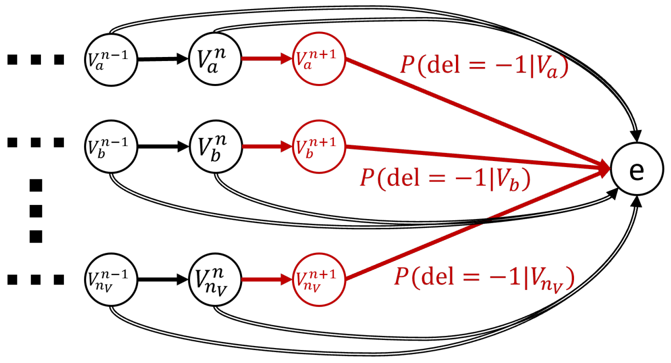


In reality, for the alpha-chain of TCR, we have a total of 103 V-genes to pick from. Therefore, without simplifying the FSA as in our illustrations, we would have 103 branches in the V-gene selection layer, with 103 probabilities between the starting state and the first nucleotide of each branch. In the actual TCR-FSA, the length of each branch differs, because V-genes do not have a fixed length in terms of the number of nucleotides.

V-gene deletions The next term in our dependency structure is that of V-gene deletion. As we have seen before, the term $P(\mathrm{del}V|V)$ corresponds to the number of deletions of the V-gene, which can be positive, negative and zero. Let us first consider non-negative deletions. Positive deletions give us the option to exit the V-gene before reaching the end of the V-gene. For now, we can consider an exit out of the V-gene to be a connection to the ending state, since we have no layers coming after the V-gene selection layer yet. In the case of 0 deletions, we simply use the same connection from the last nucleotide of all V-genes to the ending state, but with a probability for 0 deletions. For our illustration, let us assume that we can only have 1 or 0 deletions. Let us also only focus on the last part of the V-gene. We then add the following connections to add the non-negative deletions in the TCR-FSA:



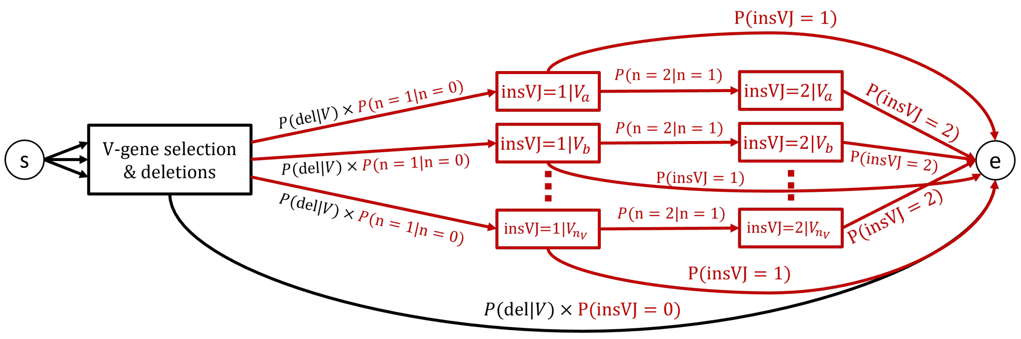
Positive deletions add P-nucleotides to the end of the V-gene, which we represent using nucleotide states. These P-nucleotides should exist between the V-gene selection layer and the ending state, but the connections we have drawn for non-negative deletions remain. When we do not exit the V-gene through non-negative deletion, we simply keep traversing further along the branch, into the P-nucleotides. This is because we cannot have both negative and positive deletions. For our illustration, let us now assume that we can have only a single positive deletion, the nucleotide $n+1$. We again focus on the end of the V-gene, and we omit the previously added probabilities (along the doubly-lined arrows) for readability purposes. The addition of a single P-nucleotide insertion for each V-gene with their probabilities then looks as follows:



In reality, we allow up to 16 positive deletions in the TCR-FSA for the alpha-chain, and up to 4 negative deletions (-4 deletions), since these are also the maximums used in IGoR.

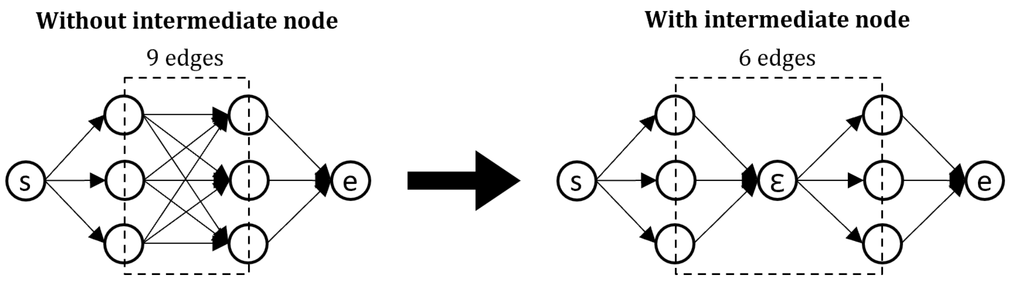
VJ-insertion Next, let us focus on the VJ-insertion layer. In the dependency structure, it is represented by two terms: $P(\text{insVJ} = n) \times \prod_{i=1}^{n} P(n_i|n_{i-1})$. The first term corresponds to the number of insertions between the V- and J-genes, while the second term corresponds to the probability of the nucleotide being inserted, which depends on the previous nucleotide in the sequence. The previous nucleotide can be an inserted nucleotide, but in the case of the first insertion, it must be a P-nucleotide or a nucleotide from the V-gene (depending on the number of deletions in the V-gene). The second term is a product term, such that it will be repeated for the number of insertions, determined by the first term.

For VJ-insertion, we can also have 0 insertions. In this case, we need to be able to exit the previous layers, the V-gene selection and deletion layers, and go straight to the end of the FSA, bypassing the insertion layer. A connection between the end node and the V-gene selection and deletion layers has already been added, but we need to multiply the probability along this connection with $P(\text{insVJ} = 0)$. For each insertion in the insertion layer, we have a possibility of exiting the insertion layer. This probability is equal to the number of insertions we have done, $P(\text{insVJ})$. Between insertions, we only have the probability of inserting a specific nucleotide, one of A, C, G, T, depending on the previous nucleotide in the sequence. Therefore, this probability is equal to $P(n_i|n_{i-1})$. For our illustration, let us assume we only have two VJ-insertions. When collapsing our V-gene selection and V-gene deletion layers for readability purposes, our illustration now looks as follows:

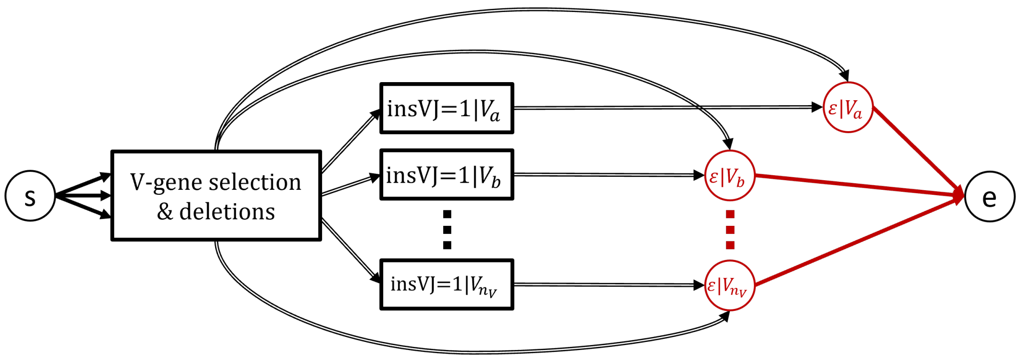


Please note that we still keep the V-gene dependency in the VJ-insertion layer, and that we still have multiple branches here. We do this because we need to insert the J-gene selection layer later, which depends on the choice of the V-gene. In reality, the FSA allows up to 39 VJ-insertions.

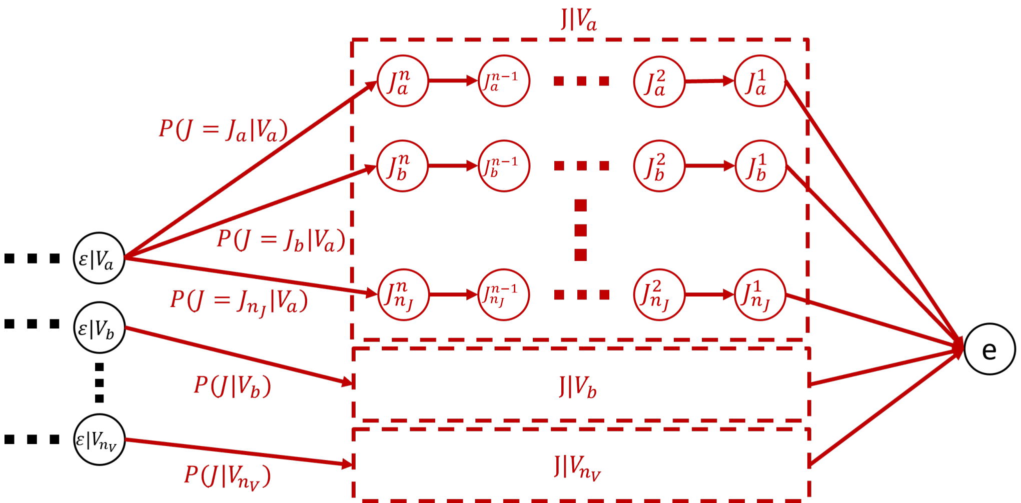
Intermediate layer In order to improve readability of the FSA and reduce the number of connections within the FSA, we let the connections between the previous layers and the upcoming layers, the J-gene selection and deletion layers, go through a layer with empty symbols (ϵ): the intermediate layer. We treat J-gene deletion similarly to V-gene deletion. However, we have seen that J-gene deletion takes place at the 5'-end side of the J-gene (the left side), which would mean that for every current branch in the FSA, we require a connection from the VJ-insertion layer and V-gene deletion layer to each possible entry point of the J-gene deletion layer. This causes an increase in connections, which is why we have decided to add an intermediate layer to the FSA structure. Below, an example is shown of how an intermediate node might reduce the number of connections between two layers of nodes.



