Carving a Niche in the Intersection of Computer Engineering and Molecular Biology

a dissertation submitted to the faculty of the
University of Minnesota by

Arnav Solanki

In partial fulfillment of the requirements for
the degree of
Doctor of Philosophy

Marc Riedel

May 2023

Acknowledgements

I would like to thank Marc Riedel for his support, guidance, and encouragement, George Vasmatzis and James Cornette for co-advising me through my degree, Chad Myers and Honghong Tinn for reviewing my numerous exams, and Murti Salapaka for offering sound advice. I would also like to thank the numerous instructors who motivated me through my research: these include Jeongsik Yong, Aaron Goldstrohm, Dan Knights, Romas Kazlauskas, R. Scott McIvor, Martina Cardone, Demoz Gebre-Egziabher, Andrew Lamperski, Kia Bazargan, and the late James Parker.

Finally, I express my gratitude to my parents, my sister, and my grandmother for their patience.

To Daisy, woof woof.

Abstract

Research at the intersection of Molecular Biology and Computer Science is evolving at a rapid pace. With newer tools like CRISPR-Cas9 making biotechnology more accessible and computational demands scaling to smaller sizes, DNA has become an ideal substrate for molecular computing. Here, a parallelized model of computing using Single input, multiple data DNA is explored. Numerous algorithms such as binary sorting, shifting, searching, and XOR are developed. An additional approach to computing through chemical reaction networks is proposed, one that allows for computing stochastically using DNA concatemers. With improvements in computing power and large data volumes being generated in bioinformatics, machine learning has taken the forefront in problems such as protein-peptide modeling. For fields such as computational immunology, reducing erroneous predictions from machine learning based tools is vital. Here the vulnerability of such tools not utilizing biochemical attributes such as hydrophobicity in their predictions is outlined. This would increase their impact in applications such as vaccine design, Furthermore, these tools were used to investigate the evolution of SARS-CoV-2 virus and its impact on Human T cell immunity. A novel approach to designing experimental controls was used to validate the preservation of T Cell epitopes in the various SARS-CoV-2 variants. These results suggest that T Cell immunity mounts a strong defense against COVID-19.

Contents

List of Figures

List of Tables

List of Abbreviations

AUC: Area under the curve

BA: Binding affinity

CRN: Chemical reaction network

DNA: Deoxyribonucleic acid

dsDNA: Double stranded DNA

EL: Eluted ligand

FP: False positives

FN: False negatives

GPU: Graphical processing unit

HLA: Human leukocyte antigens

MHC: Major histocompatibility complex

MIMD: Multiple input, multiple data

ML: Machine learning

RNA: Ribonucleic acid

ROC: Receiver operating characteristic

SIMD: Single instruction, multiple data

ssDNA: Single stranded DNA

TP: True positives

TN: True negatives

Chapter 1

Introduction

Initially, the fields of molecular biology and molecular computing might seem incredibly similar. After all, both deal with the manipulation and analysis of molecular information. Yet they exist as distinct research spheres. Molecular biology focuses on the study of small molecules such as Deoxyribonucleic acid (DNA) and proteins and the roles they play in shaping life. In contrast, molecular computing aims to develop means to store and process data on molecules. The difference in goals – the biologists seeking to explain how life works, while the computer scientists questioning how to develop and improve their computational models – leaves the fields disparate. Furthermore, neither field is limited enough to be overtaken by the other in terms of research volume or significance in applications.

All these differences make it straightforward for researchers to remain oblivious of each other's works. This dissertation stands against this notion: despite the distance between them, progression of technology and knowledge is best driven at the intersection of these fields. For instance, the tools used by both domains are shared – DNA strands, enzymes, and synthetic molecules are just as important for understanding how to treat cancer as developing a suitable DNA computing model, or machine learning pipelines and efficient computer algorithms play a role in efficient bioinformatics data processing or optimizing data storage. DNA computing cannot be truly utilized without an intimate understanding of the various mechanisms nature has evolved to store, repair, process, and replicate DNA. Similarly, bioinformatics studies cannot be reliably concluded without a fine perception of how different computational methods can influence the validation of their hypothesis. This intersection of these fields carves a niche in research; small as it is, has the potential to contribute to medicine, biotechnology, and computing.

This dissertation documents various studies conducted within this aforementioned niche. Given the disparate goals of these studies, they have been compiled into 2 parts: Part I with an emphasis on developing DNA computing models, and Part II discussing the use of machine learning in specific bioinformatics problems. Each part start off with a brief introduction (Chapters 5 and 9) motivating the research and providing the necessary background to understand the studies. Each part also concludes with a discussion (Chapters 5 and 9) highlighting the extent of the work conducted in these studies and their potential.

The studies in Part I are Chapter 3, a deep dive into building a parallel computing model using “single input, multiple data” (SIMD) DNA, and Chapter 4, an exploration of utilizing DNA chemical reaction networks to achieve true stochastic computing. The studies in Part II are Chapter 7, an analysis of the vulnerabilities of machine learning tools when modelling complex biochemsitry, and Chapter 8, a bioinformatics driven approach to investigating the lack of T cell immune evasion in later SARS-CoV-2 variants such as omicron.

Part I

DNA Computing

Chapter 2

Overview

Beginning with the seminal work of Adelman, who discussed solutions to combinatorial problems such as Boolean satisfiability and Hamiltonian paths with DNA a quarter-century ago [1], DNA computing has promised the benefits of massive parallelism in operations. More recently, there has been considerable interest in DNA storage [7, 11] and then performing computation *in vitro* [91, 108]. Instead of storing data in just the sequence of nucleotides in DNA, a particularly promising approach here has been to encode data by "nicking" DNA with editing enzymes such as PfAgo and CRISPR-Cas9 [47, 94]. This approach yields long DNA strands that store data, but also contain single stranded regions of DNA called "toeholds" that can be used to design reactions between DNA molecules [89].

This dissertation features two studies in developing models of computation using DNA as a substrate. The first study investigates parallel computing through the use of "SIMD" DNA. The study details how data can be encoded and transformed into other useful encoding schemes. It shows how data can be initialized on long strands in an easy to scale manner. Furthermore, several algorithms that can be ran on the data stored in DNA *in vitro* are presented. These algorithms include binary bubble sorting, left shifting, bitwise exclusive OR, and searching. Sorting runs in only N parallel steps, where N was the number of bits to be sorted. Shifting finishes in a single parallel step. The searching takes between $\log(n)$ and n steps to complete, where n is the length of the query string. At most N steps are needed to compute the XOR of N bits. All of these algorithms are massively parallelized and can be operated on large quantities of DNA. They are of immediate practical interest, as many forms of computation on stored data entail some form of sorting, XOR, shifting, and searching. Finally, the study includes discussions on the time complexity of these methods and limitations to overcome for future development.

The second study explores the theoretical framework of a chemical reaction network and how it could be implemented *in vitro*. It presents another data encoding scheme using DNA as a substrate, one that lends itself to the paradigm of stochastic computing. This scheme allows for the translation of any combinational logic function into a stochastic function. A simple method for transforming truth tables in to chemical reaction networks is presented, and its correctness is proven. Several examples are used to illustrate how powerful this form of computing is. Lastly, an implementation of this computational model using DNA concatemers is introduced.

2.1 Toehold Mediated Strand Displacement

Deoxyribonucleic Acid (DNA) is a polymerized macromolecule that stores genetic information in nearly all living things in a directed sequence of adenines (A), cytosines (C), guanines (G), and thymines (T). DNA primarily exists as an antiparallel double stranded molecule forming a double helical ladder structure, in which the rungs of the ladder are *nucleotide pairs* (A binds to T, C to G and vice versa) while the side rails are the *phosphodiester backbones* [105, 79].

In synthetic storage system, smaller single stranded DNA sequence units, called *domains*, are concatenated to form longer strands. All domains are restricted to less than 30 nucleotides in length. Each domain binds with its complement. In Figure 2.1, domain 1 is shown in pink, domain 2 is shown in blue, and domain 3 is shown in yellow. The complementary domains are 1^*, 2^*, and 3^*, respectively. All domains are assumed to be *orthogonal* in sequence to each other, i.e. strands from different domains exhibit negligible binding to each other. Orthogonality can be achieved by maximizing the *Hamming* distance for every pair of domains [56].

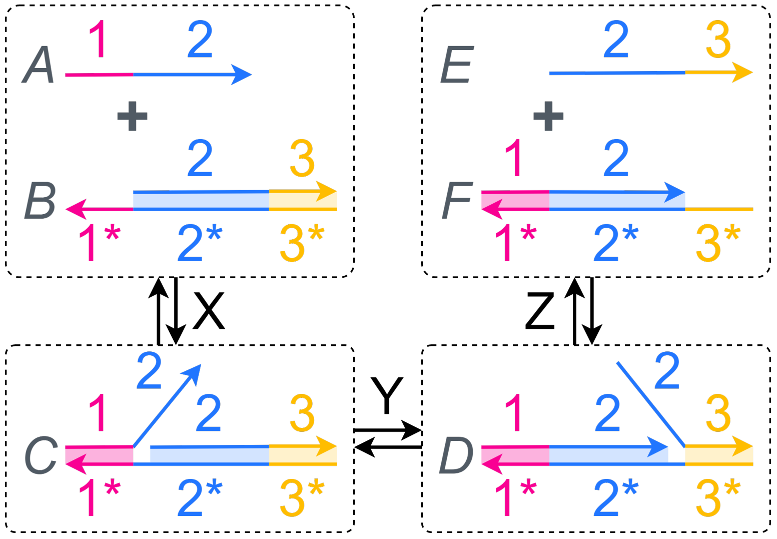


Figure 2.1: The steps of toehold-mediated strand displacement [89]. Each stage of the reaction is shown as a dashed bubble. The reaction starts in the upper left with molecules A and B, goes down to molecule C through reaction X, goes right to D through reaction Y, and finally produces E and F after reaction Z.

A *toehold* is an exposed domain, generally 10 nucleotides in length, on one of the strands in a double-stranded DNA (dsDNA) complex. Toeholds can be created with DNA nicking enzymes such as CRISPR-Cas9 [87] as in Figure **??**. With two separate applications of guided CRISPR-Cas9, the backbone of the dsDNA can be nicked before and after the toehold. At this point the strand covering the toehold can be released by mild denaturing and elution [67]. For this process, the dsDNA is held in solution by the use of magnetic beads attached to the DNA backbone. A magnetic field is applied during the washing process [55, 29].

Toehold-mediated strand displacement (TMSD) is a particular class of *in vitro* reactions that allows for rate-controlled reactions of DNA molecules with toeholds [110, 92, 97]. A single-stranded DNA (ssDNA) containing the complementary sequence to a toehold can bind at that location. If the ssDNA is longer than the toehold there will be an overhanging *flap*. This flap can displace the adjacent domains in the dsDNA through a process termed *branch migration*. The original ssDNA can completely displace the originally bound portion of the dsDNA releasing a new ssDNA. This new ssDNA can then participate in further displacement reactions, creating a cascade of displacement reactions.

Figure 2.1 illustrates the different steps involved in TMSD: the dsDNA B has 3 domains, of which 1^* is a toehold. The ssDNA A contains domain 1 and can bind to B in reaction X, resulting in molecule C containing an overhanging flap from A. Through branch migration, shown in reaction Y, the A flap can fully bind and displace the original strand to produce D. Eventually this strand can be displaced off entirely in reaction Z to produce ssDNA E and dsDNA F, which now has a new toehold in 3^*. The ssDNA E can also participate in TMSD reactions through domains 2 or 3. The rates and directions for all reactions X,Y, and Z can be controlled by factors such as the length of domains (the binding of longer domains is favorable), the C-G percentage of domains (C-G bonds consist of 3 hydrogen bonds compared to the 2 in A-T bonds and thus favors binding), temperature (denaturing is favorable at higher temperatures), elution and magnetic purification (reducing product concentration drives a reaction forward), and through enzymes (DNAse can degrade DNA to reduce product concentration).

Chapter 3

Parallelized Computing on Data Stored in DNA

This study proposed an encoding system for SIMD DNA computation, suitable for general pairwise operations [8]. The first was a binary bubble sorting algorithm (equivalent to rule 184 with elementary cellular automata [42, 45]). We showed that sorting could be performed in only N parallel steps, where N was the number of bits to be sorted. The second application was a left-shifting operation (equivalent to rule 170 with elementary cellular automata), performed in a single parallel step. The third application was a parallel search algorithm that checked if a query substring was present in a target string. In principle, the algorithm could return an answer in $\log(n)$ steps, but our implementation required between $\log(n)$ and n steps to complete, where n was the length of the query string. This paper expands upon this encoding system with a new application, a parallel Exclusive OR calculation. This XOR operation requires at most N steps to compute the XOR of N bits. All 4 of these applications are of immediate practical interest, as many forms of computation on stored data entail some form of sorting, XOR, shifting, and searching.

This work was funded by DARPA grant #W911NF-18-2-0032.

3.1 Parallel computation using SIMD

SIMD is a computer engineering acronym for Single Instruction, Multiple Data [19], a form of computation in which multiple processing elements perform the same operation on multiple data points simultaneously. It contrasts with the more general class of parallel computation called MIMD (Multiple Instructions, Multiple Data), where multiple processing elements can perform completely different operations on multiple data points simultaneously. While general MIMD parallelism might be desirable, it is often not practical. Much of the modern progress in electronic computing power has come by scaling up SIMD computation with platforms such as graphical processing units (GPUs).

SIMD implemented on DNA is intriguing. It provides a means to transform stored data, perhaps large amounts of it, with a single parallel instruction [103]. Computation using SIMD DNA is predicated on the encoding scheme for data. Data is stored in the form of bits on long strands of double-stranded DNA constructed using "domains" as defined in Section 2.1. A sequence of several (typically 5 to 7) domains maps to a "cell" storing one binary bit. Whether a cell stores a 0 or a 1 depends upon topological variations, specifically the location of nicks, i.e., breaks in the DNA backbone. The nicks always occur on one strand of a double-stranded complex (generally the top strand in most examples); the other remains untouched.

The computation is carried out by a sequence of "instructions", where each instruction implements DNA strand displacement reactions on cells. Instructions are initiated by single-stranded "instruction strands" added to the solution. After the strand displacement cascades complete, all freely floating fragments in the solution are washed away; the original strand is kept and separated via a magnetic bead. After a sequence of instructions, the data is transformed to its final state. The readout can be performed via fluorescence or with Oxford nanopore devices [4], [47].

This study's approach to computation is summarized as follows and illustrated in Figure 3.1.

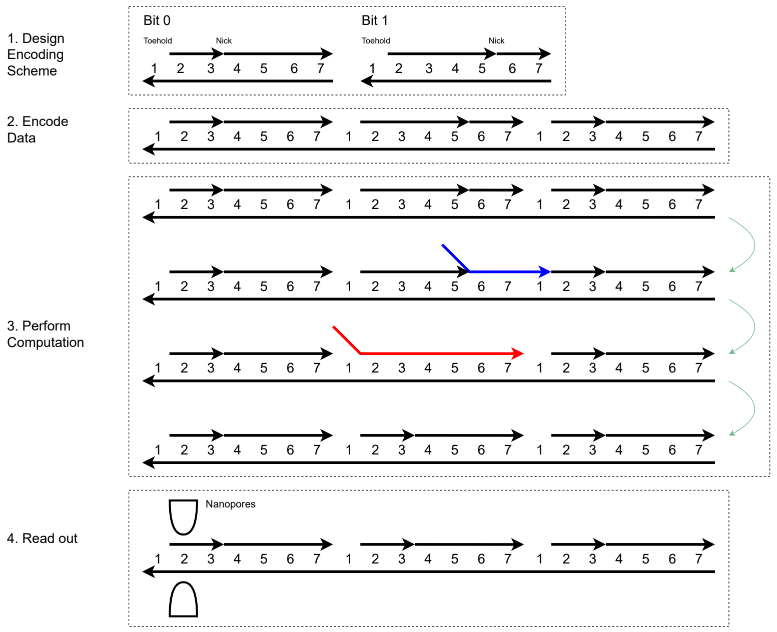


Figure 3.1: General Outline of SIMD DNA Computations. Stage 1 shows the encoding of binary bits 0 and 1 across the 7 domains per bit. Stage 2 shows an example of encoding the bits 010. Stage 3 illustrates the step in which computation is performed with strand displacement, in a general sense. Details of this step will be provided for specific algorithms in later sections. Note that, in this generic example, the location of nick in the second cell has changed at the end of stage 3. Stage 4 illustrates how nanopore sequencing could be used to perform readout.

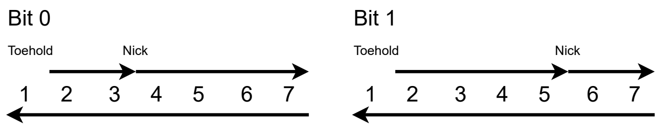


Figure 3.2: Bit representation in the encoding scheme. Horizontal lines represent DNA strands. Integers represent “domains”: specific sequences of nucleotides. Arrow heads represent nicked positions: places where the phosphodiester bond in the backbone of the DNA strand has been broken, via gene-editing techniques. Cells store binary values. Each cell consists of 7 domains. Domain 1 is always exposed, forming a toehold.

1. Design an encoding structure that best suits the algorithm.
2. Encode the data at specific locations, using enzymes to nick corresponding targets.
3. Gently denature the DNA, allowing segments between adjacent nicks to detach, exposing toeholds.
4. Execute instructions, implemented as strand-displacement operations.
5. Finally, read out data using fluorescence or with nanopores.

3.2 Design of Encoding System

Several schemes for encoding binary data were proposed in prior work [103], each chosen to minimize the number of operations for a specific algorithm. This study proposed a new encoding scheme that worked well for the broad class of algorithms that consist of parallel operations on pairs of bits. A requirement for running these algorithms was that the encoding scheme allowed the algorithm to read bits adjacent to each other. This specification came at the expense of more complexity for some algorithms, i.e., more operations per step than possible with a customized encoding.

The encoding scheme is shown in Figure 3.2. Each cell stores a single binary value (a “bit”). Each cell consists of 7 domains. The actual nucleotide sequence of the domains is not specified here for simplicity. While preparing this cell, the top DNA strand must be nicked before and after domain 1. This strand can then be displaced by denaturing, creating an exposed toehold. Domain 1 is always exposed as a toehold in this representation. Domains 2 through 7 are covered. When storing a bit 0, the top strand will be nicked between domains 3 and 4; when storing a bit 1, the nick will be between domains 5 and 6. There are four possible pairings for two adjacent cells. Each will be detected using different domain combinations: for $(0, 0)$, domains 1, 2 and 3; for $(0, 1)$, domain 1 only; for $(1, 0)$, domains 6 through 3 with wrapping at domain 7 and 1; and for $(1, 1)$, domains 6, 7 and 1.

Before delving into the various algorithms possible with this encoding, some general algorithmic steps are presented first.

3.2.1 Identifying Bit Pairs

A common task in the algorithms in this study was "identifying" pairs of adjacent bits, i.e., recognizing the specific pair of cells at a location of interest. The fact that domain 1 was always exposed was exploited to identify these specific pairs. Figure 3.3 illustrates the approach on the string 11001, which contains all 4 possible adjacent pairs: $00, 01, 10$ and 11.

Identification was performed with three instructions. In instruction 1, the strands (S_1 6 7 1 2 3) were issued to all pairs of bits. S_1 first bound at the toehold of domain 1, between each pair. If this preceding bit had a value of 1, there would have been a nick between domains 5 and 6. Through branch migration, the left side of S_1 (i.e., the (S_1 6 7) part) would have displaced the original strand covering domains 6 and 7 of the preceding bit. This is shown in Figure 3.3, instructions 1 and 2. If the value of the following bit were 0, there would have been a nick between domains 3 and 4. Through branch migration, the right side of S_1 (i.e., the (S_1 2 3) part) would displace the original strand covering domains 2 and 3. This is shown in Figure 3.3, instructions 1 and 2. Only if the preceding bit was 1 and the following bit was 0 would S_1 displace *both* these strands. For the pair $(1, 1)$, domains 2 and 3 of S_1 would be left overhanging. For the pair $(0, 0)$, domains 6 and 7 of S_1 would be left overhanging. For the pair $(0, 1)$ S_1 would not bind at

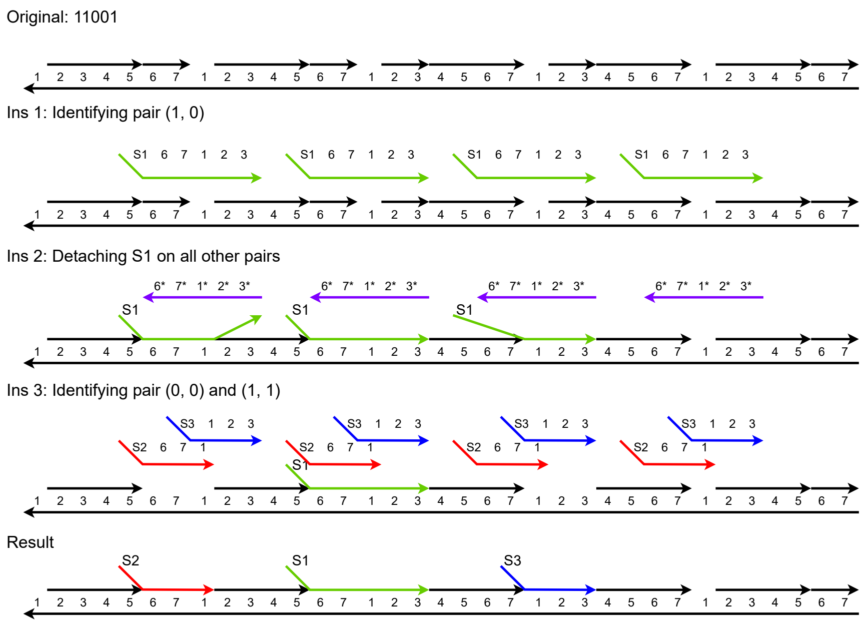

Original: 11001
Ins 1: Identifying pair (1, 0)
S1 6 7 1 2 3
Ins 2: Detaching S1 on all other pairs
6* 7* 1* 2* 3*
S1
Ins 3: Identifying pair (0, 0) and (1, 1)
S3 1 2 3
S2 6 7 1
Result
S2
S1
S3
1 2 3 4 5 6 7

Figure 3.3: Example of Identifying Different Pairs of Adjacent Bits.

all, since the only exposed toehold was domain 1. This is how the algorithm *identified* the pair $(1, 0)$.

In instruction 2, using the complementary strands (6* 7* 1* 2* 3*), the strand S_1 that attached to the pairs $(0, 0)$ and $(1, 1)$ was pulled out. This was done through the open domains 2 and 3 in the pair $(0, 0)$ and the open domains 6 and 7 in the pair $(1, 1)$ on strand S_1. After this instruction, strand S_1 remained only for the pair $(1, 0)$.

In instruction 3, two instruction strands were issued at the same time: (S_2 6 7 1) and (S_3 1 2 3). Here (S_2 6 7 1) would bind to the pair $(1, 1)$ and (S_3 1 2 3) would bind to the pair $(0, 0)$. They would not bind with any other pairs since the only exposed toehold for binding would be domain 1; they would prefer the locations with more exposed domains.

The result was that the adjacent bit pairs $(1, 1)$, $(1, 0)$ and $(0, 0)$ were each *labeled* with strands S_2, S_1 and S_3 respectively. Pairs $(0, 1)$ were labeled with an exposed toehold at domain 1. This toehold could be replaced by a strand (S_x 4 5 6 7 1) or a strand (S_x 1 2 3 4 5); the choice would be made

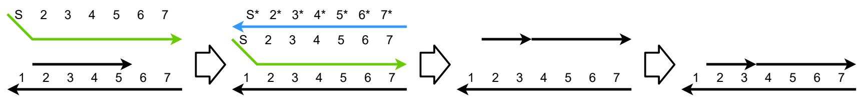


Figure 3.4: Example of Rewriting in Three Steps

depending on the use case.

3.2.2 Rewriting a cell

By exposing toeholds across domains 2 through 7 in a cell, one can rewrite the content of that cell – so change a 1 to 0 or a 0 to 1 – with three instructions. The idea is that since there are exposed domains, the content of the cell can be displaced with a single strand covering all these domains. Then that covering strand would be removed through the exposed "tag" domain (S in Figure 3.4) using a complementary strand. This would leave the cell completely exposed. A new bit value could then be written into it by hybridizing the strands according to a new encoding scheme, leaving domain 1 as a toehold and placing the nick at the desired location.

3.3 Parallel Binary Bubble Sorting

Sorting is a simple yet fundamental operation in computer science. This study focuses on sorting binary values. Sorting can be used to determine the "weight" of a vector of 0's and 1's: the count of the number of 1's relative to the length of the vector. It can also be used to compute the majority function: whether there are more 1's than 0's or not in the input set. Majority is a fundamental operation for many machine-learning algorithms.

This SIMD DNA implementation performed parallel bubble sorting on binary bits [13]. It was expressed as a pairwise operation in the form of $f(a, b) = (c, d)$, where (a, b) was the value of the input bit pair, and (c, d), the output pair. f represented the action taken on a given bit pair – to rewrite or to leave it as it is. The kinds of pairwise operations that could be performed with our encoding are discussed in Section 3.7.2.

The sorting operation was expressed in the following pairwise operation,

$$f(0,0) = (0,0), \quad f(0,1) = (0,1), \quad f(1,0) = (0,1), \quad f(1,1) = (1,1).$$

Note that only the third input pair $(1,0)$ required rewritting to $(0,1)$. The rest of the bit pairs did not need to be changed.

The following "bit swapping" explains how the sorting is performed:

- If the current bit is 1, it changes it to 0 if and only if its right neighbor is 0.
- If the current bit is 0, it changes it to 1 if and only if its left neighbor is 1.

Repeatedly performing such bit-swapping sorts the entire sequence of binary values. An example is provided in Section 10.1.6.

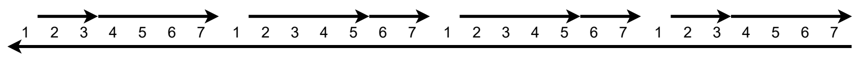
Lemma 1. *The $f(1,0) = (0,1)$ pairwise operation can only occur once in any sequence of three bits.*

Proof. It is impossible to have two consecutive, overlapping pairs $(1,0)$ spanning three bits. Therefore, the $f(1,0) = (0,1)$ pairwise operation (i.e., the bit-swap step) can only occur once in any sequence of three bits. Consequently, the bubble sort algorithm only performs non-conflicting pairwise operations. (Please see Section 3.7.2 for more details.) $\square$

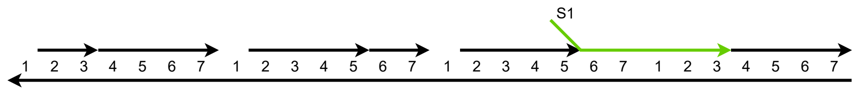
Accordingly, bubble sorting binary values in parallel did not require an odd and even index addressing scheme, as did bubble sorting arbitrary values.

Lemma 2. *Sorting completes in at most $N-1$ parallel steps where N is the total number of bits.*

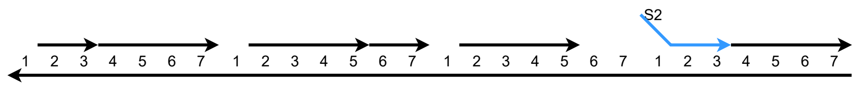
Proof. Suppose a sequence of binary bits of length N, in which all bits except the first are 0. When applying the algorithm, the 1 located at the start will be pushed back one position at a time with the $f(1,0) = (0,1)$ bit swap



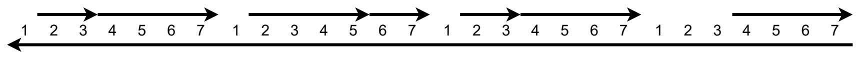
(a) Initial Sequence 0110



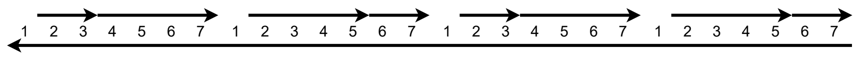
(b) After Recognizing $(1, 0)$



(c) Protection on Bit 0



(d) Flipped third bit to 0, Protection Removed



(e) Flipped fourth bit to 1, Result 0101

Figure 3.5: Outline of the SIMD DNA parallel binary sorting algorithm.

operation. Fully sorting the sequence, i.e., moving the 1 to the last position, requires $N - 1$ total swaps. Now suppose an arbitrary bit sequence being sorted. After $N - 1$ swaps, all the 1's will be at the end of the sequence. To see why, note that an $f(1, 0) = (0, 1)$ operation moves a 1 forward, while an $f(1, 1) = (1, 1)$ operation does not affect adjacent 1's. Thus, in $N - 1$ steps, all 1's will have moved to the end of the sequence. □

3.3.1 Implementation

Here is the instruction set for performing parallel binary bubble sort with SIMD DNA, using the encoding in Figure 3.2. It consisted of 12 individual instructions. These are summarized as follows:

1. Label pairs $(1, 0)$.

2. Uncover these, leaving domains 6 and 7 for the bits 1 and domains 2 and 3 for the bits 0 open in these pairs.

3. Protect the bits 0 of these pairs by covering the corresponding toehold at domains 2 and 3.

4. Flip the bits 1 to 0 in these pairs.

5. Release the protective covers; flip the bits 0 to 1 in these pairs.

For the initialization, the first two instructions described in Section 3.2.1 were used, with an additional instruction to fix open domains for bits that did not change. The rewriting method described in Section 3.2.2 was then used to flip the bits. A full description of the implementation of sorting is provided in Section 10.1.2.

3.4 Parallel Exclusive OR

The Exclusive OR operation, shortened to XOR, is a useful bit operation with many applications, including in error correction. Simply put, a multi-input XOR operation checks if there are an odd number of 1's in the input bits. With this SIMD DNA implementation, it was possible to compute the XOR of all bits on a strand. This was achieved with the following pairwise operations per step of the algorithm:

$$f(0,0) = (0,0), \quad f(0,1) = (0,1), \quad f(1,0) = (0,1), \quad f(1,1) = (0,0).$$

These pairwise operations were mostly similar to the ones for the parallel bubble sorting, in particular with the $f(1,0) = (0,1)$ that sorted all 1's and pushed them to the right. However, the $f(1,1) = (0,0)$ modification ensured that any contiguous pair of cells both containing 1 were overwritten to 0. This meant that after every individual step of the XOR algorithm, the parity of 1's was preserved, and all 1's were shifted one bit towards the right. After a certain number of steps, the last bit on the strand stored the XOR output. One issue that had to be addressed with this algorithm was how cells were paired per step. For example, if the triplet of cells (1,1,1) was recognized as two pairs of $(1,1)$, then the overwriting step would change all three 1's to 0 and break the parity of the sequence. To avoid this, all DNA strand instructions identifying (1,1) pairs were run on non-overlapping pairs of cells at each step. For example, the first step operated on pairs of cells i and $i+1$ where i was an even number, and then the next step operated on pairs i and $i+1$ where i was an odd number. However, the $f(1,0) = (0,1)$ operation could still be run on overlapping pairs.

To perform such operations, i.e., one iteration on only even-to-odd pairings and the next iteration on only odd-to-even pairings, the cells had unique sequences. The even and odd cells had different DNA base sequences – all even cells were based on one sequence, and all odd cells were based on another, distinct sequence. The nicking architecture, shown in Figure 3.2, applied despite the different base sequences. Changing the sequences ensured that instruction strands coudl be synthesized to bind to the appropriate i and $i+1$ cells discussed above.

Each iteration of the XOR algorithm is thus as follows:

- Determine a pairing that is offset by 1 cell compared to the previous iteration's pairing.
- Detect all non-overlapping pairs of (1,1) and convert them to (0,0).
- Detect all pairs of (1,0) and convert them to (0,1). For this writing process, pairs can be overlapping.

Each of these iterations pushes all 1's to the end of the strand while also overwriting any adjacent (non-overlapping) pairs of 1's. Therefore, after a sufficient number of iterations, all bits in the strand are 0's except for the last bit – that bit will only be 1 if there were an odd number of 1's to begin with. Therefore, the algorithm computes the XOR function.

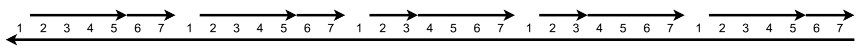
The parallel exclusive OR completes in at most N parallel steps where N is the total number of bits.

Proof. Consider the worst case of an array of N bits with two 1's, one at the start and one at the end. For the correct XOR computation, the first 1 must be moved to the end of the array, which requires at most $N-1$ steps. Then $f(1,1) = (0,0)$ requires one final step for the proper XOR output. Now any extra 1's added to the array occurr between the start and end. These will be moved to an adjacent position in those $N-1$ steps. The $f(1,1) = (0,0)$ operation for these two 1's in the middle of the array does not impede sorting the first 1 in the array. Thus computing the XOR of N bits requires at most 1 step more than the worst-case parallel sorting time. Therefore, any arbitrary array of N bits requires $(N-1)+1 = N$ steps for a correct XOR computation. □

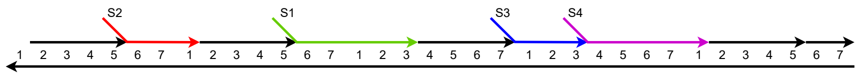
3.4.1 Implementation

The implementation of one step of the parallel exclusive OR with SIMD DNA consisted of 20 individual instructions. The gist of these instructions is summarized as follows.

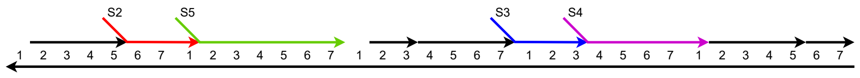
1. Label non-overlapping pairs $(1,1)$.
2. Cover all other pairs.
3. Uncover the identified $(1,1)$ pairs and expose both bits in this pair.
4. Rewrite all uncovered bits to 0.
5. Now label all pairs of $(1,0)$.
6. Uncover the $(1,0)$ pairs.
7. Protect the bits 0 of these pairs by covering the corresponding toehold at domains 2 and 3.
8. Flip the bits 1 to 0 in these pairs.
9. Release the protective covers; flip the bits 0 to 1 in these pairs.



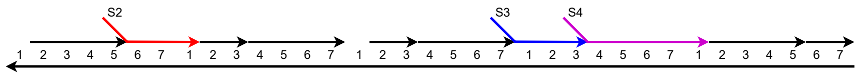
(a) Initial Sequence 11001



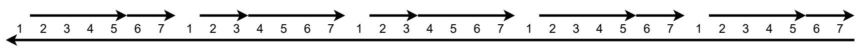
(b) After identifying all pairs



(c) Release S_1 from Pair $(1, 0)$



(d) Rewrite bit 1 in the previous pair with 0



(e) Release S_2, S_3 and S_4 then write 1, Result 10011

Figure 3.6: Outline of the SIMD DNA parallel left shift operations. The initial sequence S is 11001 and the result sequence T is 10011. The operation shifts each bit to the left one position (T[5:1]=S[4:0]), while keeping the Least Significant Bit unchanged.

Please note that in each step, the non-overlapping pairing were offset by one cell compared to the preceding step to ensure all (1,1) pairs were overwritten. A full description of the implementation of the XOR is provided in Section 10.1.3.

3.5 Parallel Left Shifting

Shifting left corresponds to multiplying a binary number by 2; shifting right corresponds to dividing it by 2. It is a useful operation in general for aligning data in a variety of algorithms [13]. Here is presented a left shift algorithm that shifts all N binary bits one position to the left, with the Least Significant Bit (LSB) remaining unchanged. This operation is, of course, a parallel left shift, moving all bits simultaneously in lockstep. The implementation required 11 instructions per shift. Note that unlike usual arithmetic or a logical left shifts that insert a bit 0 to the LSB, the left shift operation described here keeps the original LSB, thereby duplicating the LSB. The usual left shift could be implemented by adding instructions rewriting the LSB to 0 after the instructions.

The shift operation using the following pairwise operation is as follows:

$$f(0,0) = (0,X), \quad f(0,1) = (1,X), \quad f(1,0) = (0,X), \quad f(1,1) = (1,X).$$

Here X is a "don't care" bit value (to use the parlance of digital design). When computing on a specific input bit pair, the output for the X bit is not impacted by that input pair (for example, in left shifting, X is actually calculated from the bit pair to the right since that bit will be shifted to the left). For each bit pair, the operation writes the value of the right bit to the left bit. Since only the value of the left bit is changed in each bit pair, the operation is non-overlapping and can be implemented using the encoding scheme we propose. This is illustrated with the example of shifting 11001 to 10011 in Figure 3.6.

