Simulating Chemical Reaction Networks for Computing with Naturally-Occurring DNA

Owen S. Hoffend

Student ID: 5170538

Project Advisor: Marc Riedel

EE 4982V - Spring Semester 2020

Abstract

Chemical Reaction Networks (CRNs) are systems of cascading chemical reaction equations that can be designed to perform computation. This project focuses on the design of computational chemical reaction networks using DNA. The use of DNA for CRNs has been demonstrated by past studies, however all existing implementations require the costly step of synthesizing user-defined DNA nucleotide sequences, building strands one nucleotide at a time. A collaborative DARPA-funded research group is investigating cost-reduction measures by encoding data on selectively-damaged (nicked) naturally-occurring bacterial DNA. This method is orders of magnitude cheaper because no DNA synthesis is required, however performing computation is more complex. This report outlines the theory, design, and output of a simulation program that models the behavior of these nicked DNA reactions in order to better understand their potential for computation. This simulator enables rapid prototyping and error analysis of candidate designs. Using the simulator, implementations of fundemantal computational elements, as well as more complex functions such as cosine, are tested. From this work, insights into the mathematics and limitations of the proposed DNA-based computational elements are gained.

Contents

List of Figures

1 Introduction

As progress toward building ever smaller silicon transistors for modern processors continues to slow, computer engineers are increasingly seeking to develop new computer hardware substrates. Taking inspiration from the complexity seen in biological systems, research into computing with bio-molecular reactions has proven fruitful. In particular, DNA has been

well studied as a suitable physical basis for implementing networks of arbitrary user-defined chemical reactions [8]. So-called Chemical Reaction Networks (CRNs) are systems of multiple cascading chemical reactions that compute by modifying (increasing or decreasing) the chemical concentrations of molecules in a constant-volume solution [1]. Using this scheme, the designer specifies a set of chemical reaction equations implementing the desired operation, and then finds or builds molecules that interact accordingly. DNA, in particular, is a common candidate molecule for such systems due to its stability, its useful tendency to react almost exclusively with complementary strands, and the existence of well-studied techniques for building/sequencing it. Prior work has shown that DNA is capable of implementing complex combinational and sequential functions [8]. Moreover, a methodology exploiting the stochasticity of chemical reactions has led to the development of an algorithm for synthesizing DNA CRNs for arbitrary mathematical functions [7].

The existing DNA-based CRN implementation operates on data represented by custom-designed DNA base pair sequences. One critical limitation of this approach is its reliance on DNA nucleotide synthesis. At approximately $0.12 per equivalent binary bit [9] (several orders of magnitude higher than a single photolithographic transistor), the cost of synthesizing the required reactants becomes a severe hindrance to the development of appreciably sophisticated CRNs. As part of a DARPA-funded research project in collaboration with the Biotechnology Center at University of Illinois Urbana-Champaign (UIUC), Marc Riedel's Circuits, Computation, and Biology research group is currently investigating methodologies for performing computation by modifying naturally-occurring DNA molecules, such as those extracted from the bacterium *Escherichia coli*, instead of relying on costly DNA synthesis. The group has developed a data encoding scheme based on making "nicks" in the sugar-phosphate backbone of extracted bacterial DNA, and has demonstrated how computation operations such as sorting may be performed on such data [3]. Since the fundamental molecular interactions are different, designing computational hardware with nick-based encoding is considerably more complex than with traditional nucleotide synthesis. This project aims to investigate methods of adapting the rich CRN synthesis techniques built for the nucleotide-synthesis approach to work with the nick-based approach. In particular, the project's goals are to develop a simulator for modeling nicked-DNA reactions in order to enable rapid prototyping of designs and to better understand the error rates of existing design proposals.

This report outlines the design of a Monte-Carlo simulation program for modeling the interactions between nicked DNA molecules. Specifically, the goal of the simulator is to test the methodologies for encoding data and performing basic logic operations (AND and NOT) with nicked DNA that are being proposed by Prof. Riedel's research lab. Its output is used to characterize the error profile of these functions, thus identifying potential design flaws without requiring physical experiments. Such rapid testing ability also facilitates the design of new DNA functions and logic gates, some of which are explored in this report. The simulation algorithm differs from previous Monte-Carlo chemical kinetics simulators by taking into account DNA base-pair sequences and nick locations when determining which chemical reactions are available. The required process of matching DNA base-pair sequences at candidate binding sites is computationally intensive and difficult to implement efficiently. Three algorithms were tested, and the chosen method achieves nearly a 100x speedup when compared to a naiive brute-force approach. This simulator was successfully utilized to implement complicated functions such $y = cos(x)$, generate error heatmaps, and draw mathematical conclusions.

2 Background

2.1 Chemical Reaction Networks: A Computing Paradigm

Information in a chemical reaction network is typically represented as the chemical concentrations of input, output, and intermediate molecular species in a fixed-volume solution. This is a measurable quantity, and for fixed volume solutions it is directly related to the total number of moles of each molecular species. In this way, a reaction such as $X \rightarrow 2Y$ would produce a concentration of molecule Y equal to twice the concentration of molecule X after the reaction proceeds to completion. The exact molecules to be used are not specified by the ideal CRN equation, only that they are distinct and interact according to the given equations. Section 2.3 deals with the selection of suitable physical molecules. For now, the discussion is limited to ideal chemical kinetics.

As an elementary example of how multiple chemical reactions can interact to perform more complex computation, consider the following three-equation implementation of a $max(X, Y)$ function [1]:

$$\begin{aligned} \mathrm{X_1} &\longrightarrow \mathrm{X_2} + \mathrm{Z} \\ \mathrm{Y_1} &\longrightarrow \mathrm{Y_2} + \mathrm{Z} \\ \mathrm{X_2} + \mathrm{Y_2} + \mathrm{Z} &\longrightarrow \emptyset \end{aligned} \tag{1}$$

Where X_1, X_2, Y_1, Y_2, and Z each denote different molecular species, and $\emptyset$ denotes an untracked waste product. Suppose equation 1 is implemented in a well-mixed solution, with an initial state consisting of a finite number of X_1 and Y_1 molecules. After a finite time, all molecules of X_1 and Y_1 are consumed due to the action of the first two chemical reactions. Accordingly, the number of Z molecules produced is the sum of the respective counts of X_1 and Y_1 molecules. The third reaction proceeds until either X_2 or Y_2 runs out, consuming a number of Z molecules equal to $min(\#X_1, \#Y_1)$. The remaining number of Z molecules is equal to $max(\#X_1, \#Y_1)$, and therefore, in a fixed-volume solution, the output concentration $|Z|$ will be equal to $max(|X_1|, |Y_1|)$.

Based on the principles of chemical kinetics, the action of each reaction in a CRN can be described mathematically by an ordinary differential equation. For instance, as noted in [5], the behavior of reaction $X_1 + X_2 \xrightarrow{k} X_3$ can be expressed as:

$$-\frac{d[X_1]}{dt} = -\frac{d[X_2]}{dt} = \frac{d[X_3]}{dt} = k[X_1][X_2] \tag{2}$$

Where k is the "rate constant", a parameter representing the speed of the reaction. Intuitively, equation 2 outlines the reaction's dependence on the concentrations of the input molecules. Lower reactant concentrations correspond to lower probabilities of the reaction occuring, and therefore a lower rate of change for all molecules involved.