The intermediate layer contains nodes without any nucleotides (empty nodes), but still preserves the V-gene dependency. Therefore, we have a single intermediate node for each branch in the FSA. We place the intermediate layer in between the VJ-insertion layer and the ending state, and all of the connections between the previous layers and the ending state will now go to the intermediate layer, preserving the V-gene dependency. Illustrated, and when we omit the previously added probabilities (along the doubly-lined arrows), the intermediate layer looks like this:

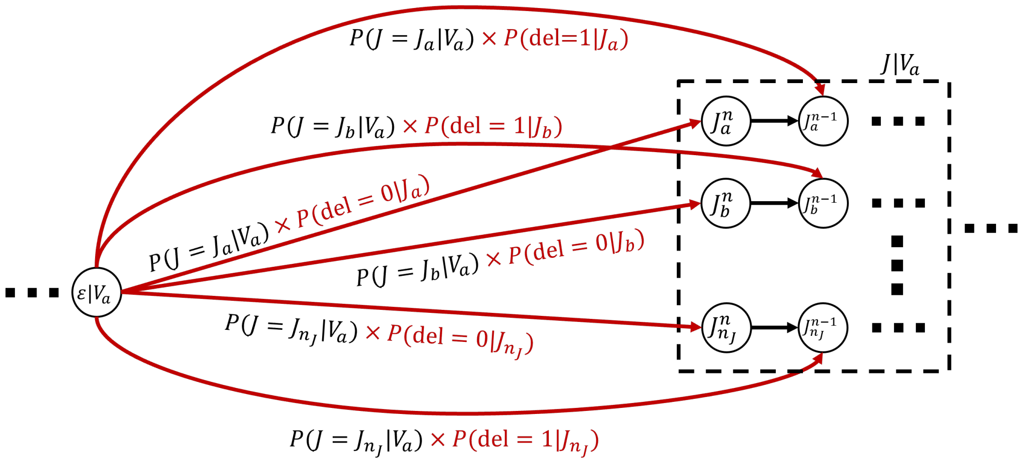


J-gene selection After the intermediate layer, we add the J-gene selection layer. It is very similar to the V-gene selection layer, except that there is now a dependency on the choice of V-gene. We kept the V-gene dependency across the entire J-gene selection layer, even though it is not necessary. In our illustration, where we do not display the part of the FSA which comes before the intermediate layer, we have highlighted a single 'branch' of the J-gene selection layer, but for readability purposes, we have collapsed the other branches into a rectangle:

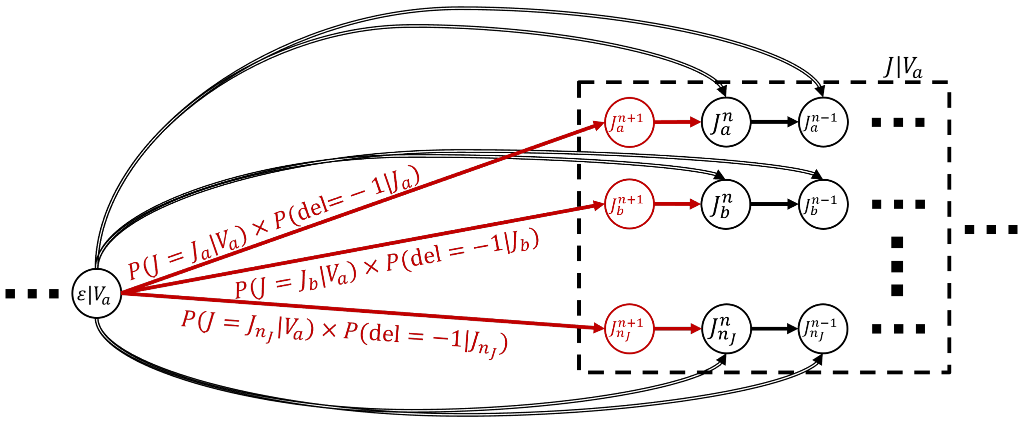


In the actual TCR-FSA, we have 68 J-genes for the alpha-chain of the TCR. This gives us $103 \times 68 = 7004$ branches in the J-gene selection layer.

J-gene deletion The last layer which we will add to the FSA, the J-gene deletion layer, is similar to the V-gene deletion layer, except that it is reversed. J-gene deletion occurs on the 5'-side of the gene, so we have to modify the FSA in between the intermediate layer and the V-gene selection layer. Again, let us first consider non-negative deletions. Instead of exiting the gene before the end of the gene, as was the case with V-gene deletion, positive J-gene deletion allows us to enter the J-gene selection layer at different points in the J-gene. As we have seen before, we enter the J-gene selection layer from the intermediate layer. In the case of 0 deletions, we simply add a probability to the existing probability for entering the J-gene selection layer. For our illustrations, let us assume that we can only have 1 or 0 deletions. We now only focus on the intermediate layer and J-gene selection layer, and we only show a single branch of the FSA, the V_a-dependency. J-gene deletion then looks as follows:


$P(J = J_a|V_a) \times P(\text{del}=1|J_a)$
$J|V_a$
$P(J = J_b|V_a) \times P(\text{del} = 1|J_b)$
$P(J = J_a|V_a) \times P(\text{del} = 0|J_a)$
$P(J = J_b|V_a) \times P(\text{del} = 0|J_b)$
$\varepsilon|V_a$
J_a^n
J_a^{n-1}
J_b^n
J_b^{n-1}
$J_{n_J}^n$
$J_{n_J}^{n-1}$
$P(J = J_{n_J}|V_a) \times P(\text{del} = 0|J_{n_J})$
$P(J = J_{n_J}|V_a) \times P(\text{del} = 1|J_{n_J})$

Let us now look at the negative deletions, the P-nucleotide insertions. Again, they are the same as with negative V-gene deletions, except that they are on the 5'-side of the J-gene. Therefore, we simply add these nodes at the beginning of the gene, and add connections to them from the intermediate layer. For our illustration, let us again assume that we only have a single positive deletion, the nucleotide $n+1$. We again focus only on the intermediate layer and the J-gene selection layer of a single branch of the FSA, and we have omitted the previously added probabilities (along the doubly-lined arrows) for readability purposes. The addition of a single P-nucleotide insertion for each J-gene with their probabilities then looks as follows:


$J|V_a$
$P(J = J_a|V_a) \times P(\text{del}=-1|J_a)$
$P(J = J_b|V_a) \times P(\text{del} = -1|J_b)$
$P(J = J_{n_J}|V_a) \times P(\text{del} = -1|J_{n_J})$
$\varepsilon|V_a$
J_a^{n+1}
J_a^n
J_a^{n-1}
J_b^{n+1}
J_b^n
J_b^{n-1}
$J_{n_J}^{n+1}$
$J_{n_J}^n$
$J_{n_J}^{n-1}$

In the actual TCR-FSA for the alpha-chain, we allow up to 18 positive deletions and up to 4 negative deletions (-4 deletions) of the J-gene.

Summary We have now covered all of the terms in the dependency structure for the alpha-chain of the TCR, finalizing the construction of the TCR-FSA structure for the TCRa. In Fig. 3, an overview of the TCR-FSA for the alpha-chain is shown.