1. Label all the bit pairs. Cover the toeholds for the pairs $(0,0)$ and $(1,1)$.
2. For the pairs $(1,0)$, flip the bits 1 to 0.
3. For the pairs $(0,1)$, flip the bits 0 to 1.
4. Finally, uncover all the toeholds for the pairs $(0,0)$ and $(1,1)$.

A full description of the implementation of shifting is given in Section 10.1.4.

3.6 Parallel Search Algorithm

Searching is fundamental to all branches of computer science that involve data storage and retrieval. The specific type of search focused on in this study was the problem of identifying whether a given substring existed in a stored string of bits. a general algorithm that returns an answer to such a question in $\log(n)$ parallel steps, where n is the substring length, is presented here. Note that a requirement of this algorithm is that the length of the query string is a power of 2. Furthermore due to practical constraints, the SIMD DNA implementation time complexity was not $O(\log(n))$; it was closer to $O(n)$. Details on this algorithm are discussed in Section 3.7.4.

The pseudocode for the search algorithm can be found in [8]. Here the algorithm will be explained through the use of the following examples.

3.6.1 Parallel search procedure

Consider searching for a query string $Q = 1101$ in the following target string A:

$$\begin{aligned} A_0 &= 10101010110110100011110101000100 \\ A_1 &= a_2a_2a_2a_2a_3a_1a_2a_2a_0a_3a_3a_1a_1a_0a_1a_0 \\ A_2 &= b_0b_0b_1b_0b_2b_1b_3b_3 \end{aligned} \tag{3.1}$$

The original string is A_0. In each step, two consecutive symbols are read and replaced with a single symbol. Here $a_0 = 00, a_1 = 01, a_2 = 10, a_3 = 11, b_0 = a_2a_2, b_1 = a_3a_1, b_2 = a_0a_3, b_3 = a_1a_0$. Note that $Q = 1101 = a_3a_1 = b_1$. After three steps, the query string can be found in the target string since there are two matches in the string A_2.

3.6.2 Search procedure with offset

It is possible that the query string does not align with divisions of length n in the target string. Thus, the operation needs repetitions with offsets. The following example illustrates the operation with an offset of 2 bits.

$$\begin{aligned} A_0 &= \cancel{10}101010110101100000011110001000100 \\ A_1 &= \cancel{10}a_2a_2a_3a_1a_1a_2a_0a_0a_3a_3a_0a_1a_0a_1a_0 \\ A_2 &= \cancel{10}b_0b_1b_2b_3b_4b_5b_5\cancel{a_0} \end{aligned} \tag{3.2}$$

Here, the replacement is given by the aggregated pairs $a_0 = 00, a_1 = 01, a_2 = 10, a_3 = 11, b_0 = a_2a_2, b_1 = a_3a_1, b_2 = a_1a_2, b_3 = a_0a_0, b_4 = a_3a_3, b_5 = a_0a_1$. Again, an instance of the query string is found in the target string.

Searching for a query string with a given offset requires at most $\log(n)$ steps. In general, for an arbitrary query string of a length n (a power of 2), the search must be performed n times with offsets ranging from 0 to $n-1$. In principle, all of these searches could be performed in parallel, as none would interfere with any other. Accordingly, the parallel implementation of searching completes in $\log(n)$ steps.

Note that the number of aggregated pair identifiers needed – the a's and b's in the example above – grows exponentially with the length of the target string. For example, to search for all possible queries of length 2, 4 identifiers are needed. For all queries of length 4, $16 + 4 = 20$ identifiers are needed. The total number of identifiers needed for queries of length n can be formulated as:

$$\sum_{i=1}^{\log(n)} 2^{2^i}.$$

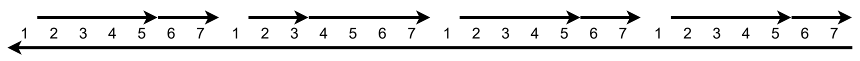
This number grows very quickly as n increases (so for longer query strings). This would seem to be a serious limitation of the search algorithm. However, this calculation assumes the search for *all* possible query strings. If the search is for a *specific* query string, then the number of identifiers required drops considerably. This is because the search only needs to identify pairs in this specific string. The maximum number of identifiers needed is:

$$\sum_{i=1}^{\log(n)} 2^i = n - 1,$$

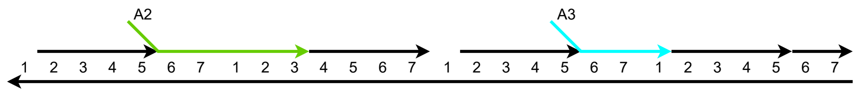
a much more manageable number.

3.6.3 Implementation

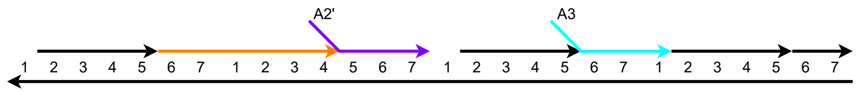
To implement the algorithm in SIMD DNA, instruction strands were not issued to all pairs of overlapping bits. Instead, the query was broken up in discrete pairs. In the example shown in Figure 3.7, for the bit sequence



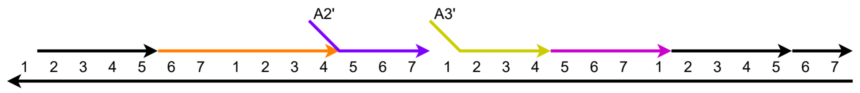
(a) Initial Sequence 1011



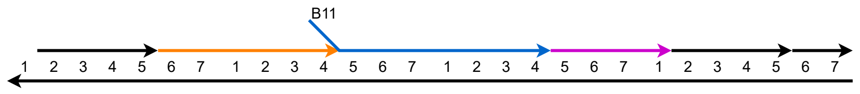
(b) Identifier A_2 captures first pair 10, A_3 captures second pair 11



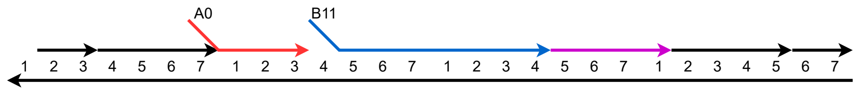
(c) covering the domain 1 between the two bit pairs



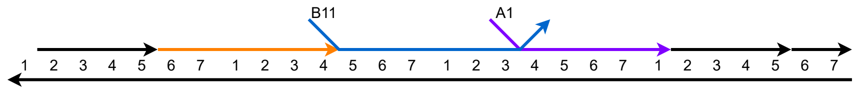
(d) Rewrite the content in the pair so that new identifiers are close to the middle



(e) Two identifier strands replaced by a single identifier if there is a perfect match



(f) Initial sequence is 0011. It will result in an open domain 4 in the cell left of the identifier



(g) Initial sequence is 1010. It will result in an overhanging domain 4 on the identifier strand itself

Figure 3.7: Example implementation of search algorithm on target sequence 1011

1011, operations on bit pair 10 and 11 were considered, but not on bit pair 01.

Figure 3.7 shows the critical steps when searching a target sequence 1011. It provides an example of a successful search and also the potential outcome of two failed searches. To implement the search operation with an offset, one can simply skip the number of bits according to the offset. Assume the word *symbol* represents the consecutive cells that are searchd for on a certain level. For example, in the first level, the symbols are 10 and 11. The bit-identifying steps described in Section 3.2.1 can be used to recognize these symbols. Now identifiers $A_0 = 00, A_1 = 01, A_2 = 10, A_3 = 11$ can be used to represent symbols in this level. Moving on to the next level, searching for consecutive symbols A_2A_3 corresponds to the target string 1011.

In the first step of the second level, first rewrite the topological structure at symbols that appear to be a query result. In this example, A_2 should be found as the left symbol, and A_3 should be found as the second symbol. Identifier A_2 is pulled out from every *odd* symbol (only look at the first, third, fifth, etc.) and the entire symbol is rewritten with the technique described in Section 3.2.2. After rewriting, the identifier A_2' covers domains (5 6 7) in the *right most* cell, as seen in Figure 3.7c. For the second symbol A_3, the step described above is repeated, except the identifier isn pulled out from every *even* symbol and the new identifier A_3' covers domains (2 3 4) in the *left most* cell. Through these steps, the identifier of the matching symbols has essentially been moved to the middle. In the final step, the new identifier strand (B_{11} 5 6 7 1 2 3 4) is issued to the location between every two symbols. It will result in a perfect binding only if there is a match at the current symbol level. Figure 3.7e shows the example of a matching result. Figures 3.7f and 3.7g show two potential examples of imperfect binding, indicating a non-matching result. They can be pulled out through the open domains either on the identifier itself or a nearby open domain on the base strand. Therefore, the presence of the identifier B_{11} indicates a successful match.

The process detailed above can be repeated to recognize multiple symbols at the same level. When moving to the next level $l + 1$, the identifiers from the level l can be used as a starting point for rewriting. To identify a symbol $S_{l+1,c} = S_{l,a}S_{l,b}$ at level $l + 1$, identifiers for $S_{l,a}$ at odd symbols and $S_{l,b}$ at even symbols at level l are pulled. Then the identifier are moved to the middle. Finally, identifiers for $S_{l+1,c}$ are given to the middle of each pair to identify the symbol.

A possible weakness of this implementation was that the strand used for rewriting could potentially be very long. The longer the query string, the longer this strand. The longer the strand, the longer strand displacement would take [77]. The time required would become prohibitive. Another issue was that our search algorithm rewrote, so destroyed, the data on the target strand. While it would be possible to reverse the process, thus restoring the original data, this process would be cumbersome and require multiple steps (see, for example, Figure 10.7). Another limitation was that the algorithm could not readily handle multiple overlapping queries within the target string.

3.7 Discussion

3.7.1 Initializing data on cells sharing the same sequences

Two of the algorithms, namely XOR and searching, relied on different cells having different underlying DNA sequences. This allowed the algorithms to perform pairwise operations that targeted specific cells, leaving others untouched. The other two algorithms, namely sorting and left-shifting, did not have this requirement. As a result, sorting and left-shifting required much smaller libraries of DNA strands. Sorting and left-shifting were true "single instruction multiple data" (SIMD) algorithms while XOR and searching were not. In order to exploit the SIMD aspect of sorting and left-shifting, the underlying DNA sequences had to be identical for all cells. The challenge was that any nicking operation performed on one cell would apply to other cells as well, so one could not readily initialize the base strand with different bit values in different cells.

One approach to overcome this was to utilize Gibson Assembly, a procedure for concatenating small DNA molecules into larger DNA molecules [44]. The small DNA molecules were synthesized with complementary "sticky" single-stranded ends. When these small DNA molecules were mixed in a solution, these sticky ends hybridized, yielding a longer DNA molecule. This approach was used to initialize data in DNA strands where all cells shared the same sequence. Molecules containing only one cell were nicked separately to store either 0 or 1. These molecules were then orderly concatenated to build strands containing multiple cells. Please refer to Section 10.1.7 for a more detailed outline on constructing a register storing two bits.

3.7.2 Ability to compute any non-conflicting pairwise operation

Sections 3.3 and 3.5 presented examples of algorithms that performed pairwise operations respectively. Given the ability to identify pairs of bits and a universal way to rewrite a cell, one can readily implement any algorithm that performs non-conflicting pairwise operations. Such operations only entail rewriting pairs of adjacent bits. The result of the operation on a specific sequence would always be the same, irrespective of the execution order. To illustrate, consider the following operation:

$$f(0,0) = (X,X), \quad f(0,1) = (X,1), \quad f(1,0) = (X,X), \quad f(1,1) = (0,X).$$

Here X indicates a "don't care" bit value – the function f for a specific input pair does not compute the output X. The operation provided above *is* conflicting. To see why, consider its effect on the sequence 011. The second bit should change to 1 when the operation is applied to the first pair $(0,1)$. However, this bit should change to 0 when the operation is applied to the second pair $(1,1)$. Depending on the order of execution, the final result is different. To ensure an operation is non-conflicting, for every three adjacent bits that two operations are performed on, the middle bit should be set to the same value.

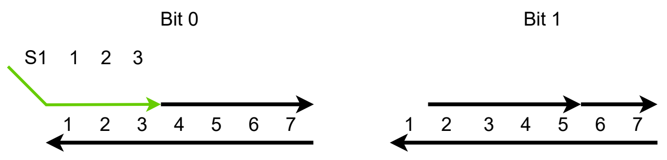


Figure 3.8: One strand can be used to differentiate two bits

Non-conflicting operations can be performed in parallel on all bit pairs. The first step identifies the four bit pairs described in Section 3.2.1. After this step, strands with four labels are supplied to cover the four bit pairs. Then, strands with specific labels are released one at a time to obtain write access to specific bit pairs. (Write access refers to a domain being exposed.) These cells are then rewritten with the operation described in Section 3.2.2. The full operation requires rewriting all four bit pairs.

This study's encoding scheme and design method are generally applicable to parallel bitwise algorithms, provided that they can be expressed in terms of such non-conflicted pairwise operations.

3.7.3 Converting to Different Encoding Schemes

A benefit of the proposed encoding scheme was that it could easily be converted to any other similar scheme since each cell always had an exposed domain 1. In the original SIMD DNA scheme proposed in [103], the authors designed two specific encoding schemes for the two applications proposed (rule 110 and a binary counter). This study's encoding scheme could be used as an intermediate form when converting to other encoding schemes, designed for particular algorithms. Figure 3.8 illustrates how a single strand (S_1 1 2 3) was used to differentiate bit values of 0 from bit values of 1. Technique discussed in Section 3.2.2 could then be used to re-write the data with a different encoding scheme, so long as the scheme also encoded each bit with 7 domains. Complete instructions for performing such encoding changes are given in Section 10.1.1.

3.7.4 Time Complexity of Parallel Search

While the time complexity of the proposed parallel search was $O(\log(n))$ in principle, where n was the query substring length, the time complexity of the SIMD DNA implementation was somewhat worse. While the abstract search algorithm found the query in the reference string by pairing individual characters in parallel, and thus completed in $O(\log(n))$ steps, the DNA implementation searched for and identified distinct symbols sequentially, that is to say, it first searched for a specific symbol across all possible locations at once, then it searched for the next symbol across all locations at once, and so on.

The abstract algorithm assumed all symbols were identified in one pass to allow for further pairing. Considering all the different symbols in a query string, counting repeated symbols, $\frac{n}{2^i}$ symbols must be searched sequentially at level i in the SIMD DNA implementation. Accordingly, the total number of sequential search steps was as high as $O(n)$. However, at each level, all the occurrences of a specific symbol were identified simultaneously. At level i, each symbol represented a binary string with a length of 2^i, so there were at most 2^{2^i} distinct symbols at level i. For example, in the first level, instead of searching for $\frac{n}{2}$ symbols, only four distinct symbols were searched for. In the second level, there are only 16 distinct symbols. Since only distinct symbols were searched, the number of steps in the first few levels was greatly reduced.

The parallel search algorithm only worked on query strings with a length that is a power of two. However, the implementation could be modified to allow for arbitrary-length query strings. Though the details were not provided in this study, they can be summarized as below.

Note that, in parallel search, the query string is searched reductively: at each level, two symbols are reduced to one symbol. When working with query strings having any arbitrary length, there might be an odd number of symbols in the current level. In this case, a method to identify the trailing odd symbol at the current level can be added. It can then be replaced in the next level. The reduction can still be completed in a logarithmic number of levels.

3.8 Conclusion

Chapter 4

Stochastic Computing on Data Stored in DNA

This study explores a link between the stochastic logic and molecular computing. Specifically, it presents a strategy for computing mathematical functions with molecular reactions by applying concepts from stochastic logic. Consider the following example: Suppose a chemical reaction network (defined in Section 4.2) that computes the function

$$f(a,b) = 1 - a - b + ab,$$

where a and b are real-valued variables. The corresponding digital function for stochastic logic can be obtained using the methods discussed in **??**. In this case, it is $f(a,b) = \mathrm{NOR}(a,b)$, expressed in the following truth table:

a	b	$\mathrm{NOR}(a,b)$
0	0	1
0	1	0
1	0	0
1	1	0

To represent a stochastic variable x that ranges from $[0,1]$ in a molecular format, a *pair* of chemical species X_0 and X_1 is used. As will be discussed in **??**, we use a *fractional* representation:

$$x = \frac{[\mathbf{X}_1]}{[\mathbf{X}_0] + [\mathbf{X}_1]}. \tag{4.1}$$

Here $[X_1]$ denotes the concentration of the molecular species X_1. Using this representation, we obtain a chemical reaction network (CRN) from the truth table above:

$$\begin{aligned}
\mathbf{A}_0 + \mathbf{B}_0 &\rightarrow \mathbf{C}_1 \\
\mathbf{A}_0 + \mathbf{B}_1 &\rightarrow \mathbf{C}_0 \\
\mathbf{A}_1 + \mathbf{B}_0 &\rightarrow \mathbf{C}_0 \\
\mathbf{A}_1 + \mathbf{B}_1 &\rightarrow \mathbf{C}_0
\end{aligned} \tag{4.2}$$

Note that the subscripts of the species match the entries of the truth table above. This CRN computes the target function, $c = 1 - a - b + ab$, in terms of the fractional variables a, b and c. Each of these corresponds to a pair of chemical species, $\{A_0, A_1\}, \{B_0, B_1\}$ and $\{C_0, C_1\}$, respectively. The central result of this paper, presented in Section 4.6, is a proof that we can implement any polynomial function, specified by a truth table, with a CRN matching its truth table template.

This study builds upon prior work, both generalizing and simplifying it.The same formalism, namely a fractional representation of values, is used in this paper as was used in [83, 82].

- [83] proposed a technique for computing functions based on a decomposition with *Bernstein polynomials* [6]. The technique implemented a broad class of functions, namely all univariate polynomials, but was quite abstruse. A target polynomial was first repackaged in Bernstein form [75]. This form was implemented in a logic circuit using a form of generalized *multiplexing* [73]. Finally, the logic circuit was translated into a CRN.

- [82] proposed an alternative technique based on factoring of polynomials with *Horner's rule.* The factored form was implemented with a cascade of 2-input logic gates. Finally, the logic gate circuit aws translated into a CRN. Although conceptually simpler than working with Bernstein polynomials, this approach was not quite so general: only a small subset of polynomials can be decomposed in the requisite way with Horner's rule.

A significant limitation of both prior approaches is the complexity of the mathematical formulation.

The approach in this paper is conceptually much simpler and cleaner. As with the NOR function example above, a target polynomial function is first mapped to a truth table. This can be done using fairly standard techniques – at least for people familiar with the theory of stochastic logic – and the results are intuitive. Then a CRN is constructed that matches the template of the truth table.

This approach is also more general. Whereas the method in [83] is limited to univariate polynomials, the method in this paper can implement any multivariate polynomial. Stochastic logic operates on functions where the domain and codomain are in the interval $[0, 1]$, i.e., the inputs and the output are probabilities. Common transcendental functions can be computed via polynomial approximations. In Supplementary Information S1, we provide CRNs for stochastic functions such as arctan, exponential, Bessel, and sinc to demonstrate our approach in detail. These functions have practical applications in fields such as machine learning, signal processing, and image processing. We discuss the implementation of these abstract chemical reaction networks with DNA strand displacement, with units called DNA concatemers.

This study is organized as follows. **??** presents background information on chemical reaction networks and stochastic logic. Section 4.5 describes our methodology for translating any function computed by a stochastic logic circuit into a set of chemical reactions. Section 4.6 provides a proof that the proposed methodology is mathematically sound, based on an analysis of the chemical kinetics. Section 4.7 analyzes sources of error stemming from differences in reaction rates in one particular case. Section 4.8 discusses the implementation with DNA strand displacement through DNA "concatemers". These concatemers implement the generic chemical reaction networks presented in the early sections. Finally, Section 4.9 provides concluding remarks and discusses future research directions.

4.1 Background

4.2 Chemical Reaction Networks

A chemical reaction network (CRN) consists of a set of reactions operating on a set of molecules. When a reaction fires, *reactant* molecules are transformed into *product* molecules. For instance, consider the reaction:

$$\mathbf{X}_1 + \mathbf{X}_2 \xrightarrow{k} \mathbf{X}_3.$$

Here one molecule of reactant $\mathbf{X}_1$ combines with one molecule of reactant $\mathbf{X}_2$, resulting in one molecule of the product $\mathbf{X}_3$. The parameter k is called the *rate constant.* A CRN consists of multiple reactions occurring simultaneously. Consider the following example of a CRN with three reactions operating on the molecule species set $\{\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \mathbf{X}_4\}$:

$$\begin{aligned}
\mathbf{X}_1 + \mathbf{X}_2 &\rightarrow \mathbf{X}_3, \\
\mathbf{X}_2 + \mathbf{X}_3 &\rightarrow 2\mathbf{X}_4, \\
3\mathbf{X}_3 + \mathbf{X}_4 &\rightarrow \mathbf{X}_1.
\end{aligned}$$

Assume that all three reactions have the same rate constant, k, an arbitrary value. To quantify the changes in concentration of all the molecular species involved in a CRN over time, the theory of *mass-action kinetics* [33] is applied: reaction rates are proportional to both the concentrations of the reactants and their rate constants. Given a CRN, a set of nonlinear differential equations for the concentrations of all molecular species can be derived. For instance, for the first reaction above, the rate of change of the concentrations of $\mathbf{X}_1$, $\mathbf{X}_2$ and $\mathbf{X}_3$ is

$$-\frac{d[\mathbf{X}_1]}{dt} = -\frac{d[\mathbf{X}_2]}{dt} = \frac{d[\mathbf{X}_3]}{dt} = k[\mathbf{X}_1][\mathbf{X}_2], \tag{4.3}$$

where $[\mathbf{X}]$ denotes the concentration of the chemical species $\mathbf{X}$. Given the initial concentration of the different molecular species, one can predict the behavior of the CRN by simulating the differential equations.

4.3 Digital Logic

The smallest unit of storage in digital logic is a bit. Bits store Boolean values – either 0 or a 1. The following definitions are pertinent when discussing digital logic:

Definition 1. *An n-input* ***combinational logic function*** *is a function*

$$F(X_1, X_2, \ldots, X_n) = Y,$$

where all inputs and outputs are Boolean values. That is, $\forall\, 1 \leq i \leq n, X_i \in \{0,1\}, Y \in \{0,1\}$.

Definition 2. *The* ***truth table*** *of a combinational logic function lists all the possible combinations of its Boolean inputs and the corresponding outputs. Each combination of inputs is called a* ***minterm****.*

Table 4.2 gives an example of the truth table of a combinational logic function.

4.4 Stochastic Logic

Stochastic logic is an active topic of research in digital design, with applications to emerging technologies [21, 73, 68]. In this paradigm, computation is performed with familiar digital constructs, such as AND, OR, and NOT gates. However, instead of having specific Boolean values of 0 and 1, the inputs are random bitstreams. A number x $(0 \leq x \leq 1)$ corresponds to a sequence of random bits. Each bit has *probability* x of being one and probability $1-x$ of being zero, as illustrated in Figure 4.1. Computation is recast in terms of the probabilities observed in these streams.

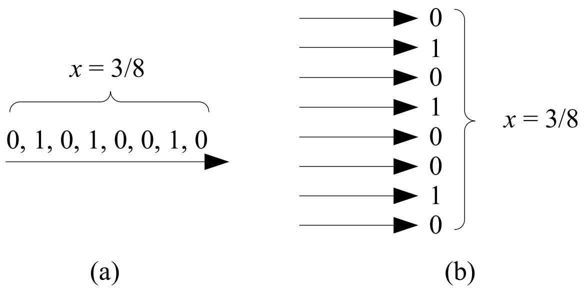


Figure 4.1: Stochastic representation: A random bitstream. A value $x \in [0, 1]$, in this case $3/8$, is represented as a bitstream. The probability that a randomly sampled bit in the stream is one is $x = 3/8$; the probability that it is zero is $1 - x = 5/8$. The bits in the streams are separated temporally (a) or spatially (b).

Consider basic logic gates. Given a stochastic input x, a NOT gate implements the function

$$\mathrm{NOT}(x) = 1 - x.$$

This means that while an individual input of 1 results in an output of 0 for the NOT gate (and vice versa), statistically, for a random bitstream that encodes the stochastic value x, the NOT gate output is a new bitstream that encodes $1 - x$.

The output of an AND gate is 1 only if all the inputs are simultaneously 1. The probability of the output being 1 is thus the probability of all the inputs being 1. Therefore, an AND gate implements the stochastic function:

$$\mathrm{AND}(x, y) = xy,$$

that is to say, multiplication. Similarly, the output of an OR gate is 0 only if all the inputs are 0. Therefore, an OR gate implements the stochastic function:

$$\mathrm{OR}(x, y) = 1 - (1 - x)(1 - y) = x + y - xy. \tag{4.4}$$

The XOR, NAND, NOR, and XNOR gates can be derived by composing the AND, OR, and XOR gates each with a NOT gate, respectively. Please refer to Table 4.3 for a full list of the algebraic expressions of these gates. An important assumption in stochastic computation is that all inputs are independent of each other, i.e., the random bitstreams are uncorrelated.

Table 4.1: Stochastic Function Implemented by Basic Logic Gates

Gate	Inputs	Function
NOT	x	$1 - x$
AND	x, y	xy
OR	x, y	$x + y - xy$
NAND	x, y	$1 - xy$
NOR	x, y	$1 - x - y + xy$
XOR	x, y	$x + y - 2xy$
XNOR	x, y	$1 - x - y + 2xy$

Definition 3. *An n-input* ***stochastic logic function*** $y = f(x_1, x_2, \ldots, x_n)$, *where* $\forall\, x_i \in [0, 1]$ *and* $y \in [0, 1]$, *is obtained from a combinational logic function* $Y = F(X_1, X_2, \ldots, X_n)$, *by setting corresponding inputs to be independent random variables* X_i *with* $\Pr(X_i = 1) = x_i$.

For a given Boolean circuit, its stochastic function can be computed as follows [74].

Theorem 1. *Given input sequences generated by independent Bernoulli random variables, the output of a stochastic logic function will also be a sequence generated by a Bernoulli random variable. The probability of the output of a stochastic logic function f being 1 is the sum of all the probabilities of the minterms that evaluate to 1 in the corresponding combination logic function F. That is,*

$$\Pr(Y = 1) = \sum_{J \in S} \left(\prod_{h=1}^{n} [\Pr(X_h = j_h)] \right) \tag{4.5}$$

where $J = (j_1, j_2, \ldots, j_n), j_i \in \{0, 1\}$ is a minterm, and $S = \{J|F(J) = 1\}$ is the set of minterms that evaluate to 1.

For example, consider a combinational circuit computing a function $F(X_1, X_2, X_3)$ with the truth table shown in Table 4.2. Let $f(x_1, x_2, x_3)$ be the stochastic function computed by this circuit, with real-valued inputs $x_1, x_2, x_3 \in [0, 1]$. Assuming each input is independent of the others, set

$$\begin{aligned}[\mathrm{P}r(X_1) = 1] = &\quad x_1,\\ [\mathrm{P}r(X_2) = 1] = &\quad x_2,\\ [\mathrm{P}r(X_3) = 1] = &\quad x_3.\end{aligned}$$

Table 4.2: Truth table for a combinational circuit, and the corresponding probability of each row.

X_1	X_2	X_3	$F(X_1, X_2, X_3)$	Probability of row
0	0	0	0	$(1 - x_1) \cdot (1 - x_2) \cdot (1 - x_3)$
0	0	1	1	$(1 - x_1) \cdot (1 - x_2) \cdot x_3$
0	1	0	0	$(1 - x_1) \cdot x_2 \cdot (1 - x_3)$
0	1	1	1	$(1 - x_1) \cdot x_2 \cdot x_3$
1	0	0	0	$x_1 \cdot (1 - x_2) \cdot (1 - x_3)$
1	0	1	1	$x_1 \cdot (1 - x_2) \cdot x_3$
1	1	0	1	$x_1 \cdot x_2 \cdot (1 - x_3)$
1	1	1	1	$x_1 \cdot x_2 \cdot x_3$

The probability that the function f evaluates to 1 is equal to the sum of the probabilities of occurrence of each row that evaluates to 1. The probability of occurrence of each row, in turn, is obtained from the assignments to the variables, as shown in Table 4.2: x_i if the corresponding variable X_i is 1 and $(1 - x_i)$ if it is 0. Now upon selecting the rows in Table 4.2 where $F(X_1, X_2, X_3) = 1$ and adding their probabilities together to obtain the expression for the stochastic function:

$$
\begin{aligned}
f(x_1, x_2, x_3) = &(1 - x_1)(1 - x_2)x_3 \quad + \\
&(1 - x_1)x_2x_3 \quad + \\
&x_1(1 - x_2)x_3 \quad + \\
&x_1x_2(1 - x_3) \quad + \\
&x_1x_2x_3 \\
= &(1 - x_2)x_3 + x_2x_3 + x_1x_2(1 - x_3).
\end{aligned}
\tag{4.6}
$$

The procedure shown for this example can be generalized to any combinational circuit to evaluate its stochastic function. Such probabilistic analysis of networks of logic gates is not new. As early as 1975, the circuit testing community had begun analyzing errors in a similar way [69, 84]. Similar techniques have also been applied to tasks such as timing and power analysis [46, 51]. However, characterizing the *outputs* of the computation this way, as probabilistic functions, is specific to the field of stochastic logic. [75] proved that any multivariate polynomial function with its domain and codomain in the unit interval $[0, 1]$ can be implemented using stochastic logic. [73] provided an efficient and general synthesis procedure for stochastic logic, the first in the field. [76] provided a method for transforming probabilities values with digital logic. Finally, [36, 59] demonstrated how stochastic computation can be performed deterministically.

4.5 Implementing Stochastic Logic with Chemical Reactions

In the introduction, we gave a brief example of translating a simple polynomial function, the NOR function, into a CRN. In this section, we step through the details of this process.

4.5.1 Fractional Representation in Solution

Two distinct molecular species $\mathbf{X}_0$ and $\mathbf{X}_1$ are used to represent a stochastic value x in a chemical system.

$$x = \frac{[\mathbf{X}_1]}{[\mathbf{X}_0] + [\mathbf{X}_1]}. \tag{4.7}$$

Here the notation $[\mathbf{X}]$ refers to the concentration of a molecular species $\mathbf{X}$. This fractional representation in prior work [83, 82]: the value x equals the ratio of the concentration of $\mathbf{X_1}$ to the total concentration of $\mathbf{X_0}$ and $\mathbf{X_1}$. As with probabilities in stochastic logic, such a fractional value can represent any real number in the unit interval $[0, 1]$.

4.5.2 Building a Chemical Reaction Network from a Truth Table

Using the fractional representational discussed in Section 4.5.1, any truth table for a combinational logic function can be transformed into a CRN. Each row of the truth table is used as a reaction, where the reactants represent a minterm, and the product represents the output of the logic function for that minterm. For example, consider the truth table for the Boolean AND operation:

A	B	$C =$AND(A, B)
0	0	0
0	1	0
1	0	0
1	1	1

Now the corresponding CRN for this truth table is generated by transforming each row into its own reaction using the fractional notation discussed in Section 4.5.1. Given the fractional representation described above, let us design a CRN that performs multiplication with an AND operation on two stochastic inputs a and b, producing an output c. The network consists of the following reactions:

$$
\begin{aligned}
\mathbf{A}_0 + \mathbf{B}_0 &\xrightarrow{k} \mathbf{C}_0, \\
\mathbf{A}_0 + \mathbf{B}_1 &\xrightarrow{k} \mathbf{C}_0, \\
\mathbf{A}_1 + \mathbf{B}_0 &\xrightarrow{k} \mathbf{C}_0, \\
\mathbf{A}_1 + \mathbf{B}_1 &\xrightarrow{k} \mathbf{C}_1.
\end{aligned} \tag{4.8}
$$

Here $\mathbf{A}_0$ and $\mathbf{A}_1$ represent stochastic value a, and so on for b and c. k is the rate constant and is required to be equal for all the reactions. Notice that there is a one-to-one mapping from the Boolean truth table of the AND gate to the indices of the chemical species. Note that, given the two inputs a and b in the fractional encoding,

$$a = \frac{[\mathbf{A}_1]}{[\mathbf{A}_0] + [\mathbf{A}_1]} \quad \text{and} \quad b = \frac{[\mathbf{B}_1]}{[\mathbf{B}_0] + [\mathbf{B}_1]}. \tag{4.9}$$

Simulating this CRN yields the output

$$c = \frac{[\mathbf{C}_1]}{[\mathbf{C}_0] + [\mathbf{C}_1]} = a \times b. \tag{4.10}$$

That is, the output value is the product of the two input values.

This strategy for implementing stochastic functions with CRN works for an arbitrary number of inputs, provided the reaction rates are the same for all reactions. This assertion is justified in Section 4.6. More examples of constructing CRNs out truth tables are listed out in Table 4.3.

4.6 Proof for the correctness of CRNs implementing truth tables

Theorem 2. *Assume an n-input stochastic function $y = f(x_1, x_2, \ldots, x_n)$ is implemented by a combinational Boolean function $Y = F(X_1, X_2, \ldots, X_n)$. The stochastic function can then be implemented with a CRN with $2n + 2$ different molecular species, in which pairs of molecular species store the input*

Table 4.3: Chemical Reaction Networks for Basic Logic Gates. Note that the indices of molecules match the truth table implementing the logic gate.

Gate	Inputs	Function	CRN
NOT	a	$b = 1 - a$	$\mathbf{A}_0 \to \mathbf{B}_1$ $\mathbf{A}_1 \to \mathbf{B}_0$
AND	a, b	$c = ab$	$\mathbf{A}_0 + \mathbf{B}_0 \to \mathbf{C}_0$ $\mathbf{A}_0 + \mathbf{B}_1 \to \mathbf{C}_0$ $\mathbf{A}_1 + \mathbf{B}_0 \to \mathbf{C}_0$ $\mathbf{A}_1 + \mathbf{B}_1 \to \mathbf{C}_1$
OR	a, b	$c = a + b - ab$	$\mathbf{A}_0 + \mathbf{B}_0 \to \mathbf{C}_0$ $\mathbf{A}_0 + \mathbf{B}_1 \to \mathbf{C}_1$ $\mathbf{A}_1 + \mathbf{B}_0 \to \mathbf{C}_1$ $\mathbf{A}_1 + \mathbf{B}_1 \to \mathbf{C}_1$
NAND	a, b	$c = 1 - ab$	$\mathbf{A}_0 + \mathbf{B}_0 \to \mathbf{C}_1$ $\mathbf{A}_0 + \mathbf{B}_1 \to \mathbf{C}_1$ $\mathbf{A}_1 + \mathbf{B}_0 \to \mathbf{C}_1$ $\mathbf{A}_1 + \mathbf{B}_1 \to \mathbf{C}_0$
NOR	a, b	$c = 1 - a - b + ab$	$\mathbf{A}_0 + \mathbf{B}_0 \to \mathbf{C}_1$ $\mathbf{A}_0 + \mathbf{B}_1 \to \mathbf{C}_0$ $\mathbf{A}_1 + \mathbf{B}_0 \to \mathbf{C}_0$ $\mathbf{A}_1 + \mathbf{B}_1 \to \mathbf{C}_0$
XOR	a, b	$c = a + b - 2ab$	$\mathbf{A}_0 + \mathbf{B}_0 \to \mathbf{C}_0$ $\mathbf{A}_0 + \mathbf{B}_1 \to \mathbf{C}_1$ $\mathbf{A}_1 + \mathbf{B}_0 \to \mathbf{C}_1$ $\mathbf{A}_1 + \mathbf{B}_1 \to \mathbf{C}_0$
XNOR	a, b	$c = 1 - a - b + 2ab$	$\mathbf{A}_0 + \mathbf{B}_0 \to \mathbf{C}_1$ $\mathbf{A}_0 + \mathbf{B}_1 \to \mathbf{C}_0$ $\mathbf{A}_1 + \mathbf{B}_0 \to \mathbf{C}_0$ $\mathbf{A}_1 + \mathbf{B}_1 \to \mathbf{C}_1$

values $x_1, x_2, \ldots, x_n$ as well as the output value y, according to the fractional representation in Equation (4.7). The CRN consists of 2^n reactions, each of the form,

$$\mathbf{X}_{1,v_1} + \mathbf{X}_{2,v_2} + \cdots + \mathbf{X}_{n,v_n} \xrightarrow{k} \mathbf{Y}_{F(V)}, \tag{4.11}$$

where $v_1, v_2, \ldots, v_n : F(V)$ is a row of the truth table for the combinational function F, and $V = (v_1, v_2, \ldots v_n)$ denotes a minterm for the function. Note that the rate constants for all reactions are equal to k, an arbitrary value.

