DNA Computing and Why It Might Be Interesting for a Coding Theorist

Raj K. Kumar, Amir Hesam Salavati
E-mail: raj.kumar@epfl.ch,hesam.salavati@epfl.ch

Supervisor: Prof. Amin Shokrollahi
E-mail: amin.shokrollahi@epfl.ch

Algorithmics Laboratory (ALGO)
Ecole Polytechnique Federale de Lausanne (EPFL)

June 3, 2011

1 Introduction

In this short report, we briefly introduce DNA computing and its advantages. We will also give some motivations about why those who work on coding theory might be interested in DNA computing and discuss a few papers that address some of the interesting related problems. Finally, a number of possible topics for future works are suggested.

The interested reader is referred to [3] for a comprehensive introduction to DNA computing, what has been achieved and the challenges remaining. A number of examples that employ DNA computing to solve some well-known computational problems are also listed in the same paper. For the sake of completeness, it must be noted that DNA computing is also closely related to DNA self-assembly which has a lot of applications in constructing nano-structures and electrical circuits (this field is usually called *DNA origami*). The interested reader is referred to [12] for further details.

2 What is DNA Computing Anyway?

DNA computing is the *art* of using DNA strands to perform computational tasks. The field was triggered by the work of Adleman who used DNA molecules to solve the Hamiltonian path problem for a graph with seven vertices [1]. Later, it was shown that DNA strands are capable of doing *universal computation* [3].

To see how on earth it is possible to compute with DNA molecules, we have to rapidly go over basics of molecular biology. If you are not interested in biochemical stuff, you might skip the next section. Just keep in mind that it is possible to design some DNA molecules, put them in a test tube, add water (may be shake it a little bit too!) and after a few moments the answer to the problem lies somewhere in the tube!

2.1 Biochemical Principles of DNA Computing

DNA molecules can be considered as a sequence of letters over a quaternary alphabet denoted by $S = \{A, C, G, T\}$, in which each letter represents a nucleotide. Among these four letters, A is said to be the complement of T and C to be the complement of G.

Now the basic principle is that each nucleotide binds to its complement when they are close enough to each other. Hence, when a single stranded DNA molecule D_1 comes close to its complement $\bar{D}_1$, i.e. a strand in which every letter in D_1 is replaced by its complement, then these two strands bind and form the double helix shape that we are all familiar with.

Another important point is to note that single stranded DNA are suitable for computing, as they can bind to each other, while double stranded ones are not appropriate, as they form stable structures and will not bind to anything else.

So how does this pairing help us to compute? The formal way to answer this question is to compare DNA and DNA manipulating mechanisms with a Turing machine as they are very similar. However, may be a better and more intuitive way of answering the question is to review Adleman's steps to solve the Hamiltonian Path Problem (HPP) using DNA molecules [1]. In short, we have a directed graph of n cities, two of which are given as the start and end points of a tour. The question is to see if there exists a path that connects these two cities and goes through all other cities exactly once. It is well-known that HPP is an NP-complete problem.

However, since DNA computing has an inherent large degree of parallelism, it might be employed to find a solution to HPP much faster than its *in silico* rivals. The idea that Adleman used was to first generate a DNA sequence for each city in the problem. This sequence has two parts which are called the *first name* and the *last name* of the city With a small modification of Adleman's approach and to make life easier, here we assume the last name is the complement of first name. Then, we cam produce a sequence for all the edges between two cities such that the first part of the sequence is the first name of the departure city and the second part is the last name of the destination. For instance, if we represent Lausanne by $ACGT-TGCA$, then $ACGT$ and $TGCA$ are the first and last names of Lausanne, respectively. Thus, if Geneva is represented by $CCTA-GGAT$, then the trip from Lausanne to Geneva is encoded by $ACGT-GGAT$. Likewise, the return trip from Geneva to Lausanne is indicated by $CCTA-TGCA$.

Now if we put all these paths in a test tube and add water(!), since two complement DNA *subsequences* bound, there would a bound between two edges that has a common city in their destination and source city, respectively. More specifically, after binding we will have a DNA strand with three parts: the first part corresponds to the first name of the departure city and is a single stranded DNA. The second part is a double helix connected to the first part and corresponds to the bound between the first and the last name of the intermediate city. Finally, the third part which is also a single stranded DNA, corresponds to the last name of the current arrival city. Figure 1 illustrates an example for the trip from Lausanne to Geneva and then to Paris, where the DNA sequence for Paris is given by $CTCT - GAGA$. Such a structure has two singled stranded parts at the beginning an end parts so it can bind to other edges and grow in size.