3.2.2 Beta-chain

The beta-chain of the TCR is the same as the alpha-chain, with the added possibility of a D-gene being present, and there being two openings between genes allowing for insertion: VD-insertion and DJ-insertion. We can easily see this difference when we look at the reordered dependency structure, where, just like with the dependency structure for the alpha-chain, we have factored out the joint probabilities:

$$\begin{aligned} P^{\beta}_{\text{recomb}} =& P(\text{V}) \times P(\text{delV}|\text{V}) \\ & \times P(\text{insVD} = n) \times \prod_{i=1}^{n} P(n_i|n_{i-1}) \\ & \times P(\text{D}|\text{J}, \text{V}) \times P(\text{delDl}|\text{D}) \times P(\text{delDr}|\text{delDl}, \text{D}) \\ & \times P(\text{insDJ} = m) \times \prod_{i=1}^{m} P(m_i|m_{i-1}) \\ & \times P(\text{J}|\text{V}) \times P(\text{delJ}|\text{J}) \end{aligned} \tag{4}$$

Since we have translated many of the terms from eq. 4 already for the TCR-FSA of the alpha-chain, we will not go over all of them. The TCR-FSA for the beta-chain will be initialized in the same way as the FSA for the alpha-chain, so we start with a starting state and an ending state, and we will build up the FSA in between these states. The first part of the dependency structure, consisting of V-gene selection and V-gene deletion, is the same as with the alpha-chain, so it entails the same FSA structure. For the beta-chain, we have 89 V-genes to choose from, and we allow up to 16 positive deletions and 4 negative deletions of the V-gene. After this, we have VD-insertion, which is the same as VJ-insertion in terms of dependencies. We then add an intermediate layer with a V-gene dependency, the VD-intermediate layer. Fig. 4 shows the overall structure of the TCR-FSA for the beta-chain. In this section, we will assume we already have built the FSA-structure up until the D-gene selection and deletion layers. From this point, the dependency structure differs from that of the alpha-chain, so we will continue our explanation per layer of the FSA structure from here.

D-gene selection Let us first consider D-gene selection without deletion. In many ways, it is similar to J-gene selection. We can enter the D-gene from the VD-intermediate layer. When looking at the term for D-gene selection in the dependency structure, $P(\text{D}|\text{J}, \text{V})$, we see that there is a dependency on both the choice of V-gene and the choice of J-gene. Even though we do not have J-gene selection in our FSA at this point, we still create a branch for it based on this dependency. Therefore, we now have a separate instance of D-gene selection for each combination of V- and J-gene. Each D-gene also has its own branch, and for the beta-chain of the TCR, we can have 3 D-genes and 15 J-genes, leading to a total of $3 \times 15 \times 89 = 4005$ branches in the D-gene selection layer. Going into a branch corresponding to a D-gene already determines the J-gene we pick later in the FSA. J-gene selection also has a dependency on the V-gene, so we keep

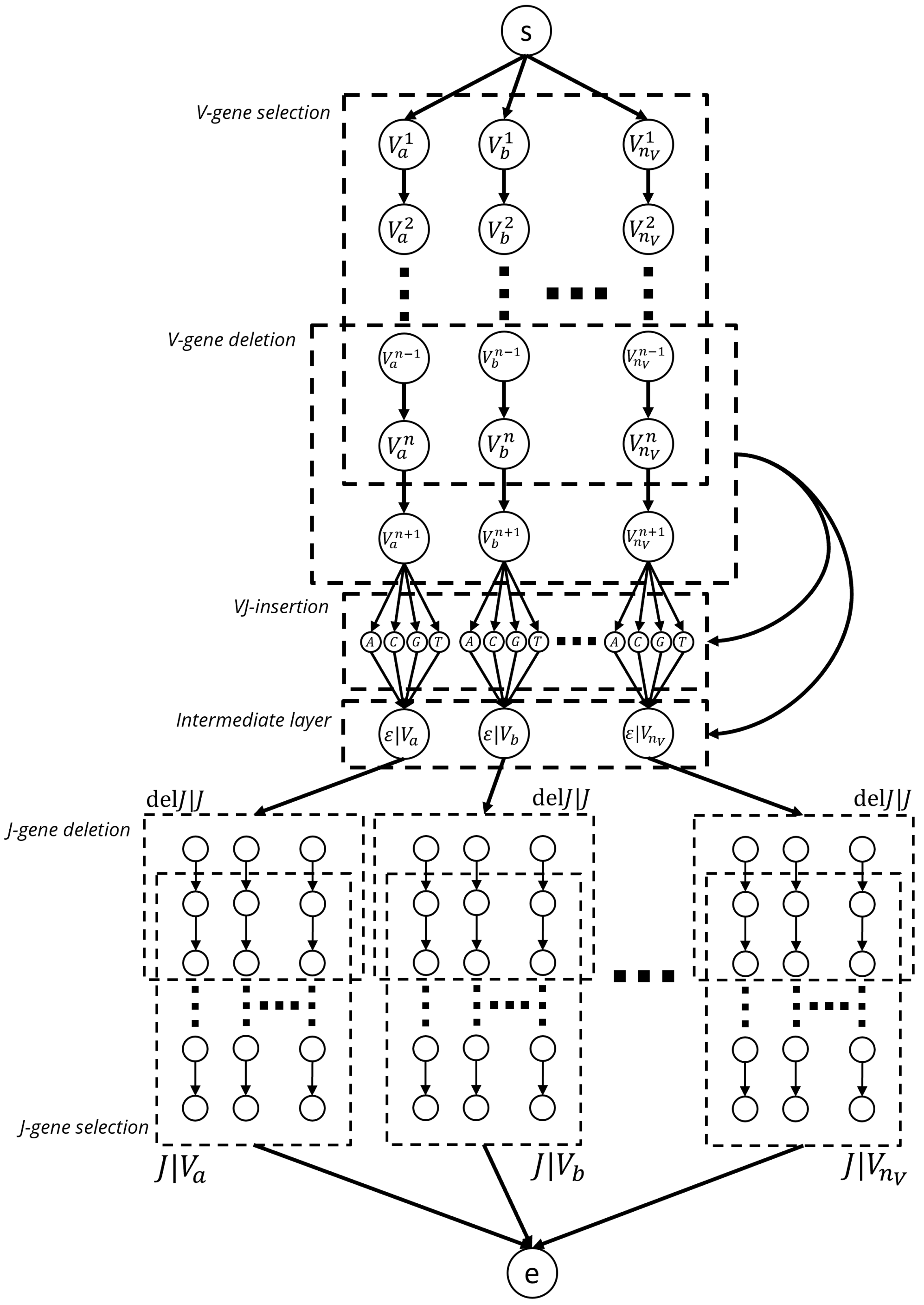
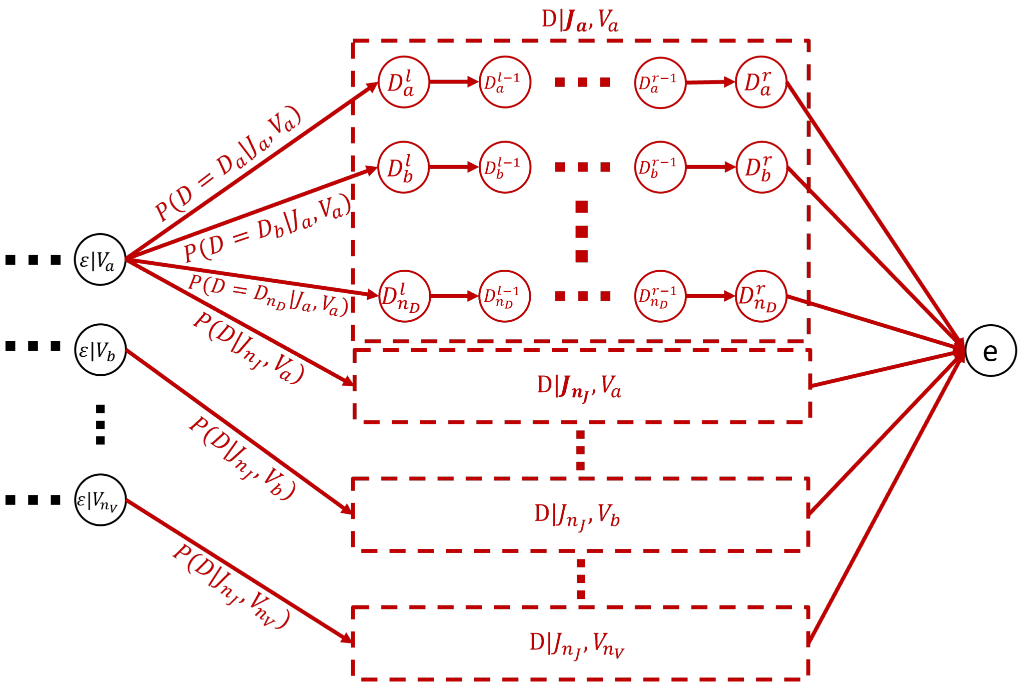


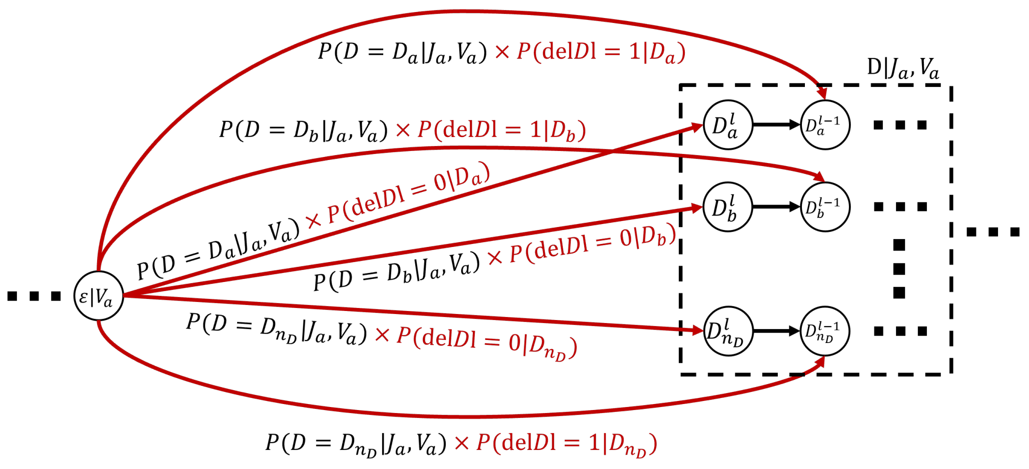
Figure 3: Overview of the TCR-FSA structure for the alpha-chain. In this simplified illustration, deletions range from -1 to 1 and there can be only a maximum of 1 VJ-insertion. For readability, all of the probabilities along the transitions have been omitted.

these dependencies throughout all of the layers up until the point where we do J-gene selection.