Let S_1 be the set of all minterms V such that $F(V) = 1$, and let S_0 be the set of all minterms V such that $F(V) = 0$. Also, $c_{i,j}$ is denoted as

$$c_{i,j} = \Pr(X_i = j) = \begin{cases} 1 - x_i & \text{if } j = 0 \\ x_i & \text{if } j = 1 \end{cases} \tag{4.12}$$

where x_i is a stochastic input, and i is the index of the input x_i in function $y = f(x_1, x_2, \ldots, x_n)$.

Proving the theorem requires demonstrating that for the given initial values of the stochastic value x_i at time $t = 0$,

$$x_i = \left. \frac{[\mathbf{X}_{i,1}]}{[\mathbf{X}_{i,0}] + [\mathbf{X}_{i,1}]} \right|_{t=0}, \tag{4.13}$$

the output of the CRN matchs the output of the stochastic function stated in Theorem 1,

$$\lim_{t\to\infty} y = \lim_{t\to\infty} \frac{[\mathbf{Y}_1]}{[\mathbf{Y}_0] + [\mathbf{Y}_1]} = \sum_{V \in S_1} \left(\prod_{h=1}^{n} c_{h,v_h} \right). \tag{4.14}$$

The proof provides an even stronger result – the limit $t \to \infty$ in Equation (4.14) is not necessary; in fact at any $t > 0$

$$y = \frac{[\mathbf{Y}_1]}{[\mathbf{Y}_0] + [\mathbf{Y}_1]}. \tag{4.15}$$

Here the proof for Theorem 2 begins.

Given the CRN described in **??**, the rate equations for each input are

$$\frac{d[\mathbf{X}_{i,j}]}{dt} = -k \cdot [\mathbf{X}_{i,j}] \cdot \prod_{h=1,h\neq i}^{n} \left([\mathbf{X}_{h,0}] + [\mathbf{X}_{h,1}]\right), j \in \{0,1\} \quad (4.16)$$

$$= -k \cdot \frac{[\mathbf{X}_{i,j}]}{[\mathbf{X}_{i,0}] + [\mathbf{X}_{i,1}]} \cdot \prod_{h=1}^{n} \left([\mathbf{X}_{h,0}] + [\mathbf{X}_{h,1}]\right). \quad (4.17)$$

Note that k, an arbitrary value, is the rate constant for each reaction. The rate equations for the output species are,

$$\frac{d[\mathbf{Y}_j]}{dt} = k \sum_{V\in S_j} \left(\prod_{h=1}^{n} [\mathbf{X}_{h,v_h}]\right), j \in \{0,1\}. \quad (4.18)$$

The following new variables are now defined,

$$p_i = \frac{[\mathbf{X}_{i,1}]}{[\mathbf{X}_{i,0}] + [\mathbf{X}_{i,1}]} \quad (4.19)$$

$$q_i = [\mathbf{X}_{i,0}] + [\mathbf{X}_{i,1}] \quad (4.20)$$

$$r_{i,j} = \begin{cases} 1 - p_i & \text{if } j = 0 \\ p_i & \text{if } j = 1 \end{cases} \quad (4.21)$$

Substituting these variables into the expressions for the concentrations:

$$[\mathbf{X}_{i,0}] = q_i(1 - p_i), \quad (4.22)$$

$$[\mathbf{X}_{i,1}] = q_i p_i, \quad (4.23)$$

$$\therefore [\mathbf{X}_{i,j}] = q_i r_{i,j}. \quad (4.24)$$

These substitutions are introduced into Section 4.6 and Equation (4.18):

$$\frac{d[\mathbf{X}_{i,j}]}{dt} = -k \cdot r_{i,j} \prod_{h=1}^{n} q_h, \quad (4.25)$$

$$\frac{d[\mathbf{Y}_j]}{dt} = k \left(\prod_{h=1}^{n} q_h\right) \sum_{V\in S_j} \left(\prod_{h=1}^{n} r_{h,v_h}\right). \quad (4.26)$$

As the concentrations $[\mathbf{X}_{i,j}]$ are functions of time, all p, q, and r are also functions of time. Consider the following two expressions derived from Section 4.6,

$$\frac{d[\mathbf{X}_{i,0}]}{dt} = -k(1-p_i)\prod_{h=1}^{n} q_h \tag{4.27}$$

$$\frac{d[\mathbf{X}_{i,1}]}{dt} = -k \cdot p_i \prod_{h=1}^{n} q_h. \tag{4.28}$$

$$\therefore \frac{dq_i}{dt} = \frac{d[\mathbf{X}_{i,0}]}{dt} + \frac{d[\mathbf{X}_{i,1}]}{dt} = -k\prod_{h=1}^{n} q_h. \tag{4.29}$$

Furthermore

$$[\mathbf{X}_{i,1}] = p_i \cdot q_i \tag{4.30}$$

$$\therefore \frac{d[\mathbf{X}_{i,1}]}{dt} = p_i\frac{dq_i}{dt} + q_i\frac{dp_i}{dt} \tag{4.31}$$

$$= p_i\left(-k\prod_{h=1}^{n} q_i\right) + q_i\frac{dp_i}{dt} \tag{4.32}$$

$$= \frac{d[\mathbf{X}_{i,1}]}{dt} + q_i\frac{dp_i}{dt}. \tag{4.33}$$

As $q_i \neq 0$,

$$\frac{dp_i}{dt} = 0, \tag{4.34}$$

that is, p_i is invariant to time. Consequently, $r_{i,j}$ is also invariant to time. This means that the stochastic value encoded by each pair of input species remains the same throughout the reaction. Therefore, for $t > 0$

$$p_i = x_i \tag{4.35}$$

$$r_{i,j} = c_{i,j} \tag{4.36}$$

Consider the new variable

$$l = \left(\prod_{h=1}^{n} q_i\right) \tag{4.37}$$

$$\therefore \frac{d[\mathbf{Y}_j]}{dt} = k \cdot l \sum_{V \in S_j} \left(\prod_{h=1}^{n} r_{h,v_h}\right). \tag{4.38}$$

Finally, we can calculate the stochastic output y as

$$y = \frac{\int_0^t \frac{d[\mathbf{Y}_1]}{dt} dt}{\int_0^t \frac{d[\mathbf{Y}_0]}{dt} dt + \int_0^t \frac{d[\mathbf{Y}_1]}{dt} dt} \tag{4.39}$$

$$= \frac{\sum_{V \in S_1} \left(\prod_{h=1}^{n} r_{h,v_h}\right) \int_0^t k \cdot l \cdot dt}{\sum_{V \in S_0} \left(\prod_{h=1}^{n} r_{h,v_h}\right) \int_0^t k \cdot l \cdot dt + \sum_{V \in S_1} \left(\prod_{h=1}^{n} r_{h,v_h}\right) \int_0^t k \cdot l \cdot dt} \tag{4.40}$$

$$= \frac{\sum_{V \in S_1} \left(\prod_{h=1}^{n} r_{h,v_h}\right)}{\sum_{V \in S_0} \left(\prod_{h=1}^{n} r_{h,v_h}\right) + \sum_{V \in S_1} \left(\prod_{h=1}^{n} r_{h,v_h}\right)} \tag{4.41}$$

$$= \sum_{V \in S_1} \left(\prod_{h=1}^{n} r_{h,v_h}\right). \tag{4.42}$$

The numerator in Section 4.6 corresponds to the sum of the minterms of all rows of the truth table F that evaluate 1, while the denominator corresponds to the sum of all minterms. As $r_{i,j}$ is only dependent on the initial input value, the denominator must sum up to 1 since it includes all the minterms. Therefore, a CRN constructed this way, corresponding to an arbitrary Boolean truth table F, will implement the stochastic function f of that truth table. The only requirement is that the rate constants of all the reactions must be equal.

4.6.1 A demonstrative example

This section provides a helpful example demonstrating the proof in Section 4.6 in action. Consider the two-input AND gate from Section 4.5.

$$\begin{aligned}
\mathbf{A}_0 + \mathbf{B}_0 &\xrightarrow{k} \mathbf{C}_0 \\
\mathbf{A}_0 + \mathbf{B}_1 &\xrightarrow{k} \mathbf{C}_0 \\
\mathbf{A}_1 + \mathbf{B}_0 &\xrightarrow{k} \mathbf{C}_0 \\
\mathbf{A}_1 + \mathbf{B}_1 &\xrightarrow{k} \mathbf{C}_1
\end{aligned} \tag{4.43}$$

The rate equations for the input and output species are:

$$\begin{aligned}
\frac{d[\mathbf{A}_0]}{dt} &= -k[\mathbf{A}_0]([\mathbf{B}_0] + [\mathbf{B}_1]) \\
\frac{d[\mathbf{A}_1]}{dt} &= -k[\mathbf{A}_1]([\mathbf{B}_0] + [\mathbf{B}_1]) \\
\frac{d[\mathbf{B}_0]}{dt} &= -k[\mathbf{B}_0]([\mathbf{A}_0] + [\mathbf{A}_1]) \\
\frac{d[\mathbf{B}_1]}{dt} &= -k[\mathbf{B}_1]([\mathbf{A}_0] + [\mathbf{A}_1]) \\
\frac{d[\mathbf{C}_0]}{dt} &= k([\mathbf{A}_0][\mathbf{B}_0] + [\mathbf{A}_0][\mathbf{B}_1] + [\mathbf{A}_1][\mathbf{B}_0]) \\
\frac{d[\mathbf{C}_1]}{dt} &= k[\mathbf{A}_1][\mathbf{B}_1].
\end{aligned} \tag{4.44}$$

Assume the previously discussed encoding to represent the stochastic values,

$$a = \frac{[\mathbf{A}_1]}{[\mathbf{A}_0] + [\mathbf{A}_1]}, \;\; b = \frac{[\mathbf{B}_1]}{[\mathbf{B}_0] + [\mathbf{B}_1]}, \;\; c = \frac{[\mathbf{C}_1]}{[\mathbf{C}_0] + [\mathbf{C}_1]},$$

as well as the sum of concentrations of each pair of input species,

$$[\mathbf{A}_0] + [\mathbf{A}_1] = q_a, \;\; [\mathbf{B}_0][\mathbf{B}_1] = q_b.$$

With these variables, Equation (4.44) becomes:

$$\begin{aligned}
\frac{d[\mathbf{A}_0]}{dt} &= -kq_aq_b \cdot (1-a)\\
\frac{d[\mathbf{A}_1]}{dt} &= -kq_aq_b \cdot a\\
\frac{d[\mathbf{B}_0]}{dt} &= -kq_aq_b \cdot (1-b)\\
\frac{d[\mathbf{B}_1]}{dt} &= -kq_aq_b \cdot b\\
\frac{d[\mathbf{C}_0]}{dt} &= kq_aq_b \cdot [(1-a)(1-b)+(1-a)b+a(1-b)]\\
\frac{d[\mathbf{C}_1]}{dt} &= kq_aq_b \cdot ab.
\end{aligned} \tag{4.45}$$

Consider the time invariance of a and b. $[\mathbf{A}_1]$ can be expressed as $a \cdot q_a$, therefore according to the chain rule for derivatives,

$$\frac{d[\mathbf{A}_1]}{dt} = q_a\frac{da}{dt} + a\frac{dq_a}{dt}. \tag{4.46}$$

According to Equation (4.45),

$$\frac{dq_a}{dt} = \frac{d[\mathbf{A}_1]}{dt} + \frac{d[\mathbf{A}_1]}{dt} = -kq_aq_b. \tag{4.47}$$

From Equations (4.45) to (4.47)

$$q_a\frac{da}{dt} = 0. \tag{4.48}$$

As q_a is not a constant equal to 0 during the reaction, $\frac{da}{dt} = 0$. This proves the time invariance of a, i.e. during the reaction, the fractional value a encoded by $[\mathbf{A}_0]$ and $[\mathbf{A}_1]$ remains constant. Similarly, b can be proven to be time-invariant.

Now c must be calculated for $t > 0$. Assume the initial concentration of

$[\mathbf{C}_0]$ and $[\mathbf{C}_1]$ are 0, then

$$\begin{aligned} c &= \frac{[\mathbf{C}_1]}{[\mathbf{C}_0]+[\mathbf{C}_1]} \\ &= \frac{\displaystyle\int_0^t \frac{d[\mathbf{C}_1]}{dt}dt}{\displaystyle\int_0^t \frac{d[\mathbf{C}_0]}{dt}dt + \int_0^t \frac{d[\mathbf{C}_1]}{dt}dt} \\ &= \frac{\displaystyle\int_0^t kq_aq_b \cdot ab \cdot dt}{\displaystyle\int_0^t kq_aq_b dt} \\ &= \frac{ab\displaystyle\int_0^t kq_aq_b dt}{\displaystyle\int_0^t kq_aq_b dt}(\text{since } a, b \text{ are constant}) \\ &= ab. \end{aligned} \tag{4.49}$$

This proves that an AND gate implements multiplication.

4.7 Error Analysis

Clearly a CRN implementing a truth table with all reaction rates set equal computes the corresponding stochastic function correctly. What would happen if this rate condition were broken? This section details simulations that were performed to test the robustness of CRNs implementing stochastic functions with the program *Mathematica* [106]. Differential equations corresponding to the reaction kinetics for CRNs were generated to investigate the impact of varying reaction rates. A detailed analysis for a specific CRN for a 3-input Exclusive-OR (XOR) is provided. The function for XOR is

$$f(x, y, z) = x + y + z - 2xy - 2xz - 2yz + 4xyz. \tag{4.50}$$

This function was deliberately chosen for error analysis because the truth table for XOR was balanced in terms of the number of 0's and 1's. Accordingly, it was the most sensitive to random variations in reaction rates. In contrast, for unbalanced functions such as AND or OR, errors could readily be masked: computing more 0's for AND or more 1's for OR would not show up statistically.

This polynomial for this function is generated by the following truth table:

x	y	z	$f(x,y,z)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

To see this, take the sum of the expressions for the minterms, i.e., the rows that evaluate to one. Recall that the expression for each row is formed by multiplying together factors corresponding to the input variables: x if the variable x is equal to 1 or $1 - x$ if the variable x is equal to 0:

$$\begin{aligned} f(x,y,z) &= (1-x)(1-y)z + \\ &\quad (1-x)y(1-z) + \\ &\quad (1-x)y(1-z) + \\ &\quad x(1-y)(1-z) + \\ &\quad xyz \\ &= x + y + z - 2xy - 2xz - 2yz + 4xyz. \end{aligned} \tag{4.51}$$

According to the method discussed in Section 4.5, this can be translated into the following CRN:

$$
\begin{aligned}
\mathbf{X}_0 + \mathbf{Y}_0 + \mathbf{Z}_0 &\xrightarrow{k_1} \mathbf{F}_0 \\
\mathbf{X}_0 + \mathbf{Y}_0 + \mathbf{Z}_1 &\xrightarrow{k_2} \mathbf{F}_1 \\
\mathbf{X}_0 + \mathbf{Y}_1 + \mathbf{Z}_0 &\xrightarrow{k_3} \mathbf{F}_1 \\
\mathbf{X}_0 + \mathbf{Y}_1 + \mathbf{Z}_1 &\xrightarrow{k_4} \mathbf{F}_0 \\
\mathbf{X}_1 + \mathbf{Y}_0 + \mathbf{Z}_0 &\xrightarrow{k_5} \mathbf{F}_1 \\
\mathbf{X}_1 + \mathbf{Y}_0 + \mathbf{Z}_1 &\xrightarrow{k_6} \mathbf{F}_0 \\
\mathbf{X}_1 + \mathbf{Y}_1 + \mathbf{Z}_0 &\xrightarrow{k_7} \mathbf{F}_0 \\
\mathbf{X}_1 + \mathbf{Y}_1 + \mathbf{Z}_1 &\xrightarrow{k_8} \mathbf{F}_1
\end{aligned}
\tag{4.52}
$$

Since the consequences of non-uniform rate constants were explored, note that the eight reactions have unique rate constants: $k_1, k_2, \ldots k_8$, respectively.

The following stochastic variables are defined:

$$x = \frac{[\mathbf{X}_1]}{[\mathbf{X}_0] + [\mathbf{X}_1]}, \; y = \frac{[\mathbf{Y}_1]}{[\mathbf{Y}_0] + [\mathbf{Y}_1]}, \; z = \frac{[\mathbf{Z}_1]}{[\mathbf{Z}_0] + [\mathbf{Z}_1]}, \; f = \frac{[\mathbf{F}_1]}{[\mathbf{F}_0] + [\mathbf{F}_1]}. \tag{4.53}$$

The procedure `NDSolveValue` in *Mathematica* was used to simulate the differential equations corresponding to CRN in Section 4.7. The rate constants were varied as well as the initial concentrations.The value of f computed by the CRN in terms of $[F_0], [F_1]$, was compared to the expected value of f from Equation (4.50). Here is a summary of the trials:

4.7.1 Trials for Error Analysis

The error was calculated as the absolute difference between the value computed by the CRN simulation and the expected value of f from Equation (4.50).

1. With all $k_i = 100$ except for $k_1 = 1000$, i.e., one rate constant being an order of magnitude higher than the others: the highest error observed

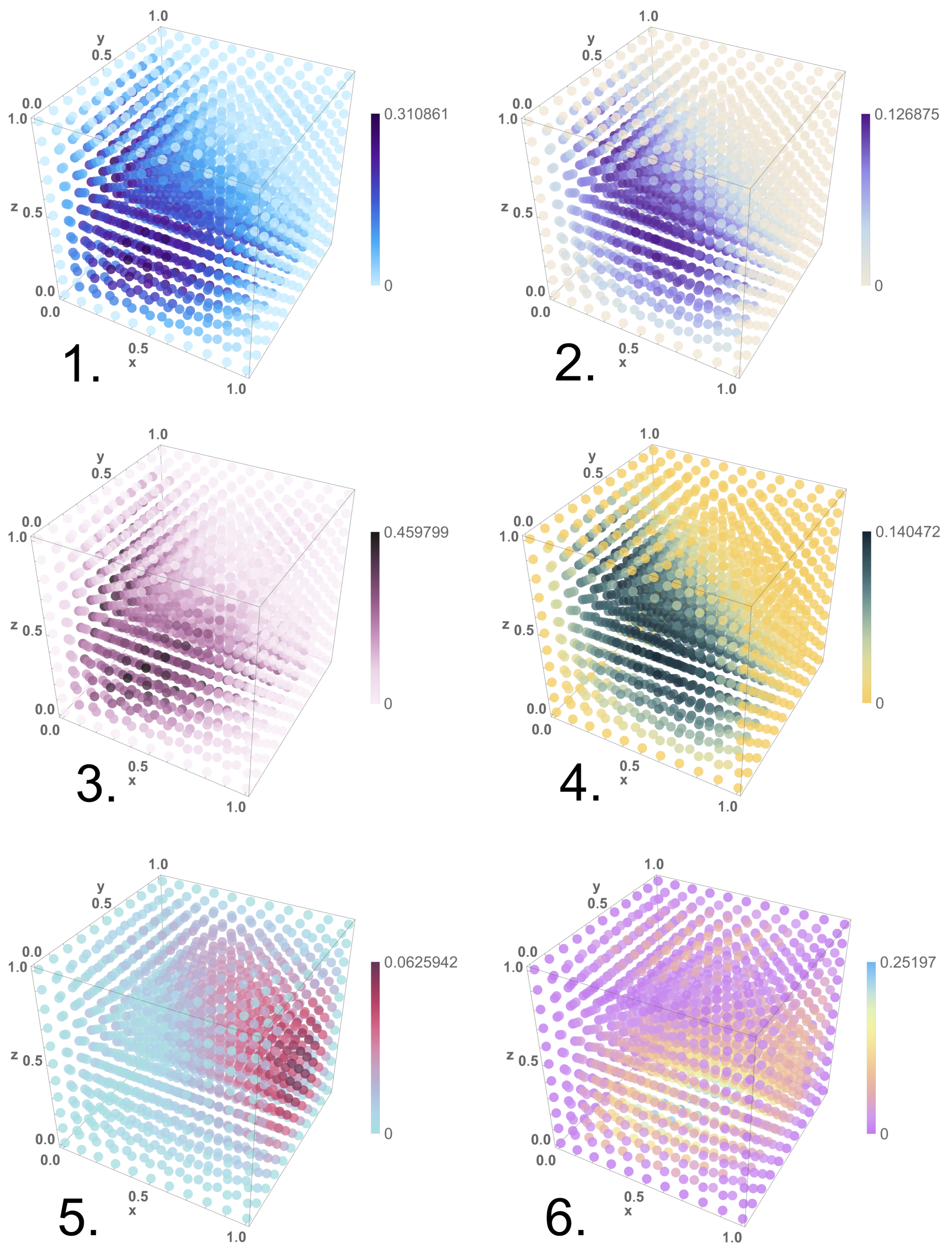


Figure 4.2: The error cubes for the six trials listed in Section 4.7.1. The three dimensions in the plots span the inputs x, y, and z, each in the interval $[0, 1]$, with a step size 0.1. The color of each point corresponds to the absolute difference between the value computed by the CRN and the expected value of f from Equation (4.50). A legend is provided for each cube. The trials were performed with theNDSolveValue function in software tool *Mathematica*.

was 0.31, with 38.1% of the input combinations having an error greater than 0.1.

2. With all $k_i = 100$ except for $k_1 = 10$, i.e., one rate constant being an order of magnitude lower than the others: the highest error observed was 0.12, with 15.7% of the input combinations having an error greater than 0.1.

3. With all $k_i = 100$ except for $k_1 = 10000$, i.e., one rate constant being two orders of magnitude higher than the others: the highest error observed was 0.45, with 45.8% of the input combinations having an error greater than 0.1.

4. With all $k_i = 100$ except for $k_1 = 1$, i.e., one rate being two orders of magnitude lower than the others: the highest error recorded was 0.12, with 22.7% of the input combinations having an error greater than 0.1.

5. With all k_i randomly generated, from a normal distribution with a mean of 100 and a low standard deviation of 10: the highest error recorded was 0.06, with no input combinations having an error greater than 0.1.

6. With all k_i randomly generated, from a normal distribution with a mean of 100 and a high standard deviation of 70 (negative values were not allowed): the highest error recorded was 0.25, with 14.4% of the input combinations having an error greater than 0.1.

The absolute difference between the output value of f, calculated with Equation (4.53), compared to the expected value of f from Equation (4.50) was calculated for a wide range of input concentrations. These are graphed in Figure 4.2. The inputs x, y, and z, calculated with Equation (4.53), were set to values in the interval $[0, 1]$ forming a cube mesh input. All input chemical species were initialized such that $[\mathbf{X}_0] + [\mathbf{X}_1] = 100$. The maximum error difference and the number of input combinations for which the error differential exceeded 0.1 were recorded. The purpose of this simulation was not to account for all possible values of the rate constants, but rather to understand the design constraints and the error margins. The key observations from the simulations are:

1. In a network with many reactions, one rate constant being slower than the others by an order of magnitude or two has a lower impact on error than if it were faster by a similar amount.
2. Error rates are low if all the rate constants are within the same order of magnitude and are distributed normally with a small standard deviation.
3. Error rates are also low when some of the fractional inputs are close to 0 or to 1. This translates to very slow or very fast reactions, respectively.
4. Even when the rate constants differ by orders of magnitude, not all inputs result in high errors.

4.8 Implementation using DNA

4.8.1 DNA Concatemers

DNA Concatemers are long strands of DNA that contain repeated base-pair sequences. These are formed when a single smaller DNA unit is capable of hybridizing with other copies of itself. Specifically, to form a DNA strand of the form **ABABAB**..., the 1-mer unit must have the following 3 regions:

1. A leading sticky end (single-stranded region) on the 1st strand with the sequence **A**.
2. A middle double-stranded section with the sequence **B**.
3. A trailing sticky end on the 2nd strand with the complement sequence $\mathbf{A}'$ such that it can bind to a leading sticky end for **A**.

This study proposed designing molecules for fractional representation as DNA concatemers [93] that could interact via strand displacement, as detailed in the next subsection. For a fractional variable a, the molecules $\mathbf{A}_0$ and $\mathbf{A}_1$ needed for the reaction network could be designed as concatemer units such that the double-stranded section for each unit was distinct, but the sticky ends for both of them were the same. This allowed the two species to cross-polymerize and formed a linear chain of DNA of randomly arranged $\mathbf{A}_0$ and $\mathbf{A}_1$ units. This was similar to the randomized digital bitstreams used in stochastic computing in which a random stream of 0's and 1's forms the basic data unit [21, 73]: $\mathbf{A}_0$ and $\mathbf{A}_1$ correspond to 0 and 1, respectively. Thus a single fractional variable wa be stored as a long DNA strand that could be amplified to improve readout [85]. This long strand could also be broken up using artificial restriction enzymes – or natural restriction enzymes, if the sticky ends are designed purposefully. Furthermore, this concatemer design allowed the use of RNA-seq [104] in the readout process to measure the fractional value stored by a DNA strand. For this purpose, a long DNA concatemer would be broken into its constituent monomers using a restriction enzyme, and then these smaller DNA units would be used instead of the standard complementary DNA in RNA-seq to determine the expression level of each unit. From this quantitative readout, the relative amount of $\mathbf{A}_1$ to $\mathbf{A}_0 + \mathbf{A}_1$ could be determined [112].

4.8.2 Procedure

Figure 4.3 illustrates the reaction $\mathbf{A}_i + \mathbf{B}_j \rightarrow \mathbf{C}_k$ implemented with DNA strand displacement and cleaving enzymes. Two species of concatemer units are transformed into another concatemer unit. The implementation consists of three stages:

1. Extracting single strands: Consider the two input concatemers $\mathbf{A}_i$ and $\mathbf{B}_j$ shown in the figure. The concatemers are designed in such a way that the sticky ends of a concatemer unit can act like open toeholds in DNA strand displacement. As a result, a single strand can be extracted from a concatemer. For example, concatemer $\mathbf{A}_i$ is formed with two single strands $[T_i, A_i]$, $[A_i^*, T_1^*]$. By adding strand $[A_i, T_1]$, strand $[T_1, A_i]$ is displaced. Similarly, strand $[T_2, B_j, T_3]$ can

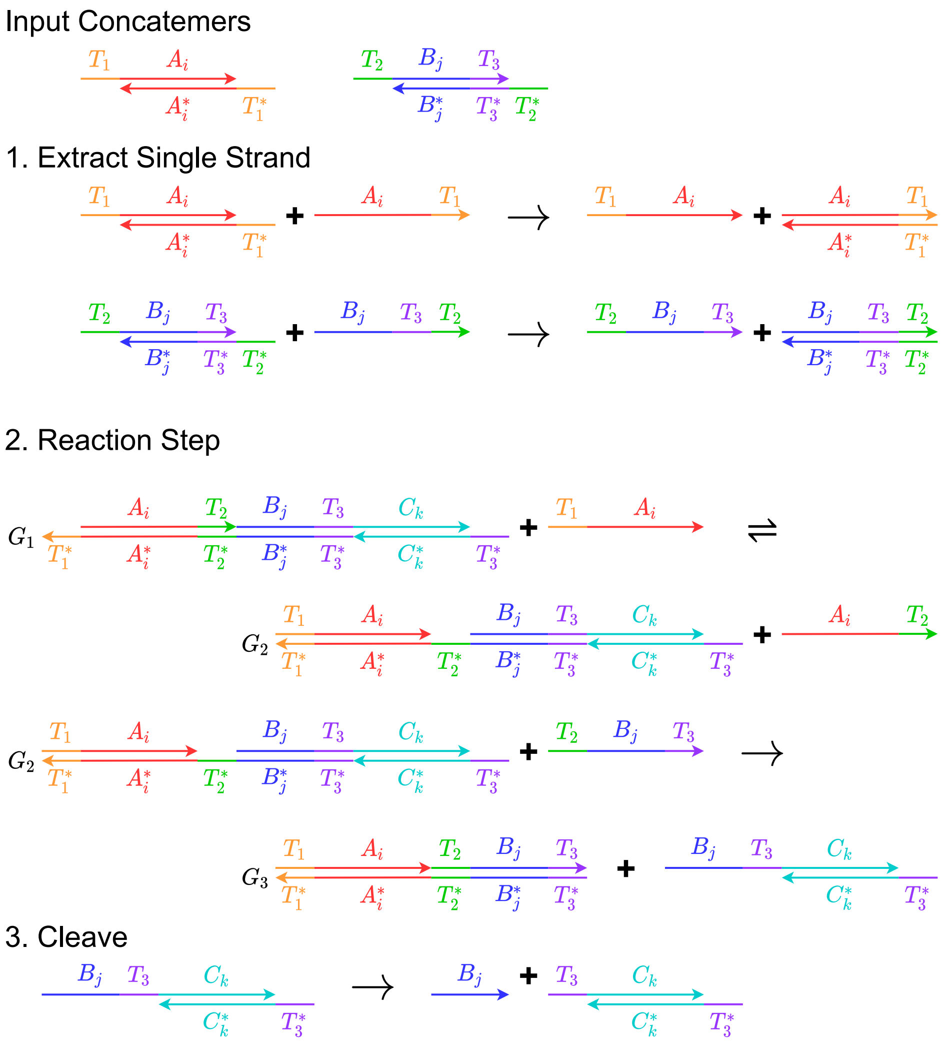


Figure 4.3: An example illustrating strand displacement reactions, implemented using concatemers. The figure is divided into an example sequence of concatemers, and three reaction steps: 1) extracting a single strand from concatemers; 2) a reaction step that consumes two single strands and outputs a complex; and 3) cleaving.

be extracted from concatemer $\mathbf{B}_j$ with strand $[B_j, T_3, T_2]$.

2. This is the strand displacement reaction that implements the main reaction. It receives two single-strand DNA molecules, $[T_1, A_i]$ and $[T_2, B_j, T_3]$ as reactants. The product is a complex containing the output concatemer. The reaction is divided into two parts. In the first part, strand $[T_1, A_i]$ displaces strand $[A_i, T_2]$ from the auxiliary complex $\mathbf{G}_1$ and forms $\mathbf{G}_2$ through a reversible reaction. Then the strand $[T_2, B_j, T_3]$ displaces the output complex which is formed by strand $[B_j, T_3, C_k]$ and $[C_k^*, T_3^*]$. This step is irreversible since the output complex cannot bind to the resulting auxiliary complex $\mathbf{G}_3$ after this step.

3. Cleaving. The output complex from the previous step contains the domain B_j in addition to the part that could form concatemer $\mathbf{C}_k$. The domain B_j is cleaved from the complex. This step yields a concatemer $\mathbf{C}_k$ with T_3 sticky end. Cleaving can be achieved by using DNA editing enzymes such as CRISPR-Cas9 and PfAgo [94].

It is assumed that the concentration of the initial auxiliary complex $\mathbf{G}_1$ is much larger than the concentration of the concatemers. With this assumption, the concentration of the auxiliary complex can be treated invariant through the reaction. Thus, the reaction rate only depends on the concentration of the single strands extracted from the concatemers. As there are four reactions to implement the two-input network shown in this example, four species of the auxiliary complex representing each reaction should be used. This ensures that the mixture of different species of $\mathbf{A}_0$ and $\mathbf{A}_1$, or $\mathbf{B}_0$ and $\mathbf{B}_1$, can react competitively. During the cleaving step, each reactant participates in only one reaction. Therefore, it should not affect the reaction rate or the fractional encoding of the output by the two product species.

The reaction itself can be extended to a multimolecular reaction by extending the chain of toehold exchange reactions. Suppose, for example, a new stochastic value d with molecules $\mathbf{D}_l$ and sticky ends T_4 were also the input alongside a and b. In the complex $\mathbf{G}_1$, domains $[T_4, D_l]$ and their complementary domains would be added between the domains B_j and T_3. That is, a new $\mathbf{G}_1$ that would react with single strands of sequence D_l and toehold T_4 would be used. In this way, $\mathbf{G}_1$ would be capable of receiving an additional strand $[T_4, D_l]$ before displacing the final product. Therefore multiple input values can be computed upon in our CRNs.

When computing with digital circuits, the length of the bitstream dictates the precision of the computation. The length of the bitstream can be chosen by the user based on their specifications. The more precision that they require, the longer the bitstream that they should use. In this DNA implementation, the concentration of DNA concatemers corresponds to the length of the bitstreams for the stochastic functions. So the limitation is experimental: how precisely the user can set and measure the input and output concentrations, respectively.

4.9 Conclusion

This paper proposed a strategy for computing mathematical functions with molecular systems based on a fractional representation, using a pair of molecular species to represent each mathematical variable. With this representation, one can apply the theory of stochastic logic design chemical reaction networks for computing functions. In particular, the translation from truth tables for stochastic functions into chemical reaction networks was shown and verified. An implementation in DNA strand displacement was provided.

Stochastic logic is an intriguing paradigm for digital computation. Instead of computing definite outputs from definite inputs – say Boolean values from Boolean values, or integers from integers – it entails computing probabilities from probabilities. There is randomness and yet the computation is robust. The computation is effected by transforming the *statistical distribution* of random bitstreams. The paradigm has been applied in a variety of domains, particularly for emerging technologies [100, 37, 86, 60]. It has been most successful for applications that entail computing mathematical functions: for instance, `arctan` for nonlinear activation functions in machine learning; *Bessel* functions for differential system models; and the `sinc` function for image and signal processing. Examples of CRN implementations of these functions are given in Chapter 10.

Chapter 5

Discussion

The previous studies demonstrate the potential for computing on a molecular level. The SIMD model allows for parallelism in DNA computing. There are, in fact, two layers of parallelism possible:

1. Bit-level Parallelism: instructions applied to all bits in an array at once.

2. Data-level Parallelism: the same instructions applied to *multiple* arrays at once.

While operations on DNA are slow and error-prone, with these levels of parallelism, perhaps DNA computation could scale to a truly impressive regime.

Over the past two decades, computing has moved from desktops and data centers into the wild. Embedded microchips – found in our gadgets, our tools, our buildings, our soils and even our bodies – are transforming our lives. And yet, there are limits to where silicon can go and where it can compute effectively. It operates based on voltage and so requires a power source. Even miniaturized to the microscale or smaller, an electronic system is often a foreign object inserted into a material, substrate, or environment. The sorts of computation discussed in these studies could find application in a novel class of computing system that is not foreign, but rather an integral part of its physical and chemical environment: a system that computes *with* its constituent molecules. In such a system, sensing, computing, and actuating occur at the molecular level, with no interfacing at all with external electronics.

The topic of stochastic logic has been advertised as a possible design paradigm for emerging technologies that promise scaling beyond complementary metal–oxide–semiconductor (CMOS), as well as the basis of non-von Neumann architectures [101, 86]. The main appeal of stochastic logic is that a wide variety of functions can be computed with simple structures. For instance, multiplication can be implemented with a single AND gate. DNA lends itself as an ideal substrate for implmenting true stochastic logic. It allows for decreasing computational complexity due to an increase in computational capacity, with no loss of run time as would be the case with bitstreams in digital computing.

The field of DNA computing is still relatively new and developing. It has the potential to revolutionize computing by offering a massively parallel computing architecture that is compact and stable. This could enable rapid and efficient processing of large amounts of data [88]. DNA computing is also energy-efficient and compares favorably to silicon based computing in terms of heat dissipation and power consumption [71]. Already DNA computing as been featured in research in a number of areas such as cryptography [107], data storage, machine learning, microfluidics [28], and drug discovery [18]. The studies in this dissertation provide a framework for a computational model from the ground up – starting with small systems that can be cascaded and built up to compute more complex problems. Challenges still lie ahead – DNA synthesis remains prohibitively expensive. It is yet to overtaking silicon based computing.

Part II

Bioinformatics

Chapter 6

Overview

Bioinformatics is a research field formed through the intersection of computer science and biology. With the ever growing understanding of biological systems, the large scale of biological data being generated, and the various demanding applications in medicine and the environment, this field is continuously expanding.

Data from various sources such as genomics and transcriptomics [34], protein structures [80], ecological [40] and environmental [20] studies have been the emphasis of bioinformatics research. The interest of this dissertation is in computational immunology: the application of bioinformatics in immunology. The particular focus of this section is Human T Cell Immunity and the MHC antigen presentation pathways.

6.1 Major Histocompatibility Complex

The Human Leukocyte Antigen (HLA) gene system encodes cell-surface proteins that play a key role in the immune system. In it, the MHC Class I and Class II pathways allow cells to present antigens derived from endogenous and exogenous proteins respectively [63]. These antigens are small peptides that are broken apart from a source protein and then strongly bound by the MHC Class I or Class II proteins. MHC Class I proteins bind peptides 8-11 amino acids long from proteins inside the cell, while MHC Class II proteins bind longer peptides (13-17 amino acids) derived from proteins outside the cell. Whenever a MHC protein binds a peptide, the resulting peptide-MHC complex is transported to the cell exterior such that the peptide can then be presented to other cells for immune surveillance. This antigen presentation allows for CD8$^+$ and CD4$^+$ T cells to identify any cells presenting pathogenic antigens and to consequently kill those infected cells. Clearly the MHC presentation pathways play a crucial role in immunity.