For large CRNs, solving the chemical kinetics ODEs becomes increasingly difficult. Fortunately, a good intuition of the behavior of these systems can be gained through Monte-Carlo style simulation. The Monte-Carlo approach maintains a count for each molecule type in the CRN, and, in each iteration, probabilistically picks a reaction to "fire" and updates the molecule counts accordingly. For instance, if the chosen reaction was $A \rightarrow B$ it would subtract one from the count for A and add one to B. This is done by considering a set of reactions R. For each reaction $r_i \in R$, compute the probability of the r_i reaction occurring during the current iteration:

$$P(r_i) = \frac{\alpha_i}{\sum_{n=0}^{|R|} \alpha_n} \tag{3}$$

Where α_i represents the number of ways to assemble the required reactants for r_i to "fire" from the current molecule counts in the simulation. The denominator summation in eq. 3 is just the total number of ways to "fire" any reaction in the CRN. α_i is computed as follows:

$$\alpha_i = k_i \prod_{j=0}^{|X_i|} \binom{x_{ij}}{n_{ij}} \tag{4}$$

In equation 4, X_i is the set of reactants for reaction r_i. The product multiplies together the number of ways to choose the required number of molecules (n_{ij}) of reactant x_{ij} for all reactants $x_{ij} \in X_i$. α_i is scaled by the rate constant k_i to better model physical CRNs. The Monte-Carlo simulation simply chooses a random number in the range $[0, 1]$ and, based on the probabilities computed with equation 3, picks which reaction to "fire". This is the basis for the operation of the simulator discussed in this report, however there are some importand differences that will be discussed in Section 4.2. For a discussion of more sophisticated CRN simulation algorithms, see Danial Gillespie's *Stochastic Simulation of Chemical Kinetics* [4].

2.2 Exploiting Molecular Randomness to Compute Functions

As shown in equation 3, at a molecular level, individual chemical reactions can be modeled as firing with an approximately random probability directly proportional to the product of the reactants' concentrations. This section explores how this randomness can be exploited to benefit computation via stochastic CRNs [7].

To provide context for stochastic CRNs, it is necessary to first turn the discussion back to the digital-electronic domain. A large body of research has been published by Marc Riedel's Circuits, Computation, and Biology Group and others surrounding the construction of electronic circuits in the digital domain that act on "random" data (for instance, see refs [6] and [2]). This so-called "stochastic logic" representation encodes data by specifying a real-valued probability of 1's versus 0's in a sequence of random binary bits, instead of encoding via the traditional positional binary approach. The stochastic representation exhibits unusual and useful properties on conventional digital hardware. For instance, when an AND gate is supplied with two stochastic bitstreams with probabilities P_1 and P_2, sampling its output will yield a probability $P_{out} = P_1 P_2$, the product of its two inputs.

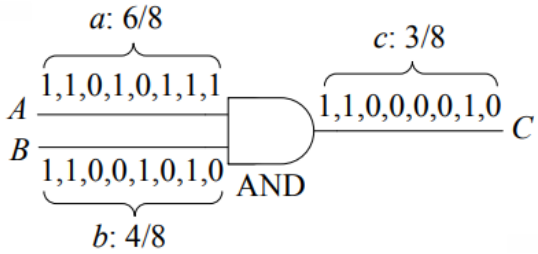

Figure 1: Computing the product of two stochastic bitstreams with an AND gate [6].

In figure 1 above, note that the output probability, $\frac{3}{8}$, is equal to the product of the two input probabilities as expected.

With stochastic logic, a traditionally-expensive floating-point multiplication operation requires only a single logic gate. In their paper *Synthesizing Logical Computation on*

Stochastic Bit Streams, Qian and Riedel show how arbitrary polynomial functions may be synthesized via stochastic logic [6]. Since increasing the number of output sample bits increases the computation's accuracy (See [2], p. 891-892), many AND operations must be performed sequentially to obtain a sufficiently accurate result, but only one physical AND gate is necessary to perform the computation. Thus, stochastic logic essentially sacrifices speed in exchange for lower silicon area and reduced power draw. For inherently imprecise and/or energy-limited applications, notably machine learning algorithms operating on battery-powered devices, the tradeoffs offered by stochastic logic can be favorable [2]. Another drawback of stochastic logic in digital electronics, however, is the need for specialized digital circuits (such as linear-feedback-shift-registers, LFSRs) to generate pseudo-random bitstreams. Such circuits take up silicon area, and precise probability values, like what might be required for the weights of a neural network, are difficult to synthesize.

In [7], the concepts of stochastic logic are applied to CRNs. In their methodology, the exploitation of random chemical interactions entirely eliminates the requirement for specialized pseudo-random number generators. Furthermore, the inherent parallelism of chemical interactions mitigates the speed issue present in digital electronics. The well-established stochastic logic toolkit enables mathematical operations ranging from real-valued multiplication to arbitrary polynomial functions to be reliably synthesized using CRNs.

For Stochastic CRNs, real values are encoded as a ratio between the chemical concentrations of two molecular species, say X_0 and X_1 (analogous to binary 0/1 in the digital representation) [7]:

$$x = \frac{|X_1|}{|X_0| + |X_1|} \tag{5}$$

To see that this is equivalent to the digital stochastic representation, recall that in a fixed-volume chemical solution, concentration is analogous to molecule count. Therefore equation 5 measures the fraction of "1" molecules out of the total number of molecules in the solution ("1's" and "0's"), and thus the probability of encountering a "1" molecule. Having established this, the "AND" gate equivalent consists of four chemical reactions, two for each input A and B:

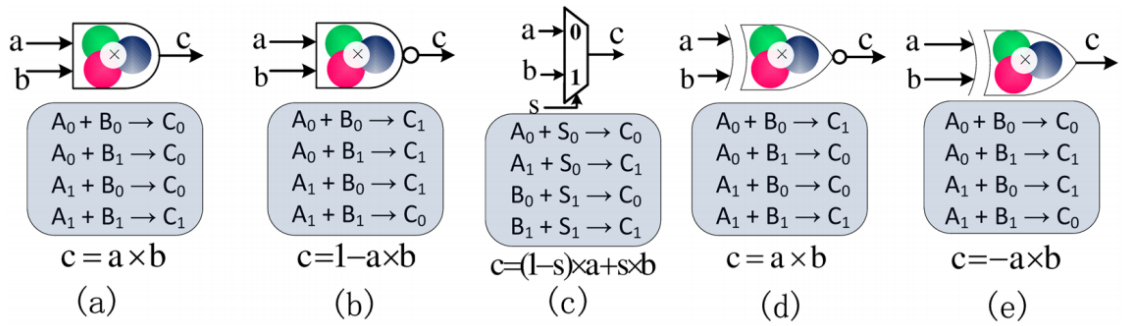

Figure 2: Fundamental molecular stochastic logic gates [7].

