DNA Computing

Abhishek Dasgupta

12th December 2009

In this report, I describe DNA Computing and how it has been implemented in rudimentary forms. The Hamiltonian Path problem is discussed in detail along with its solution using DNA. DNA computation can lead to insights and improvements in parallelism, as cloning DNA is relatively easy compared to making silicon chips. DNA computing is also much more efficient and can store more information in a smaller space than normal computers.

> It was amazing to think that the solution to a mathematical problem could be stored in a single molecule.
> – Leonard M. Adleman, *Computing with DNA*, Scientific American

DNA, the abbreviated form of the molecule deoxyribonucleic acid is the ubiquitious component of life. Starting with the now famous discovery of its double-helix structure by Watson and Crick in 1953, the understanding and role of DNA in biology has exploded. Half a century later, we are now able to wilfully change the genetic code of bacteria and use them to inject code into other organisms. While we are far away from creating an artificial life form, we can already tinker with the code of life itself.

Given that DNA acts as an information carrier for the cell and has a fast and efficient method of replication, it is not surprsing that it would be used for some kind of computation. The idea was in fact proposed in 1985 [3] by H. Bennet and Rolf Landauer, but no concrete problem was solved. In 1994, Leonard Adleman published the paper in *Science [1]* which kicked off the field of DNA computation with the solution to the Hamiltonian path problem using DNA and tools from biotechnology.

Even before computers appeared, models of computation were created by logicians such as Alan Turing, Kurt Gödel and Alonzo Church. It was Turing's construction of an idealised machine, an universal computer later known as a Turing machine, which formalised the idea of computation. One of the defining features of a Turing machine is that it is *universal*, that is it can compute anything which can be computed. A Turing machine has two tapes, one input and the other output along with a finite control. The Turing machine reads from the input tape one letter at a time and depending on the state of the finite control moves either left or right on the input tape and writes to the output tape. It can also change the state of the finite control. Using these operations it is possible to compute anything.

What struck Adleman was the similarity between a key component of the DNA replication mechanism: the DNA polymerase and the Turing machine. The function of DNA polymerase is to replicate a strand of DNA. It reads a strand of DNA which consists of the bases adenine, cytosine, guanine and thiamine labelled as A, C, G and T respectively. Adenine pairs with thiamine and guanine with cytosine. The pairing is non-covalent and occurs through hydrogen bond interactions which takes place whenever two single-stranded DNA come close, forming the double helical structure. Thus we can think of the template strand (from which DNAP makes the new DNA) as the input "tape" and the DNA strand being synthesized as the output tape. Using this and tools from biotechnology such as ligation of two DNA strands, PCR reactions and affinity separation, we can encode and operate on information in DNA, thus essentially performing a computation. Adleman showed that the Hamilton path problem could be encoded DNA and standard techniques used in biotechnology could reveal whether an Hamilton path existed. In the following section, I give an outline of the Hamilton path problem following up with the encoding and process of finding the solution to the problem.

Hamilton Path Problem

The Hamilton Path problem can be briefly stated as: given a directed graph and a start and end vertex, find whether there exists a path which starts at the start vertex and ends at the end vertex while passing through each of the other vertices exactly once (such a path is termed as Hamiltonian path).

A particular example where it helps to visualise this problem is if we consider the graph as a network of cities with the directed edges representing possible routes. Then starting at one city and stopping at another, we would have to see if we could pass through each of the remaining cities exactly once. Now, there are a lot of ways one could solve this problem, but no one has found a really efficient way of finding out whether there is a Hamiltonian path. The problem is known to be NP-complete and unless P = NP, we won't have a polynomial-time algorithm which can be run on a Turing machine.

Algorithm

1. Choose k random paths from the graph.
2. For each path in the set
   (a) If the path does not have as its terminuses the start and end vertex, then delete it from the set.
   (b) Remove the path if it is not of length equal to the number of vertices (as we must pass through each vertex exactly once).
   (c) Check that a vertex is not visited more than once.
3. If any path remains then it is Hamiltonian, thus the problem has been solved.