The mechanics of peptide binding are specific to a given MHC variant. The HLA genes are among the most diverse in the human population [26]. Thus the set of all antigens presented by a person's MHCs, labelled as their *immunopeptidome*, is unique and determines the capacity of their immune system. Since the immune response of a person to a viral infection like COVID-19, for instance, is dependent on whether the foreign antigens presented by their MHCs are distinguishable from self-peptides, understanding and predicting pMHC binding is an important topic.

In a peptide bound to a MHC protein, the starting and ending amino acids on the chain are tucked into the MHC's binding pockets. The remaining peptide chain rests along a groove exposed on MHC surface, or bulges out if it is longer than can fit in the groove. Clearly, not every residue on the peptide contributes to its binding affinity with the MHC molecule. These certain positions on a strong binding peptide that play a crucial role in binding affinity are called *anchor residues.* When observing the binding motif of a particular HLA, i.e. the consensus sequence of all its strong binding peptides, anchor residues exhibit higher specificity for certain amino acids than the remaining positions do. Furthermore, different HLA types have different anchor residue preferences, due to biochemical factors such hydrophobicity, electrostatic interactions, and shape conformity that play a role in each HLA's unique binding pocket. For example, HLA-A02:01 prefers binding 9-mers with hydrophobic residues (L,V,M, and I) on positions 2 and 9, while HLA-B27:05 prefers binding 9-mers with a simple R on position 2.

Given the large variation in MHC alleles, and their unique binding motifs, several bioinformatics tools have been developed to predict the binding of antigens by MHC proteins. While approaches such as molecular docking or other forms of structural based predictions are powerful, they are overtaken by machine learning approaches due to the large amounts of non-structural data and their slower computing times. Of the numerous machine learning tools, the state-of-the-art have certainly been the two neural network based NetMHC-4.0 [3] and NetMHCpan-4.1 [78]. Both software tools have been applied in predicting cancer immune escape mechanisms [53], checkpoint blockade immunotherapy for tumors [49], and identifying COVID-19 T-cell response targets [27].

This part of the dissertation features two studies based on these machine learning tools used for peptide-MHC binding predictions. The first study explores how neural networks can be liable to false positives or negatives if the underlying biochemistry is not accounted for in their training. Specifically, the focus of this study is hydrophobicity, and how integral it is in peptide-MHC binding predictions. It exposes how NetMHCpan-4.1 outperforms NetMHC-4.0 and this distinction is evident when hydrophobicity is used as a metric for evaluation.

The second study focuses on the use of peptide-MHC predictions in understanding T cell immune response to COVID-19. The evolution of SARS-CoV-2 over the course of the COVID-19 pandemic has produced several variants, each evading the previous variant's antibodies used for treatments. However, MHC antigens were observed to be preserved across the variants in experimental studies. This study details how antigenic sites in variants such as Omicron were mostly preserved from the original SARS-CoV-2 virus. The implication of this preservation of antigens is explored through the use of controls. Artificial variants of the COVID-19 spike protein were designed to evade T cell immunity, proving that the natural evolution of SARS-CoV-2 did not lead to a notable reduction in T cell immune responses. The significance of this study is that it showcases how the MHC presentation pathways provide a strong line of defense against viral diseases and should be an important emphasis in vaccine design.

Chapter 7

Investigating False Positives and False Negatives in Machine Learning Predictions

While these tools provide valuable pMHC predictions, they do not model pMHC binding at the molecular level or capture the entire antigen presentation pathway's effects. Hydrophobicity is a measure of how repulsive a molecule is to water, often a consequence of nonpolarity. It plays a vital role in protein binding – for example, the MHC molecule HLA-A*0201 (A2) contains hydrophobic binding pockets that bind to correspondingly hydrophobic amino acids [16]. In contrast, the MHC HLA-B*2705 (B27) prefers to bind peptides with a hydrophilic amino acid in one of its pockets [35]. Historically, immunopeptidomes have been predicted by modelling the interaction of the MHC binding pocket and peptide, particularly focusing on biochemical attributes such as sidechain conformations, solvation energies, electrostatic interactions, and hydrophobicity [109, 99]. However with improved computing power, larger datasets, and the need for interpolation due

This research was funded by the NSF Grant 2036064.

to the high polymorphism in MHC Class I alleles [64], artificial intelligence based methods have become popular over such mechanistic means of prediction. As NetMHC-4.0 and NetMHCpan-4.1 are trained with sequence data and binding scores only, they lack the means of modelling these biochemical attributes. Other software tools such as ANN-Hydro [10] have utilized hydrophobicity in their immunogenic predictions, but do not predict binding affinity and are outperformed by NetMHCpan [54]. In our use of NetMHC-4.0 we had observed a prevalence of highly hydrophobic peptides in the predicted A2 immunopeptidome. We had found this contrary to our expectations, since peptides in which all amino acids are hydrophobes would not dissolve in the aqueous cytosol within the cell and would thus likely not be available for binding with the MHC. We had therefore sought to investigate the possibility that these tools were over-estimating binding scores for such hydrophobic peptides. In a previous study [90], we had tested these tools' predictions on A2 and observed hydrophobic biases that suggested a false positive problem in NetMHC-4.0. Here, we expanded that study to look at multiple HLAs with different binding preferences in more detail. Once again, we conducted two analyses on both NetMHC-4.0 and NetMHCpan-4.1, one using training data and the other using a sample of the human proteome, to investigate the correlation of predicted strong binders and hydrophobicity. We present our results and highlight the unintended bias within NetMHC-4.0 for predicting hydrophobic peptides as strong binders, and for predicting hydrophilic peptides as non-binders.

7.1 Methods

NetMHC-4.0 and NetMHCpan-4.1 allow users to input a list of peptides or whole proteins, and test the binding of all peptides with a chosen MHC molecule. Both tools return an adjusted score between 0 (for non-binders) and 1 (for strong binders) for all peptides. A notable distinction between the two is that NetMHC is limited to predicting binding for MHC variants it is trained on, i.e. curated MHCs. In contrast, NetMHCpan is capable of interpolating predictions for uncurated MHCs if users provide the MHC amino acid sequence. This is achieved through the integration of MHC sequence as a data feature in training, and by a larger training dataset generated using a sophisticated machine learning method called NNAlign_MA [2]. NetMHCpan-4.1 consists of an ensemble of 50 neural networks, each with hidden layers containing 55 and 66 neurons, that were trained using 5-fold cross validation. NetMHC-4.0 consists of 20 neural networks, each with a single hidden layer of 5 neurons, that were trained using a nested 5-fold cross validation approach [3].

7.1.1 Data Mining

NetMHC-4.0 was trained on CD8$^+$ epitope binding affinity (BA) data from the Immune Epitope Database. This data provided binding scores for peptides to single allele MHCs, with a score that was scaled between 0 and 1 that measured how strongly the peptide bound. NetMHCpan-4.1 was trained on BA data and additional eluted ligand (EL) data from mass spectrometry experiments from multiple sources [78]. The EL data included multi-allele information that was deconvoluted into single allele datapoints using NNAlign_MA. EL score was binary (either 0 or 1) since it checked if a peptide was present in a MHC's immunopeptidome. The combined BA and EL dataset contained more than 13 million pMHC data points spread across numerous HLAs. For this study, only peptides of length 9, i.e. 9-mers, were focused upon since they are the most frequent length of antigens in human immunopeptidomes. Also, the 3 MHC molecules HLA-A02:01 (A2), HLA-B27:05 (B27), and HLA-B08:01 (B8) were prioritized. These HLAs were picked because they were highly represented in the training set (A2 ranked 1st, B27 ranked 11th, and B8 ranked 8th based on number of training datapoints), they were HLA supertypes (they represented the behavior of numerous less frequent HLA types), and they had different binding motifs (discussed in Section 7.1.2).

The training data analysis was the first analysis. For each HLA, all its 9-mers that were reported in the training dataset were collected. A2 had 52569 9-mers, B27 had 17422, and B8 had 19448. The distributions of the experimentally obtained training scores for these HLAs are shown in Figures 7.1 to 7.3. NetMHC-4.0 and NetMHCpan-4.1 were run on these 9-mers to gather each neural network's predicted binding scores with the corresponding HLAs. These scores are shown in Figures 7.1 to 7.3. Furthermore, 9-mers with large enough predicted binding scores (using a 0.5% rank to be precise) were classified as strong binders for a tested HLA by both tools. These classification thresholds were measured by finding the lowest predicted binding score for a strong binder identified by these tools. These measured thresholds are also shown in Figures 7.1 to 7.3.

Given the training data scores, confusion matrices (i.e. the number of positives and negatives, both true and false) for both neural network tools were calculated. Actual strong binders and non-binders in the context of each neural network tool were identified using the previously measured strong binding thresholds on the actual training scores for each 9-mer. For example, in Figures 7.1 to 7.3, all 9-mers on the blue plot above the red dashed line were classified as actual strong binders when testing NetMHC-4.0. The results of the confusion matrices are shown in Tables 10.1 and 10.2. The receiver operating characteristic (ROC) curve for each neural network tool for all 3 HLAs were plotted, as shown in Figures 10.18 to 10.20. The accuracy, precision, recall, and F1 score were also computed from the confusion matrices [72].

While the training data analysis was useful for identifying prediction biases, it alone was not a sufficient means for comparing NetMHC-4.0 and NetMHCpan-4.1. As NetMHC-4.0 was only trained on BA data while NetMHCpan-4.1 was trained on BA and EL data, NetMHCpan-4.1 had an advantage of having "seen" the EL peptides in its training oveThe protein sequences for all reviewed human proteins from Uniprot [12] were gathered. A 100 of them were randomly sampled, and fragmented to create a set of 50804 9-mers. These peptides were also passed through NetMHC-4.0 and NetMHCpan-4.1 for all 3 HLAs to gather their predicted scores. These scores are shown in Figures 10.9 to 10.11. Since no experimentally obtained binding scores were available for these peptides, the Pearson correlations and the confusion matrices were not calculated.

7.1.2 Hydrophobicity

As noted in Section 7.1.1, one of the reasons A2, B27, and B8 were chosen for analysis was their different binding motifs [5]. A2 has a strong affinity for 9-mers with hydrophobic amino acids such as L, V, M, and I in positions 2 and 9. B27, on the other hand, binds 9-mers with hydrophilic R at position 2. In between these two, B8 prefers to bind 9-mers with both hydrophobic amino acids L, V, M, and I at position 2 and 9, but also hydrophilic amino acids R and K at position 3 and 5. Clearly hydrophobicity plays a crucial role in distinguishing the binding preferences of different HLAs. This study focused on the role of hydrophobicity in NetMHC-4.0's and NetMHCpan-4.1's predictions.

Hydrophobicity scales assign hydrophobicity values to single amino acids. They are designed so the hydrophobicity of long peptides or protein chains can be estimated by simply linearly adding up the scores of their constituent amino acids. Scales such as Kyte-Doolittle [43], Cornette [14], and Hopp-Woods [32] are commonly used. However, the Moon scale [58] was the most appropriate for calculating hydrophobicity in this study since it specifically focuses on the sidechain hydrophobicity and polarity of single amino acids. Unlike the other scales, which are well suited for protein folding problems that do not correlate with sidechain hydrophobicity [57], the Moon scale is more representative of how small peptides would behave in an aqueous solution. The scale ranks the 20 amino acids in decreasing order of hydrophobicity as follows: F (1.43), L (1.26), I (1.15), P (1.13), Y (0.94), V (0.80), M (0.79), W (0.63), A (0.46), C (0.24), E (-0.27), G (-0.30), T (-0.33), S (-0.35), D (-0.85), Q (-0.88), N (-1.08), R (-1.19), H (-1.65), K (-1.93).

For any given 9-mer, its total hydrophobicity was calculated by adding up the Moon scale values for each of its 9 amino acids. For any given set of peptides, the mean and standard deviation of the hydrophobicity scores of all peptides in it were measured. Furthermore, any given peptide was clustered into 1 of 3 classes: Hydrophobic (total hydrophobicity greater than 3), Hydrophilic (total hydrophobicity less than -3), or Balanced (total hydrophobicity between -3 and 3). This classification distinguished peptides based on their net hydrophobicity, and allowed for investigating the impact of hydrophobicity on the differential prediction of MHC binding for different classes of peptides. These categories were added added in Figures 10.18 to 10.20 and **????** to highlight any prominent trends specific to a peptide category.

It was possible the smaller BA training dataset for NetMHC-4.0 was biased or unrepresentative of the numerous possible binding peptides. This bias could also be caused due to the binding affinity assays used to obtain BA scores, since these experiments only measure MHC-peptide affinity and do not account for the rest of the antigen presentation pathway or physiological conditions. Therefore, the hydrophobicities of the set of BA training data points (i.e. peptides NetMHC-4.0 was trained on) were compared to the set of EL training data points (i.e. more than 80% of the peptides NetMHCpan-4.1 was trained on) for all 3 HLAs.

7.2 Results

From the scores shown in Figures 7.1 to 7.3, it was clear that the pMHC binding data fit a mostly binary data classification problem, since only 15% of the analyzed peptides had a training score not equal to 0 or to 1. This was mostly due to the addition of EL data which provided a binary "yes" or "no" answer to whether a given peptide was found attached to our chosen HLAs through mass spectroscopy. NetMHC-4.0's predicted scores were dispersed smoothly between 0 and 1. In contrast, NetMHCpan-4.1 had more lopsided predictions with more non-binders assigned a binding score of 0. However, neither neural network tool predicted a definitive score of 1 to strong binders and instead used their thresholds to identify binders. NetMHCpan-4.1 predicted more strong binders than NetMHC-4.0 in all 3 cases.

These results of NetMHC-4.0 and NetMHCpan-4.1 on the sample human proteome are shown in Figures 10.9 to 10.11. Note that no experimental binding data was available for these peptides, and that the same set of these peptides was used for each HLA's predictions when comparing Figures 10.9

Table 7.1: Various binary classification metrics on the training data analysis for NetMHC-4.0.

HLA	Peptide Case	Accuracy	Precision	Recall	F1 score
A2	All	0.918	0.882	0.618	0.727
	Hydrophobic only	0.875	0.861	0.745	0.745
	Hydrophilic only	0.986	0.786	0.134	0.229
	Balanced only	0.924	0.909	0.514	0.657
B27	All	0.909	0.914	0.460	0.612
	Hydrophobic only	0.926	0.954	0.535	0.685
	Hydrophilic only	0.916	0.793	0.362	0.497
	Balanced only	0.903	0.918	0.454	0.608
B8	All	0.927	0.925	0.625	0.746
	Hydrophobic only	0.915	0.910	0.571	0.702
	Hydrophilic only	0.960	0.921	0.728	0.813
	Balanced only	0.924	0.930	0.627	0.749

Table 7.2: Various binary classification metrics on the training data analysis for NetMHCpan-4.1.

HLA	Peptide Case	Accuracy	Precision	Recall	F1 score
A2	All	0.936	0.918	0.756	0.829
	Hydrophobic only	0.871	0.922	0.735	0.818
	Hydrophilic only	0.991	0.826	0.613	0.704
	Balanced only	0.952	0.915	0.776	0.840
B27	All	0.965	0.917	0.865	0.890
	Hydrophobic only	0.969	0.964	0.832	0.893
	Hydrophilic only	0.957	0.832	0.814	0.823
	Balanced only	0.966	0.919	0.878	0.898
B8	All	0.951	0.915	0.818	0.864
	Hydrophobic only	0.935	0.930	0.728	0.816
	Hydrophilic only	0.965	0.892	0.832	0.861
	Balanced only	0.953	0.914	0.840	0.876

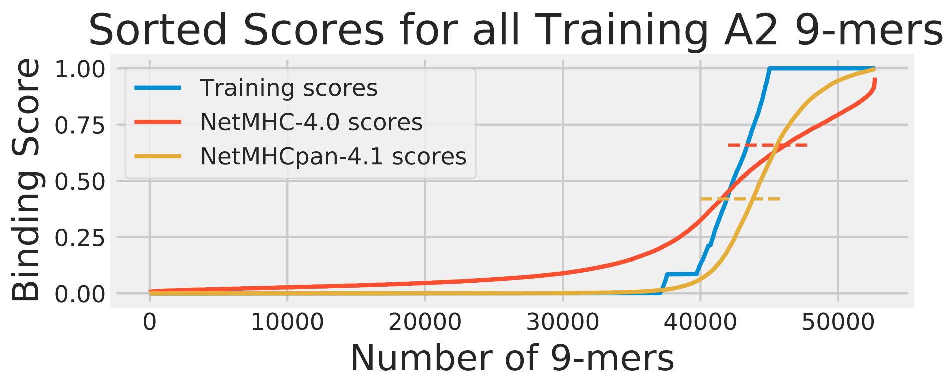


Figure 7.1: The cumulative distribution of the experimental training scores (blue), NetMHC-4.0 predicted scores (red), and NetMHCpan-4.1 predicted scores (yellow) for peptides in the training dataset for HLA A2. The strong binder thresholds for NetMHC-4.0 and NetMHCpan-4.1 are shown as dashed lines of the corresponding colors. For A2, the NetMHC-4.0 and NetMHCpan-4.1 thresholds were 0.659 and 0.419 respectively. Each plot of scores was independently sorted. Consequently, the order of peptides is not conserved across the 3 plots.

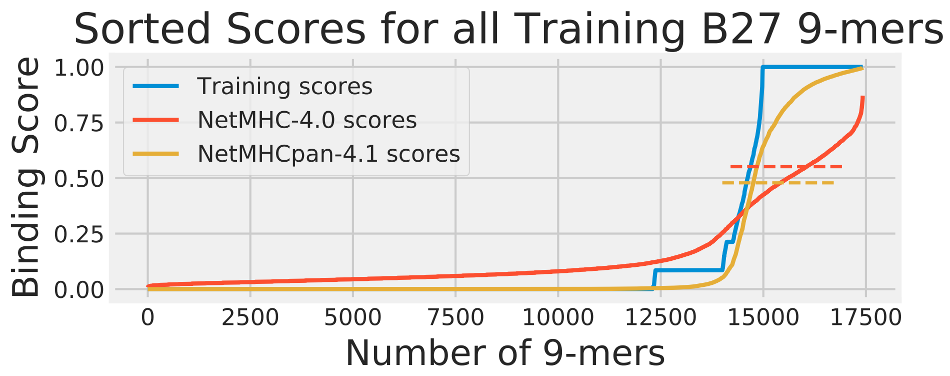


Figure 7.2: The cumulative distribution of the experimental training scores (blue), NetMHC-4.0 predicted scores (red), and NetMHCpan-4.1 predicted scores (yellow) for peptides in the training dataset for HLA B27. The strong binder thresholds for NetMHC-4.0 and NetMHCpan-4.1 are shown as dashed lines of the corresponding colors. For B27, the NetMHC-4.0 and NetMHCpan-4.1 thresholds were 0.551 and 0.478 respectively. Each plot of scores was independently sorted. Consequently, the order of peptides is not conserved across the 3 plots.

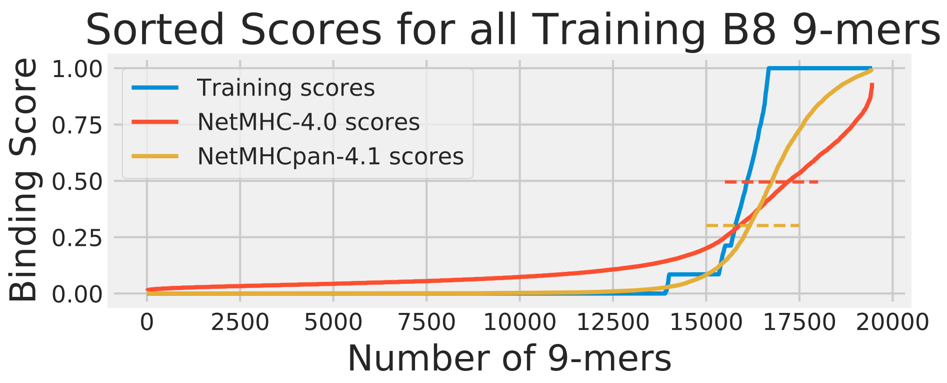


Figure 7.3: The cumulative distribution of the experimental training scores (blue), NetMHC-4.0 predicted scores (red), and NetMHCpan-4.1 predicted scores (yellow) for peptides in the training dataset for HLA B8. The strong binder thresholds for NetMHC-4.0 and NetMHCpan-4.1 are shown as dashed lines of the corresponding colors.For B8 the NetMHC-4.0 and NetMHCpan-4.1 thresholds were 0.495 and 0.301 respectively. Each plot of scores was independently sorted. Consequently, the order of peptides is not conserved across the 3 plots.

to 10.11 with Figures 7.1 to 7.3. Again, NetMHCpan-4.1 seemed more stringent in predicting non-zero binding scores. NetMHCpan-4.1 also predicted slightly more strong binders than NetMHC-4.0 for all 3 HLAs.

The performances of NetMHC-4.0 and NetMHCpan-4.1 as binary classifiers are shown in Figures 10.18 to 10.20 as ROC curves. The figure also includes the area under the curve (AUC) for each classifier. It breaks down the performance across all training peptides and even the 3 peptide categories defined in Section 7.1.2. For A2 and B27, NetMHC-4.0 and NetMHCpan-4.1 had similar AUC values (no more than 1% apart), while for B8 NetMHC-4.0 under-performed by 3%. Across all HLAs, both tools reported AUC values higher than 95%. The different peptide categories did not highlight any notable trends on the ROC plots. It is interesting to note that both tools had different performances across the 3 peptide cases in each HLA. To investigate this observation in detail, violin plots were used to visualize the predicted immunopeptidomes.

The distributions of the hydrophobicity scores of 9-mers in the training data analysis are shown in Figures 10.12 to 10.14, and those in the human proteome data analysis in Figures 7.4 to 7.6. For both analyses, the 2 sample t-test was used to compare the immunopeptidomes predicted by NetMHC-4.0 and NetMHCpan-4.1, and to identify any discrepancies in their predictions on the basis of hydrophobicity (all values used in the t-tests are listed in Tables 10.3 and 10.4).

Strong binders to A2 were expected to have two hydrophobic amino acids (L, V, M, or I) at positions 2 and 9, and thus the expected A2 immunopeptidome would be more hydrophobic than the training or sampled data (expected to be approximately centered about a Moon score of 2). In both analyses, NetMHCpan-4.1 predicted strong binders with a closer hydrophobicity score to our expected value than NetMHC-4.0 did. This difference in predictions was extremely statistically significant in both analyses (p-values less than 0.0001). That is, NetMHC-4.0's predicted strong binders for A2 were more hydrophobic than NetMHCpan-4.1's.

Strong binders to B27 were expected to have one hydrophilic amino acid (R) at position 2, and thus the expected B27 immunopeptidome would be

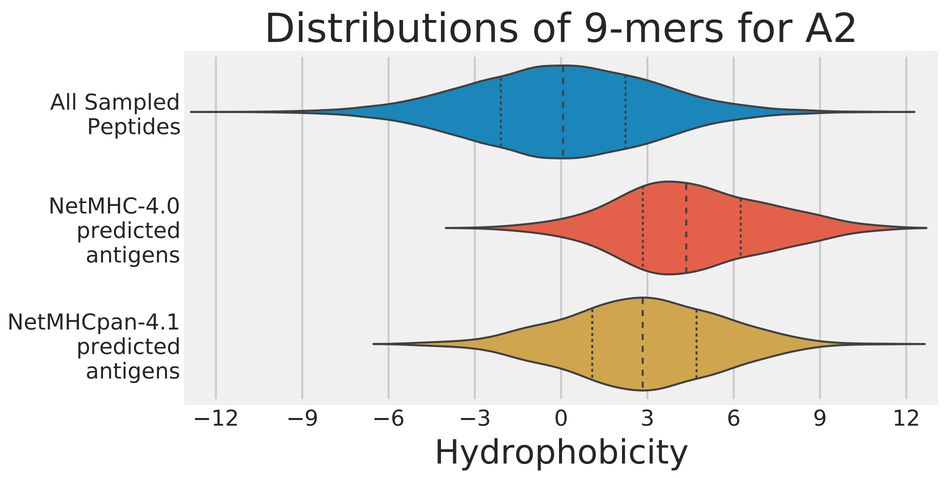


Figure 7.4: Violin plots of the hydrophobicity of the sets of strong binders predicted by NetMHC-4.0 and NetMHCpan-4.1 on the human proteome dataset for A2. The x-axis represents the hydrophobicity of a 9-mer, and the y-axis represents the frequency. The distributions of all sampled peptides (blue), strong binders predicted by NetMHC-4.0 (red), and those predicted by NetMHCpan-4.1 (yellow) are shown. The mean and two quartiles are also depicted in each distribution.

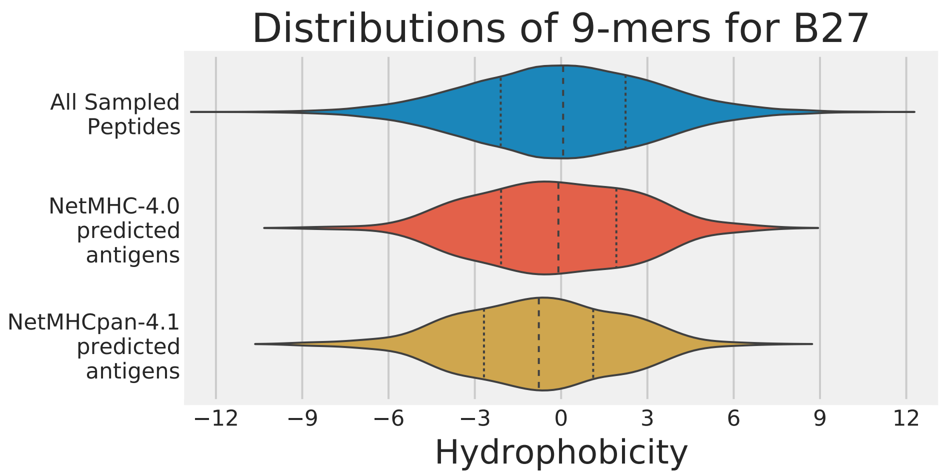


Figure 7.5: Violin plots of the hydrophobicity of the sets of strong binders predicted by NetMHC-4.0 and NetMHCpan-4.1 on the human proteome dataset for B27. The x-axis represents the hydrophobicity of a 9-mer, and the y-axis represents the frequency. The distributions of all sampled peptides (blue), strong binders predicted by NetMHC-4.0 (red), and those predicted by NetMHCpan-4.1 (yellow) are shown. The mean and two quartiles are also depicted in each distribution.

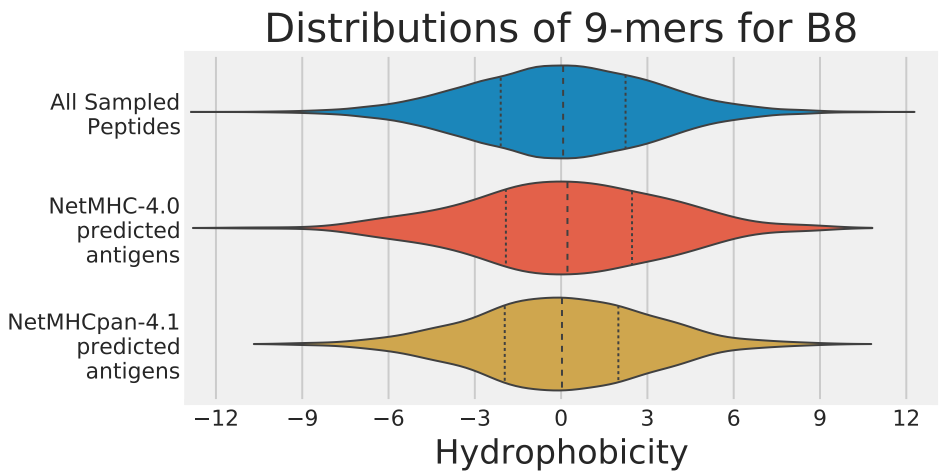


Figure 7.6: Violin plots of the hydrophobicity of the sets of strong binders predicted by NetMHC-4.0 and NetMHCpan-4.1 on the human proteome dataset for B8. The x-axis represents the hydrophobicity of a 9-mer, and the y-axis represents the frequency. The distributions of all sampled peptides (blue), strong binders predicted by NetMHC-4.0 (red), and those predicted by NetMHCpan-4.1 (yellow) are shown. The mean and two quartiles are also depicted in each distribution.

slightly more hydrophilic than the training or sampled data (expected to be approximately centered about a Moon score of -1). Neither tool exhibited this hydrophilic shift in its predictions on the training dataset, but with the human proteome, NetMHCpan-4.1 did predict strong binders centered at a hydrophobicity score of -0.775; NetMHC-4.0's mean was -0.155. The difference in predictions was statistically significant in both analyses (p-values no larger than 0.002). Again, NetMHC-4.0's predicted strong binders for B27 were more hydrophobic than NetMHCpan-4.1's.

Strong binders to B8 were expected to have two hydrophobic amino acids (L, V, M, or I) at positions 2 and 9, and two hydrophilic amino acids (R or K) at positions 3 and 5. Consequently, no major shift in hydrophobicity expected in the B8 immunopeptidomes predicted by either neural network tool. This was indeed the result observed in both analyses, and no significant difference was observed between the predictions of NetMHC-4.0 and NetMHCpan-4.1 (p-values were 0.716 and 0.425 for the training dataset analysis and the human proteome analysis respectively). In this case, NetMHC-4.0's predicted set of binders for B8 were not distinguishable from NetMHCpan-4.1's in terms of hydrophobicity.

Overall, the trend observed from these violin plots seemed to be that NetMHC-4.0 was incorrectly accounting for hydrophobicity when predicting strong binders for A2 and B27. In contrast, NetMHCpan-4.1 was predicting less hydrophobic strong binders for all HLAs in the human proteome analysis. As NetMHCpan-4.1 more closely matched the expectations for A2 and B27, its predictions were reasoned to be more accurate. Since NetMHCpan-4.1 also predicted more strong binders, the new strong binders gained in NetMHCpan-4.1's immunopeptidome were hypothesized to be slightly hydrophilic (with respect to NetMHC-4.0's immunopeptidome) and therefore skew the mean hydrophobicity lower. Each neural network tool's performance was tracked in Tables 7.1 and 7.2 to investigate this. In particular, the performances of these tools in our 3 specific peptide cases was broken down using 4 different classification metrics discussed below:

- The Accuracy metric tracks the number of true negatives and true positives identified by a classifier relative to all the tested data points. Across all HLAs, both neural network tools maintained high accuracy, though NetMHCpan-4.1 performed slightly better (by roughly 2%). For B27 in particular, NetMHCpan-4.1 had a notably higher accuracy (by about 6%) in all peptides cases. No specific improvement was observed in any individual peptide category.

- The Precision metric inversely measures the number of false positives identified by a classifier. NetMHCpan-4.1 exhibited higher precision for A2 (by 3%), roughly equivalent precision for B27, and slightly lower precision for B8 (by about 2%) in all peptide cases. The highlight here was that NetMHC-4.0 had low precision (lower than 80%) when dealing with hydrophilic peptides for A2 and B27.

- The Recall metric inversely represents the number of false negatives not identified by a classifier. NetMHCpan-4.1 showed significant improvement (consistently higher than 10%) in recall for all HLAs. For A2, these improvements were observed in classifying hydrophilic and balanced peptides. For B27 and B8, these improvements were observed in all peptide categories.

- The F1 score is a combination of precision and recall, and tracks overall performance of a classifier. For all HLAs, NetMHCpan-4.1 outperformed NetMHC-4.0 (by atleast 10%) when considering all peptides categories.

The observations from these data, in particular the accuracy and F1 score, supported the initial assumption that NetMHCpan-4.1 had stronger predictions than NetMHC-4.0 when focusing on hydrophobicity. The precision and recall scores elucidated the reasons behind this improvement: NetMHCpan-4.1 predicted fewer false positives, and much fewer false negatives for all HLAs. The sources of these false positives and negatives in NetMHC-4.0's predictions varied across the different HLAs. For A2, most false positives were found in non-balanced (hydrophobic and hydrophilic) peptides cases, while the majority of false negatives came from non-hydrophobic peptides. For B27, a few false positives were observed in the hydrophilic peptides, but most notably numerous false negatives were found across all types of peptides. For B8, false negatives in all peptides cases lowered the performance of NetMHC-4.0.

It was also important to acknowledge that NetMHCpan-4.1 had an unfair advantage over NetMHC-4.0 – the newer tool was trained on a much larger dataset. Furthermore, NetMHC-4.0 was trained on only peptide-MHC binding affinity data, while NetMHCpan-4.1 was trained on eluted ligand data that was representative of the entire antigen presentation pathways. The possibility of the small BA data in the training dataset being biased towards being hydrophobic was investigated. These values were contrasted to the mean hydrophobicity values of the EL dataset in Figures 10.12 to 10.14. In each case, the BA training data was more hydrophobic than the EL training data set. This bias was the most prominent in the A2 peptides, and least prominent in B27. This discrepancy in training data could be one of the causes for why NetMHC-4.0's predicted strong binders contained many hydrophilic false negatives.

7.3 Conclusion

A previous study by our team had identified a significant preference for hydrophobic peptides in NetMHC-4.0's predicted immunopeptidome for A2 [90]. It had argued that highly hydrophobic peptides were being classified by NetMHC-4.0 as false positives. It had suggested that highly hydrophobic peptides would never be trafficked in the aqueous cytosol of cells and were therefore obvious false positives.

This study expanded that previous research to focus on more HLA types – i.e. A2, B27, and B8. These HLAs prefer to bind hydrophobic, hydrophilic, and balanced (neither hydrophobic nor hydrophilic) peptides respectively. By comparing the predictions by NetMHC-4.0 and NetMHCpan-4.1 on both the training dataset (see Figures 10.12 to 10.14) and the sampled human proteome (see Figures 7.4 to 7.6), NetMHC-4.0's hydrophobicity bias for A2 and B27 was confirmed. In these cases, NetMHC-4.0's predicted immunopeptidome was much more hydrophobic than NetMHCpan-4.1's predictions. This hydrophobic bias was not statistically significant in the B8 immunopeptidome. These results suggested that NetMHC-4.0 struggled to predict strong binders correctly in HLAs with strong hydrophobic or hydrophilic binding motifs.

This study had used several machine learning metrics, such as accuracy, and recall, on the training dataset analysis (see Tables 7.1 and 7.2). From these results, the improvement in NetMHC-4.0's predictions (over NetMHC-4.0's) was discovered to stem from fewer false negatives in the non-balanced peptide cases, and fewer false positives in general. In particular, the biased immunopeptidome predicted by NetMHC-4.0 was not just a consequence of overestimating the binding of hydrophobic peptides, but also due to overlooking binders that were hydrophilic.

A key takeway of our analyses is that this hydrophobicity bias could only be discovered and expounded upon by focusing on hydrophobicity of peptides as a core factor in pMHC binding. Merely using machine learning metrics without accounting for such biochemical attributes would have been insufficient in capturing this bias. This is evident from how both neural network tools had similar performances across all 3 HLAs in Figures 10.18 to 10.20. Just as understanding the erroneous predictions from NetMHC-4.0 required the use of hydrophobicity as a metric, we believe that mechanistically modelling the biochemistry (to some extent) improves upon a purely data-driven artificial intelligence's prediction.

In conclusion, NetMHCpan-4.1 was the more reliable of the two neural network tools. It had stronger results across the various metrics, and the hydrophobicity of its predicted immunopeptidome matched the expected hydrophobicity values. In contrast, NetMHC-4.0 struggled to predict all strong binders for HLAs that had notable hydrophobic or hydrophilic preferences. There could be several reasons why NetMHCpan-4.1 outperformed NetMHC-4.0. NetMHCpan-4.1 had a larger training set. BA data alone could not model the effects of the entire antigen presentation pathway as EL data could have. For example, the binary values of EL data might have trained NetMHCpan-4.1 to be more decisive in its predictions. Tracking the MHC sequence could have allowed NetMHCpan-4.1 to model binding mechanics of MHC binding pockets. EL data might have set NetMHCpan to capture aspects of the entire antigen presentation pathway instead of estimating pMHC binding strength alone. The generation of negative training data [2] for NetMHCpan-4.1 could have resolved false positives that NetMHC-4.0 was vulnerable to.