Intuitively, the reactions from figure 2(a) produce output C_1 with a probability equal to the probability of a molecule of A_1 and a molecule of B_1 reacting with each other in solution. In all other cases, the reactions produce C_0. Such behavior is analogous to the digital stochastic AND gate. Figure 2 also shows several other fundamental CRN stochastic logic gates. Of particular note, (b) implements the logical inverse of function (a), a NAND

gate. These two molecular gates, AND and NAND, are sufficient for approximating arbitrary mathematical functions. To see how, first consider the following decomposition of a polynomial $P(x)$, rewriting in terms of only multiplication and "one-minus" operations.

$$\begin{aligned} P(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ... + a_n x^n \\ P(x) &= b_0(1 - b_1 x(1 - b_2 x(1 - b_3 x ... (1 - b_{n-1} x(1 - b_n x))))...) \end{aligned} \tag{6}$$

Where $b_0 = a_0$, and $b_i = \frac{a_i}{a_{i-1}}$. The property expressed by equation 6 is known as *Horner's Rule* (see ref [7]). The nested algebraic form consisting of multiplication (AND) and multiply-one-minus (NAND) operations allows the polynomial to be directly translated to stochastic CRN logic gates. For an arbitrary mathematical function, the process consists of writing it as a Taylor Expansion of degree n, converting it to an implementable form via Horner's Rule, then finally synthesizing the required stochastic CRN logic-gate reactions. Figure 3 below shows an example implementation of the function $y = cos(x)$ with stochastic CRN AND and NAND gates:

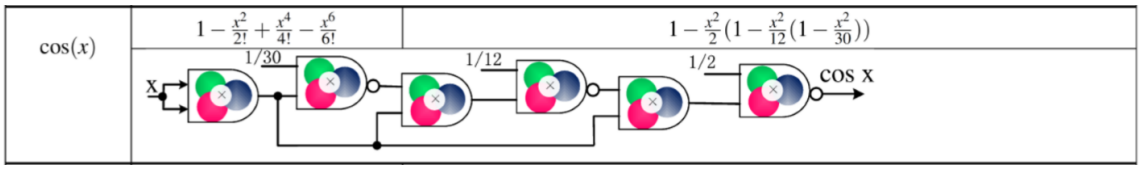

Figure 3: A cosine function implemented as a stochastic CRN [7].

Where the upper left and right functions are the 6th-degree Taylor series expansion and the equation rewritten with Horner's rule, respectively. Observe how the converted polynomial maps onto the AND and NAND gates. The input x, as well as all fractional input constants, are expressed as a ratio of concentrations according to equation 5.

Lastly, an important note is that the logical NAND gate is equivalent to an AND gate and a NOT gate chained together. For computation using nicked DNA (see sections 2.4, 3.1, and 3.2), a pure NOT gate is far easier to implement than a pure NAND gate, and so AND and NOT are used as the fundamental gates for implementing polynomials.

2.3 Physical Implementation of CRNs with DNA

So far, the discussion and experimentation surrounding CRNs has been limited to the domain of mathematical theory and computer simulations. The difficulty in synthesizing physical CRNs lies in the creation of large numbers of unique and stable molecular species, with arbitrary chemical dynamics. A large body of research has been published considering DNA as a suitable medium for the physical implementation of CRNs. DNA is readily readable and writable by current technology, and, critically, single-stranded DNA (ssDNA) base-pair sequences will only react with their complement sequences (ACG will only pair with TGC, etc) to form dual-stranded DNA (dsDNA). The latter property allows large numbers of custom molecular species to be specified without fear of undesired interference.

The following figure, adapted from [1], illustrates the basic method of action for DNA-based CRNs:

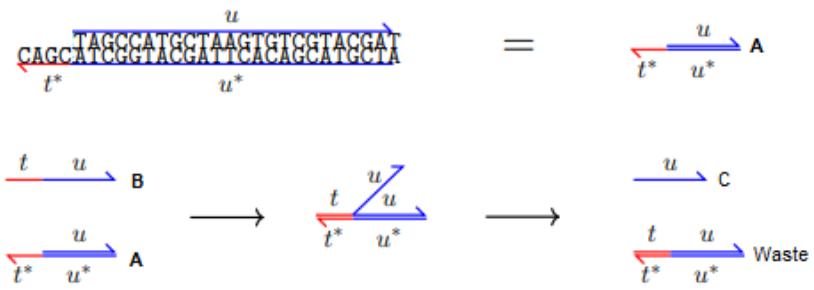

Figure 4: Basic DNA interactions for CRNs [1].

In figure 4, the distinct molecular species in the system are labelled as A, B, and C, whereas functional sections of the DNA strands are labelled with lowercase letter variables. t^* and u^* are the nucleotide complements of t and u, respectively. The exposed (red) section of ssDNA on molecule A is known as a "toehold", and represents a location where another DNA strand may bind to the molecule in order to initiate a reaction. In the bottom left of the figure, when section t of molecule B comes into contact with t^* on molecule A, a "DNA strand displacement" reaction is initiated. The binding of t with t^* essentially creates a force which "unzips" molecule C off of molecule A. Thus, the above example implements the simple reaction $A + B \rightarrow C$.

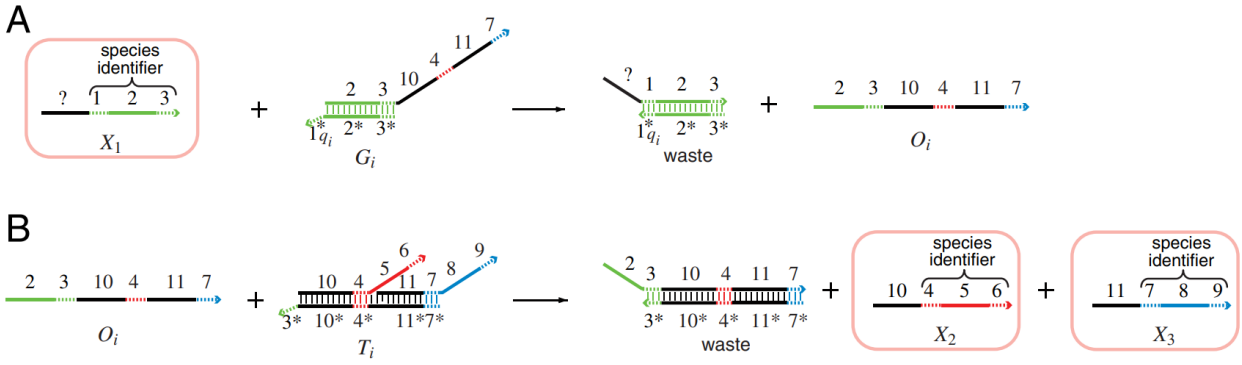

Figure 5: DNA Reactions Implementing a CRN Equation [8].