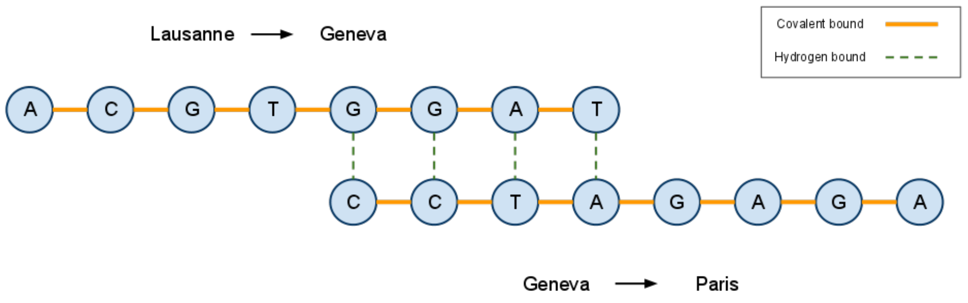

Figure 1: A DNA sequence that encodes the trip from Lausanne to Geneva and then to Paris.

Therefore, the set of these connected edges will grow by adding other possible edges one at a time. After *a few seconds*, we will have the solution to the HPP in our hands, well of course together with all other solutions which we have to get rid of somehow. Hence, all remains to do is to filter the unwanted solutions and check to see if the correct solution actually lies in the test tube. Long story short, there are a number of ways to accomplish this goal and Adleman extracted the correct solution to the HPP, which was a DNA strand containing the correct set of edges.

2.2 Advantages of DNA Computing

DNA computing has a number of key advantages, some of which are listed below:

- Speed and parallelism: The major advantage of DNA computing is its essential parallelism because all possible combinations develop in parallel via chemical reactions.

- Energy efficiency: in principle, one joule is sufficient for 10^{19} molecular operations while the same amount of energy is only enough for 10^{9} computer operations [1].

- Information density: the amount of information packed in one gram of DNA is five times better than the magnetic storage densities of the time Adleman invented DNA computing [1].

Although DNA computing has several advantages, it must be noted that so far, we have been able to construct only few working examples and for small problems solely. What works on a graph of seven vertices in DNA computers encounters serious problems for problems with 100 vertices. For instance, it is estimated that in order to extend Adleman's approach to solve Hamiltonian path problem over a graph of 70 vertices, we will need 10^{25} kilograms of DNA [3]!

3 How DNA Computing Is Related to Us?

OK! Solving HPP in a test tube might seem fascinating but what is in it for those who work on coding theory, communication systems and computer science? The answer to this question comes in two parts. Obviously, one can benefit from computational powers of DNA computers (assuming they come to computer stores some time in near future) to solve some difficult problems in an energy-efficient fast manner. However, for a *researcher* the more important question is how we can apply our knowledge to improve various aspects of DNA computing. This section aims to briefly answer this question by reviewing a number of current problems in DNA computing that might benefit from applying methods related to coding theory and computer science.

3.1 Using Coding Techniques to Overcome the Issue of Noise in DNA Computing

For any viable application, DNA computing must be highly reliable. This is in contrast to the unpredictable nature of the environment DNA computation is performed, i.e. in a solution and by using chemical reactions. There are many sources of error in such an environment but the two most important ones are: undesired reactions between two DNA strands, which results in unwanted helical DNA molecules in the output, and DNA strands reacting with themselves and folding in $3D$ forms which will make that particular strand useless for the rest of computation.

It is well known that two DNA strands that have a small distance from each other will most probably react. Hence, we must develop techniques to overcome this and other sources of unreliability. In this regard, the situation is similar to transmitting data over noisy channels where we would like to eliminate noise. As a result, coding algorithms might be used to design DNA strands with good distance properties to avoid unwanted reactions.

The main design criteria for *good* DNA strands are:

- Large minimum distance between strands themselves and between strands and their complements: this property lowers the probability of unwanted inter-strands reactions.
- Small free energy for the strands: This property ensures that a strand is a stable structure and will not fall back upon itself.
- Small length: this property translates into ease of use and cost efficiency.
- Relatively constant GC-content: GC-content is the number of G and C nucleotides in each sequence and is related to thermodynamic properties of the system. Having a constant GC-content among all strands ensures that all chemical reactions proceed with similar paces.