Chapter 8

Investigating T Cell Immune Responses to the various SARS-CoV-2 variants

The Omicron variant of SARS-CoV-2 caused the largest wave of COVID-19 infections from November 2021 to February 2022 [66]. Since then, Omicron has been responsible for more than 95% of all recorded cases of COVID-19 according to GISAID [41]. It was observed to have a higher infectivity rate and a lower disease severity than the original COVID-19 variant [17]. Notably, Omicron could reinfect patients who had already recovered from COVID-19 and infect people who had received double-doses of COVID-19 vaccines [17]. These factors had selected for its spread over other SARS-CoV-2 variants.

COVID-19 vaccines allowed treated people to develop immunity against COVID-19 [81, 25]. The vaccines developed by BioNTech-Pfizer and Moderna-NIAID employed new mRNA vaccine technology, while those developed by Janseen-Johnson & Johnson, and Novavax featured previously established methods such as viral vectors and protein adjuvants. All of these vaccines utilized the novel Spike Glycoprotein of SARS-CoV-2 to activate a treated

This research was funded by the NSF Grant 2036064.

person's immune system [62, 38]. They trained the immune system to recognize the spike protein and develop antibodies to neutralize it. This meant that upon COVID-19 infection, a vaccinated individual's immune system could mount a stronger and faster defense. Vaccines also allowed the body to produce T cells that kill COVID-19 infected cells through the Major Histocompatibility Complex (MHC) presentation pathways.

MHC proteins of different alleles prefer binding different peptides [5]. For example, HLA-A*0201 prefers binding peptides with hydrophobic residues while HLA-B*2705 prefers binding peptides with hydrophilic residues. MHC Class I proteins also prefer binding peptides 9 amino acids long while Class II proteins prefer binding peptides of length 15. Furthermore, the MHC genes are among the most polymorphic genes in the human genome [26]. Due to all these reasons, the antigens presented from the SARS-CoV-2 spike protein by the MHC proteins can differ vastly from person to person. Individuals presenting a higher number of antigens elicit a stronger T Cell response to COVID-19 infection. Previous studies have concluded that the Omicron Variant is resistant to numerous monoclonal antibodies used in COVID-19 therapy, and evades antibodies from infected or vaccinated individuals as well [48, 31, 95]. However, T Cell responses to Omicron are mostly preserved in these individuals, suggesting that most T Cell epitopes from the original SARS-CoV-2 spike proteins are conserved in Omicron [15, 39, 9, 23, 61, 96]. These observations motivate the importance of T Cell immunity in COVID-19 cases, as this class of immunity has not been strongly evaded by the SARS-CoV-2 virus with its new variants.

This study sought to investigate the cause of this lack of evasion of T Cell Immunity by the Omicron variants. NetMHCpan-4.1 and NetMHCiipan-4.0 were used to predict MHC Class I and MHC Class II antigens respectively from various spike protein variants [78]. These state-of-the-art tools that had previously been used in vaccine design and neoantigen identification [70, 53, 49]. The predictions of these tools were used to investigate the antigens derived from the numerous SARS-CoV-2 Variants of Concern across various MHC alleles with high frequencies in human populations. These predictions estimated the different T-Cell responses elicited by the variants in most humans, and confirmed no significant loss of T-Cell epitopes in the Omicron spike protein. An original form of positive control was devised by targeting the various epitopes identified in the spike protein, and deliberately mutating these regions of the protein to cause a loss of antigens binding to MHC proteins. These mutant “evading” spike proteins did lose T Cell epitopes across numerous HLAs when tested using NetMHCpan-4.1 and NetMHCiipan-4.0, unlike Omicron. These results proved that the Omicron spike protein lacked mutations that would lead to evasion of T Cell Immunity. This implied that other stronger selective factors (such as higher infectivity rate, stronger ACE2 binding, and antibody evasion) had played a role in the evolution of the Omicron variant. The results of this study reassure that T Cell immunity forms a strong bastion against SARS-CoV-2, and should remain the focus of COVID-19 therapy.

8.1 Methods

8.1.1 SARS-CoV-2 Spike Protein and Variants

The spike protein in the original SARS-CoV-2 virus is a 1273 amino acid long protein [111, 65], and features a receptor binding domain which allows the virus to infect human cells through the angiotensin-converting enzyme 2 (ACE2) receptor [102]. Over the course of the COVID-19 pandemic from 2019 to 2022, the various Variants of Concern (VOCs) of SARS-Cov-2 reported by WHO evolved their own mutant spike proteins, with most mutations either stabilizing their spike protein or increasing its binding affinity with the ACE2 receptor. Clearly, the SARS-CoV-2 spike protein played a key role in the COVID-19 viral infection, and consequently it had been the focus of numerous studies and clinical therapies. Notably, the most widely used COVID-19 vaccines in the United States (such as those manufactured by Pfizer, and Moderna) encoded RNA transcripts for this spike protein, with minor stabilizing mutations. This study sought to assess the various T-Cell antigens produced from spike protein variants from VOCs and common vaccines.

To begin, all the key mutations with respect to the original spike protein in the different variants were identified. For each of the SARS-CoV-2 variants, the deletions and point mutations that had a frequency of greater than 33% in all the spike protein samples for that variant in the GISAID database were recorded . This was achieved through the use of the mutation tracker on outbreak.info [22, 98] and the list of shared mutations on the covariants website [30]. The mutations that were observed across the different variants were (in order of position along the spike protein):

1. B.1.1.7 (Alpha variant): H69del, V70del, Y144del, N501Y, A570D, D614G, P681H, T716I, S982A, and D1118H. There were 3 deletions and 7 point mutations.

2. B.1.351 (Beta variant): L18F, D80A, D215G, L241del, L242del, A243del, K417N, E484K, N501Y, D614G, and A701V. There were 3 deletions and 8 point mutations. Mutation L18F was not listed on the covariants website.

3. P.1 (Gamma variant): L18F, T20N, P26S, D138Y, R190S, K417T, E484K, N501Y, D614G, H655Y, T1027I, and V1176F. There were 0 deletions and 12 point mutations.

4. B.1.617.2 (Delta variant): T19R, T95I, G142D, E156G, F157del, R158del, L452R, T478K, D614G, P681R, and D950N. There were 2 deletions and 9 point mutations. Mutations T95I and G142D were not listed on the covariants website.

5. BA.1 (Omicron variant): A67V, H69del, V70del, T95I, G142D, V143del, Y144del, Y145del, N211I, L212del, G339D, S371L, S373P, S375F, K417N, N440K, G446S, S477N, T478K, E484A, Q493R, G496S, Q498R, N501Y, Y505H, T547K, D614G, H655Y, N679K, P681H, N764K, D796Y, N856K, Q954H, N969K, and L981F. There were 6 deletions and 30 point mutations.

I gratefully acknowledge all data contributors, i.e., the Authors and their Originating laboratories responsible for obtaining the specimens, and their Submitting laboratories for generating the genetic sequence and metadata and sharing via the GISAID Initiative, on which this research is based.

6. BA.2 (“Stealth” Omicron variant): T19I, L24S, P25del, P26del, A27del, G142D, V213G, G339D, S371F, S373P, S375F, T376A, D405N, R408S, K417N, N440K, S477N, T478K, E484A, Q493R, Q498R, N501Y, Y505H, D614G, H655Y, N679K, P681H, N764K, D796Y, Q954H, and N969K. There were 3 deletions and 28 point mutations.

The mutations in the newest variants of concern of 2022, i.e. BA.2.75 and BA.5, were also identified in a follow-up study. The majority of the mutations reported in these Omicron subvariants were shared by BA.2. These were:

1. BA.2.75: T19I, L24S, P25del, P26del, A27del, G142D, K147E, W152R, F157L, I210V, V213G, G257S, G339H, S371F, S373P, S375F, T376A, D405N, R408S, K417N, N440K, G446S, S477N, T478K, E484A, Q498R, N501Y, Y505H, D614G, H655Y, N679K, P681H, N764K, D796Y, Q954H, and N969K. There were 3 deletions and 33 point mutations.

2. BA.5: T19I, L24S, P25del, P26del, A27del, H69del, V70del, G142D, V213G, G339D, S371F, S373P, S375F, T376A, D405N, R408S, K417N, N440K, L452R, S477N, T478K, E484A, F486V, Q498R, N501Y, Y505H, D614G, H655Y, N679K, P681H, N764K, D796Y, Q954H, and N969K. There were 5 deletions and 29 point mutations.

The synthetic mutations in the common COVID-19 vaccines used in the United States were tracked as well. The vaccines investigated in this study were:

1. BNT162b2 (Manufactured by BioNTech-Pfizer) and mRNA-1273 (Manufactured by Moderna-NIAID): These feature the mutations K986P and V987P.

2. Ad26.COV2.S (Manufactured by Janssen-Johnson & Johnson): This features the mutations R682S, R685G, K986P, and V987P.

3. NVX-CoV2373 (Manufactured by Novavax): This features the mutations R682Q, R683Q, R685Q, K986P, and V987P.

With these key mutations compiled, a consensus spike protein sequence for each variant was generated by performing the corresponding single amino acid deletions or substitutions on the original spike protein. This statistics-based approach allowed for investigating the impact of the core, defining mutations in a variant without relying on an individual reported sample to incompletely represent that variant. However this approach did not track amino acid insertions. For example, several BA.1 variant samples reported small insertions in the N-Terminal Domain [24] but these could not be found in the mutation reporting data. Given this specific observation, the 3 a.a. sequence EPE was inserted at position 214 in the BA.1 representative spike protein. This was the only insertion performed in our analysis.

8.1.2 MHC-Peptide Binding Affinity Prediction

NetMHCpan-4.1 and NetMHCiipan-4.0 were used to investigate MHC Class I and Class II antigens respectively. Both of these tools predicted the binding probability of a peptide with a given MHC molecule on a continuous scale of 0 to 1 – 0 meant no affinity, and 1 meant the strongest binding possible. Both these tools classified strong binding peptides to a MHC molecule using a 0.5 percentile threshold with respect to the training data for that MHC.

For the MHC Class I analysis, the following HLA serotypes were shortlisted: HLA-A*0101 (A1), HLA-A*0201 (A2), HLA-A*0301 (A3), HLA-A*2402 (A24), HLA-A*2601 (A26), HLA-A*3001 (A30), HLA-B*1501 (B15), HLA-B*3501 (B35), HLA-B*4001 (B40), HLA-B*4402 (B44), and HLA-B*5101 (B51). For the MHC Class II predictions, HLA-DRB1*0101 (DR1), HLA-DRB1*0301 (DR3), HLA-DRB1*0401 (DR4), HLA-DRB1*0701 (DR7), HLA-DRB1*0801 (DR8), HLA-DRB1*1101 (DR11), DRB1*1201 (DR12), HLA-DRB1*1301 (DR13), HLA-DRB1*1302 (DR1302), and HLA-DRB1*1501 (DR15) were selected. These HLA supertypes were chosen for this study because they were amongst the most frequent alleles while also representing a broad and diverse sample of the human population's HLA types in the United States [50].

For each of the spike protein variants, the fasta-file sequence of the protein was put through both prediction tools, and the binding data for all the aforementioned MHC molecules was gathered. For MHC Class I predictions, only 9-mers – contiguous sequences of 9 amino acids generated from the protein – were analyzed as they bound the strongest to MHC Class I molecules compared to all other peptide lengths. For the same reason, 15-mers were the focus for MHC Class II molecules. The binding scores gathered from this prediction data allowed for identifying the antigens presented by various MHC molecules from each spike protein.

8.1.3 Generating Evaders as Positive Controls

As highlighted in Section 8.1.2, some form of control was needed to assess the significance of the number of predicted antigens being consistent across different variants. The approach chosen to resolve this was to design new spike protein variants as a form of positive control. These spike proteins featured the same number of mutations as the most mutated SARS-CoV-2 variant. However, these spike proteins were specifically engineered to knock down the number of antigens across multiple HLA types. Such spike proteins acted as positive controls since they would reflect the strongest MHC pathway knock down relative to all our tested variants. These engineered spike proteins were labelled as "evaders". Two sets of evaders were engineered – one for the MHC Class I pathway, and the other for Class II.

To design a Class I evader spike protein, sites on the original spike protein that were critical for deriving antigens had to be identified. The location of all strong binding peptides from the original spike protein for each MHC Class I HLA along the 1273 amino acid sequence was mapped on to the protein sequence. This highlighted *regions of antigenicity* – positions from the original spike protein that constituted peptides that were presented as antigens in the MHC Class I pathway. From here, the anchor residues in these regions of antigenicity were singled out. This information was compiled into a set of 11 lists – each list corresponding to an HLA type, and the numbers in each list representing the position of anchor residues for that HLA in the original spike protein. These anchor residues were mutated to generate the best evader proteins. A limit of 36 mutations (30 point mutations and 6 deletions, which were the number of mutations of BA.1) was used. The number of mutations allocated to evade an HLA was also scaled according to the number of antigens predicted for that HLA by NetMHCpan-4.1 (see Section 8.2.1). Therefore 3, 3, 4, 3, 4, 4, 4, 5, 2, 2, and 2 positions (a total sum of 36) were sampled randomly from the 11 lists respectively. These positions were then mutated. For each point mutation, the new substitute amino acid was randomly chosen from a set of amino acids that were antagonistic to the HLA that was anchored by that site. For example, a residue chosen for mutation that was originally an anchor residue for A2 binding was only allowed to mutate to amino acids besides V, L, M, and I. With this procedure a 100 Class I evader proteins were generated. The full details on the mutations protocol are discussed in Table 8.1.

To design a Class II evader spike protein,the same procedure to identify anchor residues was followed. In this case there were 10 lists and 6, 2, 3, 5, 2, 2, 3, 2, 6, and 5 numbers (again adding up to 36) were sampled from each respectively. The same mutation protocol was repeated to engineer a 100 Class II evaders.

The procedure discussed in Section 8.1.2 was repeated on the designed evaders to investigate the antigens predicted from them. Only the NetMHCpan-4.1 predictions for the Class I evaders, and NetMHCiipan-4.0 predictions for the Class II evaders were tested.

8.1.4 Statistics and Ranking

For each of the 100 Class I evader proteins, the number of strong binders predicted by NetMHCpan-4.1 was tracked across all the 11 aforementioned HLAs. Here the goal was to analyze whether the 10 *nonrandom* variants (i.e. the original spike, the natural variants, and the vaccine variants) exhibited a different antigen profile compared to the evader proteins. The Wilcoxon rank-sum test from the SciPy library was used to analyze whether any two sets of spike proteins possessed the same distribution of antigens. For each of the 11 HLAs, the number of nonrandom variants' predicted binders with the

Table 8.1: Anchor Residue positions identified from strong binding peptides for all investigated HLAs. The amino acids that were restricted from these positions (when creating evaders) are also listed.

HLA	Anchor Residues	Restricted Amino Acids
A1	8	Y
A2	1, 8	L, M, I, V
A3	8	K, R
A24	1	Y, F, W,L
A26	8	Y, F, M
A30	0,2, 8	R, K, V
B15	8	Y, F
B35	1, 8	P, A, Y, F, M
B40	1	E
B44	1	E
B51	1	P, A
DR1	0, 3, 5, 8	F, Y, L, G, A, V, I
DR3	3	D
DR4	0	F, Y, I, L
DR7	0, 3	F, Y, I , L, S, T, V
DR8	5, 8	K, R, D, E
DR11	5	K, R
DR12	0, 3	I, L, V
DR13	5	R, K
DR1302	0, 3	I, F, L, N, D
DR15	3	Y, F

evasive variants' predicted binders were compared. Furthermore, the BA.1 Omicron variant was compared to both the nonrandom set and the evasive set of proteins. The aim of Wilcoxon test was to highlight that the set of nonrandom variants reported a different distribution of number of antigens for an individual HLA than the evasive proteins. Simply phrased, would the evasive variants reported a different number of antigens across the various HLAs?

Before delving into this query, it had to be noted that the test could only focus on an individual HLA each time. Therefore 3 different multi-HLA metrics were utilized to understand the significance across all the HLAs. The first metric was a simple sum – for each variantall the antigens across different HLAs were added up to track the *total number of antigens*. This metric was useful for visualizing how many antigens were being lost over different evaders (the larger the total, the more the antigens), but was mostly determined by the few HLAs that presented many more antigens than others. So, the *cartesian distance* between a variant and the original spike was used as the second metric. Each variant was treated as a data point with each HLA as a coordinate, and the root of the sum of the squares of its difference from the original spike was calculated. The larger the cartesian distance, the more the antigen profile of a variant was different from the original spike. This metric did not scale down different HLAs with greater antigen variance, so a third rank metric to merge the multidimensional HLA data was devised. For each individual HLA, all the spike proteins (from the 110 total nonrandom and evasive variants) were ranked based on the number of predicted antigens for that HLA in ascending order. Proteins that shared the same number of binders were assigned the same average rank. With all 110 proteins being ranked across all 11 HLAs, the various ranks (for each HLA) for a single protein were added up into a single *sum of ranks* score. The Wilcoxon rank-sum test was repeated on all 3 of these metrics to identify if the antigen profiles of the nonrandom and evasive variants were distinguishable when considering all HLAs. Again, Omicron BA.1 was compared to see which set of proteins it matched better with.

he analysis discussed above was repeated for the 100 Class II evader

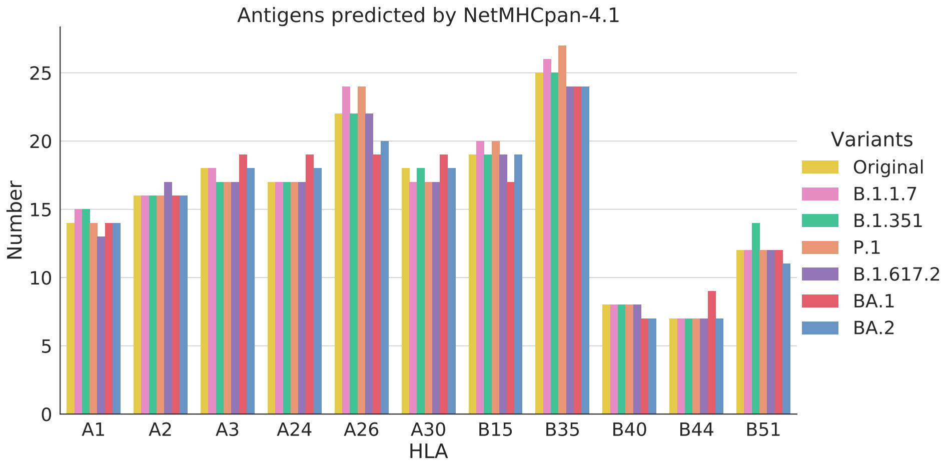


Figure 8.1: MHC Class I antigens predicted from the multiple SARS-CoV-2 variant proteins by NetMHCpan-4.1. The x-axis tracks the various common Class I HLA types while the y-axis reports the number of antigens. The results for different variants (from the original spike to both omicron variants) are shown in different colored bins for each HLA.

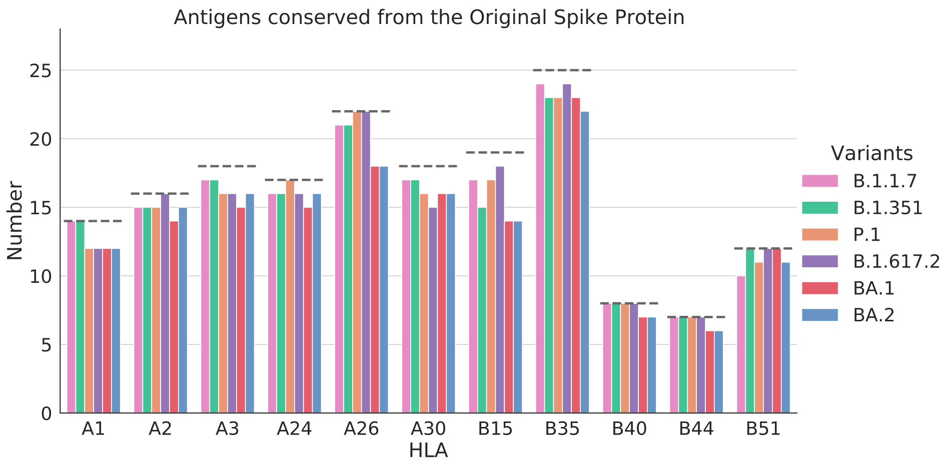


Figure 8.2: MHC Class I antigens from the original spike that were preserved in each variant spike according to NetMHCpan-4.1. For each HLA on the x-axis, the dashed line represents the number of antigens predicted from the original spike. The different variants are represented by the different colored bins. Each bin measures how many of the original spike antigens were conserved in a variant's predicted set of antigens.

proteins across the 10 Class II HLAs as well.

8.2 Results

8.2.1 MHC Class I

NetMHCpan-4.1 Results on Spike Protein Variants

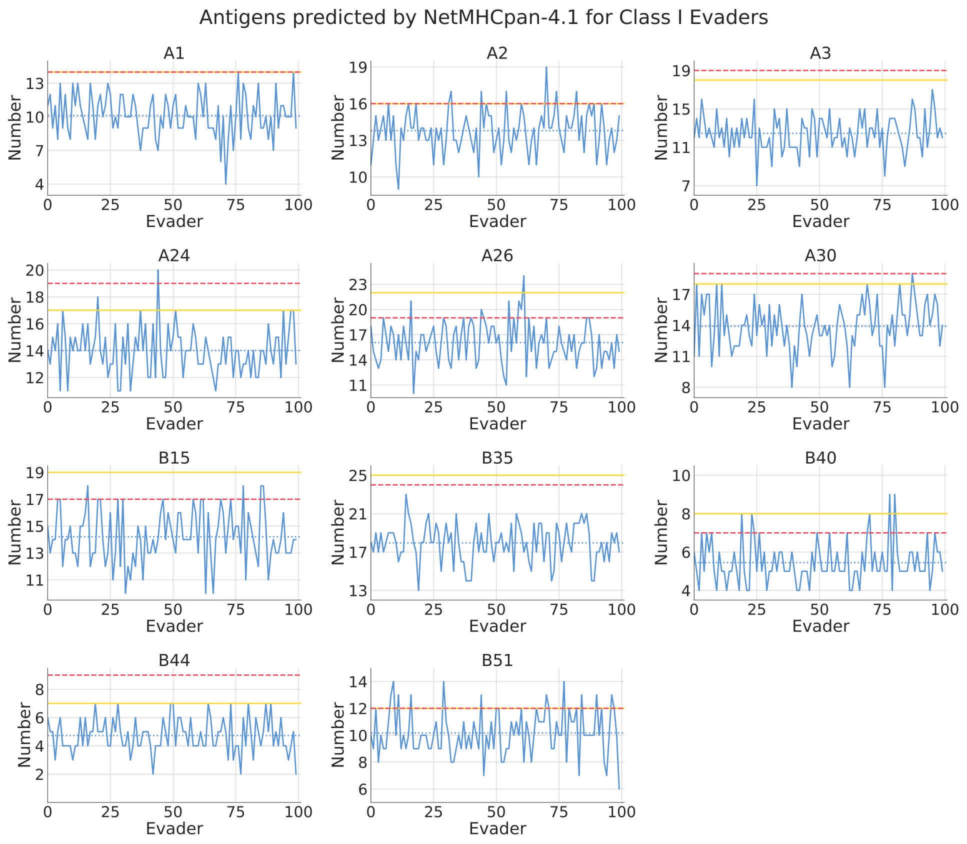


Figure 8.3: The number of strong binding peptides from each Class I evader protein predicted by NetMHCpan-4.1 for all target HLAs. The x-axis represents all 100 evaders and the y-axis represents the number of antigens. The evader predictions are shown in the blue line plot, with the mean number of antigens for all evaders in dotted blue. The number of predicted binders from the original spike is shown in solid yellow, and the number from Omicron BA.1 is shown in dashed red.

For each of the constructed spike protein variants, the number of antigens predicted by NetMHCpan-4.1 is shown in Figure 8.1. The results for the vaccines and the newer Omicron subvariants are presented in Table 10.7. For each of the 11 Class I molecules, these numbers of antigens were mostly unchanged across the different spike variants (the largest drop was observed for A26 between the original spike protein and BA.1 – a loss of 3 binders from the original 22, which represents a 13% drop in number of antigens). That is, no particular variant exhibited a significant knockout of predicted Class I antigens compared to the original spike protein's antigens. In particular, the two Omicron variants did not exhibit a drop in total number of antigens for molecules A1, A2, A3, A24, A30, and B44. In some cases (namely for A26, B35, and B51) the Alpha, Beta, and Gamma variants even gained more predicted antigens over the original spike protein. This suggested that the number of antigens derived from the different spike proteins for an individual Class I HLA's predictions is relatively consistent. The mutations present in newer variants, including Omicron, did not lower the number of antigens presented by the Class I pathway.

The number of original SARS-CoV-2 peptides that were preserved in the newer variants was also investigated. That is, for each spike variant, the antigens that were also present in the original spike were counted up. These results are shown in Figure 8.2 and Table 10.8. The two Omicron variants show a drop in the number of antigens across all HLAs. This meant that certain mutations in these variants occured in regions of antigenicity of the spike protein. However, since the total number of antigens from these variants was equivalent to the original spike protein's antigens (see Figure 8.1), these mutations did not knockout the antigenicity of the spike protein. Instead, they merely altered the antigen footprint of the spike protein variants without actually impacting their net antigenicity.

NetMHCpan-4.1 Results on Evaders

The numbers of Class I antigens that were predicted by NetMHCpan-4.1 from the set of a 100 Class I evader proteins are shown in Figure 8.3. For each HLA, the number of antigens counted in both the original spike and the omicron BA.1 spike are reported. Furthermore, the mean numbers of antigens from the evaders are also shown. Clearly for all HLAs, the set of evaders reported a consistently lower number of antigens compared to both natural spike proteins. These results suggested that the targeted random mutations used to construct the evaders effectively knocked down the antigenicity of the evader spikes across multiple HLAs. Note that since the 36 targeted mutations in an evader protein were distributed to lower antigenicity across all HLAs, no complete knockout of antigens was observed for a single HLA. Instead, a notable (but not complete) knockdown was observed for all HLAs. In summary, the mutations sampled contributed to successful MHC Class I evasion by the evader proteins.

Statistical Significance

The Wilcoxon rank-sum test was used to measure how successful the evaders were in lowering antigenicity of their spike proteins. The results for the single HLA tests are shown in Table 10.5. For all 11 HLAs, the set of evader proteins and the set of nonrandom proteins (natural variants and vaccines) formed statistically different distributions. This suggested that the evaders significantly knocked down antigenicity of the spike protein for each HLA. Furthermore, for each case the omicron variant compared more favorably with the nonrandom set than the evader set.

The 3 different multi-HLA metrics (as discussed in Section 8.1.4) were also tested to ensure that the observations for single HLAs applied to the entire catalogue of MHC Class I molecules. The distribution of these metrics for all evaders and nonrandom proteins is shown in Figure 8.4. The first plot shows that the total antigens count, i.e. the sum of antigens from all tested HLAs, of all the evaders were lower than all nonrandom proteins. The second plot shows that all evaders had a higher cartesian distance from the original spike protein's antigen profile in comparison to the other nonrandom variants. Lastly, the third plot depicts that all evaders possessed a lower sum of ranks score than the nonrandom proteins. All three of these metrics

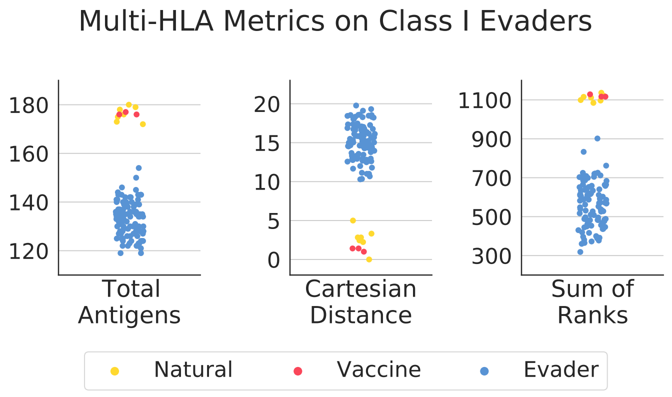


Figure 8.4: The distribution of the spike protein variants using the total antigens, cartesian distance, and sum of ranks metrics respectively. All Class I evaders are shown in blue, all natural SARS-CoV-2 are shown in yellow, and all vaccine spikes are shown in red. For each metric, the distributions of the evader proteins and the nonrandom (natural plus vaccine) are notably disparate.

therefore confirmed that the evaders knocked down antigenicity of the spike protein across all tested HLAs simultaneously. The Wilcoxon rank-sum test reported the same conclusion as the single HLA tests, as shown in Table 8.2. That is, the Class I evaders bore mutations that consistently knocked down the antigenicity of the spike protein across a whole catalogue of Class I MHCs. In fact, these evaders achieved these results while possessing the same number of mutations (36) as Omicron BA.1. Therefore the mutations seen in Omicron (and all other natural variants discussed in the paper) did not contribute to evasion of the MHC Class I pathway in one or many HLA alleles.

8.2.2 MHC Class II

NetMHCiipan-4.0 Results on Spike Protein Variants

The prediction results of NetMHCiipan-4.0 for all variants are shown in Figure 8.5. Again, the number of predicted strong binders across different variants was relatively equivalent for each HLA allele (the largest drop was

Table 8.2: P-values for the Wilcoxon rank-sum test for multi-HLA metrics in Class I.

Metric	Nonrandom vs. Evasive	Omicron vs. Nonrandom	Omicron vs. Evasive
Total Antigens	2.0×10^{-7}	4.2×10^{-1}	8.6×10^{-2}
Cartesian Distance	2.0×10^{-7}	1.5×10^{-1}	8.6×10^{-2}
Sum of Ranks	2.0×10^{-7}	4.2×10^{-1}	8.6×10^{-2}

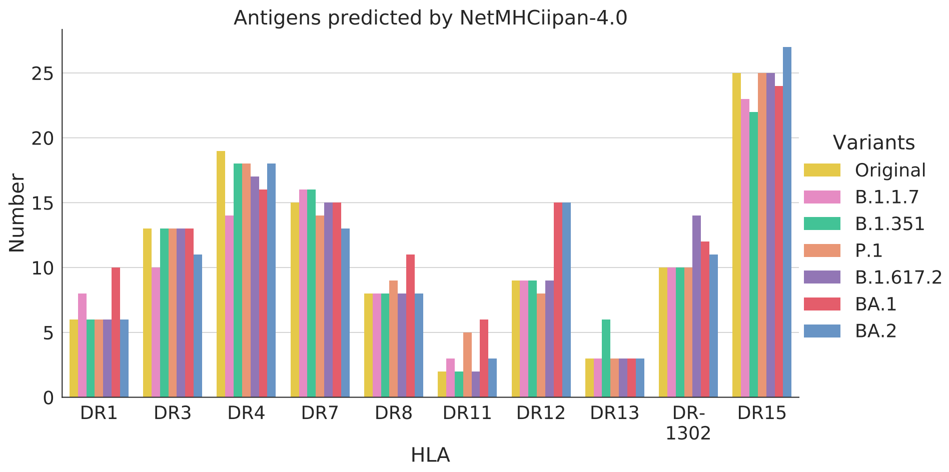


Figure 8.5: MHC Class II antigens predicted from the multiple SARS-CoV-2 variant proteins by NetMHCiipan-4.0. The x-axis tracks the various common Class II HLA types while the y-axis reports the number of antigens. The results for different variants are shown in different colored bins for each HLA.

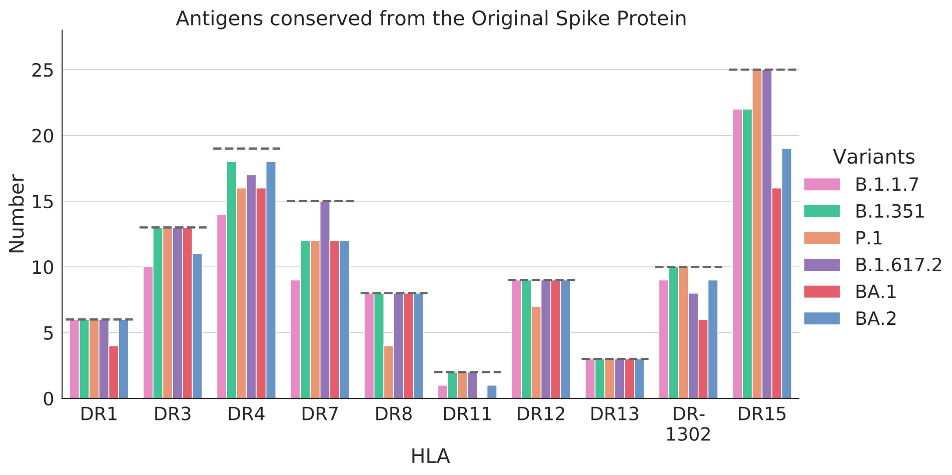


Figure 8.6: MHC Class II antigens from the original spike that were preserved in each variant spike according to NetMHCpan-4.1. For each HLA on the x-axis, the dashed line represents the number of antigens predicted from the original spike. The different variants are represented by the different colored bins. Each bin measures how many of the original spike antigens were conserved in a variant's predicted set of antigens.

observed for DR4 between the original spike protein and B.1.1.7 – a loss of 5 binders from the original 19, which represents a 26% drop in number of antigens). For DR1, DR8, DR11, DR12, DR13, and DR1302, the two Omicron variants did not lead to a loss in number of antigens compared to the original spike protein. In the case of DR11, which bound the least number of antigens from the original spike at 2, all natural variants (i.e. Alpha through both Omicrons) did not yield fewer antigens. Thus similar to the Class I predictions, the Class II predictions suggested that the mutations occurring in the natural SARS-CoV-2 variants did not lead to loss of presented antigens compared to the original spike protein.

Again, the antigens from the original spike that were preserved in the newer variants were also traced. These results are shown in Figure 8.6. As was observed in the Class I results, the newer variants, in particular omicron, had fewer of the original antigens in several HLAs (such as DR4, DR7 and DR15). However, the net antigenicity of each variant was relatively equivalent as seen in Figure 8.5. This meant that the mutations they carried merely altered the antigens that were presented and did not lower overall antigenicity.

NetMHCiipan-4.0 Results on Evaders

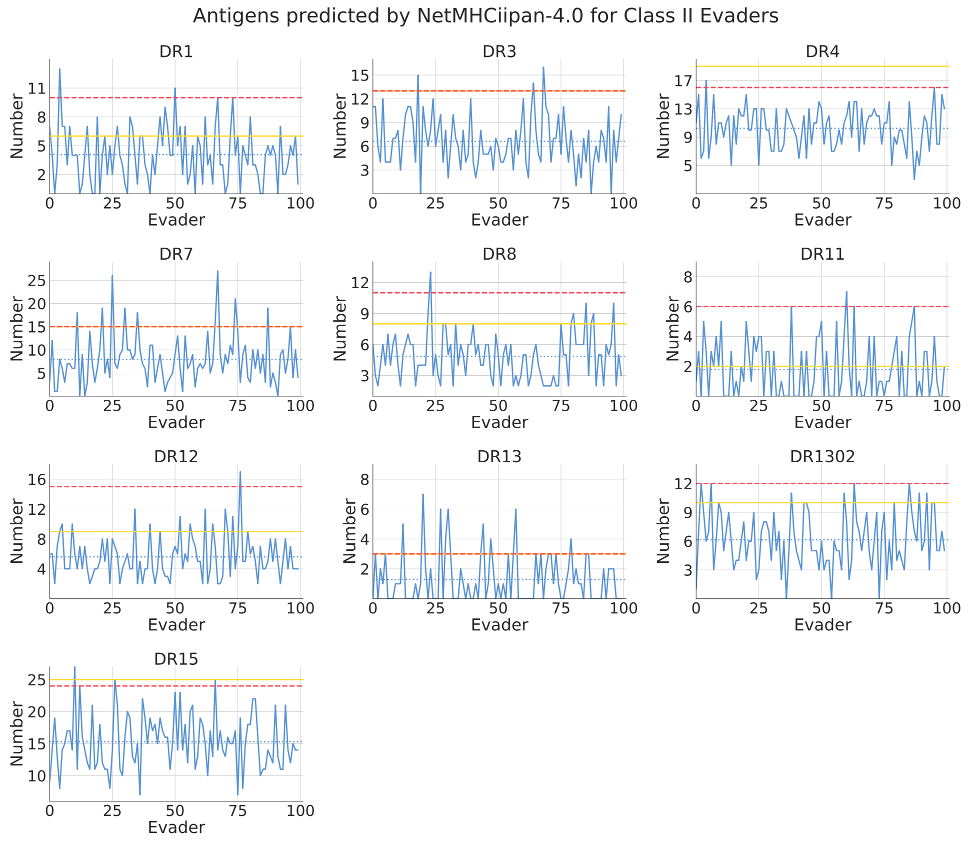


Figure 8.7: The number of strong binding peptides from each Class II evader protein predicted by NetMHCiipan-4.0 for all target HLAs. The x-axis represents all 100 evaders and the y-axis represents the number of antigens. The evader predictions are shown in the blue line plot, with the mean across all evaders in dotted blue. The number of predicted binders from the original spike is shown in solid yellow, and the number from Omicron BA.1 is shown in dashed red.