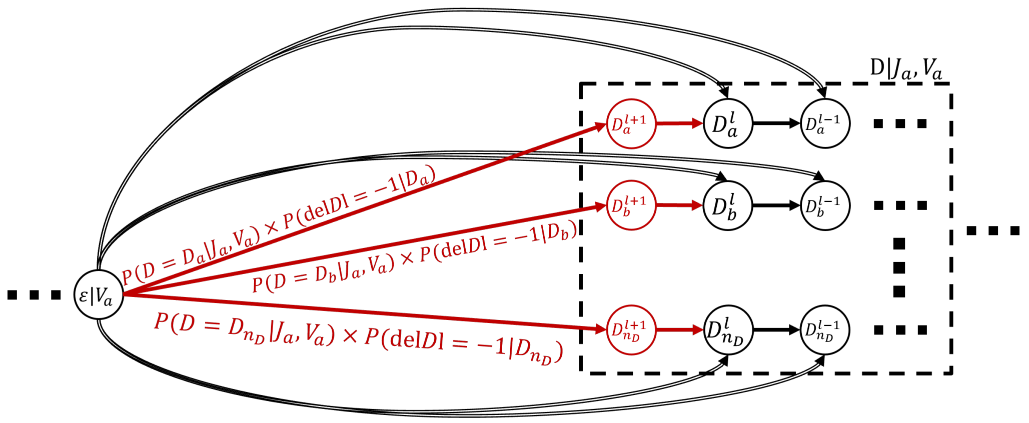
Let us now illustrate the D-gene selection layer. For our illustration, we have D-genes $D_a, D_b, ..., D_{n_D}$, J-genes $J_a, ..., J_{n_J}$ and V-genes $V_a, V_b, ..., V_{n_V}$. The D-genes have nucleotides ranging from $l, l-1, ..., r-1, r$, with l and r corresponding to the leftmost and rightmost nucleotides in the D-gene. The probabilities for going into the D-gene will be equal to $P(\mathrm{D}|\mathrm{J}, \mathrm{V})$, in which the V-gene choice is already determined by the V-gene dependency in the VD-intermediate layer. Please recall that in our illustrations, all probabilities along the transitions are 1, unless noted otherwise. Let us now focus only on the VD-intermediate layer and the ending state with the added D-gene selection layer. Our illustration is then as follows:



5' D-gene deletion The D-gene can have deletions on both the 5'- and the 3'-side. We will focus on these terms separately, starting with the 5'-side (left), as this term has less dependencies. When we ignore the extra dependencies the D-gene itself has when compared to the J-gene, it is the same as J-gene deletion. Let us illustrate non-negative 5'-side D-gene deletion first. In our illustration, we allow up to 1 maximum deletion. We zoom in on a single D-gene selection branch, that of $\mathrm{D}|\mathrm{J}_a, \mathrm{V}_a$. Again, we can now enter the D-gene selection layer at different points within the gene, because of positive deletion. We add the probabilities to the connections going into the D-gene selection layer, giving us the following illustration:


$P(D = D_a|J_a, V_a) \times P(\mathrm{del}Dl = 1|D_a)$
$P(D = D_b|J_a, V_a) \times P(\mathrm{del}Dl = 1|D_b)$
$P(D = D_a|J_a, V_a) \times P(\mathrm{del}Dl = 0|D_a)$
$P(D = D_b|J_a, V_a) \times P(\mathrm{del}Dl = 0|D_b)$
$P(D = D_{n_D}|J_a, V_a) \times P(\mathrm{del}Dl = 0|D_{n_D})$
$P(D = D_{n_D}|J_a, V_a) \times P(\mathrm{del}Dl = 1|D_{n_D})$
$\mathrm{D}|J_a, V_a$
$\varepsilon|V_a$
D_a^l
D_a^{l-1}
D_b^l
D_b^{l-1}
$D_{n_D}^l$
$D_{n_D}^{l-1}$

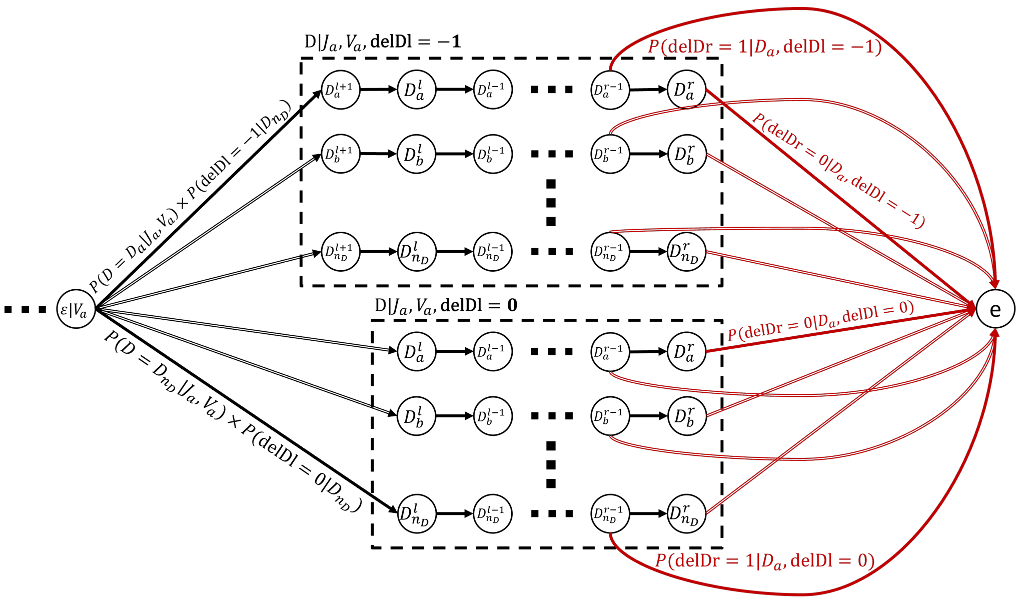
Negative 5'-side D-gene deletions are again similar to negative J-gene deletions. We can show this again through an illustration, where we focus on the same D-gene selection branch, $\mathrm{D}|\mathrm{J}_a, \mathrm{V}_a$. Let us assume we allow for a single P-nucleotide insertion. When omitting the previously added probabilities (along the doubly-lined arrows) for readability purposes, the addition of negative deletion can now be illustrated as follows:


$P(D = D_a|J_a, V_a) \times P(\mathrm{del}Dl = -1|D_a)$
$P(D = D_b|J_a, V_a) \times P(\mathrm{del}Dl = -1|D_b)$
$P(D = D_{n_D}|J_a, V_a) \times P(\mathrm{del}Dl = -1|D_{n_D})$
$\mathrm{D}|J_a, V_a$
$\varepsilon|V_a$
D_a^{l+1}
D_a^l
D_a^{l-1}
D_b^{l+1}
D_b^l
D_b^{l-1}
$D_{n_D}^{l+1}$
$D_{n_D}^l$
$D_{n_D}^{l-1}$

In our actual TCR-FSA, we allow up to a maximum of 16 deletions and 4 palindromic insertions of the 5'-side of the D-gene.

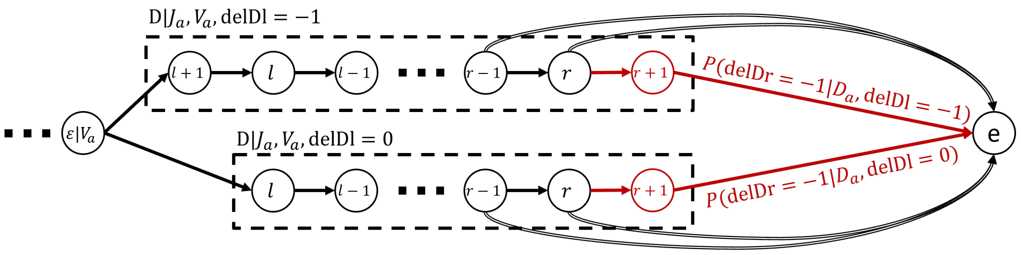
3' D-gene deletion As we can see from eq. 4, 3'-side D-gene deletion, $P(\mathrm{delDr}|\mathrm{delDl}, \mathrm{D})$ introduces another dependency, that of 5'-side gene deletion. So, deletion on the right side of the D-gene is dependent on both the choice of D-gene and the number of deletions on the left side of the gene. We give each number of 5'-side D-gene deletions its own branch, such that the dependency holds when we perform 3'-side D-gene deletion.

