John Burns

Computational Biology

Professor András Aszódi

December 5, 2017

“In Silico” Simulation of L.M. Adleman’s *Computing With DNA[[1]](#footnote-1)*

A computer’s processing power and capacity are bounded by the quality of its mechanisms. Since Babbage’s Analytical Engine, great minds and monetary sums have been thrown at the evolution (and miniaturization) of computing machines and their myriad components. With the introduction of the transistor, the exponential increase of processing power and miniaturization in computers ultimately became known as Moore’s Law. Though Moore’s Law has held true for approximately one hundred and twenty years, there is a hard limit on the number of transistors that can be jammed onto a single chip. The exponential increase in processing power and miniaturization of microprocessors as described by Moore’s Law will inevitably converge. Fortunately, scientists and engineers have begun to explore various approaches to surmount problems introduced by the transistor density threshold. Some of these scientists, including computer scientist, biotechnologist, and mathematician Leonard M. Adleman, believe that the solution lies outside the realm of traditional computing technologies. Quoting Adleman from a 1998 *Scientific America* article, “The computer that you are using to read these words [the brain] bears little resemblance to a PC. Perhaps our view of computation is too limited. What if computers were ubiquitous and could be found in many forms?”[[2]](#footnote-2) Adleman believed that he could synthesize a method of computing directly with molecules, utilizing the immense natural computational power of the organic world to create sub-microscopic computers. In 1993 upon visiting a molecular biology lab for the first time, a curious Adleman began to explore methods of carrying out computations on the molecular level. In 1994, he achieved his goal by solving the infamous Hamiltonian Path Problem with a novel method of computing using DNA.

Overview and Tools:

My work is an “in silico” simulation of Leonard Adleman’s first experiments with DNA computing. In this experiment Adleman used strands of DNA to encode nodes in a directed graph. The goal was to use these DNA sequences encoding the nodes of a graph to assemble larger sequences that corresponded to Hamiltonian Paths existing in the input directed graph, in turn solving the famous Hamiltonian Path Problem. The operations of the computation were carried out using commonplace biology protocols and enzymes. I referenced two of Adleman’s published journal articles to help me construct my simulation: *Computing With DNA[[3]](#footnote-3)* and *Molecular Computation of Solutions to Combinatorial Problems*.*[[4]](#footnote-4)* My simulation was written in only Python (Python3.5.1 interpreter) and utilized the following external libraries:

* networkx – a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks
* pydna - contains classes and code for representing double stranded DNA and functions for simulating homologous recombination between DNA molecules
* random - this module implements pseudo-random number generators for various distributions
* matplotlib - a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms

Execution:

The simulation is meant to be run via terminal. The .py file to be executed is Main.py and it takes 3 arguments:

1. path to .txt file holding the graph (included in the Networks directory)
2. start vertex
3. end vertex

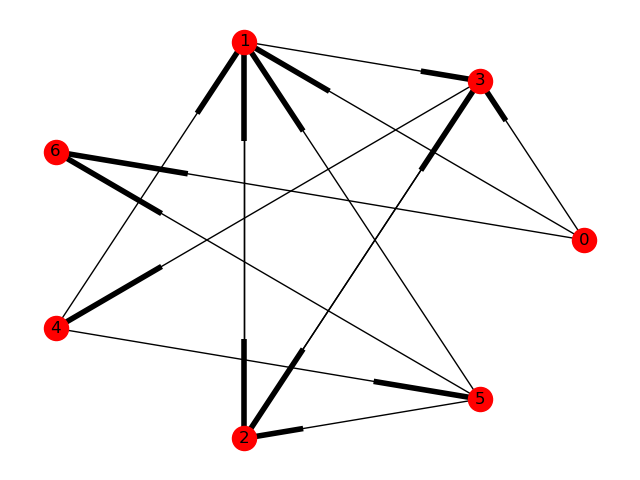
The simulation prints to the terminal some filtered ligations as well as any Hamiltonian Paths found. If any paths were found, they will be saved as .png files in the ‘graphs’ directory, along with a .png of the original graph. They are labeled appropriately.