Figure 5 illustrates a second example, this time implementing the reaction equation $X_1 \rightarrow X_2 + X_3$. Note how X_1 releases O_i from G_i, which then releases both products from T_i. This example introduces the notion of a "species identifier", a general way encode a variable quantity in a CRN. These consist of two toe-hold binding sections (dotted colored lines), and one identification section (solid colored lines). Species identifiers serve as a universal structure for input/output molecules in DNA CRN reactions, and are used to enable reaction cascading. DNA molecules that are not species identifiers are intermediate molecules that don't represent inputs/outputs. These extra molecules are supplied in abundant concentrations with respect to the reaction input/outputs, to prevent affecting the computation. Relative reaction rates (k, as in equation 2) are tuned by modifying the length and/or base pair sequence of the toeholds.

For reaction equations with more than one reactant, such as $2X_1 \rightarrow X_3$, step A from figure 5 may be modified such that only half of molecule O_i is produced (DNA sections 2, 3, and 10), and another step could be added that mirrors step A everywhere except for producing the other half of O_i (sections 4, 11, and 7). In this way, both the first and second halves of O_i would be required to release the CRN's product X_3, because the first

half of O_i must expose a toehold on molecule T_i at 4^* for the second half to be able to dislodge X_3. Note that in this case the X_2 byproduct is ignored.

By chaining DNA reactions such as those shown in figures 4 and 5, it is possible to synthesize physical implementations of the complex stochastic CRNs proposed in the previous section (2.2).

2.4 DNA Nicking: Cheaply Encoding Data on Naturally-Occurring DNA

One major drawback of the DNA nucleotide-based data representation is the difficulty of synthesizing all of the required DNA molecules for the desired reactions to work. At approximately \$0.12 per equivalent binary bit [9], even moderately large amounts of data and/or computation are prohibitively expensive. To solve this problem, Tabatabaei et al. describe a chemical process for modifying pre-existing bacterial DNA in their paper *DNA Punch Cards: Encoding Data on Native DNA Sequences via Topological Modifications* [9]. Nicks in bacterial DNA, such as *E Coli*, are achieved by cutting the sugar-phosphate backbone of the molecule through the use of an enzyme. The presense or absense of these nicks along the length of a DNA strand represents information, which can then be read back via a chemical process involving "solid-state nanopores" [9]. The encoding/decoding scheme is shown below:

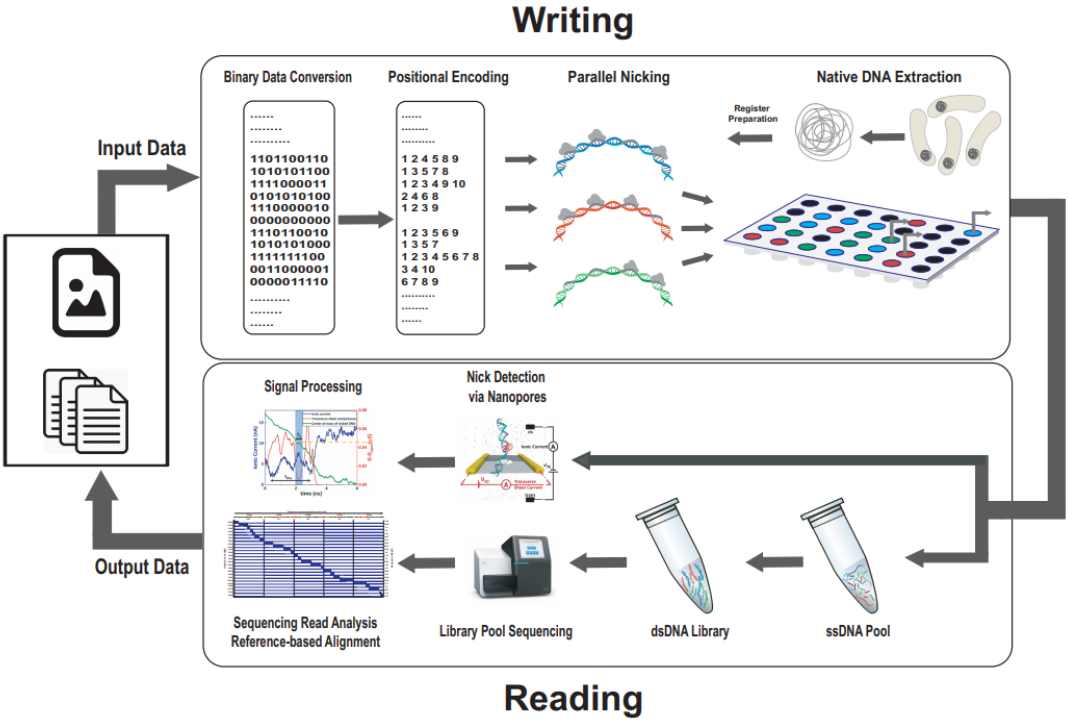

Figure 6: Block-diagram process of nicking DNA for data storage [9].

This DNA nicking method was originally developed as an extreme-density DNA data-storage solution, theoretically capable of up to 4 *Exabytes*, or 10^{18} bytes, per gram of DNA. However, the technology benefits computation for the same reasons it benefits storage: cost per equivalent binary bit. At only $\$1.5 * 10^{-6}$ per bit, synthesis using the DNA nicking method costs approximately 5 orders of magnitude less.

3 DNA Computation Methodology

3.1 Encoding Data with Nicked DNA

The Riedel research group has proposed a theoretical platform for implementing computation on nicked DNA. The process allows the implementation of stochastic AND and NOT gates (see section 2.2), which in turn allows arbitrary Taylor polynomials to be implemented. The basic premise of the encoding scheme is to store real values within the range $[0, 1]$ as a ratio of the number of exposed bases (toeholds) versus non-exposed bases along a DNA strand. As an example, consider the following figure:

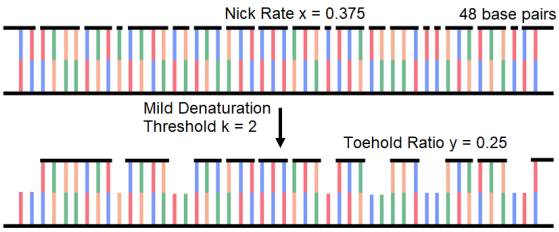

Figure 7: Utilization of nicks for storing data within DNA.

Figure 7 is adapted from an unpublished presentation by Marc Riedel for the DARPA group. The top half of figure 7 shows a DNA sequence with numerous nicks along one of its phosphate backbones. To create the nicks, a chemical process (specifically, a controlled hydroxyl radical reaction) is used that results in nicks occurring *randomly* with probability x, known as the *nick rate*. Following this, the strand is heated to partially seperate the two halves of the DNA molecule in a process known as *denaturation*. Since DNA binding is more stable for longer base-pair sequences, the sections of the molecule that are disconnected by nicks are more likely to break free from the main molecule. In the figure 7, sections of $k = 2$ base-pairs or smaller break free, while all others stay bound. The value of k, which is termed the denaturation threshold, can be fine-tuned by controlling the temperature of the denaturation; higher heat will supply enough energy dislocate larger sequences. After denaturation, the main molecule is left with a fraction $T(x, k)$ of exposed toeholds. In the above figure, there are 48 bases total, and 12 are exposed after denaturing the strand, so the value represented is $T = 0.25$. Figure 12 and Equation 7 in Section 5.1 describe the relationship between x, T, and k.