As always, let us focus on non-negative 3'-side D-gene deletion first. We will now illustrate the addition of 3'-side D-gene deletion to our FSA structure. We essentially pull apart the existing D-gene selection and 5'-side D-gene deletion layers, creating new branches for each number of deletions on the 5'-side. For each resulting branch, we can then perform 3'-side D-gene deletion as we would do normally. For our illustration, let us zoom in on the same D-gene selection branch as in the previous illustrations, $\mathrm{D}|\mathrm{J}_a, \mathrm{V}_a$. When we split up this branch per each number of deletions in the D-gene, we get three new branches: $\mathrm{D}|\mathrm{J}_a, \mathrm{V}_a, \mathrm{delDl} = -1$; $\mathrm{D}|\mathrm{J}_a, \mathrm{V}_a, \mathrm{delDl} = 0$ and $\mathrm{D}|\mathrm{J}_a, \mathrm{V}_a, \mathrm{delDl} = 1$. In our illustration, we focus on the first two of these branches, and we add non-negative deletions on the 3'-side to them. Each of these branches can now only be entered using their corresponding number of 5'-side deletions. This causes the 5'-side dependency to exist for each branch, allowing us to perform 3'-side deletion. In our illustration of non-negative 3'-side D-gene deletion, we have omitted the probabilities along the doubly-lined arrows, to prevent clutter and improve readability:



$\mathrm{D}|J_a, V_a, \mathbf{delDl = -1}$
$P(\mathrm{delDr} = 1|D_a, \mathrm{delDl} = -1)$
$P(\mathrm{delDr} = 0|D_a, \mathrm{delDl} = -1)$
$P(D = D_a|J_a, V_a) \times P(\mathrm{delDl} = -1|D_{n_D})$
$\varepsilon|V_a$
$\mathrm{D}|J_a, V_a, \mathbf{delDl = 0}$
$P(\mathrm{delDr} = 0|D_a, \mathrm{delDl} = 0)$
$P(D = D_{n_D}|J_a, V_a) \times P(\mathrm{delDl} = 0|D_{n_D})$
$P(\mathrm{delDr} = 1|D_a, \mathrm{delDl} = 0)$
e

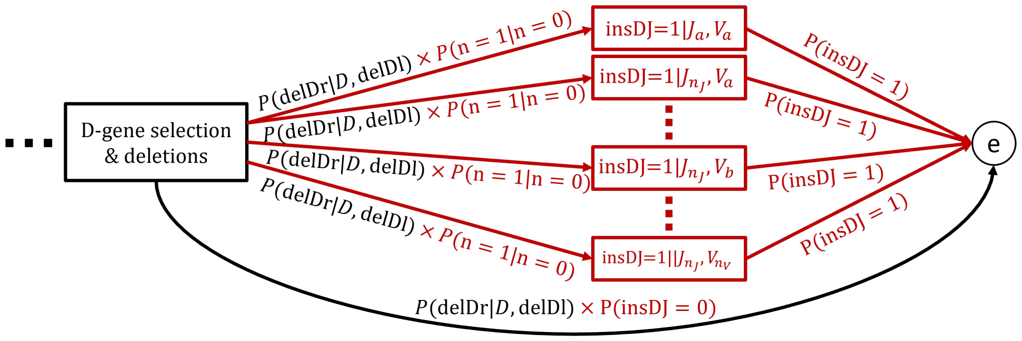
Next, let us consider negative deletions of the 3'-side of the D-gene. As with the other types P-nucleotide insertions, we add them to the right side of the D-gene, and add the probabilities resulting from this addition. Let us illustrate this with a single P-nucleotide insertion. We again focus on the same two branches as before, $\mathrm{D}|\mathrm{J}_a, \mathrm{V}_a, \mathrm{delDl} = -1$ and $\mathrm{D}|\mathrm{J}_a, \mathrm{V}_a, \mathrm{delDl} = 0$, but we have now collapsed them into a single D-gene for readability purposes. Again, the probabilities along the doubly-lined arrows have been omitted for the same reason. Our illustration of this part of the current FSA structure is then as follows:



$\mathrm{D}|J_a, V_a, \mathrm{delDl} = -1$
$P(\mathrm{delDr} = -1|D_a, \mathrm{delDl} = -1)$
$\varepsilon|V_a$
$\mathrm{D}|J_a, V_a, \mathrm{delDl} = 0$
$P(\mathrm{delDr} = -1|D_a, \mathrm{delDl} = 0)$
e

In our actual TCR-FSA for the beta-chain, we allow up to 16 positive and 4 negative 3'-side D-gene deletions. Because of the length of the D-genes in terms of their number of nucleotides, it is possible for the entire D-gene to be deleted. In our FSA, we add connections to bypass the D-gene selection entirely, with the probabilities of the number of deletions required to do so. Due to its added complexity to the FSA structure, these connections are not present in our illustrations of the individual layers of the TCR-FSA, but we do show them in fig. 4.

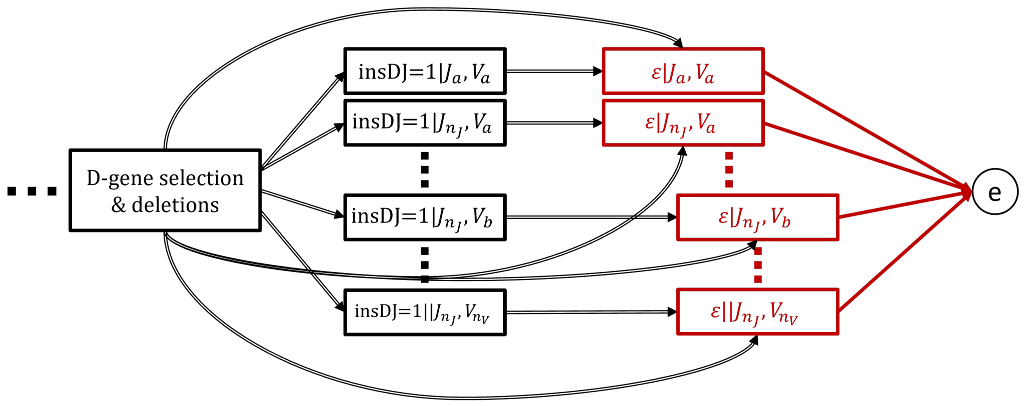
DJ-insertion The DJ-insertion layer is the same as the VD-insertion layer, as it does not introduce any new dependencies. However, we must keep in mind that we still have a dependency on the V-gene and the selection of the J-gene at this point of the FSA, which is necessary for J-gene selection. Therefore, we add a DJ-insertion layer for each branch that is required for J-gene selection. At this point in the FSA, we can collapse a lot of branches due to some of them not being required anymore, in particular the branches introduced by the dependency on the D-gene and the dependency on 5'-side deletion of the D-gene. Apart from the extra dependencies, all of the aspects of DJ-insertion are the same as those of VD-insertion. Let us now illustrate this with a single DJ-insertion, and where we entirely collapse the D-gene selection and deletion layers for readability purposes:



D-gene selection & deletions
$P(\mathrm{delDr}|D, \mathrm{delDl}) \times P(\mathrm{n}=1|\mathrm{n}=0)$
$\mathrm{insDJ}=1|J_a, V_a$
$\mathrm{insDJ}=1|J_{n_J}, V_a$
$\mathrm{insDJ}=1|J_{n_J}, V_b$
$\mathrm{insDJ}=1||J_{n_J}, V_{n_V}$
$P(\mathrm{insDJ}=1)$
e
$P(\mathrm{delDr}|D, \mathrm{delDl}) \times \mathrm{P}(\mathrm{insDJ}=0)$

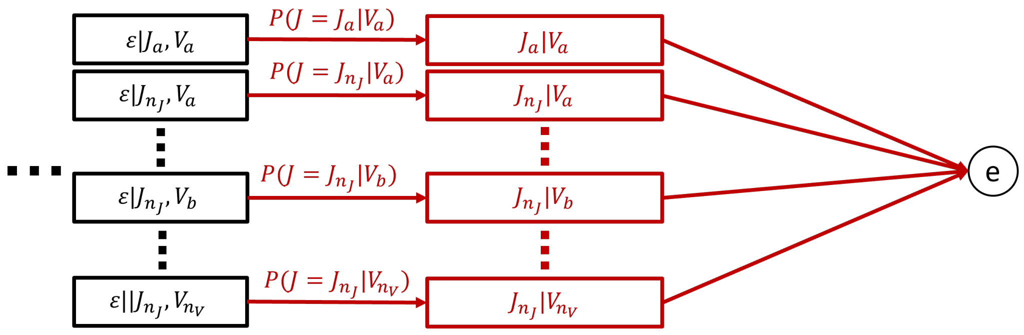
In our full FSA, we can have up to 30 DJ-insertions. For the first DJ-insertion, we always look at the last nucleotide in the current sequence. Therefore, in the case of full deletion of the D-gene, we look at the last nucleotide of the V-gene (accounting for deletion).