Hamiltonian Path Problem:

Adleman used DNA computing to solve the Hamiltonian Path Problem. He chose this problem for his experiments because there it is NP-complete. He insisted that, “The problem should not appear to be contrived to fit the machine, and it should demonstrate the potential of this novel method of computation.”[[5]](#footnote-5) A Hamiltonian Path is a path that begins at the “start” vertex and ends at the “end” vertex and passes through every other node exactly once. The Hamiltonian Path Problem is, given a graph with directed edges and a “start” and an “end” vertex determine the existence of a Hamiltonian Path in the directed graph.[[6]](#footnote-6) The Hamiltonian Path Problem was shown to be NP-complete (all known algorithms have exponential worst-case time complexity), and therefore has no efficient algorithmic solution. For some graphs of just under one hundred nodes, an average electronic computer might take years to find an existing Hamiltonian Path.

Adleman’s Approach:

Figure 1

 Figure 1 shows the graph that Adleman used to model his experiment. There exists a Hamiltonian Path beginning at *Vin* = 0 and ending at *Vout* = 6, given by edges, 0→1, 1→2, 2→3, 3→4, 4→5, 5→6. Adleman chose to model his approach after a nondeterministic algorithm to solve the Hamiltonian Path Problem.

The algorithm is as follows:

1. Generate random paths through the graph
2. Keep only those paths that begin with *Vin* and end with *Vout*
3. If the graph has *n* vertices, then keep only those paths that enter exactly *n* vertices
4. Keep only those paths that enter all the vertices of the graph at least once
5. If any paths remain, say “Yes”; otherwise, say “No”

Following the algorithm, Adleman implemented every in sequence. His implementation was as follows[[7]](#footnote-7):

1. Each vertex *i* in the graph was associated with a random 20-mer sequence of DNA denoted *Oi* and for each edge *i*→*j* in the graph, an oligonucleotide *Oi*→j was created that was the 3´ 10-mer of *Oi* (unless *i*=0, in which case it was all of *Oi*), followed by the 5´ 10-mer of *Oj* (unless *j*=6, in which case it was all of *Oj*). The 20-mer oligonucleotide with the sequence that is Watson-Crick complementary to *Oi* was denoted *Ōi*.For each vertex *i* in the graph (except *i*=0 and *i*=6) and for each edge *i*→*j* in the graph 50 pmol of *Ōi* and 50 pmol of *Oi*→j respectively were mixed together in a single ligation reaction.[[8]](#footnote-8)

* The ligation reaction results in the formation of DNA molecules encoding random paths through the graph
* The choice of random 20-mer oligonucleotides for encoding the graph was 1) because 420 20-mer oligonucleotides exist so there is a low probability that oligonucleotides associated with different vertices would share long common subsequences (which could result in unintended binding during ligation), and 2) choosing 20-mers assured that binding between “splint” and “edge” oligonucleotides would involve 10 nucleotide pairs and would be stable at room temperature.

1. The product of Step 1 (DNA molecules encoding random paths through the graph) were amplified by polymerase chain reaction (PCR)[[9]](#footnote-9) using primers *O0* and *Ō6*.

* The result was only those molecules encoding paths that began with vertex 0 and ended with vertex 6 are amplified

1. The product of Step 2 (DNA molecules encoding random paths through the graph with paths that begin with vertex 0 and end with vertex 6 are amplified) was run on an agarose gel[[10]](#footnote-10), and the one hundred and forty base pair band (corresponding to double-stranded DNA encoding paths entering exactly seven vertices) was excised and soaked in doubly distilled H2O to extract DNA.

* The product was PCR-amplified and gel-purified several times to enhance its purity

1. The product of Step 3 was affinity-purified with a biotin-avidin magnetic beads system. This was accomplished by generating single-stranded DNA from the double-stranded product of Step 3 and then incubation the single-stranded DNA with *Ō1* conjugated to magnetic beads. Only those single-stranded DNA molecules that contained the sequence *O1* (and hence encoded paths that entered vertex 1 at lease once) annealed to the bound *Ō1* and were retained. This process was repeated successively with all following vertices 2,3,4, and 5.
2. The product of Step 4 (DNA sequences that encode every node) was amplified by PCR and run on a gel.