NetMHCiipan-4.0's prediction results on our 100 class II evaders are shown in Figure 8.7. Again, the number of predicted antigens is shown alongside the mean for all evaders and compared with the number of antigens from the original spike and the Omicron variant. For all HLAs except DR11, the sets of evaders reported a lower number of antigens than the Omicron variant. Interestingly, for DR7 and DR13 several of the evaders possessed more antigens than Omicron (for DR13, this could be because there were so few antigens already that it was easier to mutate new DR13 antigens from other HLA mutation sites in the evaders). Nonetheless, these results pointed out that the 36 mutations applied in most evaders led to a drop in antigenicity across all HLAs. Some HLAs, such as DR1, DR11, and DR13 even reported a complete knockout of antigens in many evaders.

Statistical Significance

The Wilcoxon rank-sum test results for the single class II HLA tests are shown in Table 10.6. Similarly to the class I results, the set of class II evader proteins and the set of nonrandom proteins formed statistically different distributions. The only outlier here was DR11 – possibly due to the low number of antigens for DR11 by Omicron. For all the other HLAs, this suggested that the evaders significantly knocked down antigenicity of the spike protein. For these cases the omicron variant compared more favorably with the nonrandom set than the evader set as well.

Again, the 3 different multi-HLA metrics were calculated. The distributions of these metrics for all evaders and nonrandom proteins is shown in Figure 8.8. The first plot shows that the total antigens count of all the evaders were lower than all nonrandom proteins. The second plot shows that all evaders had a higher cartesian distance from the original spike protein's antigen profile in comparison to the other nonrandom variants. Lastly, the third plot depicts that all evaders possessed a lower sum of ranks score than the nonrandom proteins. Again, all three metrics confirmed the antigenicity knockdown across a catalogue of Class II MHc molecules by the evaders. The Wilcoxon rank-sum test on these metrics, as shown in Table 8.3, led to the same conclusion as the single HLA tests. The Class II evaders, with just

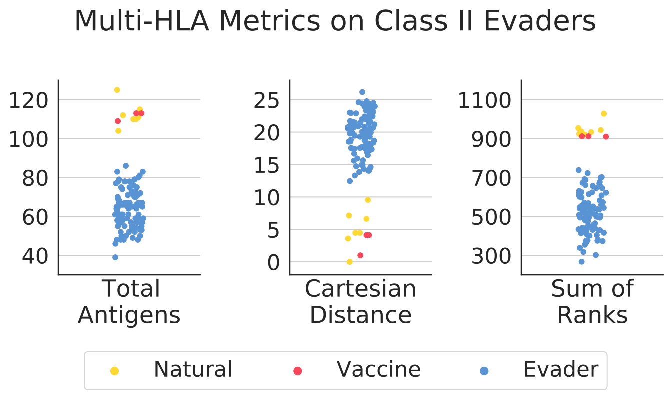


Figure 8.8: The distribution of the protein variants using the total antigens, cartesian distance, and sum of ranks metrics respectively. All Class II evaders are shown in blue, all natural SARS-CoV-2 are shown in yellow, and all vaccine spikes are shown in red. For each metric, the distributions of the evader proteins and the nonrandom (natural plus vaccine) are notably disparate.

Table 8.3: P-values for the Wilcoxon rank-sum test for multi-HLA metrics in Class II.

Metric	Nonrandom vs. Evasive	Omicron vs. Nonrandom	Omicron vs. Evasive
Total Antigens	2.0×10^{-7}	1.5×10^{-1}	8.6×10^{-2}
Cartesian Distance	2.0×10^{-7}	1.5×10^{-1}	8.6×10^{-2}
Sum of Ranks	2.0×10^{-7}	1.5×10^{-1}	8.6×10^{-2}

36 mutations from the original spike, were able to evade the MHC Class II pathway in numerous HLAs. Even in the case of DR11, the one outlier, the evaders were able to knockout antigens in several instances. These results suggested that the mutations reported in Omicron did not allow for Class II evasion.

8.3 Conclusion

T Cells play a vital role in immunity to viral infections. The MHC antigen presentation pathway allows for identifying more antigens, in particular peptides from the inside of a viral protein, than antibodies allow. Furthermore, T Cell levels have also been noted to last longer than antibodies levels for COVID-19 vaccine immunization [**?**]. As previous studies and our analysis have shown, T Cell immunity is particularly resilient to the numerous variants of concern for SARS-CoV-2 [39]. In particular, the two Omicron variants (BA.1 and BA.2) generated equivalent numbers of antigens in comparison to the original Wuhan spike protein according to MHC Class I and Class II antigen prediction models. Despite many peptide antigens from the original spike being lost in the newer variants, the mutations in these variants did not decrease the number of antigens across several common HLA types.

These observations were further corroborated by deliberately knocking out T Cell antigens out of the spike protein through targeted mutations. The engineered evasive spike protein variants, despite being limited to the same number of mutations as Omicron BA.1, successfully lowered the number of antigens across numerous highly frequent HLAs. These results suggested that the mutations in the Omicron spike variant were not selected for T Cell immunity evasion, and were probably more impactful in spike protein stability or ACE2 receptor binding affinity. This explained why despite the Omicron variant being capable of neutralizing several pharmaceutical antibodies and infecting vaccinated patients, it did not evade T Cell immunity obtained by the common COVID-19 vaccines. Lastly, the protocol for engineering evader proteins devised in this study opens up the possibility of using experimental control *in silica*. This will be beneficial for quickly understanding the evasion of T Cell immunity by future SARS-CoV-2 variants.

Chapter 9

Discussion

Prediction of the binding of peptides to MHC Class I and II proteins is important because it presents insight into how the human immune system recognizes and responds to diseases such as COVID-19 and cancer. In the latter study, simple machine learning tools such as NetMHCpan were capable of explaining why T cell immune responses to SARS-CoV-2 were preserved. The drawbacks of machine learning tools, including the lack of interpretable results, can be mitigated with stronger training practices such as accounting for the complex biochemistry underlying problems such as peptide-MHC binding. Such improvements can only be achieved in bioinformatics when knowledge from both computer science and biology are merged.

The gold standard of peptide-MHC modelling is, of course, structural data. However, the cost and infrastructure needed for generating such experimental data is a limiting factor. This, alongside the explosion of data gathered from simpler, more accessible experiments such as binding affinity assays and mass spectrometry, has carved a niche for machine learning tools in bioinformatics. Though tools such as PANDORA [52] might provide more reliable predictions through their efficient energy minimization simulations, the future lies in hybrid technologies that offer the same accuracy but staggering speed of machine learning.

Bibliography

[1] Adleman, L. M. Molecular computation of solutions to combinatorial problems. *Science* (1994), 1021–1024.

[2] Alvarez, B., Reynisson, B., Barra, C., Buus, S., Ternette, N., Connelley, T., Andreatta, M., and Nielsen, M. NNAlign_MA; MHC peptidome deconvolution for accurate MHC binding motif characterization and improved T-cell epitope predictions. *Molecular & Cellular Proteomics 18*, 12 (2019), 2459–2477.

[3] Andreatta, M., and Nielsen, M. Gapped sequence alignment using artificial neural networks: application to the MHC class I system. *Bioinformatics 32*, 4 (2016), 511–517.

[4] Athreya, N., Milenkovic, O., and Leburton, J.-P. Detection and mapping of dsDNA breaks using graphene nanopore transistor. *Biophysical Journal 116*, 3 (2019), 292a.

[5] Bassani-Sternberg, M., Chong, C., Guillaume, P., Solleder, M., Pak, H., Gannon, P. O., Kandalaft, L. E., Coukos, G., and Gfeller, D. Deciphering HLA-I motifs across HLA peptidomes improves neo-antigen predictions and identifies allostery regulating HLA specificity. *PLoS computational biology 13*, 8 (2017), e1005725.

[6] Bernstein, S. N. Démonstration du théorème de Weierstrass fondée sur le calcul des probabilités. *Communications of the Kharkov Mathematical Society 13* (1912), 1–2.

[7] Ceze, L., Nivala, J., and Strauss, K. Molecular digital data storage using DNA. *Nature Reviews Genetics 20*, 8 (Aug 2019), 456–466.

[8] Chen, T., Solanki, A., and Riedel, M. Parallel pairwise operations on data stored in DNA: Sorting, shifting, and searching. In *27th International Conference on DNA Computing and Molecular Programming (DNA 27)* (2021), Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[9] Choi, S. J., Kim, D.-U., Noh, J. Y., Kim, S., Park, S.-H., Jeong, H. W., and Shin, E.-C. T cell epitopes in SARS-CoV-2 proteins are substantially conserved in the omicron variant. *Cellular & molecular immunology 19*, 3 (2022), 447–448.

[10] Chowell, D., Krishna, S., Becker, P. D., Cocita, C., Shu, J., Tan, X., Greenberg, P. D., Klavinskis, L. S., Blattman, J. N., and Anderson, K. S. TCR contact residue hydrophobicity is a hallmark of immunogenic CD8+ T cell epitopes. *Proceedings of the National Academy of Sciences 112*, 14 (2015), E1754–E1762.

[11] Church, G., Gao, Y., and Kosuri, S. Next-generation digital information storage in DNA. *Science (New York, N.Y.) 337* (08 2012), 1628.

[12] Consortium, U. UniProt: a worldwide hub of protein knowledge. *Nucleic acids research 47*, D1 (2019), D506–D515.

[13] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[14] Cornette, J. L., Cease, K. B., Margalit, H., Spouge, J. L., Berzofsky, J. A., and DeLisi, C. Hydrophobicity scales and computational techniques for detecting amphipathic structures in proteins. *Journal of molecular biology 195*, 3 (1987), 659–685.

[15] Cox, R. J., and Brokstad, K. A. Not just antibodies: B cells and T cells mediate immunity to COVID-19. *Nature Reviews Immunology 20*, 10 (2020), 581–582.

[16] Doytchinova, I. A., and Flower, D. R. A comparative molecular similarity index analysis (comsia) study identifies an hla-a2 binding supermotif. *Journal of computer-aided molecular design 16*, 8 (2002), 535–544.

[17] Fan, Y., Li, X., Zhang, L., Wan, S., Zhang, L., and Zhou, F. SARS-CoV-2 omicron variant: recent progress and future perspectives. *Signal Transduction and Targeted Therapy 7*, 1 (2022), 1–11.

[18] Fitzgerald, P. R., and Paegel, B. M. Dna-encoded chemistry: drug discovery from a few good reactions. *Chemical reviews 121*, 12 (2020), 7155–7177.

[19] Flynn, M. J. Some computer organizations and their effectiveness. *IEEE Transactions on Computers C-21*, 9 (1972), 948–960.

[20] Fulekar, M. *Bioinformatics in Life and Environmental Sciences.* Springer, 2009.

[21] Gaines, B. Stochastic computing systems. In *Advances in Information Systems Science*, vol. 2. Plenum Press, 1969, ch. 2, pp. 37–172.

[22] Gangavarapu, K., Latif, A. A., Mullen, J. L., Alkuzweny, M., Hufbauer, E., Tsueng, G., Haag, E., Zeller, M., Aceves, C. M., Zaiets, K., et al. Outbreak.info genomic reports: scalable and dynamic surveillance of SARS-CoV-2 variants and mutations. *Research square* (2022).

[23] Gao, Y., Cai, C., Grifoni, A., Müller, T. R., Niessl, J., Olofsson, A., Humbert, M., Hansson, L., Österborg, A., Bergman, P., et al. Ancestral SARS-CoV-2-specific T cells cross-recognize the omicron variant. *Nature medicine 28*, 3 (2022), 472–476.

[24] Gerdol, M., Dishnica, K., and Giorgetti, A. Emergence of a recurrent insertion in the N-terminal domain of the SARS-CoV-2 spike glycoprotein. *Virus research 310* (2022), 198674.

[25] GeurtsvanKessel, C. H., Geers, D., Schmitz, K. S., Mykytyn, A. Z., Lamers, M. M., Bogers, S., Scherbeijn, S., Gommers, L., Sablerolles, R. S., Nieuwkoop, N. N., et al. Divergent SARS-CoV-2 omicron–reactive T and B cell responses in COVID-19 vaccine recipients. *Science immunology 7*, 69 (2022), eabo2202.

[26] Gourraud, P.-A., Khankhanian, P., Cereb, N., Yang, S. Y., Feolo, M., Maiers, M., D. Rioux, J., Hauser, S., and Oksenberg, J. HLA diversity in the 1000 genomes dataset. *PloS one 9*, 7 (2014), e97282.

[27] Grifoni, A., Weiskopf, D., Ramirez, S. I., Mateus, J., Dan, J. M., Moderbacher, C. R., Rawlings, S. A., Sutherland, A., Premkumar, L., Jadi, R. S., et al. Targets of T cell responses to SARS-CoV-2 coronavirus in humans with COVID-19 disease and unexposed individuals. *Cell 181*, 7 (2020), 1489–1501.

[28] Grover, W. H., and Mathies, R. A. An integrated microfluidic processor for single nucleotide polymorphism-based dna computing. *Lab on a Chip 5*, 10 (2005), 1033–1040.

[29] Haber, C., and Wirtz, D. Magnetic tweezers for DNA micromanipulation. *Review of Scientific instruments 71*, 12 (2000), 4561–4570.

[30] Hodcroft, E. B. CoVariants: SARS-CoV-2 mutations and variants of interest, 2021.

[31] Hoffmann, M., Krüger, N., Schulz, S., Cossmann, A., Rocha, C., Kempf, A., Nehlmeier, I., Graichen, L., Moldenhauer, A.-S., Winkler, M. S., et al. The omicron variant is highly resistant against antibody-mediated neutralization: Implications for control of the COVID-19 pandemic. *Cell 185*, 3 (2022), 447–456.

[32] Hopp, T. P., and Woods, K. R. A computer program for predicting protein antigenic determinants. *Molecular immunology 20*, 4 (1983), 483–489.

[33] Horn, F., and Jackson, R. General mass action kinetics. *Archive for rational mechanics and analysis 47*, 2 (1972), 81–116.

[34] Horner, D. S., Pavesi, G., Castrignano, T., De Meo, P. D., Liuni, S., Sammeth, M., Picardi, E., and Pesole, G. Bioinformatics approaches for genomics and post genomics applications of next-generation sequencing. *Briefings in bioinformatics 11*, 2 (2010), 181–197.

[35] Jardetzky, T., Lane, W., Robinson, R., Madden, D., and Wiley, D. Identification of self peptides bound to purified hla-b27. *Nature 353*, 6342 (1991), 326–329.

[36] Jenson, D., and Riedel, M. A deterministic approach to stochastic computation. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (2016), pp. 1–8.

[37] Jia, X., Wang, Y., Huang, Z., Zhang, Y., Yang, J., Qu, Y., Cockburn, B., Han, J., and ZHAO, W. *Spintronic Solutions for Stochastic Computing.* 02 2019, pp. 165–183.

[38] Kashte, S., Gulbake, A., El-Amin III, S. F., and Gupta, A. Covid-19 vaccines: rapid development, implications, challenges and future prospects. *Human cell 34*, 3 (2021), 711–733.

[39] Keeton, R., Tincho, M. B., Ngomti, A., Baguma, R., Benede, N., Suzuki, A., Khan, K., Cele, S., Bernstein, M., Karim, F., et al. SARS-CoV-2 spike T cell responses induced upon vaccination or infection remain robust against omicron. *MedRxiv* (2021).

[40] Kembel, S. W., Cowan, P. D., Helmus, M. R., Cornwell, W. K., Morlon, H., Ackerly, D. D., Blomberg, S. P., and

Webb, C. O. Picante: R tools for integrating phylogenies and ecology. *Bioinformatics 26*, 11 (2010), 1463–1464.

[41] Khare, S., Gurry, C., Freitas, L., Schultz, M. B., Bach, G., Diallo, A., Akite, N., Ho, J., Lee, R. T., Yeo, W., et al. GISAID's role in pandemic response. *China CDC Weekly 3*, 49 (2021), 1049.

[42] Krug, J., and Spohn, H. Universality classes for deterministic surface growth. *Physical Review A 38*, 8 (1988), 4271.

[43] Kyte, J., and Doolittle, R. F. A simple method for displaying the hydropathic character of a protein. *Journal of molecular biology 157*, 1 (1982), 105–132.

[44] Li, L., Jiang, W., and Lu, Y. A modified gibson assembly method for cloning large DNA fragments with high gc contents. *Synthetic Metabolic Pathways: Methods and Protocols* (2018), 203–209.

[45] Li, W. Power spectra of regular languages and cellular automata. *Complex Systems 1*, 1 (1987), 107–130.

[46] Liou, J.-J., Cheng, K.-T., Kundu, S., and Krstic, A. Fast statistical timing analysis by probabilistic event propagation. In *Design Automation Conference* (2001), pp. 661–666.

[47] Liu, K., Pan, C., Kuhn, A., Nievergelt, A. P., Fantner, G. E., Milenkovic, O., and Radenovic, A. Detecting topological variations of DNA at single-molecule level. *Nature communications 10*, 1 (2019), 1–9.

[48] Liu, L., Iketani, S., Guo, Y., Chan, J. F.-W., Wang, M., Liu, L., Luo, Y., Chu, H., Huang, Y., Nair, M. S., et al. Striking antibody evasion manifested by the omicron variant of SARS-CoV-2. *Nature 602*, 7898 (2022), 676–681.

[49] Łuksza, M., Riaz, N., Makarov, V., Balachandran, V. P., Hellmann, M. D., Solovyov, A., Rizvi, N. A., Merghoub, T., Levine, A. J., Chan, T. A., et al. A neoantigen fitness model predicts tumour response to checkpoint blockade immunotherapy. *Nature 551*, 7681 (2017), 517–520.

[50] Maiers, M., Gragert, L., and Klitz, W. High-resolution HLA alleles and haplotypes in the united states population. *Human immunology 68*, 9 (2007), 779–788.

[51] Marculescu, R., Marculescu, D., and Pedram, M. Logic level power estimation considering spatiotemporal correlations. In *International Conference on Computer-Aided Design* (1994), pp. 294–299.

[52] Marzella, D. F., Parizi, F. M., Van Tilborg, D., Renaud, N., Sybrandi, D., Buzatu, R., Rademaker, D. T., AC't Hoen, P., and Xue, L. C. Pandora: a fast, anchor-restrained modelling protocol for peptide: Mhc complexes. *Frontiers in Immunology 13* (2022).

[53] McGranahan, N., Rosenthal, R., Hiley, C. T., Rowan, A. J., Watkins, T. B., Wilson, G. A., Birkbak, N. J., Veeriah, S., Van Loo, P., Herrero, J., et al. Allele-specific HLA loss and immune escape in lung cancer evolution. *Cell 171*, 6 (2017), 1259–1271.

[54] Mei, S., Li, F., Leier, A., Marquez-Lago, T. T., Giam, K., Croft, N. P., Akutsu, T., Smith, A. I., Li, J., Rossjohn, J., et al. A comprehensive review and performance evaluation of bioinformatics tools for HLA class I peptide-binding prediction. *Briefings in bioinformatics 21*, 4 (2020), 1119–1135.

[55] Miller, M., Sheehan, P., Edelstein, R., Tamanaha, C., Zhong, L., Bounnak, S., Whitman, L., and Colton, R. A DNA array sensor utilizing magnetic microbeads and magnetoelec-

tronic detection. *Journal of Magnetism and Magnetic Materials 225*, 1-2 (2001), 138–144.

[56] Mohammadi-Kambs, M., Hölz, K., Somoza, M. M., and Ott, A. Hamming distance as a concept in DNA molecular recognition. *ACS omega 2*, 4 (2017), 1302–1308.

[57] Monera, O. D., Sereda, T. J., Zhou, N. E., Kay, C. M., and Hodges, R. S. Relationship of sidechain hydrophobicity and α-helical propensity on the stability of the single-stranded amphipathic α-helix. *Journal of peptide science: an official publication of the European Peptide Society 1*, 5 (1995), 319–329.

[58] Moon, C. P., and Fleming, K. G. Side-chain hydrophobicity scale derived from transmembrane protein folding into lipid bilayers. *Proceedings of the National Academy of Sciences 108*, 25 (2011), 10174–10177.

[59] Najafi, M. H., Jenson, D., Lilja, D. J., and Riedel, M. D. Performing stochastic computation deterministically. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 27*, 12 (2019), 2925–2938.

[60] Najafi, M. H., and Lilja, D. J. High-speed stochastic circuits using synchronous analog pulses. In *ASP-DAC 2017, 22nd Asia and South Pacific Design Automation Conference* (2017).

[61] Naranbhai, V., Nathan, A., Kaseke, C., Berrios, C., Khatri, A., Choi, S., Getz, M. A., Tano-Menka, R., Ofoman, O., Gayton, A., et al. T cell reactivity to the SARS-CoV-2 omicron variant is preserved in most but not all individuals. *Cell 185*, 6 (2022), 1041–1051.

[62] Ndwandwe, D., and Wiysonge, C. S. Covid-19 vaccines. *Current opinion in immunology 71* (2021), 111–116.

[63] Neefjes, J., Jongsma, M. L., Paul, P., and Bakke, O. Towards a systems understanding of mhc class i and mhc class ii antigen presentation. *Nature reviews immunology 11*, 12 (2011), 823–836.

[64] Nielsen, M., Andreatta, M., Peters, B., and Buus, S. Immunoinformatics: predicting peptide–MHC binding. *Annual Review of Biomedical Data Science 3* (2020), 191–215.

[65] Okada, P., Phuygun, S., Thanadachakul, T., Parnmen, S., Wongboot, W., Waicharoen, S., Wacharapluesadee, S., Uttayamakul, S., Vachiraphan, A., Chittaganpitch, M., et al. Early transmission patterns of coronavirus disease 2019 (COVID-19) in travellers from Wuhan to Thailand, January 2020. *Eurosurveillance 25*, 8 (2020), 2000097.

[66] Organization, W. H., et al. COVID-19 weekly epidemiological update, edition 110, 21 september 2022.

[67] Pan, L., Wang, Z., Li, Y., Xu, F., Zhang, Q., and Zhang, C. Nicking enzyme-controlled toehold regulation for DNA logic circuits. *Nanoscale 9*, 46 (2017), 18223–18228.

[68] Parhi, M., Riedel, M. D., and Parhi, K. K. Effect of bit-level correlation in stochastic computing. In *2015 IEEE International Conference on Digital Signal Processing (DSP)* (2015), IEEE, pp. 463–467.

[69] Parker, K. P., and McCluskey, E. J. Probabilistic treatment of general combinational networks. *IEEE Transactions on Computers 24*, 6 (1975), 668–670.

[70] Paul, S., Grifoni, A., Peters, B., and Sette, A. Major histocompatibility complex binding, eluted ligands, and immunogenicity: benchmark testing and predictions. *Frontiers in immunology 10* (2020), 3151.

[71] Perumal, A. S., Wang, Z., Ippoliti, G., van Delft, F. C., Kari, L., and Nicolau, D. V. As good as it gets: a scaling com-

parison of dna computing, network biocomputing, and electronic computing approaches to an np-complete problem. *New Journal of Physics 23*, 12 (2021), 125001.

[72] Powers, D. M. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061* (2020).

[73] Qian, W., Li, X., Riedel, M. D., Bazargan, K., and Lilja, D. J. An architecture for fault-tolerant computation with stochastic logic. *IEEE Transactions on Computers 60*, 1 (2011), 93–105.

[74] Qian, W., and Riedel, M. D. The synthesis of robust polynomial arithmetic with stochastic logic. In *DAC'08* (2008), pp. 648–653.

[75] Qian, W., Riedel, M. D., and Rosenberg, I. Uniform approximation and Bernstein polynomials with coefficients in the unit interval. *European Journal of Combinatorics 32*, 3 (2011), 448–463.

[76] Qian, W., Riedel, M. D., Zhou, H., and Bruck, J. Transforming probabilities with combinational logic. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 30*, 9 (2011), 1279–1292.

[77] Radding, C. M., Beattie, K. L., Holloman, W. K., and Wiegand, R. C. Uptake of homologous single-stranded fragments by superhelical dna: Iv. branch migration. *Journal of molecular biology 116*, 4 (1977), 825–839.

[78] Reynisson, B., Alvarez, B., Paul, S., Peters, B., and Nielsen, M. NetMHCpan-4.1 and NetMHCIIpan-4.0: improved predictions of MHC antigen presentation by concurrent motif deconvolution and integration of MS MHC eluted ligand data. *Nucleic acids research 48*, W1 (2020), W449–W454.

[79] Richmond, T. J., and Davey, C. A. The structure of DNA in the nucleosome core. *Nature 423*, 6936 (2003), 145–150.

[80] Rigden, D. J., and Rigden, D. J. *From protein structure to function with bioinformatics.* Springer, 2009.

[81] Sahin, U., Muik, A., Derhovanessian, E., Vogler, I., Kranz, L. M., Vormehr, M., Baum, A., Pascal, K., Quandt, J., Maurus, D., et al. COVID-19 vaccine BNT162b1 elicits human antibody and TH1 T cell responses. *Nature 586*, 7830 (2020), 594–599.

[82] Salehi, S. A., Liu, X., Riedel, M., and Parhi, K. Computing mathematical functions using DNA via fractional coding. *Nature Scientific Reports 8*, 8312 (2018).

[83] Salehi, S. A., Riedel, M., and Parhi, K. Chemical reaction networks for computing polynomials. *ACS Synthetic Biology 6*, 1 (2017).

[84] Savir, J., Ditlow, G., and Bardell, P. H. Random pattern testability. *IEEE Transactions on Computers 33*, 1 (1984), 79–90.

[85] Schlecht, U., Mok, J., Dallett, C., and Berka, J. ConcatSeq: A method for increasing throughput of single molecule sequencing by concatenating short DNA fragments. *Scientific reports 7*, 1 (2017), 1–10.

[86] Shanbhag, N. R., Abdallah, R. A., Kumar, R., and Jones, D. L. Stochastic computation. In *Proceedings of the 47th Design Automation Conference* (2010), pp. 859–864.

[87] Shen, B., Zhang, W., Zhang, J., Zhou, J., Wang, J., Chen, L., Wang, L., Hodgkins, A., Iyer, V., Huang, X., et al. Efficient genome modification by CRISPR-cas9 nickase with minimal off-target effects. *Nature methods 11*, 4 (2014), 399–402.

[88] Shipman, S. L., Nivala, J., Macklis, J. D., and Church, G. M. Crispr–cas encoding of a digital movie into the genomes of a population of living bacteria. *Nature 547*, 7663 (2017), 345–349.

[89] Solanki, A., Chen, T., and Riedel, M. Cascadable stochastic logic for dna storage. In *2021 International Conference on Visual Communications and Image Processing (VCIP)* (2021), IEEE, pp. 1–5.

[90] Solanki, A., Riedel, M., Cornette, J., Udell, J., Koratkar, I., and Vasmatzis, G. The role of hydrophobicity in peptide-MHC binding. In *International Symposium on Mathematical and Computational Oncology* (2021), Springer, pp. 24–37.

[91] Soloveichik, D., Seelig, G., and Winfree, E. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences 107*, 12 (2010), 5393–5398.

[92] Srinivas, N., Ouldridge, T. E., Šulc, P., Schaeffer, J. M., Yurke, B., Louis, A. A., Doye, J. P., and Winfree, E. On the biophysics and kinetics of toehold-mediated DNA strand displacement. *Nucleic acids research 41*, 22 (2013), 10641–10658.

[93] Sun, L., and Åkerman, B. Characterization of self-assembled DNA concatemers from synthetic oligonucleotides. *Computational and structural biotechnology journal 11*, 18 (2014), 66–72.

[94] Tabatabaei, S. K., Wang, B., Athreya, N. B. M., Enghiad, B., Hernandez, A. G., Fields, C. J., Leburton, J.-P., Soloveichik, D., Zhao, H., and Milenkovic, O. DNA punch cards for storing data on native DNA sequences via enzymatic nicking. *Nature communications 11*, 1 (2020), 1–10.

[95] Tao, K., Tzou, P. L., Kosakovsky Pond, S. L., Ioannidis, J. P., and Shafer, R. W. Susceptibility of SARS-CoV-2 omicron variants to therapeutic monoclonal antibodies: systematic review and meta-analysis. *Microbiology Spectrum 10*, 4 (2022), e00926–22.

[96] Tarke, A., Sidney, J., Methot, N., Yu, E., Zhang, Y., Dan, J., Goodwin, B., Rubiro, P., Sutherland, A., Wang, E., et al.

Impact of sars-cov-2 variants on the total cd4 (+) and cd8 (+) t cell reactivity in infected or vaccinated individuals. cell rep med. 2021; 2 (7): 100355, 2021.

[97] Thachuk, C., Winfree, E., and Soloveichik, D. Leakless DNA strand displacement systems. In *International Workshop on DNA-Based Computers* (2015), Springer, pp. 133–153.

[98] Tsueng, G., Mullen, J., Alkuzweny, M., Cano, M., Benjamin, H. R., Emily, O., Curators, L., Alaa, A., Zhou, X., Qian, Z., et al. Outbreak. info research library: A standardized, searchable platform to discover and explore COVID-19 resources and data. *BioRxiv* (2022).

[99] Vasmatzis, G., Zhang, C., Cornette, J. L., and DeLisi, C. Computational determination of side chain specificity for pockets in class I MHC molecules. *Molecular immunology 33*, 16 (1996), 1231–1239.

[100] Venkatesan, R., Venkataramani, S., Fong, X., Roy, K., and Raghunathan, A. Spintastic: Spin-based stochastic logic for energy-efficient computing. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2015), IEEE, pp. 1575–1578.

[101] Von Neumann, J. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies 34* (1956), 43–98.

[102] Walls, A. C., Park, Y.-J., Tortorici, M. A., Wall, A., McGuire, A. T., and Veesler, D. Structure, function, and antigenicity of the SARS-CoV-2 spike glycoprotein. *Cell 181*, 2 (2020), 281–292.

[103] Wang, B., Chalk, C., and Soloveichik, D. SIMD——DNA: Single instruction, multiple data computation with DNA strand displacement cascades. In *DNA Computing and Molecular Programming*

(Cham, 2019), C. Thachuk and Y. Liu, Eds., Springer International Publishing, pp. 219–235.

[104] Wang, Z., Gerstein, M., and Snyder, M. Rna-seq: a revolutionary tool for transcriptomics. *Nature reviews genetics 10*, 1 (2009), 57–63.

[105] Watson, J. D., and Crick, F. H. The structure of DNA. In *Cold Spring Harbor symposia on quantitative biology* (1953), vol. 18, Cold Spring Harbor Laboratory Press, pp. 123–131.

[106] Wolfram, S. *Mathematica: a system for doing mathematics by computer.* Addison Wesley Longman Publishing Co., Inc., 1991.

[107] Xiao, G., Lu, M., Qin, L., and Lai, X. New field of cryptography: Dna cryptography. *Chinese Science Bulletin 51* (2006), 1413–1420.

[108] Yurke, B., Turberfield, A. J., Mills, A. P., Simmel, F. C., and Neumann, J. L. A DNA-fuelled molecular machine made of DNA. *Nature 406*, 6796 (2000), 605–608.

[109] Zhang, C., Vasmatzis, G., Cornette, J. L., and DeLisi, C. Determination of atomic desolvation energies from the structures of crystallized proteins. *Journal of molecular biology 267*, 3 (1997), 707–726.

[110] Zhang, D. Y., and Winfree, E. Control of DNA strand displacement kinetics using toehold exchange. *Journal of the American Chemical Society 131*, 47 (2009), 17303–17314.

[111] Zhou, P., Yang, X.-L., Wang, X.-G., Hu, B., Zhang, L., Zhang, W., Si, H.-R., Zhu, Y., Li, B., Huang, C.-L., et al. A pneumonia outbreak associated with a new coronavirus of probable bat origin. *nature 579*, 7798 (2020), 270–273.

[112] Zuckermann, M., Hlevnjak, M., Yazdanparast, H., Zapatka, M., Jones, D. T., Lichter, P., and Gronych, J. A novel cloning

strategy for one-step assembly of multiplex crispr vectors. *Scientific reports 8*, 1 (2018), 1–8.

Chapter 10

Appendix

10.1 Supplementary Material for Chapter 3

10.1.1 Instructions for Converting to Another Scheme

Instruction 1 identifies and distinguishes the two different bits. In instruction 1, strand (S_1 1 2 3) is issued. In bit 0, the strand will displace the short strand over domains 2 and 3 but does not edit bit 1 since domain 1 is the only open domain for binding. In instruction 2, all domains in bit 1 are replaced by a single strand covering all domains with identifier S_a. Then in instruction 3, the strand S_1 is detached, so domains 1, 2, and 3 on bit 0 are exposed. In Instruction 4, all domains in bit 0 are replaced by a single strand covering all the domains with the identifier S_b. Then any encoding scheme with 7 domains in 1 cell could be written to the bits by first detaching strand S_a and writing the encoding for bit 1, then detaching strand S_b and writing the encoding for bit 0.

10.1.2 Detailed Implementation of Each Step for Parallel Sorting

Here is an instruction set for parallel binary bubble sort with the previously defined encoding scheme. It is implemented with 12 individual operations. Details of the design are shown in Figure 10.2.