While this is not an efficient algorithm and is not even guaranteed to get a correct answer, if we take a large enough k then there is a high probability that we will find the Hamiltonian path if one exists. This is the algorithm that was implemented by Adleman using DNA.

Tools

Just as the Turing machine operates on tapes using a finite control, we too need some kind of chemical to operate on DNA. Thankfully, over billion years of evolution there are already several molecules which can help us in "processing" information in DNA. Some of the tools are

Watson-Crick pairing Every DNA strand has its complement (for example the strand ACGT will have as its complement TGCA)

Polymerases Polymerases copy information from one DNA strand to another. For example DNA polymerase makes a complementary DNA strand from a template. DNA polymerase also needs something known as primer, a short piece of DNA to which the polymerase attaches and starts synthesising the complementary strand.

Ligases Ligases bind molecules. For example, DNA ligase binds two pieces of DNA together. Thus it is used by the cell to repair breaks in the DNA which can be caused due to irradiation or chemical damage.

Nucleases These cut nucleic acids. There are two kinds of nucleases: exonucleases and endonucleases. Exonucleases start gobbling up DNA from one end while endonucleases are more specific, cutting DNA at places where they recognize certain marker sequences. For example, EcoRI from *Escherichia coli* cuts DNA after the G in the sequence GAATTC.

Gel electrophoresis This process is used to separate DNA molecules by size. A mixture of DNA molecules of various size is put in a gel and a potential applied. The negatively charged DNA molecules migrate towards the anode, shorter strands moving quicker than the longer ones.

DNA synthesis We now have the capability to synthesise arbitrary sequences of DNA from commercial synthesis facilities, which we can use to our advantage while designing a DNA computer to solve a particular algorithm.

Mapping the problem

The first task in using DNA to solve the Hamiltonian path problem is the task of encoding the input. We've to do such a step in virtually every problem, like encoding the input for a Turing machine using a finite

set of symbols. We assign each vertex an unique eight letter DNA code. The first four letters identify the first name of the vertex while the last four letters, the last name. The first names and the last names assigned to a vertex are unique i.e. no two vertices can have the same first or last name.

Each directed edge is assigned an eight character code obtained from concatenating the last name of the origin vertex and the first name of the destination vertex. Then the computation can be performed by adding the complementary DNA vertex names and the edge names along with water, ligase, salt and a few other ingredients to a test tube. The computation finishes within a second and the solution to the problem is encoded in a molecule in the tube.

How it works

Consider the edge AB which has a code of GCATGACG and the complementary code of B which is CTGCGTAG. By design the last part of the edge code GACG and B's first name CTGC are complementary. Thus they'll anneal. Similarly when this complex encounters the edge code of BC (whose first name shall again be complementary of the complex), it will anneal with the edge code of BC. In effect the molecule being created represents one possible path through the graph. What remains is to filter out the non-Hamiltonian paths which can be done following a sequence of processes.

First, we use PCR (polymerase chain reaction [7])to amplify those DNA whose start and end vertex matches our choice. Two primers are used: the last name of the first vertex and the complement of the first name of the end vertex. PCR clones DNA which has the primer attached and in this case DNA which has both the primers attached (i.e. having matching start and end vertices) is cloned at a far faster rate than DNA which has only one of the primers attached. DNA with no primers attached is not cloned at all. Thus we end up with a far higher concentration of DNA for which the start and end vertices match.

Now we run the DNA through a gel electrophoresis process which separates DNA on the basis of length. We pick the DNA which has the correct length (corresponding to the number of vertices) and then check whether all the vertices are there in the path or not. This can be done using an affinity separation process. In affinity separation, we attach probes encoding the complementary code of a vertex to microscopic iron balls approximately a micron in diameter. When DNA is passed through a tube containing these balls, only those DNA which have the particular vertex anneal to the probes while the rest are washed out. The temperature was then raised and a solvent added to wash out the DNA which attached to the balls. The resulting DNA was then further screened for each vertex. If any DNA remained after all the screenings, then it had to encode a Hamiltonian path.