In what follows, we give an overview of what has been done with respect to using coding theoretical approaches to design proper DNA strands, also known as DNA codewords.

Milenkovic et al. [6] have proposed three novel ways based on well-known coding techniques to design DNA molecules with the desired properties:

- *Cyclic codes*: The authors employ cyclic codes over $GF(4)$ and map them to DNA sequences by assigning each of the 4 coding symbols to one of the four nucleotides. Cyclic codes are particularly interesting in the context of DNA computing because they make the testing procedure for $3D$ structure formation easier.

- *Generalized Hadamard matrices*: Another method that the authors propose for picking codewords is to construct an $n \times n$ Hadamard matrix in a way that rows are cyclic shifts of each other. Furthermore, due to the special construction method, the minimum distance is the same between all rows. Now, each row is mapped to a DNA sequence and the result is a DNA code with equal distances among all the codewords.

- *Binary mapping* The authors fist map the nucleotides to binary sequences in the following way: $A \rightarrow 00, T \rightarrow 01, C \rightarrow 10, G \rightarrow 11$. Then a binary code is developed in a way that the number of G and C nucleotides in the final DNA sequences is constant for all codewords.

In [9] the authors apply error correcting codes to decrease the error rate in DNA computing. Using the proposed error correction technique with vector quantization methods, they also propose a new way to implement DNA databases with associative search queries (similar to their neural networks counterparts). Their idea of implementing an error correcting method in the context of molecular biology is very interesting: First, they generate some codewords over an alphabet of 4 letters using any appropriate coding method. Then, they synthesize DNA strands that are composed of two parts with exactly the same size: the first part is the original codeword and the second one is a corrupted version of the codeword with some noise probability (the same as the one that occur during the computation process). If for each codeword we produce many such DNA strands, we get many versions of the channel output in the second part of the strand. In the process of error correction, we are given a probe and would like to find the codeword it represents. Since we have many corrupted versions of the codeword, chances are that one of them matches the probe. In that case, they react and form a double helix with the original codeword attached to it as a single strand. We can then extract the double helix part and retrieve the single strand.

Aside error correcting codes, other techniques have also been used to design proper DNA codewords. For instance, in [7], the authors generate all possible DNA sequences of length 20 (both *in vitro* and via simulations) and then filter the inappropriate sequences out. What remains are proper DNA codewords. Filtering out is done by putting all generated sequences in a tube and extracting all those that bind together. Although the suggested method is quite accurate, it has a major drawback as it is not scalable and is very time consuming. For that reason and the insight we obtain from analytically designing codewords, one might still prefer coding theoretical methods to find appropriate DNA codewords.

3.2 DNA Computing and Dynamic Biochemical Networks

Implementing a dynamic network of several nodes have also attracted several researchers. In brief, we are interested in a network of n nodes that are capable of accomplishing certain tasks by coordinating among themselves. Codes on graphs, neural networks and Gene Regulatory Networks (GRN) are all specific examples of such a structure. Assuming such a network can be implemented, one can imagine using codes on graphs to do error correction for DNA strands *in vitro*, instead of just designing *error avoiding* DNA sequences in advance that minimize the probability of error but once it has occurred, they can not correct it.

Another possibility is to implement a neural network using DNA strands. Then, the power of artificial neural networks might be employed to accomplish certain tasks in DNA computing. The main motivation here is that neural networks are extremely robust and fault tolerant in their computations, two properties that are highly welcomed in noisy environments of DNA computing.

As a result, a number of researchers have tried different approaches to mimic the operation of neural networks with DNA strands. Mills et al. have made an attempt to accomplish this goal by proposing a novel mechanism to develop a neural network in which the axons and neurons are replaced by the diffusion and molecular recognition of DNA [4].

In a very interesting paper, Kim, Hopfield and Winfree have developed another framework to implement any neural network using DNA molecules [2]. They key element is a *DNA switch* which is a normal single stranded DNA molecule that can be in either of the two states ON of OFF. Each such DNA switch has several input/output terminals by means of which it affect other DNA switches (by releasing RNA molecules if you are curious to know how). In neuroscience parlance, DNA switches act as neurons and the concentration of RNA strands act as spikes. Similar to neural networks then, we will have a network of DNA switches that can collaborate with each other to perform computational tasks. The authors have provided several examples of such computational tasks.