DJ-intermediate layer At this point in the FSA, we run into the same problem as with the FSA for the alpha-chain of the TCR. If we would immediately hook up the connections from the previous layers to the J-gene selection, which has multiple entry points (because of deletion), we would get an increase in connections. Again, we solve the problem by adding another intermediate layer. It is the same as the intermediate layer for the alpha-chain and the VJ-intermediate layer for the beta-chain, except that we again have an extra dependency, the choice of J-gene. Illustrated, and with the D-gene selection and D-gene deletion layers collapsed, the DJ-intermediate layer looks as follows:



D-gene selection & deletions
insDJ=1$|J_a, V_a$
insDJ=1$|J_{n_J}, V_a$
insDJ=1$|J_{n_J}, V_b$
insDJ=1$||J_{n_J}, V_{n_V}$
$\varepsilon|J_a, V_a$
$\varepsilon|J_{n_J}, V_a$
$\varepsilon|J_{n_J}, V_b$
$\varepsilon||J_{n_J}, V_{n_V}$
e

J-gene selection The last steps in building the FSA-structure for the beta-chain are J-gene selection and J-gene deletion. We will not go over them in great detail, since the structure is the same as with the alpha-chain. The only difference is that we have already chosen the J-gene earlier in the FSA. Therefore, all we have to do is to connect the correct branches corresponding to specific J-genes in the DJ-intermediate layer to the correct J-genes in the J-gene selection layer. Because everything else is the same as with J-gene selection and deletion in the alpha-chain, we will not cover J-gene deletion any further. We will, however, illustrate how the DJ-intermediate layer connects to the J-gene selection layer. Let us consider the same DJ-intermediate nodes as before. They connect to the J-gene selection layer in the following way, where each rectangle now corresponds to a complete J-gene sequence:



In the TCR-FSA for the beta-chain, we have 15 J-genes, with a maximum of 18 positive deletions and 4 negative deletions.

Summary We have now covered all of the terms in the dependency structure for the beta-chain of the TCR, finalizing the construction of the TCR-FSA structure for the TCRb. In Fig. 4, an overview of the TCR-FSA for the beta-chain is shown.

3.2.3 Probability data

To finalize the construction of our FSAs, we replace the probability placeholders with actual probabilities from the data found in IGoR. For some edges in the FSA, we have multiple probabilities, which we multiply to obtain a single probability. The end result are two FSAs, one for the TCRa and one for the TCRb, with only probabilities along

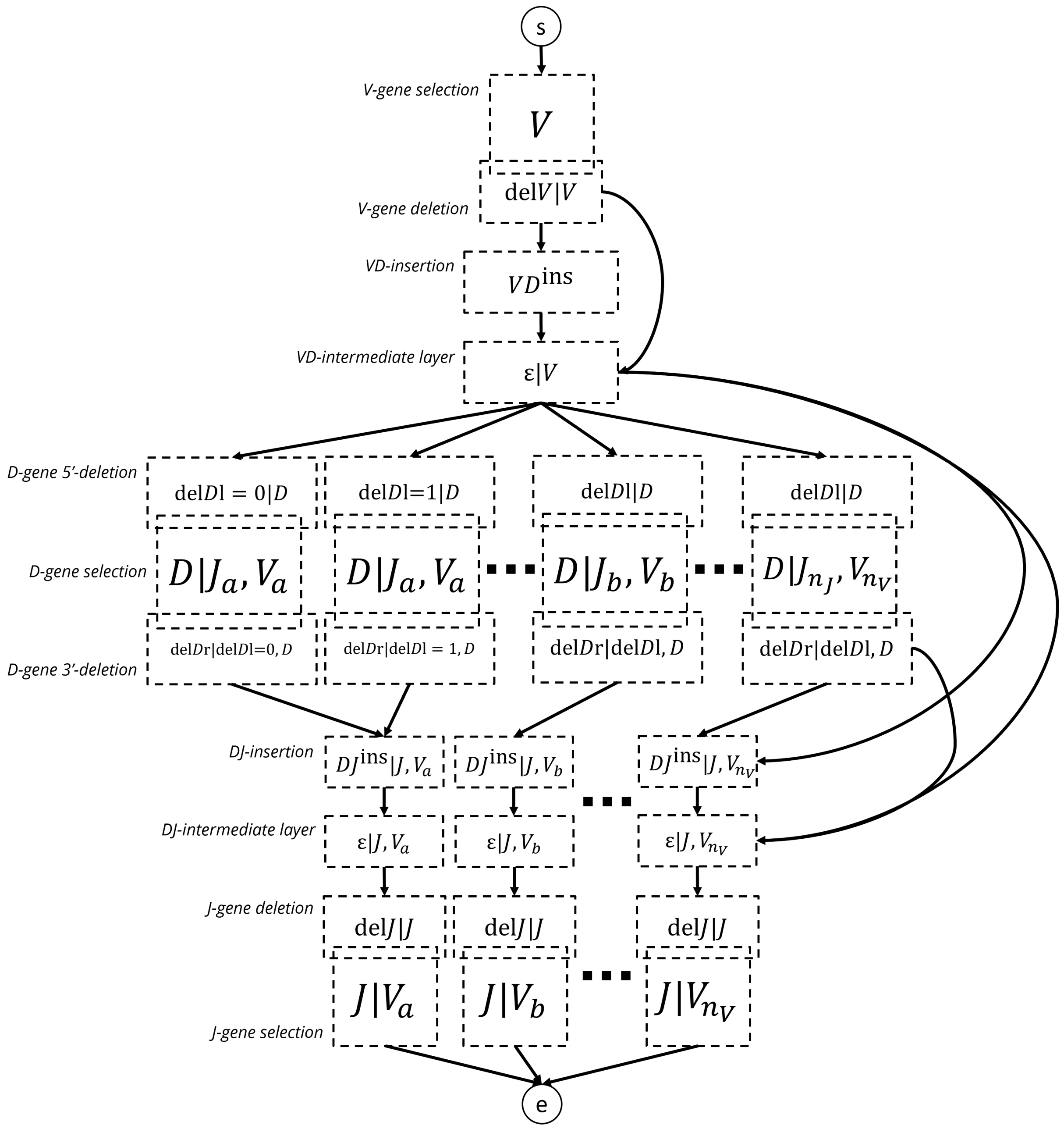


Figure 4: Simplified overview of the TCR-FSA structure for the beta-chain and its dependencies. For readability, all of the probabilities along the transitions have been omitted.

their edges. The edges with '1' along them remain unchanged. These FSAs can then be used for analysis.

3.3 FSA analysis

Both sequence analysis and sequence (or repertoire) generation can be performed using our TCRa- and TCRb-FSAs. Here, we will discuss both algorithms separately.

3.3.1 Sequence generation

The sequence generation algorithm used on the FSAs is a form of probabilistic FSA traversal, in which we try to find a path from the start node to the end node, using the probabilities in the FSA. Once we have found such a path, we know that it represents a sequence, and therefore, we have generated a sequence.

First of all, we set the start node, s, to be the "current node". Next, we look at all of the neighbours of the current node, and pick the next "current node" according to the probabilities along the edges. We repeat this process until the current node is the end node, e. Please keep in mind that the FSAs are all forms of acyclic directed graphs, so when we look for neighbours in the FSA, we will never return a previous node from the path. In reality, the algorithm is a bit more complex than this, as, due to some probabilities being zero for some of the edges in the FSA, we can also have dead ends. In some cases, this prohibits us from finding any reachable neighbours for some nodes in the FSA. When this is the case, we backtrack until we can go deeper into the graph again. When the current node only has a single reachable neighbour, we always choose this neighbour.

This algorithm only gives correct results when the FSA has been adapted prior to using the algorithm, which has not yet been done. As of now, some probabilities appear at points in the FSA such that they are not taken into account when traversing certain parts of the FSA, even though they might influence the traversal choices involved. Therefore, the algorithm is incorrect when used on the current FSAs. The probabilities within the FSA would have to be moved towards the left of the FSA, such that going into a new branch also takes into account these probabilities. However, the algorithm will still be able to perform roughly the same calculations, resulting in the same average run time as described in the results-section of this work.

3.3.2 Sequence analysis

When given a T-cell receptor, the sequence analysis algorithm calculates the corresponding probability using the FSA. It is a form of depth-first search in the FSA, with a lot of optimizations such that computation time is decreased. Many of these optimizations will not be discussed here, as they are not theoretical in nature.