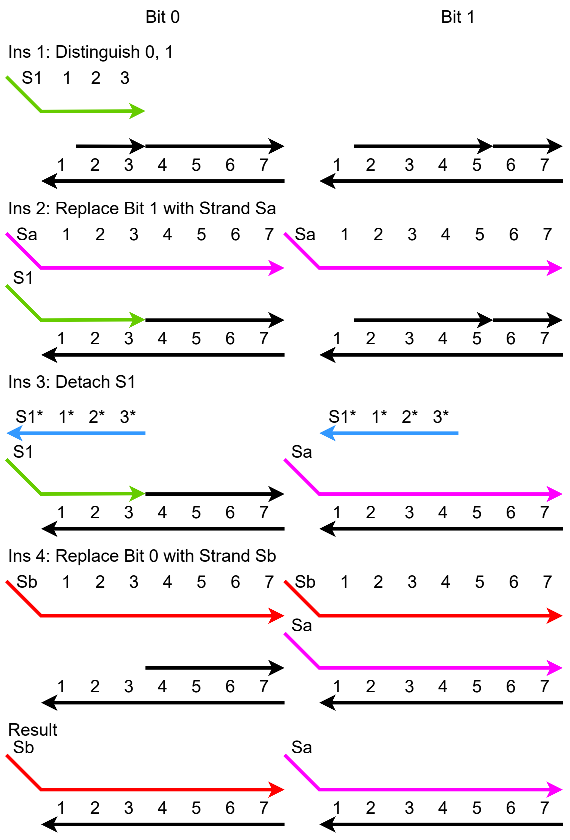


Figure 10.1: Current coding scheme can be converted to another coding scheme

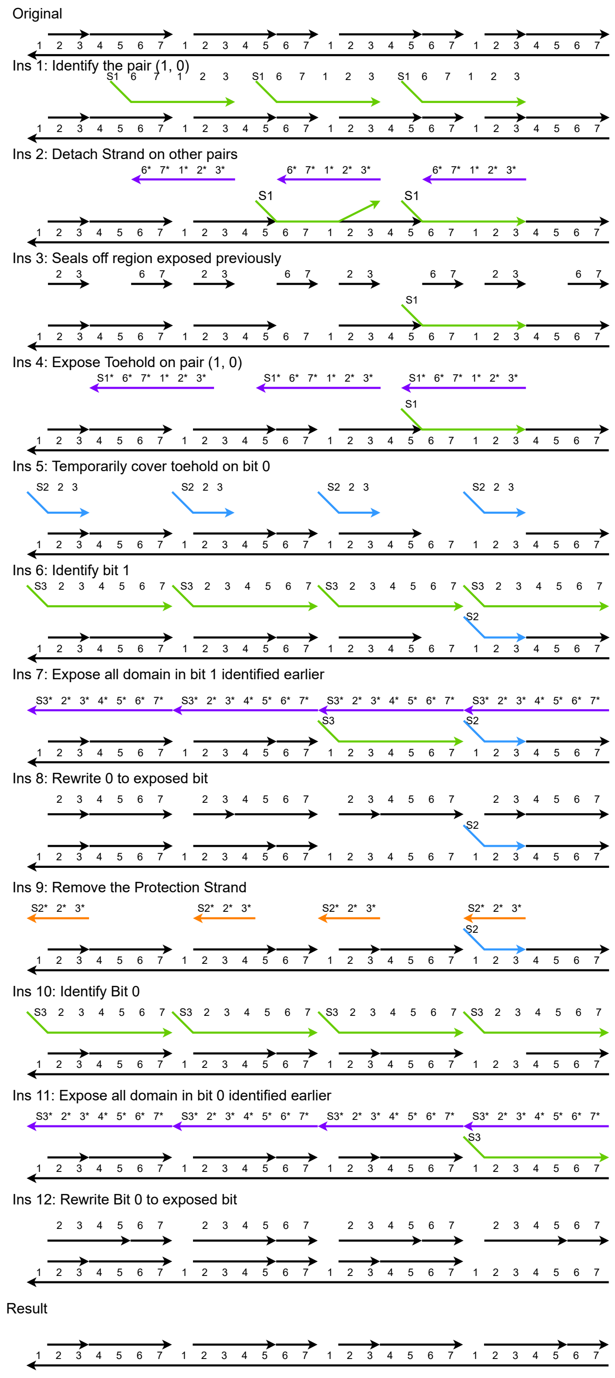

Original
Ins 1: Identify the pair (1, 0)
Ins 2: Detach Strand on other pairs
Ins 3: Seals off region exposed previously
Ins 4: Expose Toehold on pair (1, 0)
Ins 5: Temporarily cover toehold on bit 0
Ins 6: Identify bit 1
Ins 7: Expose all domain in bit 1 identified earlier
Ins 8: Rewrite 0 to exposed bit
Ins 9: Remove the Protection Strand
Ins 10: Identify Bit 0
Ins 11: Expose all domain in bit 0 identified earlier
Ins 12: Rewrite Bit 0 to exposed bit
Result

Figure 10.2: Instructions for Parallel Sorting

The 12 instructions fall into 2 stages. The first stage is “identifying.” During instructions 1-4, all the pairs (0, 1) are identified, and in both bit 0 and 1, a toehold is exposed for writing new data. More specifically, Instructions 1 and 2 identify the combination of $(1, 0)$. In instruction 1, (S_1 6 7 1 2 3) is issued to each pair of bits. In pair $(0, 0)$, S_1 and domains 6, 7 are exposed. In pair $(0, 1)$, since the only open domain is 1, it will not form a strong enough bond. In pair $(1, 0)$, only S_1 is exposed. In pair $(1, 1)$, S_1 and domains 2, 3 are exposed. In instruction 2, strand (6* 7* 1* 2* 3*) is issued to each pair of bits. Since pair $(1, 0)$ is the only pair that does not have exposure 5 or 2, this strand will detach strand S_1 in each pair except pair $(1, 0)$. After Instruction 2, the toehold between a bit value of 1 and a bit value of 0 in the pair $(1, 0)$ is replaced by a strand with an identifier of S_1. Instruction 3 seals off the domain exposed in the other pairs during Instruction 1 and 2 so that it will not be edited later. In instruction 4, the strand with identifier S_1 is detached, exposing domains 6 and 7 in the left cell containing bit 1, or domains 2 and 3, in the right cell containing bit 0. After this instruction, toeholds are exposed only in the 1s and 0s in pair $(1, 0)$. Other bits are not affected.

The second stage is flipping the bits in the pair $(1, 0)$. In instruction 5, in the case of a bit value of 0, domains 2 and 3 are temporarily covered by a strand with identifier S_2 so that the writing process will not interfere with the identified 0s at this moment. In instruction 6, a bit value of 1 is replaced by a strand with identifier S_3 via the open toehold at domains 6 and 7. The strand is then detached in instruction 8, exposing all the domains of that bit. Then, the bit value of 0 is written to the location of a bit value of 1 in instruction 8. In instruction 9, the temporary cover for a bit 0 is lifted. Then, in instructions 10 through 12, a bit 1 is written to the location of a bit value of 0 using the same scheme as instructions 6 through 8. Throughout the process, only bits identified in the first stage with toeholds exposed are affected.

10.1.3 Detailed Implementation of Each Step for Parallel Exclusive OR

The instructions are shown below, alongside an example of the Exclusive OR algorithm for sequence 11101 to 00000 in two iterations.

In each XOR iteration, the f(1,1) = (0,0) rewriting must be performed on non-overlapping pairs of bits. In the first iteration, the pairing is as follows: cell 0 with cell 1, cell 2 with cell 3, and so on. This means that all instruction strands only operate on these pairs. For this algorithm specifically, this can be achieved by using different sequences for the even versus the odd cells on the strand. In instruction 1, the strand (S_1 6 7 1 2 3) is issued to identify (1,0) pairs. In instruction 2, strand (6* 7* 1* 2* 3*) is issued to detach any S_1 strands with exposed domains of 6 and 7, or 2 and 3. In instruction 3, the strands (S_2 6 7 1) and (S_3 1 2 3) are issued to identify (1,1) and (0,0) pairs respectively. Finally, (0,1) pairs are identified with strand (S_4 4 5 6 7 1) for instruction 4. Now that all 1 domain toeholds are covered, strand (S_2* 6* 7* 1*) is issued in instruction 5 to detach all S_2 and expose (1,1) pairs. In instruction 6, strand (S_5 2 3 4 5 6 7 1 2 3 4 5 6 7) is issued to cover both cells in (1,1) pairs. Both S_5 and S_4 are now detached using strands (S_5* 2* 3* 4* 5* 6* 7* 1* 2* 3* 4* 5* 6* 7*) and (S_4* 4* 5* 6* 7*) in instruction 7. Then in instruction 8, all exposed cells are written to 0 using strands (2 3) and (4 5 6 7). In instruction 9, all S_1 and S_3 are detached using (S_1* 6* 7* 1* 2* 3*) and (S_3* 1* 2* 3*). By covering all exposed domains using strands (2 3) and (6 7) in instruction 10, all (1,1) pairs identified in the register are rewritten to (0,0) pairs. At this point, instructions 1-11 of the parallel sorting in Section 10.1.2 are implemented to write all (1,0) pairs to (0,1). For these sorting steps, the cell pairing can be overlapping. The result of this whole iteration of the XOR algorithm is a DNA sequence that has the same bit parity as the input, but is more ordered (i.e., closer to being sorted), and contains the same or fewer 1's. In Figure 10.3, the first iteration is carried out with non-overlapping pairs for cells 0 with 1, and so on. However, in Figure 10.4, depicting a second iteration of the XOR algorithm, the pairing is: cell 1 with cell 2, cell 3 with cell 4, and so on. In

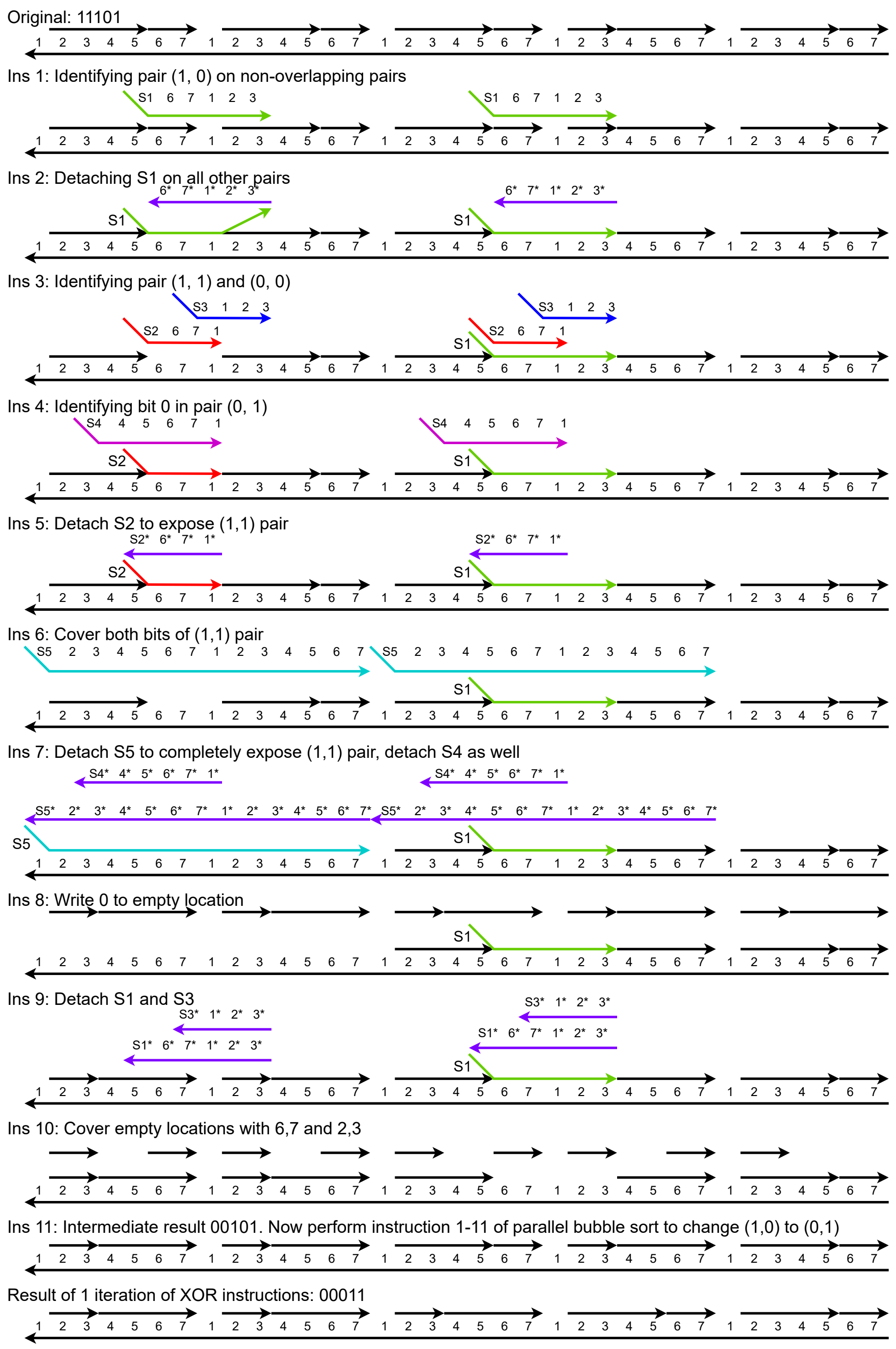

Original: 11101
Ins 1: Identifying pair (1, 0) on non-overlapping pairs
Ins 2: Detaching S1 on all other pairs
Ins 3: Identifying pair (1, 1) and (0, 0)
Ins 4: Identifying bit 0 in pair (0, 1)
Ins 5: Detach S2 to expose (1,1) pair
Ins 6: Cover both bits of (1,1) pair
Ins 7: Detach S5 to completely expose (1,1) pair, detach S4 as well
Ins 8: Write 0 to empty location
Ins 9: Detach S1 and S3
Ins 10: Cover empty locations with 6,7 and 2,3
Ins 11: Intermediate result 00101. Now perform instruction 1-11 of parallel bubble sort to change (1,0) to (0,1)
Result of 1 iteration of XOR instructions: 00011
S1
S2
S3
S4
S5

Figure 10.3: Instructions for the Exclusive OR. The first iteration converts 11101 to 00011.

the third iteration, the pairing can return to the original pairing in the first iteration. For a n bit register, after n iterations of the XOR algorithm, the last cell contains the output of the n bit XOR.

10.1.4 Detailed Implementation of Each Step for Parallel Left Shift cell

The instructions are shown as followed, with an example of shifting 11001 to 10011.

The first three instructions are exactly the same as those for identifying bit pairs in Section 3.2.1. In instruction 1, the strand (S_1 6 7 1 2 3), which identifies the different patterns of two bits, is issued to each pair of bits. In instruction 2, strand (6* 7* 1* 2* 3*) is issued, detaching strands with open domains 6 and 7, or 2 and 3. After this instruction, strands with identifier S_1 only remain at pair $(1, 0)$. In instruction 3, we issue two species of strands at the same time: (S_2 6 7 1) and (S_3 1 2 3). (S_2 6 7 1) will bind with pair $(1, 1)$ and (S_3 1 2 3) will bind with pair $(0, 0)$. S_2 will not form a stable binding with pair $(0, 0)$ or $(0, 1)$ because the binding area is only one domain. Same goes with S_3 and pair $(1, 1)$ or $(0, 1)$. After this instruction, only domain 1 between pair $(0, 1)$ is still exposed. In instruction 4, strand (S_4 4 5 6 7 1) is issued. Through the open domain 1 between pair $(0, 1)$, the strand in bit 0 is replaced by S_4. After this step, the first bit in pair $(1, 0)$ is identified with the strand S_1, and the first bit in pair (0, 1) is replaced with the strand S_4.

Instructions 5 to 9 rewrite the first bit in pair $(1, 0)$ to 0. In instruction 5, the strand S_1 is detached, exposing domains 6, 7, 1, 2 and 3. The exposed domains 2 and 3 are sealed off in instruction 6 to not interfere with subsequent instructions. In instruction 7, strand (S_5 2 3 4 5 6 7) is issued through the open toehold on domains 6 and 7 in the bit 1 in pair $(1, 0)$, and displaces the strand in that bit. Since domains 2 and 3 are sealed off, bit 0 will not be modified in this instruction. In instruction 8, strand S_5 is detached, leaving the domains in the bit open. In instruction 9, strands (2 3) and (4 5 6 7), which represent 0, are written to the bit containing open domains.

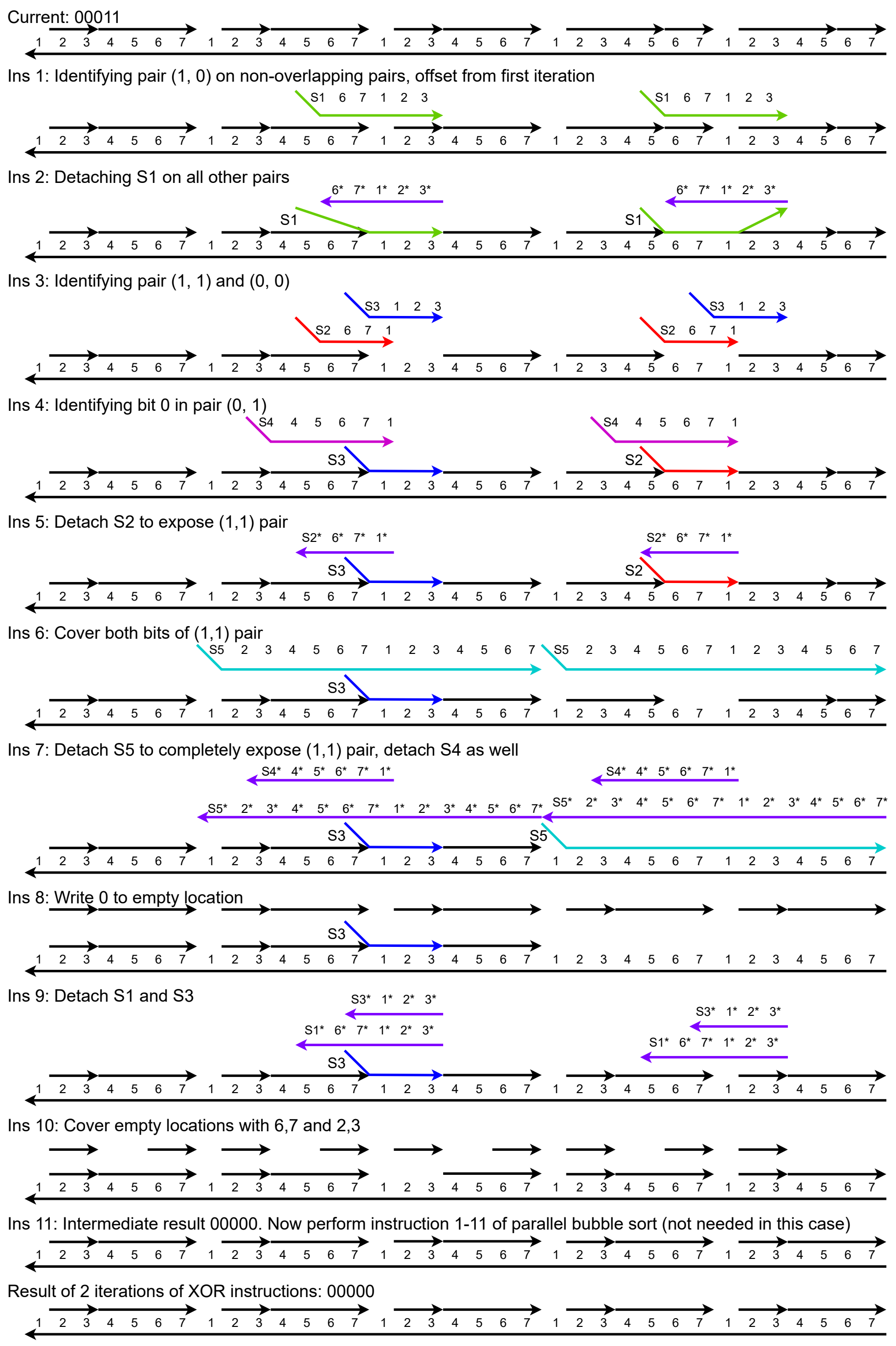

Current: 00011
Ins 1: Identifying pair (1, 0) on non-overlapping pairs, offset from first iteration
Ins 2: Detaching S1 on all other pairs
Ins 3: Identifying pair (1, 1) and (0, 0)
Ins 4: Identifying bit 0 in pair (0, 1)
Ins 5: Detach S2 to expose (1,1) pair
Ins 6: Cover both bits of (1,1) pair
Ins 7: Detach S5 to completely expose (1,1) pair, detach S4 as well
Ins 8: Write 0 to empty location
Ins 9: Detach S1 and S3
Ins 10: Cover empty locations with 6,7 and 2,3
Ins 11: Intermediate result 00000. Now perform instruction 1-11 of parallel bubble sort (not needed in this case)
Result of 2 iterations of XOR instructions: 00000

Figure 10.4: Instructions for the Exclusive OR. The second iteration converts 00011 to 00000.

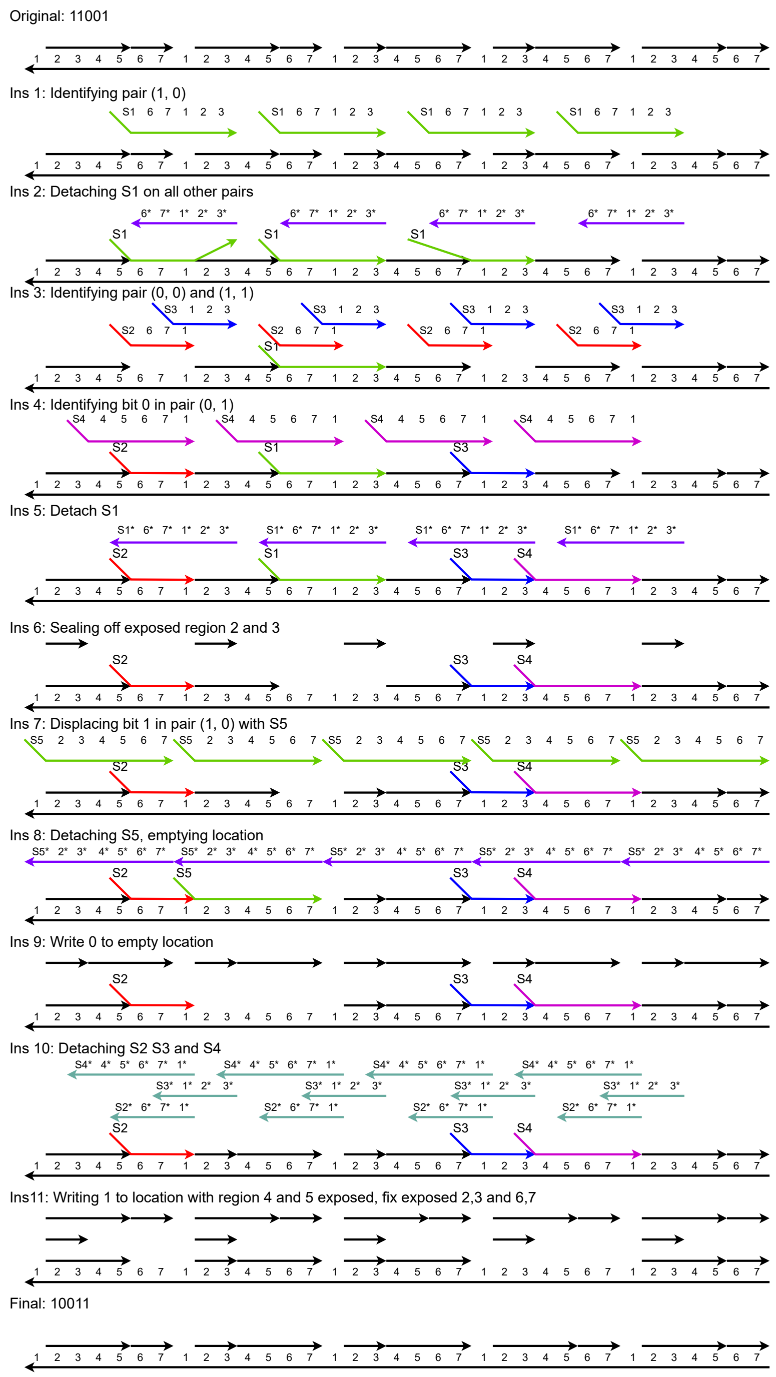

Original: 11001
Ins 1: Identifying pair (1, 0)
Ins 2: Detaching S1 on all other pairs
Ins 3: Identifying pair (0, 0) and (1, 1)
Ins 4: Identifying bit 0 in pair (0, 1)
Ins 5: Detach S1
Ins 6: Sealing off exposed region 2 and 3
Ins 7: Displacing bit 1 in pair (1, 0) with S5
Ins 8: Detaching S5, emptying location
Ins 9: Write 0 to empty location
Ins 10: Detaching S2 S3 and S4
Ins11: Writing 1 to location with region 4 and 5 exposed, fix exposed 2,3 and 6,7
Final: 10011
S1
S2
S3
S4
S5
1 2 3 4 5 6 7

Figure 10.5: Instructions for the Left Shift cell

In the final two instructions, we write 1 to the first bit in pair $(0, 1)$. In instruction 10, 3 strands are issued to each pair of bits: (S_2* 6* 7* 1*), (S_3* 1* 2* 3*) and (S_4* 4* 5* 6* 7* 1*). S_2, S_3 and S_4 are detached through these strands. Since S_4 covers the bit 0 in pair (0, 1), after this step, domains 3 and 4 are exposed in these bits, ready to be written to 1. In the final step, strands (2 3), (2 3 4 5), and (6 7) are issued to each cell. Strands (2 3) and (6 7) will fix the exposed domains from strand S_2 or S_3, and strand (2 3 4 5) will write bit 1 to the bit with domain 3 and 4 exposed. Details of the design are shown in Figure 10.5.

For all the pairs of $(0, 0)$ and $(1, 1)$, the first bit in those pairs will not be modified since the toehold 1 will be covered with S_2 or S_3 in the process.

10.1.5 Detailed Implementation of the Second Level in Parallel Search

Here the *second* level of the parallel search operation is discussed. The first level of the search operation uses the instructions that were described in Section 3.2.1, except now only strands to non-overlapping bit pairs are issued. Identifiers $A_0 = 00, A_1 = 01, A_2 = 10, A_3 = 11$ are used to represent symbols in this level. For instance, to search for the target string 1011, the symbols A_2 in odd symbols and A_3 in even symbols are searched. The cases of A_2 in even symbols and A_3 in odd symbols are covered by searching with an offset.

In the first instruction of the second level, the A_2 is uncovered in the odd symbols, creating an open region. In instruction 2, a long strand is used to cover the entire right half of the symbol, from the start of identifier A_2 to the rightmost cell. This strand is pulled out in instruction 3. In instruction 4, an identifier A_2' is used to cover domains 5, 6, 7 in the rightmost cell while covering all other domains.

Instructions 5 to 8 are essentially the same as instructions 1 to 4, but with two significant differences. Firstly, since A_3 is the second symbol in the current level of query, only even-numbered symbols (2, 4, 6, etc.) are searched. Secondly, instead of rewriting the right half of the symbol, the

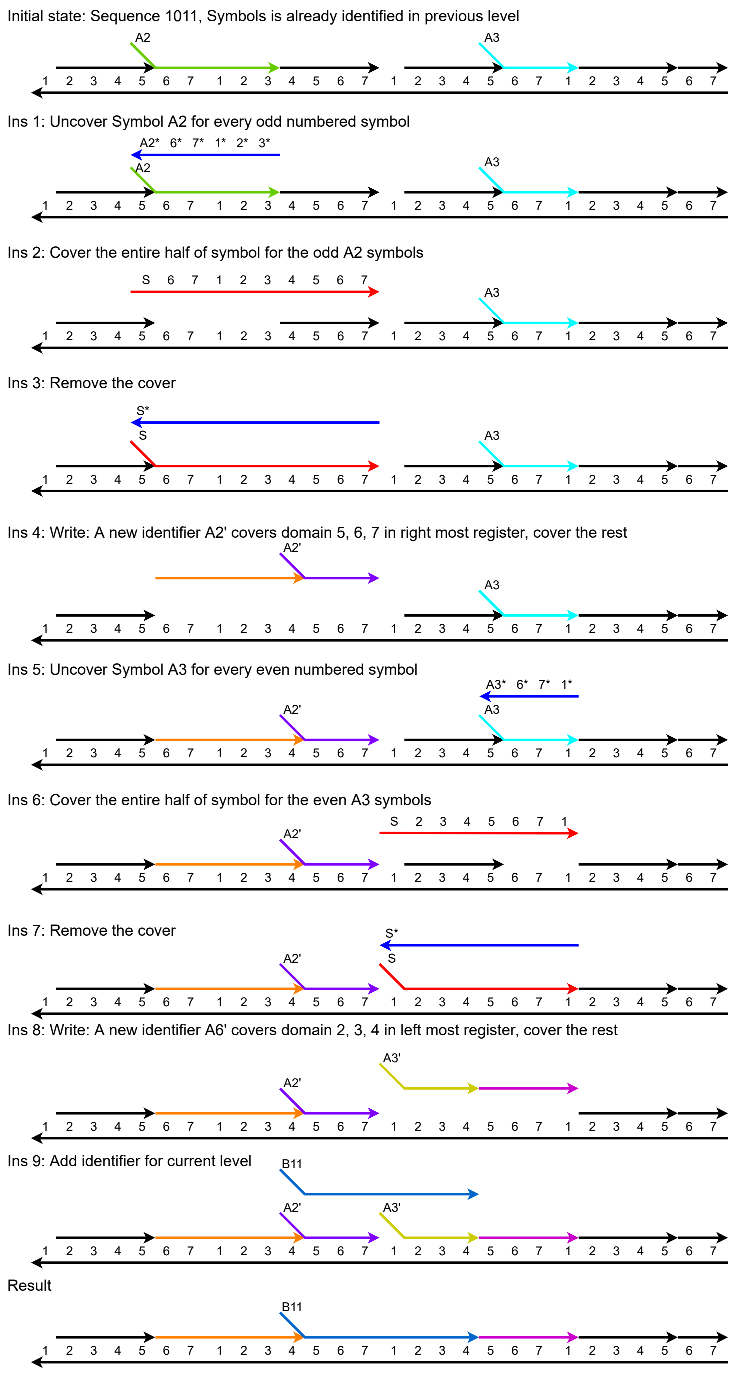

Initial state: Sequence 1011, Symbols is already identified in previous level
A2
A3
Ins 1: Uncover Symbol A2 for every odd numbered symbol
A2* 6* 7* 1* 2* 3*
Ins 2: Cover the entire half of symbol for the odd A2 symbols
S 6 7 1 2 3 4 5 6 7
Ins 3: Remove the cover
S*
S
Ins 4: Write: A new identifier A2' covers domain 5, 6, 7 in right most register, cover the rest
A2'
Ins 5: Uncover Symbol A3 for every even numbered symbol
A3* 6* 7* 1*
Ins 6: Cover the entire half of symbol for the even A3 symbols
S 2 3 4 5 6 7 1
Ins 7: Remove the cover
Ins 8: Write: A new identifier A6' covers domain 2, 3, 4 in left most register, cover the rest
A3'
Ins 9: Add identifier for current level
B11
Result
1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7

Figure 10.6: Instructions for a search operation of target sequence 1011

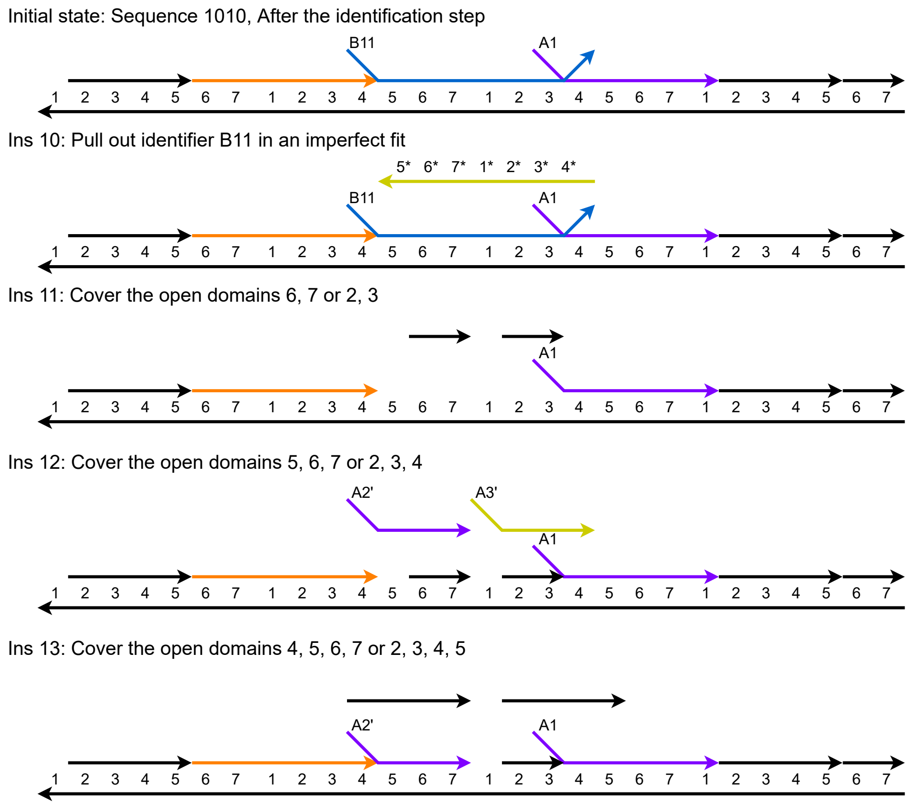

Initial state: Sequence 1010, After the identification step
B11
A1
1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7
Ins 10: Pull out identifier B11 in an imperfect fit
5* 6* 7* 1* 2* 3* 4*
B11
A1
Ins 11: Cover the open domains 6, 7 or 2, 3
A1
Ins 12: Cover the open domains 5, 6, 7 or 2, 3, 4
A2'
A3'
A1
Ins 13: Cover the open domains 4, 5, 6, 7 or 2, 3, 4, 5
A2'
A1

Figure 10.7: Instructions for the cleanup process for a failed searching. These instructions won't affect the result of a successful search.

left half is written. The new identifier A'_3 is made to cover domains 2, 3, 4 in the left-most cell. In instruction 9, identifier (B_{11} 5 6 7 1 2 3 4) is used to recognize the two consecutive symbols A_2 and A_3. Since, in the regular encoding, no strand starts from domain 5 or ends at domain 4, it will only form a perfect binding with a matched result.

After the identifier B_{11} binds, imperfect bindings also need to be cleaned up in case of a mismatch. Figure 10.6 shows the instructions for the cleanup process. In instruction 10, the complementary strand (5* 6* 7* 1* 2* 3* 4*) is used to pull out the imperfect bond identifier B_{11}. Then strands covering the exposed domain are issued. First strands covering fewer domains are issued, followed by strands covering more domains in the next instruction. This results in a perfect fit; the strands will not be pulled out in potentially unrelated rewriting processes.

10.1.6 Example of Parallel Bubble Sort on an arbitrary bit-string

Consider the 12-bit long string S = 110010010110. In each iteration of bubble sort, we first identify all $(1, 0)$ pairs (shown in red) and then rewrite them to $(0, 1)$ (shown in blue). For this string, the numerous iterations of the sorting algorithm are:

$$110010010110 \rightarrow 101001001101$$
$$101001001101 \rightarrow 010100101011$$
$$010100101011 \rightarrow 001010010111$$
$$001010010111 \rightarrow 000101001111$$
$$000101001111 \rightarrow 000010101111$$
$$000010101111 \rightarrow 000001011111$$
$$000001011111 \rightarrow 000000111111.$$

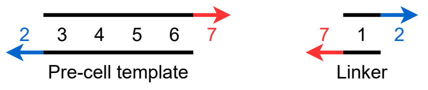
After 7 iterations, the final sorted string is 000000111111.

10.1.7 Gibson Assembly of a 2 bit register

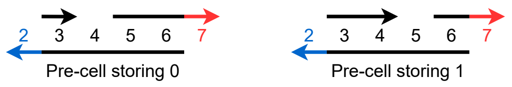
Gibson Assembly of DNA molecules is achieved through the use of “sticky ends” – single stranded sequences at the ends of these molecules that allow them to concatenate. To create registers storing unique bit sequences, we use two different molecules to start off: pre-cell molecules (domains 2 to 7, with sticky ends on domains 2 and 7), and linker molecules (domains 7 1 2, with sticky ends on domains 7 and 2). To store a bit value of 0 in a pre-cell, a toehold on domain 4 is created. To store a bit value of 1, a toehold on domain 5 is created. This is shown in Figure 10.8b.

Before concatenating two different pre-cells, their particular sticky ends must be “sealed” – those ends are no longer single stranded and cannot link together anymore. Sealing a particular sticky end can easily be done by adding a single strand of DNA that binds to that sticky end. For example, by sealing the sticky end on domain 2 of a pre-cell, that pre-cell can no longer concatenate with itself when the linker molecule is mixed. In Figure 10.8c, pre-cell A only has a sticky end on domain 7, and pre-cell B only has a sticky end on domain 7. When these pre-cells are mixed together with the linker molecule, they will bind to each other in the order A to B. This creates a pre-register of those two pre-cells. The starting end of the pre-register has a domain 1 concatenated through a “cap” molecule (domains 1 and 2, with a sticky end at domain 2) as shown in Figures 10.8e and 10.8f. After this stage, the pre-register can be treated with DNA ligase to seal all nicks. The resulting DNA strand contains the cells A and B which contain toeholds at domain 4 and 5 respectively. All 1 domains across all cells in this strand can be exposed into to toehold domains through nicking and gentle denaturing. Finally, this DNA molecule (which encodes 01 based on the pre-cell encoding scheme) can be converted to the bit encoding scheme used in this study (Figure 3.2) through the procedure described in Sections 3.2.2 and 3.7.3. This entire procedure yields a 2 bit register storing the bits 0 and 1 in that order.

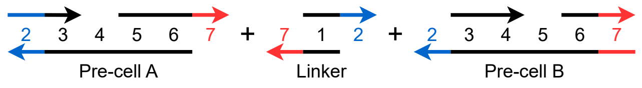
This approach can be used to construct registers of any arbitrary number of bits despite all cells having the same sequence. This is because pre-



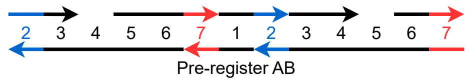
(a) The two main types of molecules used for Gibson Assembly



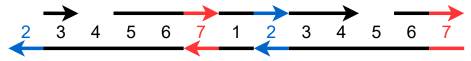
(b) How to store bit values 0 or 1 through toeholds 4 or 5 respectively.



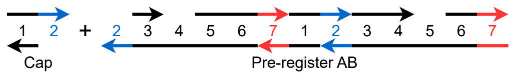
(c) Pre-cell A stores 0, Pre-cell B stores 1. They are mixed together with linker molecules to concatenate them. The blunt ends (domain 2 on A, domain 7 on B) prevent linking of two same pre-cells.



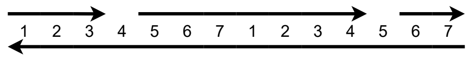
(d) The resulting pre-register AB.



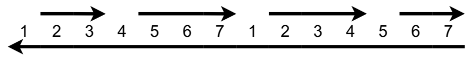
(e) Creating a sticky end on the first domain 2 of the pre-register.