* The result was amplified Hamiltonian Paths

Under the assumptions that the ligation of two molecules is considered as a single operation and about half of the 4x1014 edge oligonucleotides in Step 1 were ligated, approximately 1014 operations were executed. The number of operations per second during ligation exceeded that of the modern supercomputers at the time, by a large degree. Adleman also calculated that with his DNA computing method, approximately 1 joule was sufficient to perform around 2x1019 operations (the theoretical maximum is 34x1019 operations per joule by the second law of thermodynamics), where supercomputers at the time could only execute 109 operations per joule. Considering storage efficiency, storing information in molecules of DNA allows for an information density of approximately one bit per nm3. Magnetic storage methods at the time such as video and cassette tapes had an information density of approximately 1 bit per 1012 nm3.[[11]](#footnote-11) Performance-wise, his molecular computations crushed even the supercomputers of the time, but simple operations such as 100-digit multiplication would prove to be a much more difficult task using molecules than it would be on an electric computer.

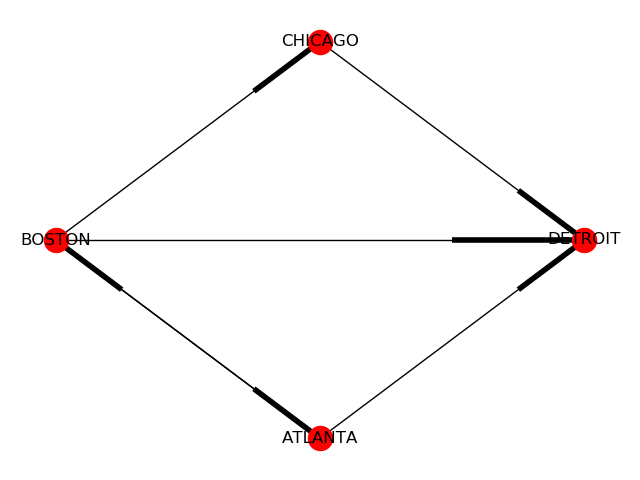
My Approach (Simulation):

Figure 2

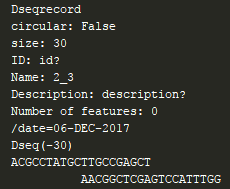
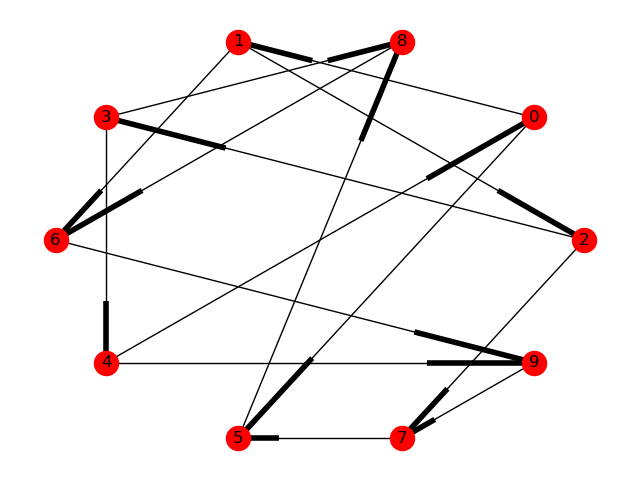
 In my simulation of Adleman’s DNA computing method, there were some steps that I could not duplicate in earnestness because DNA on a computer is not DNA from a cell. Since my “reactions” did not happen in a tube, my simulation was not as fast as the actual DNA computing model. I ran my simulation on three different directed graph models (Figure 1, Figure 2, and Figure 3—which is the Peterson Graph), all of which contain Hamiltonian Paths. In the graph in Figure 1 *Vin* = 0and *Vout* = 6. In Figure 2 *Vin* = ATLANTAand *Vout* = DETROIT. In Figure 3 *Vin* = 0and *Vout* = 6. For each graph model, my simulation detected and successfully graphed the Hamiltonian Paths. I also ran simulation where I changed the *Vin* and *Vout* so that no Hamiltonian Path existed (Figure 1 with *Vin* = 2and *Vout* = 3), and the simulation successfully completed without detecting a Hamiltonian Path. The step-by-step implementation of my simulation is as follows:

Figure 4

Figure 3

1. Each vertex *i* in the graph is associated with a random 20-mer sequence of DNA denoted *Oi* and for each edge *i*→*j* in the graph, an oligonucleotide *Oi*→j is created that iss the 3´ 10-mer of *Oi*,followed by the 5´ 10-mer of *Oj*. The 20-mer oligonucleotide with the sequence that is the Watson-Crick complementary to *Oi* is denoted *Ōi*.For each edge *i*→*j* in the graph, a double-stranded DNA sequence object (pydna.Dseqrecord) is created, where *Oi*→j is the oligonucleotide’s Watson (top) sequence and *Ōj* is the Crick (bottom) sequence and an ovhg offset of -10-mer is applied, as seen in Figure 4. When the ligation reaction begins, every *Oi*→j already has a “splint” to help bond two compatible edges. Every one of these Dseqrecords is added to a list and pydna.Assembly is used to simulate the ligation reaction that occurs in Step 1 of Adleman’s approach. The pydna.Assembly method uses an assembly algorithm based on graph theory where each overlapping region forms a node and sequences separating the overlapping regions form edges. During the assembly step, a parameter is set such that only terminal overlaps (overlaps of 10-mer) can be bonded, this ensures clean bonds. The result is a set of linear products, or DNA sequences encoding random paths along the graph. Sample output can be seen in Figure 5, which shows the assembly of a Hamiltonian Path from the graph in Figure 2 before PCR.

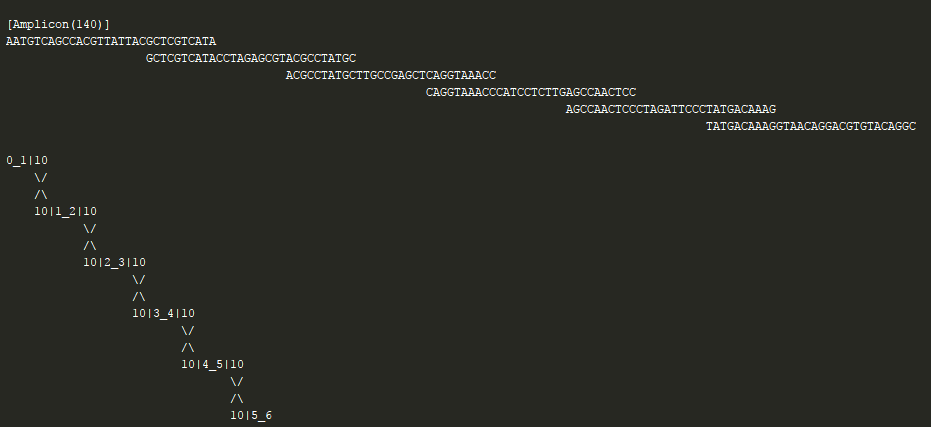
* The ligation reaction results in the formation of DNA molecules encoding random paths through the graph
* 20-mer oligonucleotides were necessary, I experimented with 10-mer, and results due to common subsequences.

Figure 5

1. The product of Step 1 (DNA molecules encoding random paths through the graph) were amplified by polymerase chain reaction (PCR) using primers *Oin* and *Ōout*. This is done using the pydna.Anneal module using the products as Step 1 as templates, and *Ōout* and *Oin* as the primers. The pydna.Anneal module provides functions for PCR.

* The result is only molecules encoding paths that begin with *Vin* and end with *Vout*.

1. Step 3 is implemented using a filtering process on the products of Step 2. The process filters out all molecules not equal to (20\*num\_nodes)-mer.

* The result is a set of molecules encoding paths that begin with *Vin* and end with *Vout* with the required length for a Hamiltonian Path in given the number of vertices in the graph.

1. The product of Step 3 is put through another filtering process in which each molecule is checked for the containment of every vertex *Vi* in the graph (aside from *Vin* and *Vout*).

* Same effect as Step 4 in Adleman’s study, just done without laboratory equipment
* The result is a set of molecules encoding paths that begin with *Vin* and end with *Vout*, have the required length for a Hamiltonian Path given the number of vertices in the graph, and passes through every vertex.
* The result is a potential set of Hamiltonian Paths

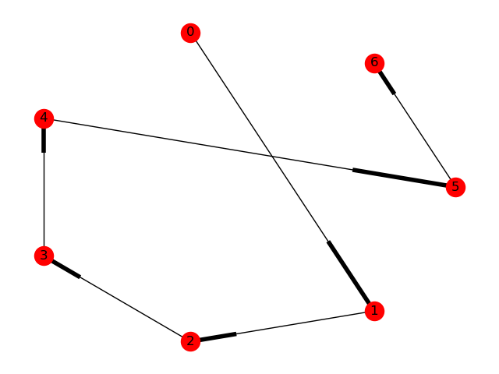
1. The product of Step 4 (DNA sequences that encode every node) are decoded and translated back into graphs using python.networkx and then plotted using python.matplotlib. This allowed the user to physically see any Hamiltonian Paths that may have been found by the simulation. Figure 6, shows the solution to the Hamiltonian Path Problem for the graph shown in Figure 1. Graph figures are stored in a “graphs” directory in the root project folder, and are generated and saved upon success of every simulation.