The algorithm works by traversing the FSA from the starting node in a depth-first manner, to save memory. It keeps track of its current path, its current position in the given nucleotide sequence, and the current probability of the current path. It can only traverse further into the FSA if the next nucleotide matches the next nucleotide in the given sequence. Once it has reached the end node, this means that a path has been found which matches the nucleotide sequence, and the current probability is the final probability of the path. However, there might be multiple ways for the FSA to produce the same nucleotide sequence. Therefore, we must search the entire reachable graph and sum all of the probabilities of all the paths from the start node to the end node corresponding to the given nucleotide sequence. Just like with the sequence generation algorithm, we can run into dead ends, where it might not be possible to go deeper into the FSA, for example when the neighbouring nucleotides do not match the next nucleotide in the provided sequence. In this case, we simply backtrack, as is customary with depth-first search.

One optimization which we will discuss here is the addition of a probability threshold. Some probabilities of the paths leading to a sequence might be very insignificant. Since we keep track of the current probability, we can easily see if going deeper into the FSA using a particular edge causes the current probability to go under a probability threshold. When this is the case, we do not allow the algorithm to traverse this edge, after which the search continues. This can significantly speed up the search, especially for the TCRb-FSA.

4 Results

In this section, we shall provide results related to the size of both FSAs and run time of several tasks related to both FSAs, such as sequence analysis and sequence generation. All of the results were obtained using mid-range hardware, consisting of a laptop computer with a Intel i7-8550U processor with 8GB LPDDR3 RAM.

4.1 FSA size

As expected, from table 1, we see a clear size difference between the TCRa-FSA and the TCRb-FSA. Both FSAs are saved locally as a JSON-file, which contains all nodes and edges. The difference in file size can be explained due to the vast difference between the number of nodes and edges between the two FSAs. The TCRb-FSA has 1.5 times the nodes and more than 5 times the number of edges compared to the TCRa-FSA. The graph density, which relates the number of possible edges in the graph based on the number of nodes to the actual number of edges in the graph, where 1 is a complete graph and 0 is a graph with no edges, is similar for both FSAs.

TCRa-FSA & TCRb-FSA size comparison		
Characteristic	TCRa	TCRb
Local file size	78.6MB	499MB
Number of nodes	499,068	791,016
Number of edges	733,494	4,152,099
Graph density	2.9×10^{-6}	6.6×10^{-6}

Table 1: Comparison between the TCRa-FSA and the TCRb-FSA in terms of local file size (JSON), number of nodes and edges, and graph density.

4.2 Run times

Tables 2 and 3 show the run times of several tasks related to the FSAs, for both the TCRa-FSA and the TCRb-FSA. These tasks include FSA construction (building the FSA structure and calculating all the probabilities along the edges according to the methods described previously), loading the FSA (which involves reading in the local JSON file), generation of a single sequence as well as generation of 100 sequences, and the recombination probability analysis of a single sequence (see “FSA analysis” under the methods-section for a description of sequence generation and sequence analysis). For sequence analysis, a random sequence is analysed. All of these tasks were performed 10 times, and the average run time across these 10 runs with a standard deviation is provided in tables 2 and 3.

When looking at FSA construction, we can see a clear difference between the TCRa- and TCRb-FSAs. Due to its increased complexity (which also follows from the size difference), it takes a longer time to construct the TCRb-FSA. The same can be said for loading the FSAs. The TCRb-FSA is more than 6 times larger in terms of local file size and therefore takes more than 3.5 times the time to load. The run times for sequence generation do not differ much across both FSAs, which can be explained due to the fact that this algorithm simply traverses the FSA from start to end, making it linear in computation time. Even though the TCRb-FSA has much more nodes and edges than the TCRa-FSA, the overall length of the sequences generated does not differ much on this timescale. Sequence generation is very quick, even on this hardware, for both FSAs, and large repertoires can be generated in a matter of seconds.

A vast difference can be observed in terms of sequence analysis between the TCRa-FSA and the TCRb-FSA. In the results from tables 2 and 3, no threshold is used, which means that the full graph is being explored, as long as the nodes and edges are consistent with the provided sequence. The difference in run time can therefore be explained due to the difference in the number of nodes and edges between the two FSAs.

TCRa-FSA run time statistics across 10 runs	
Task	Avg. run time +/- SD
FSA Construction	6.32s +/- 0.63
Loading time	6.66s +/- 0.55
1 Sequence generation	0.02s +/- 0.01
100 Sequence generation	1.06s +/- 0.07
1 Sequence analysis, no threshold	2.88s +/- 1.07

Table 2: Run time statistics across several tasks for the TCRa-FSA. Each task is performed 10 times, and the average run time and the standard deviation is provided here.

TCRb-FSA run time statistics across 10 runs	
Task	Avg. run time +/- SD
FSA Construction	59.02s +/- 3.38
Loading time	24.71s +/- 1.48
1 Sequence generation	0.02s +/- 0.01
100 Sequence generation	1.00s +/- 0.19
1 Sequence analysis, no threshold	460.14s +/- 131.99

Table 3: Run time statistics across several tasks for the TCRb-FSA. Each task is performed 10 times, and the average run time and the standard deviation is provided here.

From figures 5 and 6, we can see that adding a threshold can make a big difference in the run time of sequence analysis, for both the TCRa-FSA and the TCRb-FSA. Overall, a larger threshold means a lower run time, but the difference is more evident for the TCRb-FSA. With a larger threshold, a smaller portion of the graph has to be explored, since many potential paths are off-limits due to them contributing only a very insignificant amount to the total probability. For the TCRa-FSA, the sequence analysis run time will always stay within the order of seconds, but certain threshold values can make sequence analysis take longer than analysis without a threshold. This observation can be explained due to the fact that a check is performed for each reachable node, whether its probability brings us under the significance threshold. Therefore, it only makes sense to use large thresholds for the TCRa-FSA, since making this value too small will make sequence analysis slower than without a threshold. Using large thresholds, we can bring down the sequence analysis run time of the TCRb-FSA into the order of seconds. However, it quickly increases with lower thresholds, which makes sequence analysis at low thresholds go into the order of minutes. We also see a clear increase in the standard deviation for these lower thresholds, as some sequences might require more of the FSA to be explored than others. For large thresholds, the differences in analysis time are generally much smaller. Another interesting observation is that, for the TCRb-FSA, even at very small thresholds, we still see a benefit in terms of run times when compared to using no threshold at all. Since the FSA is very large and complex, the check performed when using the threshold does not have a very significant contribution in terms of run time related to the total time it takes to explore the FSA, even for these very low thresholds.

The use of a threshold value does come with a trade-off, as it introduces a risk that certain sequences might not be found in the FSA at all. This can happen when all of the total probabilities of paths in the FSA corresponding to the sequence are below the probability threshold. The algorithm will not be able to find a single path corresponding to the sequence, and no probability can be found. In the case of a sequence being found, the difference between the actual probability and the probability with a threshold generally depends on how close the threshold is to the actual probability. The closer the probability threshold, the higher the error, as the neglected probabilities have greater significance over the total probability.

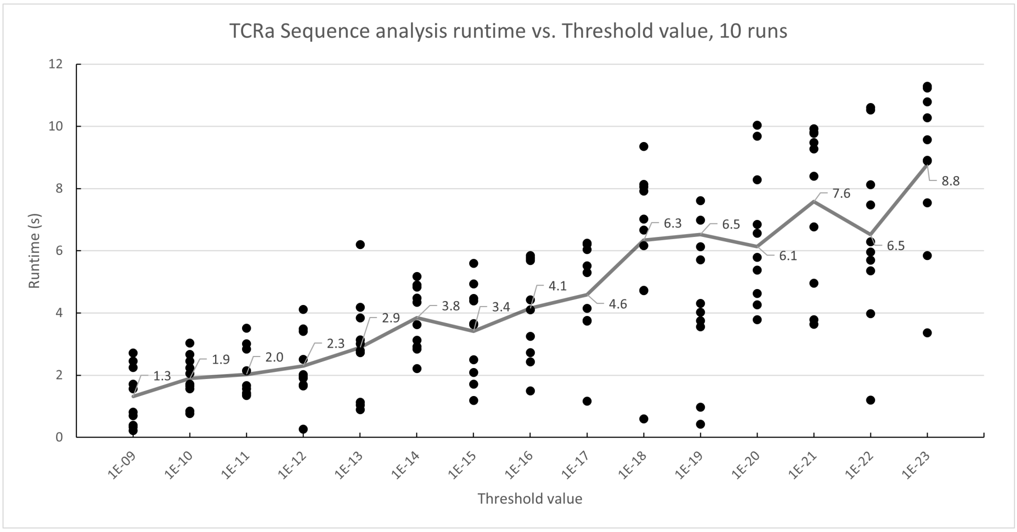


Figure 5: Average single sequence analysis run time vs. Threshold value across 10 runs (with each run represented by a dot) for each threshold value, TCRa-FSA.