The same research team has also proposed another approach to design biochemical neural networks, but based on direct chemical reactions among strands instead of DNA switches affecting other switches indirectly and by releasing RNA molecules [8]. The main advantage of the latter approach is its scalability to much larger networks. However, there is serious drawback and that's the fact that the state of neurons can not be updated dynamically and once DNA switches decide their states, they freeze. Therefore, the former method is suitable for small dynamic neural networks while the latter is more appropriate for larger scale feed forward networks. Another important point is that all the mentioned methods only talk about how we can have a general DNA neural network. The issue of fault tolerance remains an open issue.

In [9] the authors use error correcting algorithm to implement an association mechanism similar to neural associative memory. In a nutshell, the problem to solve is that we have a huge database formed as encoded DNA strands. We would like to retrieve the closest stored vector to the query pattern. Similar to other DNA-based systems, we must have appropriate DNA codewords to represent stored data patterns. The Authors have proposed a method for associative searches when all we are required is to return all stored vectors that are in distance d from the query vector. The idea is to cluster patterns that are in distance d from each other, assign them a center codeword to represent the cluster and then do a normal DNA-based error correction using the query vector and the encoded version of the center codewords. The authors have also extend the suggested approach to the case that we are required to return a vector that is in distance *at most* d from the query vector.

While one has to artificially construct biochemical neural networks, gene regulator networks are capable of accomplishing similar tasks and they are already working in living species. So one might think of harnessing the power of GRNs to perform computational tasks. This is specially interesting as some of the models used to analyze the dynamical behavior of GRNs are based on neural networks or codes on graphs [5]. Consequently, we might think about constructing artificial gene regulatory networks, which are usually called transcription circuits, or manipulate the already available GRNS. However, not many works are yet done with this respect and those that have followed this path only work for small networks [2]

4 What Is Allowed in DNA Computing?

So computing with DNA is possible and there are a number of problems that might be interesting for researchers in coding theory. But what are the operations that one is permitted to use in order to apply coding techniques to improve the performance of DNA computing? Although the set of currently allowed operations may vary and one might also think of *inventing* appropriate procedures, the widely-used set of biochemical operations is as follows [3]:

1. *Annealing (DNA pairing)*: if a DNA sequence meets its complement, they pair up by developing hydrogen bounds. This is formally called *annealing.*

2. *Melting*: this is the inverse of annealing. Note that during melting, a double stranded DNA becomes two single stranded ones as the temperature to break the bound between two strands is far more lower that the one to break bounds among nucleotides of a single strands.

3. *Completing a DNA sequence (Polymerase)*: If two DNA sequences bound, the DNA polymerase enzymes may be used to complete the shorter sequence to have the same length as the longer sequence.

4. *Concatenating two strands (Ligate)*: Ligation concatenates two single stranded DNA strands together by developing covalent bounds (which is much stronger than the hydrogen bound). Note that while DNA pairing results in a double stranded helix, ligation results is another single stranded DNA.

5. *Cutting strands (Nuclease)*: This process cuts DNA strands in half whenever a specified subsequence is found by nuclease enzymes. Note that these sequences vary for different enzymes. As for 1993, there were over 2300 different known nuclease enzymes sensitive to more than 200 different subsequences.

6. *Sort by length (Gel electrophoresis)*: This process sorts DNA strands by length. Then we can extract DNA strands with correct length to be read using magnetic bids and other cumbersome reading procedures.

7. *DNA synthesis*: This process gets a DNA sequence in paper and produces the corresponding DNA molecule. Destroy: subsets of strands can be destroyed or digested by employing special enzymes.

And here are some *higher order operations*:

9. *Amplifying a subset of strands (PCR)*: this procedure generates multiple copies of a subset of strands in the test tube and works as follows. Given a strand with a specific start and end subsequences (called primers), polymerase enzymes copy everything that lies between these sequences (called the template).

10. *Separate by subsequence*: using specific approaches (such as employing magnetic beads), one can divide the test tube into two parts: the part that contains all strands with a given subsequence inside them and the other part with the rest of the strands. [me: but in general, this process is very difficult to do].