Figure 6

In Conclusion:

The primary difference between DNA computing performed in silico and DNA computing performed in a lab with physical enzymes and nucleotide strands is that a computer simply cannot simulate certain processes unique to the natural world. The organic cells are finely-tuned machines that function seemingly independent of an operator. It is a known well-known transgression to confuse magic and science, though at times I find myself unable to sidestep the notion of a resting supernatural essence at the nucleus of the natural world. There is a potency in living cells that can be felt. Holding an organism, even one as tiny as an ant or as passive as a tree, feels different from handling say, a pen. It is this essence that biotechnologist such as Adleman aim to capture. The primary goal of Biotechnology is to harness the incomprehensible processing power of the of living cells, and utilize it to advance mankind’s computing technologies. Alongside modern electronic computers the sub-microscopic machinery present in living cells could usher humans into a novel era of technology. DNA is arguably the most efficient methods of information storage known to mankind. As Adleman proved in his experiments, calculations on the molecular level can be performed 1) substantially faster than the same calculations executed on electronic computers, 2) on a much smaller scale than a modern electronic computer (and DNA does not produce heat of the same degree as transistors do), and 3) without the limitations thrust on electronic computers by binary operations. Three billion years of evolution has yielded an almost infinite supply of microscopic machines unbounded the state of modern technology. We can utilize these resources to revolutionize modern science and climb to the next plateau in human evolution, because electronic computers just cannot seem to get us off the one we are on.

# References:

Adleman, Leonard M. 1998. "Computing With DNA: The manipulation of DNA to solve mathematical problems is redifining what is meant by "computation"." *Scientific America* (Scientific America, Inc) 54-61. Accessed November 25, 2017.

Adleman, Leonard M. 1994. "Molecular Computation of Solutions to Combinatorial Problems." *Science* 266: 1021-1024. Accessed November 25, 2017.

1. Leonard M. Adleman, “Computing With DNA: The manipulation of DNA to solve mathematical problems is redifining what is meant by ‘computation’,” *Scientific America*, (1998): 54-61. [↑](#footnote-ref-1)
2. Adleman, “Computing With DNA: The manipulation of DNA to solve mathematical problems is redifining what is meant by ‘computation’,” 54. [↑](#footnote-ref-2)
3. Adleman, “Computing With DNA: The manipulation of DNA to solve mathematical problems is redifining what is meant by ‘computation’,” 54-61. [↑](#footnote-ref-3)
4. Leonard M. Adleman, “Molecular Computation of Solutions to Combinatorial Problems,” *Science*,(1994): 1021-1024. [↑](#footnote-ref-4)
5. Adleman, “Computing With DNA: The manipulation of DNA to solve mathematical problems is redifining what is meant by ‘computation’,” 57. [↑](#footnote-ref-5)
6. Adleman, “Computing With DNA: The manipulation of DNA to solve mathematical problems is redifining what is meant by ‘computation’,” 58. [↑](#footnote-ref-6)
7. Leonard M. Adleman, “Molecular Computation of Solutions to Combinatorial Problems,” *Science*,(1994): 1021-1022. [↑](#footnote-ref-7)
8. Ligase binds molecules together. It takes two strands of DNA in proximity and bonds them into a single strand. Ligase is used in the body to repair broken DNA strands. [↑](#footnote-ref-8)
9. PCR is used to replicate DNA molecules begin and end with given forward and backward primers. [↑](#footnote-ref-9)
10. A solution of DNA molecules is placed in one end of a slab of gel and a current is applied. The negatively charged DNA molecules move toward the anode with shorter strands moving more quickly than longer ones. This process separates DNA by length. [↑](#footnote-ref-10)
11. Leonard M. Adleman, “Molecular Computation of Solutions to Combinatorial Problems,” *Science*,(1994): 1023. [↑](#footnote-ref-11)