5 Conclusion

In this work, we presented a weighted Finite-State Automaton (FSA) for sequence analysis and sequence generation of the alpha- and beta-chains of T-Cell Receptors (TCRa and TCRb). The FSA structure is built using dependency structures introduced in previous work [12], and the FSAs use a layered structure based on the several elements present in the TCR-dependency structures. The FSA structure comes with placeholders, which can be replaced with TCR repertoire probability data, such that calculations

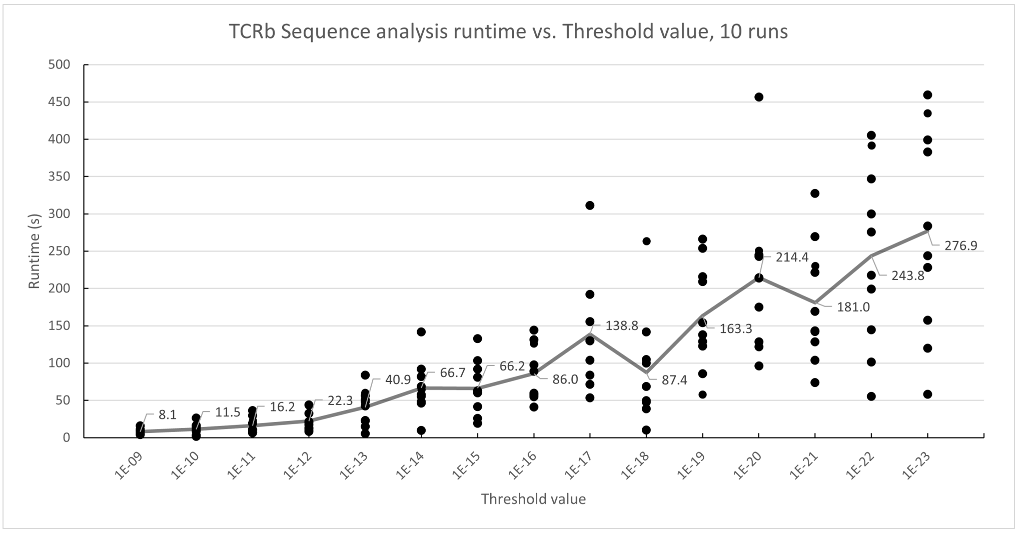


Figure 6: Average single sequence analysis run time vs. Threshold value across 10 runs (with each run represented by a dot) for each threshold value, TCRb-FSA.

can be performed using the FSA. In this work, we show that the resulting FSAs are able to calculate the recombination probability of a given TCR-sequence. Using a threshold parameter, this calculation can be performed in a matter of seconds for both the alpha- and beta-chain, using conventional mid-tier hardware. Additionally, the FSAs can generate repertoires of TCR-sequences in a short amount of time, but as of now, the sequence generation algorithm is incorrect.

There are improvements which can be applied directly to the FSAs themselves. For example, we could get rid of insignificant edges or edges with zero-probability before analysis, which might also shorten the run time of these algorithms. We could also apply determinization and/or minimization to our FSAs. As of now, many transitions between states in the FSA have a probability of 1, but these states could be collapsed into a single state consisting of multiple nucleotides, which could considerably bring down the number of nodes and edges in the FSA. Other states of the FSA might also be merged through minimization, as long as the dependencies in the FSA remain.

There are also improvements which can be made to the analysis algorithms provided in this work. First of all, sequence analysis is quite slow without the use of the threshold parameter, especially for the FSA of the beta-chain, which is caused by the large size and complexity of the FSA. However, determining the value of the threshold parameter beforehand might not be desirable, since some sequences only have a very small probability to be produced through recombination, and might therefore show up without a probability because of their insignificance. Therefore, we suggest a dynamic threshold, which is adapted during sequence analysis. Since there are multiple ways in which a sequence might be formed using recombination, one way might be way more probable than another. When we find a probable way of forming a sequence, we could set the threshold value accordingly, so that very improbable recombination probabilities which might come after do not contribute to the total probability anymore, shortening the run time of the algorithm. The run times of both algorithms could also still be tested on better hardware, to see whether this makes a significant difference.

This work offers a perspective on how FSAs might be used as stochastic models for V(D)J recombination, resulting in the creation of weighted FSAs which serve this exact purpose. This opens the way for using FSAs for V(D)J recombination within large-scale models of the adaptive immune system.

References

[1] NWO | Artificial Immunological Intelligence.

[2] Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter. Introduction to Pathogens. *Molecular Biology of the Cell. 4th edition*, 2002. Publisher: Garland Science.

[3] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. OpenFst: A General and Efficient Weighted Finite-State Transducer Library. In Jan Holub and Jan Žďárek, editors, *Implementation and Application of Automata*, Lecture Notes in Computer Science, pages 11–23, Berlin, Heidelberg, 2007. Springer.

[4] Dmitriy A. Bolotin, Stanislav Poslavsky, Igor Mitrophanov, Mikhail Shugay, Ilgar Z. Mamedov, Ekaterina V. Putintseva, and Dmitriy M. Chudakov. MiXCR: software for comprehensive adaptive immunity profiling. *Nature Methods*, 12(5):380–381, May 2015. Number: 5 Publisher: Nature Publishing Group.

[5] Xavier Brochet, Marie-Paule Lefranc, and Véronique Giudicelli. IMGT/V-QUEST: the highly customized and integrated system for IG and TR standardized V-J and V-D-J sequence analysis. *Nucleic Acids Research*, 36(suppl_2):W503–W508, July 2008.

[6] J. E. Deakin, Z. E. Parra, J. a. M. Graves, and R. D. Miller. Physical mapping of T cell receptor loci (TRA@, TRB@, TRD@ and TRG@) in the opossum (Monodelphis domestica). *Cytogenetic and Genome Research*, 112(3-4):342K–342K, 2006. Publisher: Karger Publishers.

[7] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata.* Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2009.

[8] Marc Duez, Mathieu Giraud, Ryan Herbert, Tatiana Rocher, Mikaël Salson, and Florian Thonier. Vidjil: A Web Platform for Analysis of High-Throughput Repertoire Sequencing. *PLOS ONE*, 11(11):e0166126, November 2016. Publisher: Public Library of Science.

[9] Namita T. Gupta, Jason A. Vander Heiden, Mohamed Uduman, Daniel Gadala-Maria, Gur Yaari, and Steven H. Kleinstein. Change-O: a toolkit for analyzing large-scale B cell immunoglobulin repertoire sequencing data. *Bioinformatics*, 31(20):3356–3358, October 2015.

[10] Aric Hagberg and Drew Conway. NetworkX: Network Analysis with Python. *URL: https://networkx. github. io*, 2020.

[11] Nadia Kurd and Ellen A. Robey. T-cell selection in the thymus: a spatial and temporal perspective. *Immunological Reviews*, 271(1):114–126, May 2016.

[12] Quentin Marcou, Thierry Mora, and Aleksandra M. Walczak. High-throughput immune repertoire analysis with IGoR. *Nature Communications*, 9(1):561, February 2018. Number: 1 Publisher: Nature Publishing Group.

[13] Kenneth Murphy and Casey Weaver. *Janeway's Immunobiology*. Garland Science, March 2016. Google-Books-ID: GmPLCwAAQBAJ.

[14] Anand Murugan, Thierry Mora, Aleksandra M. Walczak, and Curtis G. Callan. Statistical inference of the generation probability of T-cell receptors from sequence repertoires. *Proceedings of the National Academy of Sciences*, 109(40):16161–16166, October 2012. Publisher: Proceedings of the National Academy of Sciences.

[15] Joy A. Pai and Ansuman T. Satpathy. High-throughput and single-cell T cell receptor sequencing technologies. *Nature Methods*, 18(8):881–892, August 2021. Number: 8 Publisher: Nature Publishing Group.

[16] Duncan K. Ralph and Frederick A. Matsen Iv. Consistency of VDJ Rearrangement and Substitution Parameters Enables Accurate B Cell Receptor Sequence Annotation. *PLOS Computational Biology*, 12(1):e1004409, January 2016. Publisher: Public Library of Science.

[17] David G. Schatz and Yanhong Ji. Recombination centres and the orchestration of V(D)J recombination. *Nature Reviews Immunology*, 11(4):251–263, April 2011. Number: 4 Publisher: Nature Publishing Group.

[18] David G. Schatz and Patrick C. Swanson. V(D)J Recombination: Mechanisms of Initiation. *Annual Review of Genetics*, 45(1):167–202, 2011. _eprint: https://doi.org/10.1146/annurev-genet-110410-132552.

[19] Zachary Sethna, Yuval Elhanati, Curtis G Callan, Jr, Aleksandra M Walczak, and Thierry Mora. OLGA: fast computation of generation probabilities of B- and T-cell receptor amino acid sequences and motifs. *Bioinformatics*, 35(17):2974–2981, September 2019.

[20] Adrien Six, Encarnita Mariotti-Ferrandiz, Wahiba Chaara, Susana Magadan, Hang-Phuong Pham, Marie-Paule Lefranc, Thierry Mora, Véronique Thomas-Vaslin, Aleksandra Walczak, and Pierre Boudinot. The Past, Present, and Future of Immune Repertoire Biology – The Rise of Next-Generation Repertoire Analysis. *Frontiers in Immunology*, 4, 2013.

[21] Niclas Thomas, James Heather, Wilfred Ndifon, John Shawe-Taylor, and Benjamin Chain. Decombinator: a tool for fast, efficient gene assignment in T-cell receptor sequences using a finite state machine. *Bioinformatics*, 29(5):542–550, March 2013.