11. *Append*: this process attaches a small subsequence to the end of *all strands in the tube.*

12. *Mark*: this process attaches a small subsequence to the end of *a subset of strands.*

13. *Unmark*: this is just opposite of marking and removes the tags attached to the given strands.

So if someone is interested in applying various methods in computer science of coding theory to DNA computing, he must only use the set of operations given above. The interested reader is referred to [3] for a list of applications that solve some computer scientific problems using the operations above.

5 Possible Subjects for Future Works

In this section, we describe a couple of suggestions on the topic that might be interesting to pursue for someone with background in coding theory and computer science.

5.1 Coding Theory to Design Proper DNA Codewords

The first idea, which is in line with [6] and [9] and many other similar works, is to use coding techniques to design DNA strands that first of all minimize probability of an error happening at all (by having large enough minimum distances) and secondly, in case of errors, can correct them to some extent. The main challenge then is to design proper codes based on the desired criteria in DNA computing, i.e. large minimum distance, constant number of G and C nucleotides in codewords, etc.

5.2 Neural Coding Theory to Perform Iterative Error Correction

The second idea, which is more related to our current research on neural networks, is to employ the framework developed in [2], [4] or [8] to design *iterative* error correcting algorithms for DNA computing. A small modification makes the same framework also appropriate for performing various computational tasks or acting as a means to battle some diseases that affect the normal operation of a cell.

Moreover, the idea about designing DNA neural networks is closely related to Gene Regulatory Networks (GRN) and how they evolve over time. We discussed possible applications of coding techniques in gene regulatory networks in detail in [10] and [11]. There, the idea was to see if GRNs use coding techniques to overcome the issue of noise and if so, what type of coding algorithm is used. However, as correctly argued previously by several faculty members, it is not necessarily the case that GRNs use error correcting codes as we know them. Nevertheless, in the realm of DNA computing, since we are synthesizing DNA molecules ourselves, we can use any coding technique that we want.

References

[1] L. M. Adleman, "Computing with DNA", Scientific American, Scientific American, Vol. 279, No. 8, 1998, pp. 34-41.

[2] J. Kim, J. J. Hopfield, E. Winfree, "Neural network computation by in vitro transcriptional circuits", Advances in Neural Information Processing Systems (NIPS), Vol. 17, 2004, pp. 681-688.

[3] C. C. Maley, "DNA computations: theory, practice, and prospects", J. Evolutionary Computation, Vol. 6, No. 3, 1998, pp. 201-229.

[4] A. P. Mills Jr., M. Turberfield, A. J. Turberfield, B. Yurke, P. M. Platzman, "Experimental Aspects of DNA Neural Networks", Soft Computing - A Fusion of foundations, Methodologies and Applications, Vol. 5, No. 1, 2001, pp. 10-18.

[5] O. Milenkovic, N. Kashyap, B. Vasic, "On DNA Computers Controlling Gene Expression Levels", Proc. IEEE Conference on Decision and Control, and the European Control Conference, 2005, pp. 1770-1775.

[6] O. Milenkovic, N. Kashyap, "On the Design of Codes for DNA Computing" Lecture Notes in Computer Science, Vol. 3969, 2006, pp. 100-119.

[7] A. J. Neel, M. H. Garzon, "DNA-based memories: a survey", Studies in Computational Intelligence, Vol. 113, 2008, pp. 259-275.

[8] L. Qian, D. Soloveichik, E. winfree, "Efficient turing-universal computation with DNA polymers", Lecture Notes in Computer Science, Vol. 6518, 2011, pp. 123-140.

[9] J. H. Reif, T. H. LaBean, "Computationally Inspired Biotechnologies: Improved DNA Synthesis and Associative Search Using Error-Correcting Codes and Vector-Quantization", Lecture Notes in Computer Science, Vol. 2054, 2001, pp. 145-172.

[10] Amir Hesam Salavati, Applications of Coding Theory in Biological Systems, PhD candidacy exam report, Ecole Polytechnique Federale de Lausanne (EPFL), June 2010

[11] Amir Hesam Salavati, Applications of Coding Theory in Molecular Biology: An Overview Technical Report, Ecole Polytechnique Federale de Lausanne (EPFL), March 2010

[12] A. H. Salavati, "A summary of the papers presented at the workshop *Statistical Mechanics and Computation of DNA self-assembly*", Technical report, EPFL, June 2011.