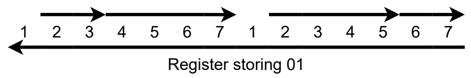
(f) Using a cap molecule to add a domain 1 at the start of the pre-register



(g) The resulting pre-register after ligation



(h) Toeholds are created on all 1 domains.



(i) The pre-cell encoding is changed to the encoding proposed in Figure 3.2.

Figure 10.8: Using Gibson Assembly to construct a register storing 01 from cells with the same sequence.

registers can also be concatenated in the same manner as pre-cells as shown in Figure 10.8c. For this, the sealed ends of a pre-register must be unsealed (through the use of an exonuclease) to create sticky ends again.

10.2 Supplementary Material for Chapter 4

Examples of CRNs for polynomial approximations of nonlinear functions.

10.2.1 ArcTan Function

arctan(x) can be approximated as:

$$\arctan(x) \approx x - \frac{1}{3}x^3 = x(1 - \frac{1}{3}x^2).$$

After assigning the stochastic variables $x_1 = x, x_2 = \frac{1}{3}, x_3 = x_4 = x$, the stochastic logic function is:

$$\text{AND}(x_1, \text{NAND}(x_2, \text{AND}(x_3, x_4))).$$

According to the truth table, the corresponding CRN is:

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} \rightarrow \mathbf{Y}_0$$

10.2.2 Exponential Function

$\exp(-x)$ is approximated as:

$$\exp(-x) \approx 1 - x + \frac{1}{2}x^2 - \frac{1}{6}x^3 = 1 - x(1 - \frac{1}{2}x(1 - \frac{1}{3}x))$$

Assigning the stochastic variables $x_1 = x, x_2 = \frac{1}{2}, x_3 = x, x_4 = \frac{1}{3}, x_5 = x$ yields the stochastic logic function:

$$\mathrm{NAND}(x_1, \mathrm{NAND}(x_2, \mathrm{AND}(x_3, \mathrm{NAND}(x_4, x_5)))).$$

According to the truth table, the corresponding CRN is:

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_0$$

10.2.3 Bessel Function

Consider the example of the Bessel function of the first kind with parameter $\alpha = 1$. Its approximation is:

$$J_1(x) \approx \frac{1}{2}x - \frac{1}{16}x^3 = \frac{1}{2}x(1 - \frac{1}{8}x^2).$$

Assigning the stochastic variables $x_1 = \frac{1}{2}, x_2 = x, x_3 = \frac{1}{8}, x_4 = x_5 = x$ yields the stochastic logic function:

$$\text{AND}(x_1, \text{AND}(x_2, \text{NAND}(x_3, \text{AND}(x_4, x_5)))).$$

According to the truth table, the corresponding CRN is:

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \to \mathbf{Y}_0$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \rightarrow \mathbf{Y}_0$$

10.2.4 Sinc Function

The mathematical expression of the sinc function is:

$$\mathrm{sinc}(x) = \frac{\sin(x)}{x}.$$

Its approximation is:

$$\mathrm{sinc}(x) \approx 1 - \frac{1}{6}x^2 + \frac{1}{120}x^4 = 1 - \frac{1}{6}x^2(1 - \frac{1}{20}x^2)$$

Assigning the stochastic variables $x_1 = x_2 = x, x_3 = \frac{1}{6}, x_4 = \frac{1}{20}, x_5 = x_6 = x$ yields the stochastic logic function:

$$\mathrm{NAND}(\mathrm{AND}(x_1, x_2), \mathrm{AND}(x_3, \mathrm{NAND}(x_4, \mathrm{AND}(x_5, x_6)))).$$

According to the truth table, the corresponding CRN is:

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \rightarrow \mathbf{Y}_0$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \rightarrow \mathbf{Y}_1$$

Table 10.1: Confusion matrices for the training data analysis for NetMHC-4.0. The abbreviations used are: True Negatives (TN), False Positives (FP), False Negatives (FN), and True Positives (TP).

HLA	Peptide Case	TN	FP	FN	TP
A2	All	42621	770	3540	5728
	Hydrophobic only	8050	516	1094	3199
	Hydrophilic only	5213	3	71	11
	Balanced only	29316	250	2371	2508
B27	All	14573	118	1476	1255
	Hydrophobic only	2581	12	215	247
	Hydrophilic only	1946	24	162	92
	Balanced only	10028	82	1098	914
B8	All	15923	169	1257	2099
	Hydrophobic only	2864	35	265	353
	Hydrophilic only	2101	18	78	209
	Balanced only	10950	116	913	1534

10.3 Supplementary Material for Chapter 7

The changes in predictions observed in Figures 7.1 to 7.3 suggested that NetMHCpan-4.1 approximated the training scores better than NetMHC-4.0 did inFigures 7.1 to 7.3 (the yellow plot fitted the S-curve transition of the blue plot more tightly than the red plot did). To confirm this, the Pearson Correlations of the training data with NetMHC-4.0, and with NetMHCpan-4.1 were calculated. The correlation coefficients for NetMHC-4.0 and NetMHCpan-4.1 for A2 were 0.8492 and 0.8637 respectively. For B27, these coefficients were 0.8165 and 0.8844. For B8, these were 0.8492 and 0.863. It was likely that the stronger correlation for NetMHCpan-4.1 was a consequence of NetMHCpan-4.1 having "seen" EL peptides which NetMHC-4.0 would not have been trained on.

Table 10.2: Confusion matrices for the training data analysis for NetMHCpan-4.1. The abbreviations used are: True Negatives (TN), False Positives (FP), False Negatives (FN), and True Positives (TP).

HLA	Peptide Case	TN	FP	FN	TP
A2	All	41166	730	2630	8133
	Hydrophobic only	7467	316	1345	3731
	Hydrophilic only	5193	12	36	57
	Balanced only	28464	402	1247	4332
B27	All	14367	221	384	2450
	Hydrophobic only	2564	15	80	396
	Hydrophilic only	1905	45	51	223
	Balanced only	9880	161	253	1828
B8	All	15495	280	667	3006
	Hydrophobic only	2785	38	189	505
	Hydrophilic only	2058	32	53	263
	Balanced only	10645	210	424	2234

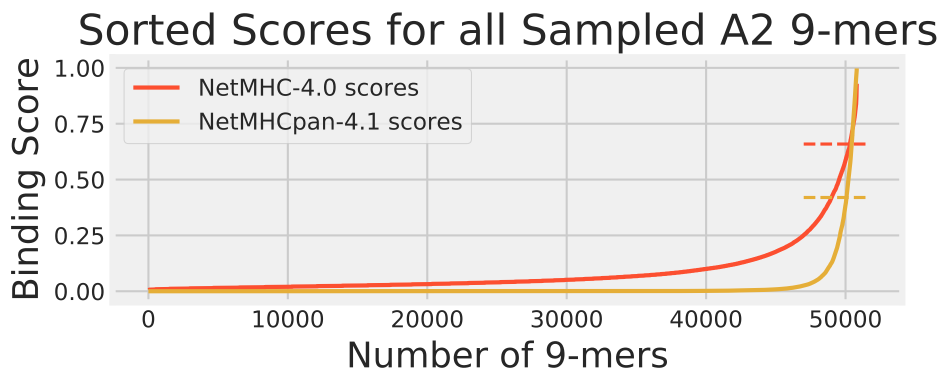


Figure 10.9: The cumulative distribution of NetMHC-4.0 predicted scores (red) and NetMHCpan-4.1 predicted scores (yellow) for peptides in the human proteome dataset for HLA A2. The strong binder thresholds for NetMHC-4.0 and NetMHCpan-4.1 are shown as dashed lines of the corresponding colors. These thresholds are the same as those in Figure 7.1. Each plot of scores was independently sorted. Consequently, the order of peptides is not conserved across the 2 plots.

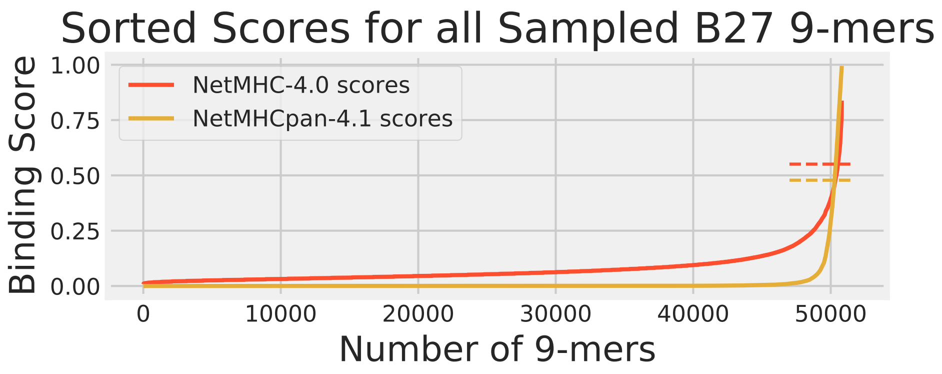


Figure 10.10: The cumulative distribution of NetMHC-4.0 predicted scores (red) and NetMHCpan-4.1 predicted scores (yellow) for peptides in the human proteome dataset for HLA B27. The strong binder thresholds for NetMHC-4.0 and NetMHCpan-4.1 are shown as dashed lines of the corresponding colors. These thresholds are the same as those in Figure 7.2. Each plot of scores was independently sorted. Consequently, the order of peptides is not conserved across the 2 plots.

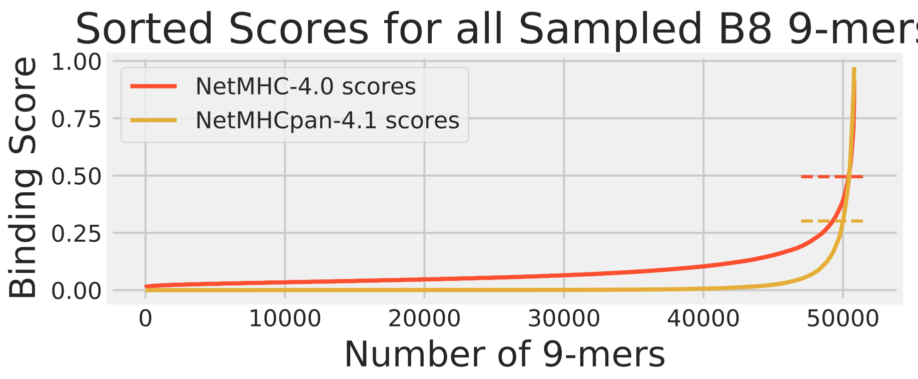


Figure 10.11: The cumulative distribution of NetMHC-4.0 predicted scores (red) and NetMHCpan-4.1 predicted scores (yellow) for peptides in the human proteome dataset for HLA B8. The strong binder thresholds for NetMHC-4.0 and NetMHCpan-4.1 are shown as dashed lines of the corresponding colors. These thresholds are the same as those in Figure 7.3. Each plot of scores was independently sorted. Consequently, the order of peptides is not conserved across the 2 plots.

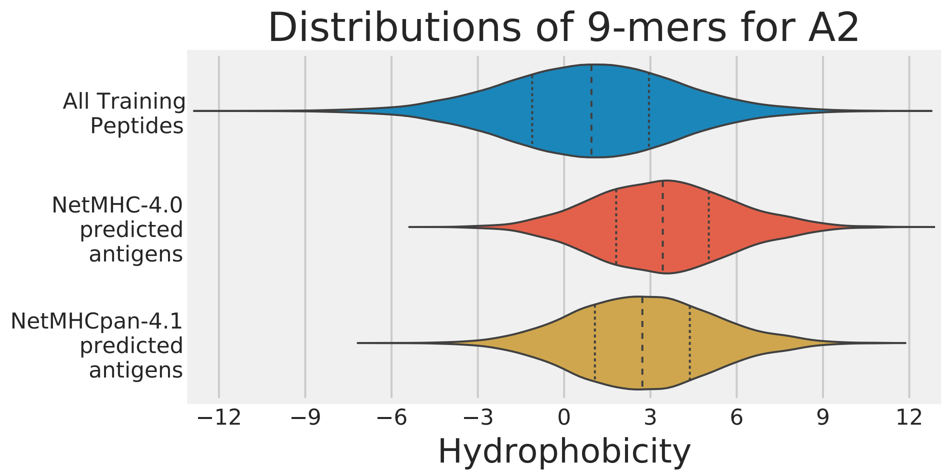


Figure 10.12: Violin plots of the hydrophobicity of the sets of strong binders predicted by NetMHC-4.0 and NetMHCpan-4.1 on the training dataset for A2. The x-axis represents the hydrophobicity of a 9-mer, and the y-axis represents the frequency. The distributions of all training peptides (blue), strong binders predicted by NetMHC-4.0 (red), and those predicted by NetMHCpan-4.1 (yellow) are shown. The mean and two quartiles are also depicted in each distribution.

Table 10.3: Sizes, Means, and Standard Deviations of the hydrophobicity of the sets of peptides reported in the Training Data Analysis. The abbreviations used here are: NetMHC-4.0 predicted Strong Binders (N-4.0 SB), and NetMHCpan-4.1 predicted Strong Binders (NP-4.1 SB).

HLA	Value of Set	All	N-4.0 SB	NP-4.1 SB
A2	Size	52659	6498	8863
	Mean	0.902	3.458	2.756
	Std. Dev.	3.063	2.365	2.426
B27	Size	17422	1373	2671
	Mean	0.364	0.725	0.450
	Std. Dev.	2.923	2.593	2.593
B8	Size	19448	2268	3286
	Mean	0.393	0.544	0.570
	Std. Dev.	2.946	2.652	2.566

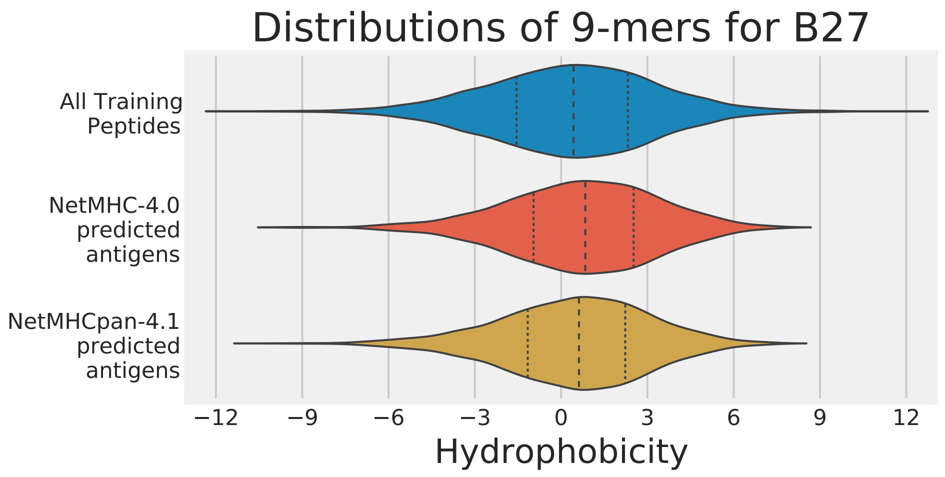


Figure 10.13: Violin plots of the hydrophobicity of the sets of strong binders predicted by NetMHC-4.0 and NetMHCpan-4.1 on the training dataset for B27. The x-axis represents the hydrophobicity of a 9-mer, and the y-axis represents the frequency. The distributions of all training peptides (blue), strong binders predicted by NetMHC-4.0 (red), and those predicted by NetMHCpan-4.1 (yellow) are shown. The mean and two quartiles are also depicted in each distribution.

Table 10.4: Sizes, Means, and Standard Deviations of the hydrophobicity of the sets of peptides reported in the Human Proteome Analysis. The abbreviations used here are: NetMHC-4.0 predicted Strong Binders (N-4.0 SB), and NetMHCpan-4.1 predicted Strong Binders (NP-4.1 SB).

HLA	Value of Set	All	N-4.0 SB	NP-4.1 SB
A2	Size	50804	486	730
	Mean	0.052	4.519	2.789
	Std. Dev.	3.212	2.517	2.647
B27	Size	50804	290	528
	Mean	0.052	-0.155	-0.775
	Std. Dev.	3.212	2.740	2.729
B8	Size	50804	411	801
	Mean	0.052	0.182	0.031
	Std. Dev.	3.212	3.332	3.011

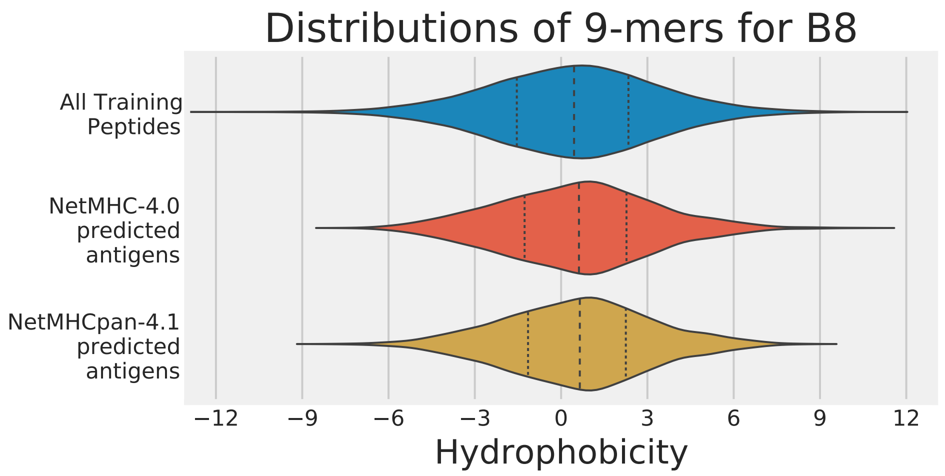


Figure 10.14: Violin plots of the hydrophobicity of the sets of strong binders predicted by NetMHC-4.0 and NetMHCpan-4.1 on the training dataset for B8. The x-axis represents the hydrophobicity of a 9-mer, and the y-axis represents the frequency. The distributions of all training peptides (blue), strong binders predicted by NetMHC-4.0 (red), and those predicted by NetMHCpan-4.1 (yellow) are shown. The mean and two quartiles are also depicted in each distribution.

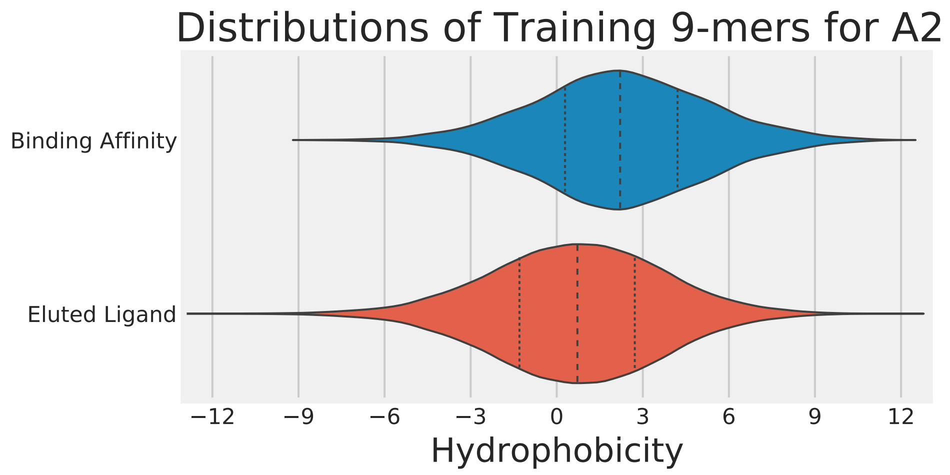


Figure 10.15: Violin plots of the hydrophobicity of the sets of all BA (blue) vs. EL (red) training data predicted by NetMHC-4.0 and NetMHCpan-4.1 on the human proteome dataset for A2. The x-axis represents the hydrophobicity of a 9-mer, and the y-axis represents the frequency. The mean and two quartiles are also depicted in each distribution.

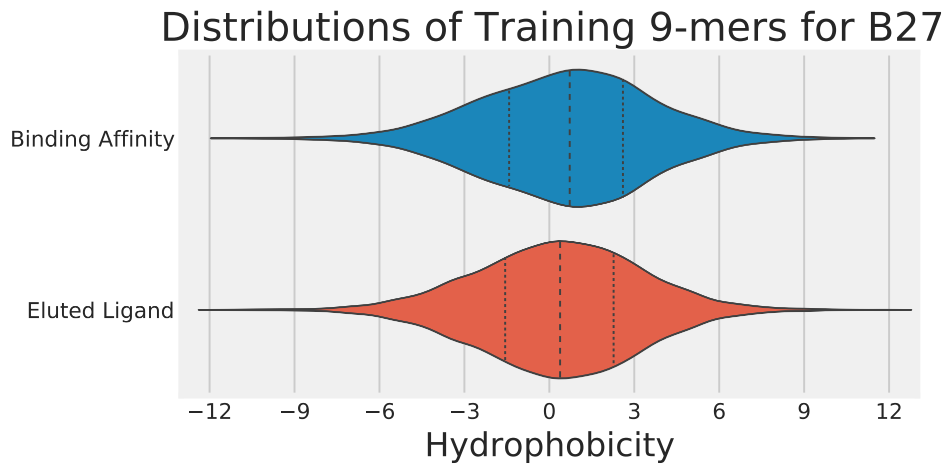


Figure 10.16: Violin plots of the hydrophobicity of the sets of all BA (blue) vs. EL (red) training data predicted by NetMHC-4.0 and NetMHCpan-4.1 on the human proteome dataset for B27. The x-axis represents the hydrophobicity of a 9-mer, and the y-axis represents the frequency. The mean and two quartiles are also depicted in each distribution.

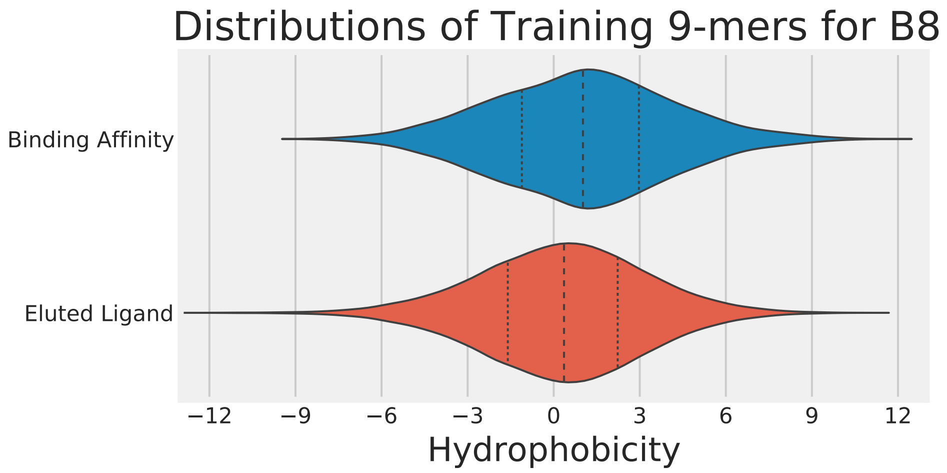
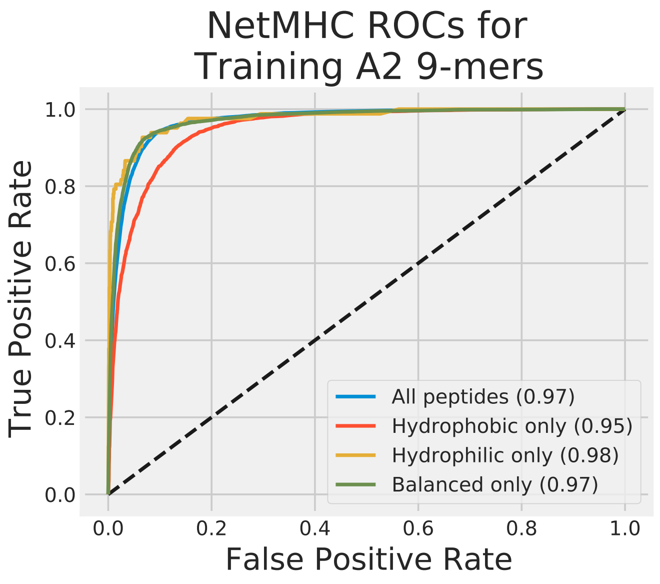


Figure 10.17: Violin plots of the hydrophobicity of the sets of all BA (blue) vs. EL (red) training data predicted by NetMHC-4.0 and NetMHCpan-4.1 on the human proteome dataset for B8. The x-axis represents the hydrophobicity of a 9-mer, and the y-axis represents the frequency. The mean and two quartiles are also depicted in each distribution.



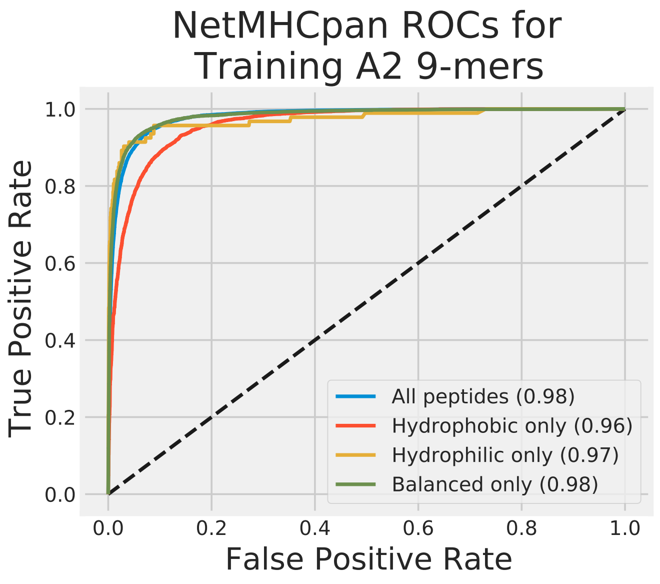
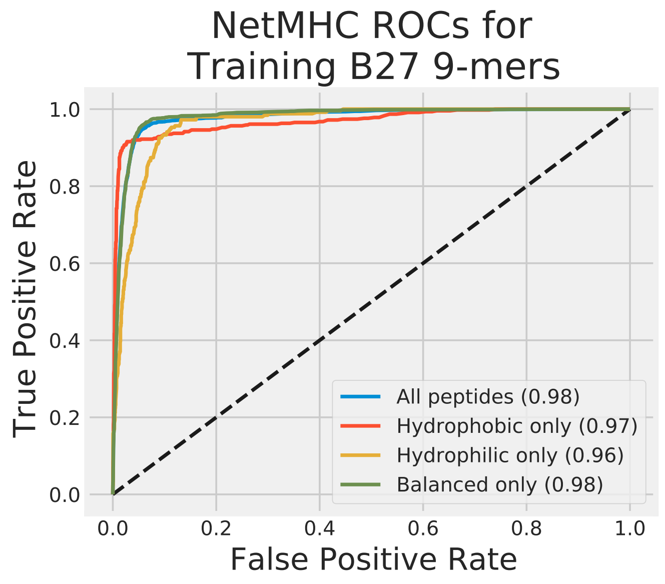


Figure 10.18: Receiver Operating Characteristic curves for NetMHC-4.0 (top) and NetMHCpan-4.1 (bottom) for A2 based on the training dataset. In each subfigure, plots are drawn for all peptides (blue), hydrophobic peptides only (red), hydrophilic peptides only (yellow), and balanced peptides only (green). A completely random classifier is also plotted for reference (dashed black). For each plot, the Area Under the Curve (AUC) is also noted in the legend.



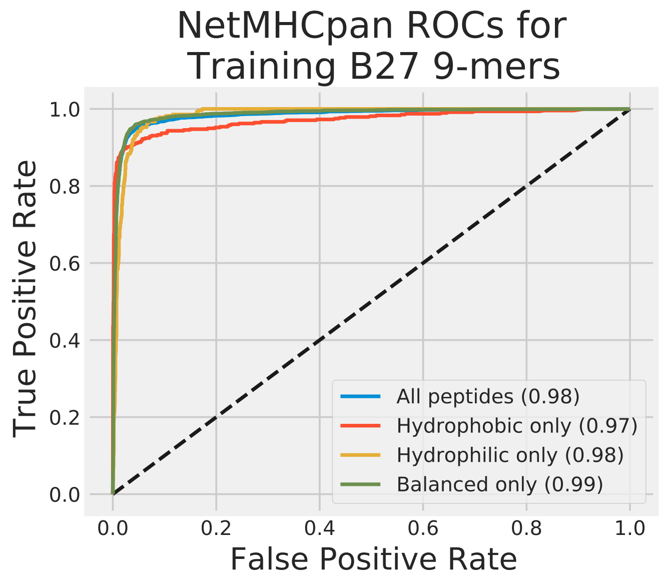
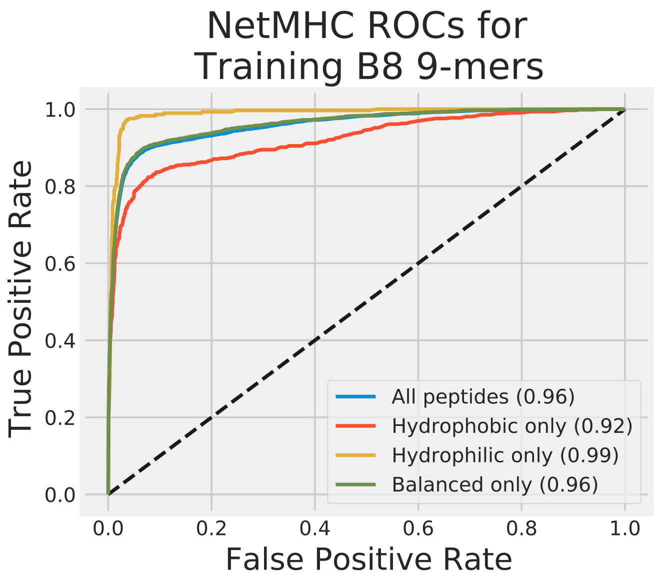


Figure 10.19: Receiver Operating Characteristic curves for NetMHC-4.0 (top) and NetMHCpan-4.1 (bottom) for B27 based on the training dataset. In each subfigure, plots are drawn for all peptides (blue), hydrophobic peptides only (red), hydrophilic peptides only (yellow), and balanced peptides only (green). A completely random classifier is also plotted for reference (dashed black). For each plot, the Area Under the Curve (AUC) is also noted in the legend.



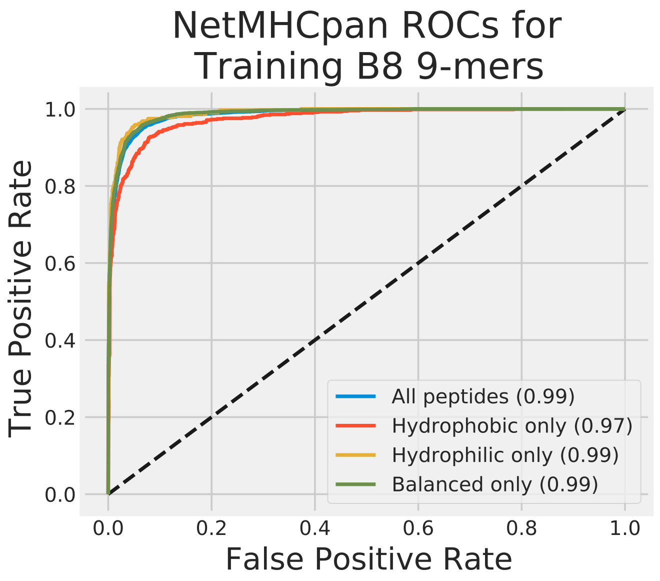


Figure 10.20: Receiver Operating Characteristic curves for NetMHC-4.0 (top) and NetMHCpan-4.1 (bottom) for B8 based on the training dataset. In each subfigure, plots are drawn for all peptides (blue), hydrophobic peptides only (red), hydrophilic peptides only (yellow), and balanced peptides only (green). A completely random classifier is also plotted for reference (dashed black). For each plot, the Area Under the Curve (AUC) is also noted in the legend.

10.4 Supplementary Material for Chapter 8

Table 10.5: P-values for the Wilcoxon rank-sum test for individual HLAs in Class I. The lower the p-value, the more "separated" the two sets being compared.

HLA	Nonrandom vs. Evasive	Omicron vs. Nonrandom	Omicron vs. Evasive
A1	4.3×10^{-7}	8.7×10^{-1}	9.2×10^{-2}
A2	2.9×10^{-5}	8.7×10^{-1}	1.7×10^{-1}
A3	2.1×10^{-7}	1.5×10^{-1}	8.6×10^{-2}
A24	1.9×10^{-9}	1.5×10^{-1}	9.2×10^{-2}
A26	7.0×10^{-7}	1.5×10^{-1}	1.8×10^{-1}
A30	7.6×10^{-6}	1.5×10^{-1}	8.9×10^{-2}
B15	3.4×10^{-7}	1.5×10^{-1}	1.7×10^{-1}
B35	2.0×10^{-7}	2.6×10^{-1}	8.6×10^{-2}
B40	2.7×10^{-6}	2.0×10^{-1}	1.7×10^{-1}
B44	1.7×10^{-6}	1.5×10^{-1}	8.6×10^{-2}
B51	1.2×10^{-4}	6.3×10^{-1}	2.4×10^{-1}

Table 10.6: P-values for the Wilcoxon rank-sum test for individual HLAs in Class II.

HLA	Nonrandom vs. Evasive	Omicron vs. Nonrandom	Omicron vs. Evasive
DR1	2.9×10^{-3}	1.5×10^{-1}	1.0×10^{-1}
DR3	3.4×10^{-6}	7.5×10^{-1}	1.0×10^{-1}
DR4	3.6×10^{-7}	2.6×10^{-1}	9.6×10^{-2}
DR7	3.1×10^{-5}	1.0×10^{0}	1.6×10^{-1}
DR8	1.3×10^{-5}	1.5×10^{-1}	9.2×10^{-2}
DR11	1.1×10^{-1}	1.5×10^{-1}	1.0×10^{-1}
DR12	5.3×10^{-5}	2.0×10^{-1}	9.2×10^{-2}
DR13	1.9×10^{-4}	8.7×10^{-1}	2.3×10^{-1}
DR1302	1.5×10^{-5}	2.6×10^{-1}	9.9×10^{-2}
DR15	9.4×10^{-7}	4.2×10^{-1}	1.1×10^{-1}

Table 10.7: Antigens predicted by NetMHCpan-4.1 for the vaccine and new Omicron subvariant spikes. The vaccine labels are BNT (BNT162b2), Ad26 (Ad26.COV2.S), and NVX (NVX-CoV2373).

HLA	BNT	Ad26	NVX	BA.2.75	BA.5
A1	14	14	14	15	15
A2	16	16	16	17	16
A3	18	18	18	17	18
A24	17	17	17	17	18
A26	22	22	22	22	22
A30	18	17	17	18	18
B15	19	19	19	19	19
B35	25	25	25	27	26
B40	8	8	8	8	7
B44	7	7	7	7	7
B51	13	13	13	11	11

Table 10.8: Class I Antigens conserved from the Original Spike Protein in the vaccine and new Omicron subvariant spikes. The vaccine labels are BNT (BNT162b2), Ad26 (Ad26.COV2.S), and NVX (NVX-CoV2373).

HLA	BNT	Ad26	NVX	BA.2.75	BA.5
A1	14	14	14	11	12
A2	16	16	16	16	16
A3	18	18	18	16	16
A24	17	17	17	16	16
A26	22	22	22	19	19
A30	18	17	17	15	16
B15	19	19	19	13	14
B35	25	25	25	23	23
B40	8	8	8	7	7
B44	7	7	7	6	6
B51	12	12	12	11	11

Table 10.9: Antigens predicted by NetMHCiipan-4.0 for the vaccine and new Omicron subvariant spikes. The vaccine labels are BNT (BNT162b2), Ad26 (Ad26.COV2.S), and NVX (NVX-CoV2373).

HLA	BNT	Ad26	NVX	BA.2.75	BA.5
DR1	6	6	6	6	6
DR3	13	13	13	11	11
DR4	19	23	23	18	18
DR7	15	15	15	13	13
DR8	8	8	8	8	8
DR11	1	1	1	3	3
DR12	9	9	9	15	15
DR13	3	3	3	3	3
DR1302	10	10	10	12	15
DR15	25	25	25	27	24

Table 10.10: Class II Antigens conserved from the Original Spike Protein in the vaccine and new Omicron subvariant spikes. The vaccine labels are BNT (BNT162b2), Ad26 (Ad26.COV2.S), and NVX (NVX-CoV2373).

HLA	BNT	Ad26	NVX	BA.2.75	BA.5
DR1	6	6	6	6	6
DR3	13	13	13	11	11
DR4	19	19	19	18	18
DR7	15	15	15	12	12
DR8	8	8	8	8	8
DR11	1	1	1	1	1
DR12	9	9	9	9	9
DR13	3	3	3	3	3
DR1302	10	10	10	5	7
DR15	25	25	25	19	16