3.2 Computing with Nicked DNA

As discussed in section 2.2, the fundamental operations that are required to perform arbitrary stochastic computation are AND (performs multiplication, $T_{out} = T_a T_b$) and NOT (performs $T_{out} = 1 - T_a$). Figure 8 below is a high-level illustratration of the basic sequence of operations required to implement these gates:

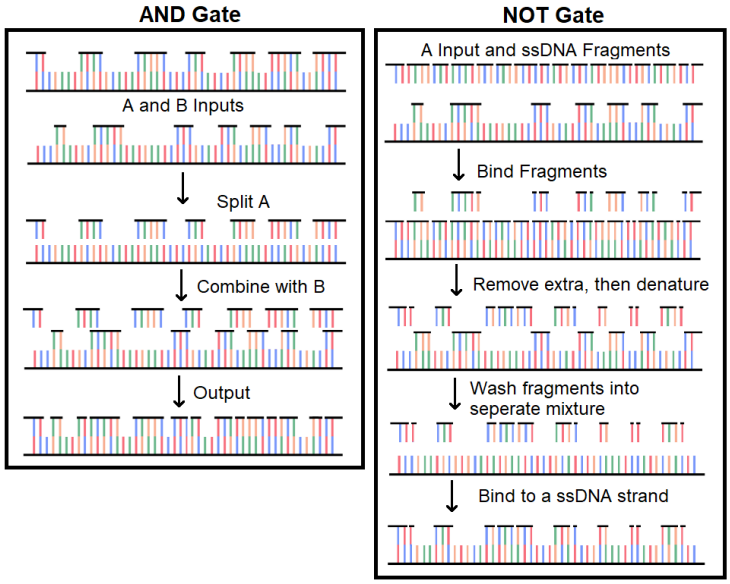

Figure 8: Computing basic stochastic logic gates with nicked DNA.

For the AND gate in Figure 8, a solution containing only strand A is first completely *denatured* (split) by heating. It is then mixed with a solution containing strand B and allowed to recombine (*hybridize*). After this, overlapping sections are trimmed off (not shown), and the result is the product of the toehold ratios for A and B. Intuitively, this is the case because a toehold in the output will only occur if a toehold existed in the same location on both A and B individually. Here, the random locations of the toeholds on molecules A and B are analogous to the random bitstreams in conventional digital stochastic logic, as described earlier.

Implementing the NOT operation for computing $T_{out} = 1 - T_a$ involves a similar set of fundamental operations. The process begins by hybdirizing an input molecule A (possessing a toehold ratio y) with random fragments of ssDNA until all toeholds are filled. Following this, the ssDNA fragments are removed from the solution by chemical means (leaving the dsDNA strand). The dsDNA strand is then heated to release the ssDNA fragments that bound to it in the first step. Critically, these small fragments are guaranteed to be shorter than the denaturation threshold of k bases long. Thus, floating in the solution are a set of ssDNA strands that form the exact complement of the toeholds on the original strand. These are then washed into a different solution containing full-length ssDNA strands cloned from the same original source as the dsDNA strand. After binding, the result is a strand representing the value $1 - T_a$.

This project also introduces a third fundamental gate option, the NOR gate, which is theorized to have lower error rates relative to the AND gate. The construction of the NOR gate is very similar to that of the NOT gate, with the only difference being that filled-toehold fragments from *two* input strands, A and B, are washed into the final solution and allowed to bind to the final ssDNA molecule. Because of this, the final molecule will, on average, have an exposed base (toehold) only if the equivalent base is covered on both input molecules, thus producing the logical behavior of a NOR gate. Stochastic NOR gates implement the function $T_{out} = (1 - T_a)(1 - T_b)$.

It is the primary intent of the simulator to test the reliability and error profile of these elementary gate design ideas, both on their own and within larger chemical reaction networks implementing complex functions.

4 DNA Computation Simulator

4.1 Simulator Software Overview

For this project, a Monte-Carlo Style simulator was built to model the interactions between nicked DNA. This software was written in Python, and the complete source code is available on GitHub. A UMN GitHub account may be required to view the code. Please contact Owen Hoffend if any issues regarding access to the source files arise.

The overall goal of the software design was to build a command-line based tool that allows a user to input a simple .JSON file with desired function definitions and other configuration options for simulation. For instance, the cosine function shown in Figure 3 can be inputted into the simulator in Horner's Rule form using the code shown in Figure 9 below:

```
"funcs":{
  "cos" : "(1-x^2/2(1-x^2/12(1-x^2/30)))",
  …
}
```

Figure 9: Code for configuring the simulator to simulate a cosine function.

Following invocation, the program will parse the user-specified input functions and initialize the simulation according to the desired settings. It is programmed to run on multiple threads, which can reduce runtime on systems with multi-core CPUs. Once finished, the program will export the simulation results to a file. Most commonly, this output is a .CSV (excel) file containing a list of (input, output) pairs from a parameter sweep of the specified function. Figure 10 shows a high-level block diagram of the software architecture, which illustrates this workflow:

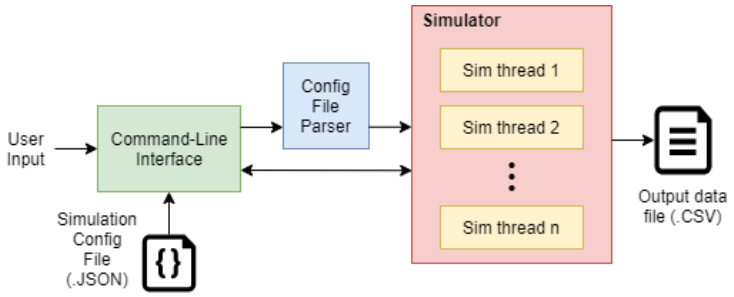

Figure 10: High-level software architecture for the simulator.

The next section describes the operation of the red simulator block in Figure 10 in more detail.

4.2 Monte-Carlo Simulation Algorithm

The basic operation of the simulator largely follows the process for Monte-Carlo chemical kinetics simulation, as described in section 2.1, however there are some important differences. Most imporantly, data in nicked-DNA reactions are no longer merely represented as a chemical concentration, but rather as a property of the molecular structure of the DNA (toehold ratio). Previous Monte-Carlo chemical kinetics simulators operated by having the set of all possible chemical reaction equations, R, defined in an input file at the time

of startup. For this simulator, this is impossible because R must dynamically track the potential reactions between single-stranded DNA (ssDNA) molecules floating around in the mixture and toeholds on double-stranded DNA (dsDNA) strands. R must therefore be re-calculated after every reaction step, which is considerably more challenging computationally. The computational efficiency of doing so is highly dependent on the data structure used. Three distinct potential algorithms were tested: A naiive brute force approach, exhaustively enumerating all possible binding configurations, a radix-search tree approach, and a hash-table approach. Compared to the other two implementations, the hash-table approach achieved the best performance on single AND-gate simulation by a 100x margin, likely due to its $O(1)$ average lookup time. The hash table approach is illustrated below:

```
Toehold t1: ATTA   Toehold t2: CAT
Combinations       Combinations
0123               012
A                  C
AT                 CA
ATT                CAT
ATTA                A
 T                  AT
 TT                  T
 TTA
  T
  TA
   A

Hashtable {
    "A"    : [(t1, 0), (t1, 3), (t2, 1)],
    "AT"   : [(t1, 0), (t2, 1)],
    "ATT"  : [(t1, 0)],
    "ATTA" : [(t1, 0)],
    "T"    : [(t1, 1), (t1, 2), (t2, 2)],
    "TT"   : [(t1, 1)],
    "TA"   : [(t1, 2)],
    "C"    : [(t2, 0)],
    "CA"   : [(t2, 0)],
    "CAT"  : [(t2, 0)]
}
```

Figure 11: Hash table data structure for matching single-stranded DNA strands to toeholds.