Complexity

DNA computing is essentially by its very nature, massively parallelised. Thus it acts like a nondeterministic Turing machine in a sense. In [2], algorithms are rated in two ways: the number of biological steps taken and the number of DNA strands used. For the Hamiltonian path problem, the number of biological steps taken is $O(n)$ while the number of strands is $O(n!)$. The number of biological steps is linear in n because of the n affinity separation steps that we have to undergo. We require at least $n!$ strands which represent the order in which the vertices are to be visited. Lipton [6] showed that SAT (satisfiability for formulas in conjunctive normal form) can be done in time linear in the size of the formula. An improvement in Lipton's method is that the number of strands used is only 2^n where n is the number of variables. There have been efforts towards formalising computation in DNA. Computation through DNA can be regarded as a sequence of test tubes where each subsequent test tube is created from earlier ones by some biological operation. The operations can be classified as

Extract Extract strands with given substring

Length Separate the strands by length

Pour Pour two test tubes into one, with no change of the individual strands

Amplify PCR used to make copies of strands or selected subregions

Anneal Represents all the operations that combine a test tube of single stranded DNA with other prepared strands and let them anneal together to form double strands

Cut Apply a restriction enzyme to cut strands in the test tube

Join Represents the annealing steps combining two test tubes that are unlikely to be possible in practice

Current Research and the Future

Researchers have found several NP-complete problems which can be solved using DNA. Lipton found out how to solve SAT. NP-hard problems like MAX-Clique or MAX- Circuit-Satisability (given a circuit, nd a satisfying assignment in which the largest number of variables are set to true) have also been solved using DNA [2].

There are issues still to be resolved – especially the problem of errors, as DNA is fragile and breaks easily. Some research has taken place on how to cope with errors. Algorithms have also to be classified and more rigorously analysed. In [5] algorithms are classified according to two ways: the first, according to volume changes during the computation. In *decreasing volume* algorithms, the number of strands in the test tube decreases as the algorithm progresses; *constant volume* algorithms maintain the volume and *mixed* algorithms are those which fit into neither case. Secondly, an algorithm is *uniform* if for every test tube, any two different strands have the same number of copies. This classification is important with respect to error correction as decreasing volume and uniform algorithms are more resistant to errors [4][5].

One of the main appeals of DNA computing is its extraordinary energy efficiency and information density. A joule can sustain approximately 2×10^{19} ligation operations. Traditional supercomputers execute on the order of $10^9 - 10^{10}$ operations per joule. Not only DNA but also molecular computation is gaining ground. Erik Winfree and others have synthesised self-organising molecular complexes. Recently Shapiro *et al* published [8] the first prototype of a DNA computer which analyses the levels of RNA and can accordingly regulate gene expression.

Computation is being looked at in new ways. No longer confined to the silicon chip which has been the mainstay of computer evolution in the past half-century, researchers are looking at hitherto unsuspected places, hidden forms of computation that had eluded us so far. In Adleman's words:

> What if computers were ubiquitous and could be found in many forms? Could a liquid computer exist in which interacting molecules perform computations?

References

[1] LM Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, 1994.

[2] Dan Boneh, Christopher Dunworth, Richard J. Lipton, and Jiri Sgall. On the Computational Power of DNA.

[3] Charles H. Bennett and Rolf Landauer. The fundamental physical limits of computation. *Scientific American*, 1985.

[4] D. Boneh and R. J. Lipton. Making DNA computers error resistant. CS TR 491 95, Princeton University, 1995.

[5] D. Boneh, R. J. Lipton, and J. Sgall. Error resistant and uniform DNA computers.

[6] RJ Lipton. DNA solution of hard computational problems. *Science*, 268(5210):542–545, 1995.

[7] Wikipedia. Polymerase chain reaction — wikipedia, the free encyclopedia, 2009. [Online; accessed 11-December-2009].

[8] Yaakov Benenson, Binyamin Gil, Uri Ben-Dor, Rivka Adar, and Ehud Shapiro. An autonomous molecular computer for logical control of gene expression. *Nature*, 429(6990):423–429, 27 May 2004.