The hash table is built by cycling through all toeholds and adding all possible ways to bind to each toehold. In Figure 11 these are shown under $t1$ and $t2$. While doing this may seem to be no better than a brute-force approach, the advantage is that the construction of the table only needs to occur once. Each time a molecule bind occurs, the algorithm only removes the existing entries that overlap with the binding site. The matching algorithm works by iterating through all single-stranded DNA strands, looking up their sequence in the toehold hashtable, and then adding all sites for the given sequence to R. Each site returned from the hash table includes information about where the binding site is, such as a start index (shown) and molecule index (not shown).

From here, normal chemical kinetics simulation can proceed. The probability of selecting a given reaction is dependent on k, the rate constant, and here k is estimated to be proportional to the number of bases in the sequence. For instance, toehold $t1$ from Figure 11 would have a higher k than $t2$ because it is a longer sequence.

In addition to the main DNA binding function, the simulator also has a number of internal instructions for performing operations such as mixing multiple solutions containing different molecules together, heating a solution (for denaturation), nicking DNA in a solution, etc. Higher-level functions such as the AND and NOT gates discussed in the previous section are implemented as sequences of such instructions. At the top level, functions such as $y = cos(x)$ are implemented by sequencing calls to the AND and NOT gate functions.

5 Simulation Results

5.1 Toehold Ratio vs. Nick Rate Relationship

The simulator software was first used to generate graphs of the relationship between the toehold ratio T, nick rate x and denaturation threshold k for a single strand of DNA. This was done by repeatedly simulating a random nick and denature process for variety of toehold ratios from 0.0 to 1.0 (x axis) and for thresholds from 1 to 10 (dark blue to light grey lines). For each data point, a 12-sample average was taken to improve accuracy. The number 12 was chosen to align with the number of logical cores on an Intel i7 8750H CPU, allowing for parallel computation of the independent samples.

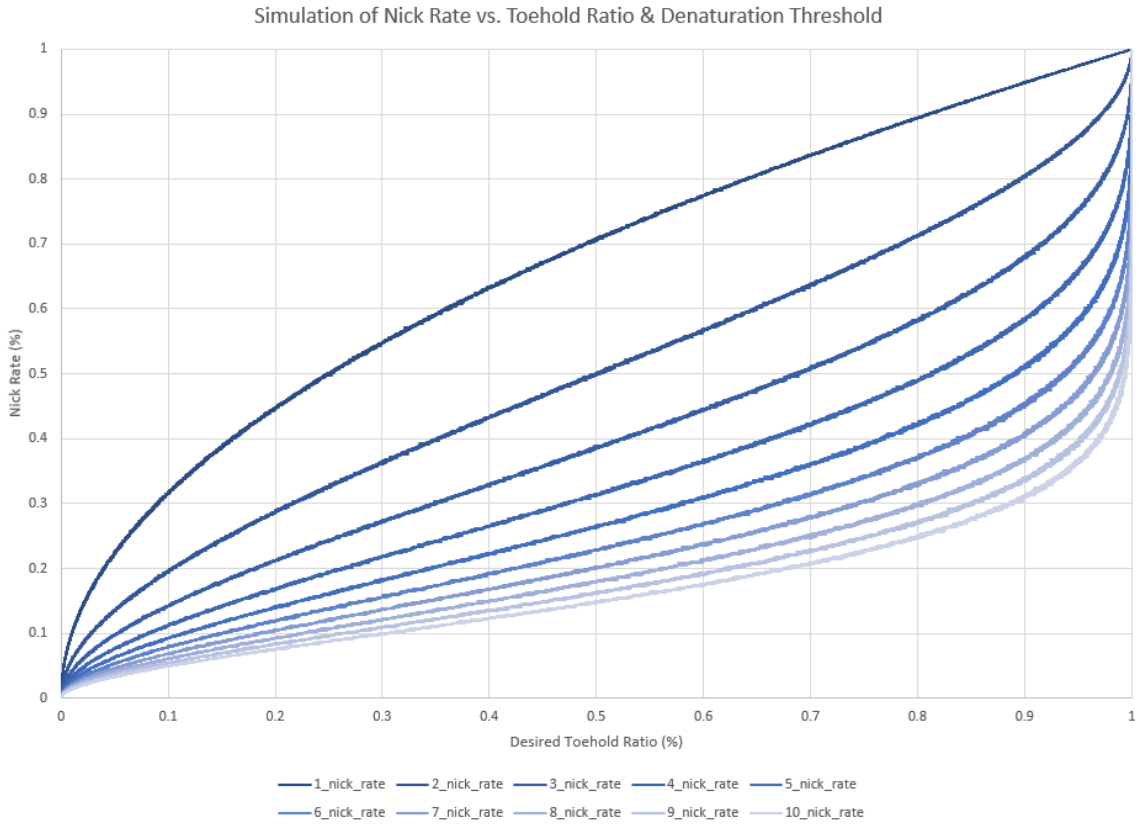

Figure 12: Simulated relationship between nick rate and toehold ratio.

Function-fitting the data shown in Figure 12 gives an equation:

$$T(x,k) = 1 - (1+kx)(1-x)^k \tag{7}$$

This equation is an unpublished result that was initially determined experimentally by other members the Riedel group and is confirmed here through function-fitting the simulator's output data. Doing so produces a near-identical overlay on the above graph. Additionally, a brief proof of the equation was developed, and is given as follows:

Begin by noting that $1 - T(x,k)$ represents the probability of encountering a covered base (not a toehold) at any location s_i along the DNA molecule. Now consider the range of bases s_{i-k} to s_{i+k} where the presense of a nick could affect whether or not s_i is covered. Nicks outside of this region are beyond the threshold range k. See Figure 13 for an illustration:

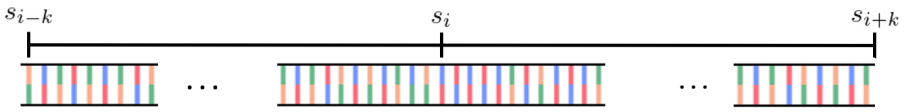

Figure 13: Range of nick locations affecting base s_i

The conditions under which s_i remains covered are the following: Either (A) There is no nick in the preceding k bases (the range s_{i-k} to s_i) or (B) There *is* a nick in the preceding k bases but no nick in the following k bases (the range s_i to s_{i+k}). Since the probability of finding no nick within k bases is $(1-x)^k$, and the probability of finding at least one nick in k bases is kx, we obtain the following equations for cases (A) and (B):

$$\begin{aligned} P[Case\ A] &= (1-x)^k \\ P[Case\ B] &= kx(1-x)^k \end{aligned} \tag{8}$$

Computing the sum of these two cases yields $1 - T(x,k)$ as expected, concluding the direct proof:

$$\begin{aligned} P[s_i\ is\ covered] &= (1-x)^k + kx(1-x)^k = (1+kx)(1-x)^k \\ P[s_i\ is\ exposed] &= 1 - [(1+kx)(1-x)^k] = T(x,k) \end{aligned} \tag{9}$$

Within the simulator, it is desirable to correct for this nick-rate-to-toehold relationship so that the programmer can simply input the desired toehold ratio value instead of having to know the correct nick rate. For instance, when $k = 6$, a toehold ratio of 0.5 requires a nick rate of approximately 0.25. Unfortunately, solving Equation 7 for x in terms of T and k becomes intractable for large k. Instead, a large lookup table of values is pre-computed, and the desired toehold ratio is found during simulation initialization using a basic search. It is this lookup table that the graph in Figure 5.1 is based on.

5.2 Fundamental Gate Accuracy

Following the lookup table computation, the average error rates of the basic AND, NOT, and NOR operations were simulated with respect to input magnitude. Figure 14 below shows three heatmaps with the results that were obtained from the simulator's parameter sweep output. The colored values represent the absolute value difference between the simulated gate's output and an ideal "correct" computation of the function. Lower error values are better. Again, 12-sample averages were used.

NOT Gate - Max Error: 0.087

INPUT	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1
NOT ERR	0	0.007	0.007	0.005	0.004	0.025	0.004	0	0.005	0.016	0.005	0.01	0.005	0.002	0.001	0.008	0.017	0.029	0.05	0.064	0.087

AND Gate – Max Error: 0.222

AND ERR	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0	0	0	0	0	0	0	0	0	0	0
0.1	0	0.01	0.02	0.03	0.04	0.05	0.059	0.065	0.049	0.018	0.01
0.2	0.019	0.019	0.04	0.058	0.074	0.085	0.095	0.081	0.053	0.01	0.009
0.3	0.055	0.013	0.041	0.071	0.087	0.096	0.101	0.086	0.046	0.005	0.012
0.4	0.1	0.004	0.037	0.063	0.084	0.098	0.101	0.095	0.042	0.022	0.002
0.5	0.136	0.029	0.016	0.048	0.072	0.086	0.09	0.071	0.041	0.012	0.008
0.6	0.166	0.055	0.003	0.034	0.06	0.068	0.097	0.06	0.026	0.019	0.014
0.7	0.201	0.087	0.032	0.011	0.042	0.063	0.065	0.043	0.01	0.022	0.011
0.8	0.21	0.101	0.053	0.012	0.024	0.037	0.036	0.039	0.004	0.034	0.003
0.9	0.219	0.133	0.082	0.019	0.002	0.013	0.012	0.006	0.015	0.033	0.004
1	0.222	0.161	0.102	0.066	0.031	0.028	0.002	0.012	0.039	0.051	0.003

NOR Gate – Max Error: 0.174

NOR ERR	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0	0.004	0.012	0.004	0.005	0.008	0.006	0.005	0.029	0.052	0.092
0.1	0.002	0.014	0.038	0.034	0.037	0.063	0.047	0.05	0.036	0.002	0.045
0.2	0.005	0.031	0.049	0.059	0.084	0.107	0.097	0.108	0.083	0.035	0.003
0.3	0.003	0.041	0.064	0.093	0.103	0.125	0.14	0.126	0.099	0.064	0
0.4	0.001	0.052	0.081	0.112	0.132	0.158	0.143	0.146	0.11	0.06	0
0.5	0.005	0.054	0.104	0.151	0.159	0.174	0.165	0.145	0.1	0.05	0
0.6	0.014	0.046	0.097	0.142	0.157	0.161	0.151	0.12	0.08	0.04	0
0.7	0.003	0.064	0.102	0.131	0.137	0.147	0.12	0.09	0.06	0.03	0
0.8	0.015	0.041	0.077	0.107	0.116	0.1	0.08	0.06	0.04	0.02	0
0.9	0.052	0.004	0.033	0.064	0.06	0.05	0.04	0.03	0.02	0.01	0
1	0.093	0.046	0.006	0	0	0	0	0	0	0	0

Figure 14: Heatmaps of simulated DNA logic gate error vs. input value.

Several important conclusions can be drawn from the data shown in Figure 14. First, the NOT gate error is relatively low overall, and increases as the input magnitude approaches 1. Similarly, the AND gate has higher error rates when one of its two inputs is close to 1. When both inputs are near 0.5, the AND gate also has a slightly increased error rates. Both of these observations will be used to explain an error profile observed in a more complex function in the next section. Another useful conclusion is a practical comparison between the AND gate and the NOR gate. While the NOR gate has a lower maximum error overall, its error is less ideally distributed in the center of its operating region, instead of near the edges like the AND gate's error. For this reason, the NOR gate design was not pursued further.

For all three gates, a major source of error is DNA binding in different positions than expected. For traditional digital stochastic bitstreams, the logic works because the random bitstreams are not encumbered by systematic biases with respect to bitwise pairings within the computation. For example, a digital stochastic AND gate will always pair bits with indices A_i with B_i at timestep i. Similarly, the proposed DNA gates assume that a dislodged ssDNA strand from one moleule will always bind to an equivalent location at a receiving ssDNA strand, but this isn't always the case. Dislodged ssDNA strands, especially short ones, have some probability of reacting in incorrect locations, causing systematic result skewing as observed in Figure 14.

5.3 Cosine Function Implementation

As a final demonstration of the simulator's capabilities, the following figure shows a simulation of a nicked-DNA implementation of the function $y = cos(x)$ on the interval $[0, 1]$. The Taylor polynomial and molecular gate architecture shown in Figure 3 of Section 2.2 was used:

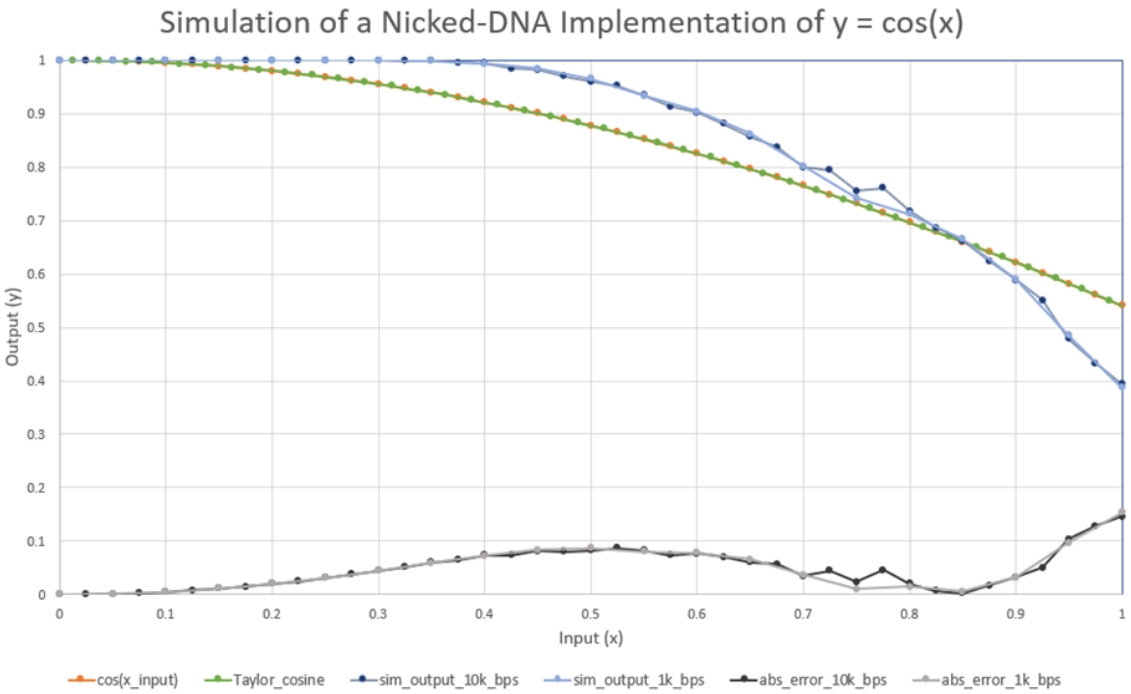

Figure 15: Simulated implementation of $cos(x)$ using nicked DNA molecular gates.

The nicked-DNA cosine function follows the general curve of the real cosine function.

The simulation was run twice with DNA molecules of 1000 base pairs and 10,000 base pairs, to compare the accuracy of the two lengths. Since the error curves between the two runs follow the same profile and do not differ significantly, the source of error is elsewhere. The exact Taylor approximation that was implemented is also graphed, showing that the polynomial appoximation to cosine is not the source of error either.

Notice that the error plot shown in Figure 15 exhibits two local minima around $x = 0.5$ and $x = 1.0$. Compare this to the error heatmaps for the AND and NOT gates from Figure 14, and a clear similarity can be observed. Looking at the circuit for the cosine function (Figure 3), the quantity x^2, computed with the first AND gate, has a high fan-out to the rest of the circuit, which may explain why the error profile resembles that of an AND gate.

Another potential culprit of inaccuracy is correlation between random values introduced by copying the x^2 result from the first AND gate to multiple other inputs. To illustrate, consider how $AND(0.5, 0.5)$, which ideally produces a result of $0.5 * 0.5 = 0.25$ could instead produce a result of 0.5 if both inputs are exactly the same. When they are the same (perfect correlation), a 1 from the first input will always be paired with a 1 in the second input, so the AND gate has no effect. Further analysis is required to determine if correlation is occurring in the cosine function. Solving this error problem is critical for the practicality of nicked-DNA computation.

6 Conclusion

The primary outcome of this project was a Monte-Carlo simulation program for modeling the behavior of nicked-DNA reactions. It was built with the intention of assisting Prof. Riedel's research group with testing designs for logic gates and higher-level computational elements utilizing such reactions. The algorithm that was implemented allows for fast simulation due to careful attention to performance optimizations during development.

Following the simulator development, it was subsequently used to generate plots of the relationship between the nick rate and the toehold ratio of a DNA strand, for a given threshold k. These plots are useful because the relationship is nonlinear, and having accurate knowledge of it is a necessity to accurately provide input to a nicked-DNA chemical reaction network. By working backwards from a function-fit equation, a concise proof of the identified relationship was found during the course of this project.

The simulator was also used to generate plots of the error of three basic nicked-DNA logic gates: AND, NOT, and NOR. The NOR gate was proposed and tested as a result of this project, but was shown to have a slighly worse error profile, with respect to input magnitude, when compared to the AND gate. These error profiles provided insight into potential sources of physical limitations of nicked-DNA reactions, and were also used to explain the error profile of more complex compound functions such as cosine.

One of the most pertinent avenues of future work involves conducting physical experiments with nicked-DNA reactions to realize the proposed designs. Doing so would provide valuable information both for the designs themselves and for tweaking the operation of the simulator to better align with actual lab results. Physical experiments were originally planned as part of this project, however they were not able to be done due to the COVID-19 situation. Other potential future projects could involve designing improved molecular gates to reduce the error observed in the simulated output, compared to the AND/NOT/NOR gates described here. Such work would directly benefit the feasibility of DNA-based computation.

References

[1] Brijder, Robert. "Computing with Chemical Reaction Networks: a Tutorial." Natural Computing 18, no. 1 (2019): 119–37. https://doi.org/10.1007/s11047-018-9723-9.

[2] Brown, B.d., and H.c. Card. "Stochastic Neural Computation. I. Computational Elements." IEEE Transactions on Computers 50, no. 9 (2001): 891–905. https://doi.org/10.1109/12.954505.

[3] Chen, Tonglin, and Marc Riedel. "Parallel Binary Sorting and Shifting with DNA." International Workshop on Bio-Design Automation, July 2019.

[4] Gillespie, Daniel T. "Stochastic Simulation of Chemical Kinetics." Annual Review of Physical Chemistry 58, no. 1 (2007): 35–55. https://doi.org/10.1146/annurev.physchem.58.032806.104637.

[5] Jiang, Hua, Sayed Ahmad Salehi, Marc D. Riedel, and Keshab K. Parhi. "Discrete-Time Signal Processing with DNA." ACS Synthetic Biology 2, no. 5 (February 2013): 245–54. https://doi.org/10.1021/sb300087n.

[6] Qian, Weikang and Marc D. Riedel. "Synthesizing Logical Computation on Stochastic Bit Streams." (2010).

[7] Salehi, Sayed Ahmad, Xingyi Liu, Marc D. Riedel, and Keshab K. Parhi. "Computing Mathematical Functions Using DNA via Fractional Coding." Scientific Reports 8, no. 1 (2018). https://doi.org/10.1038/s41598-018-26709-6.

[8] Soloveichik, D., G. Seelig, and E. Winfree. "DNA as a Universal Substrate for Chemical Kinetics." Proceedings of the National Academy of Sciences 107, no. 12 (April 2010): 5393–98. https://doi.org/10.1073/pnas.0909380107.

[9] Tabatabaei, S Kasra, Boya Wang, Nagendra Bala Murali Athreya, Behnam Enghiad, Alvaro Gonzalo Hernandez, Jean-Pierre Leburton, David Soloveichik, Huimin Zhao, and Olgica Milenkovic. "DNA Punch Cards: Encoding Data on Native DNA Sequences via Nicking." BioRxiv, 2019. https://doi.org/10.1101/672394.