American Journal of Nanomaterials, 2015, Vol. 3, No. 1, 1-14
Available online at http://pubs.sciepub.com/ajn/3/1/1

DOI:10.12691/ajn-3-1-1

A Proper Approach on DNA Based Computer

Shyam Nandan Kumar*

M.Tech-Computer Science and Engineering, Lakshmi Narain College of Technology-Indore (RGPV, Bhopal), MP, India
*Corresponding author: shyamnandan.mec@gmail.com

Received January 08, 2015; Revised February 26, 2015; Accepted March 01, 2015

Abstract Computer applications have become an essential part of our daily lives, and their use is flourishing day by day. In Conventional Computer, there are lots of limitations like: Operational Speed, Power Consumption, Parallel Processing, Hardware Size, Data Storage Limitation, etc. To emphasize the problem, the paper provides A Proper Approach on DNA Based Computer. DNA Based Computer is a nano-computer that uses DNA (DeoxyriboNucleic Acids) to store information and perform complex calculations with low power consumption in a matter of seconds. The paper proposed Design Methodology of DNA Computer. Also various features of DNA Computer and operation of its processing units are reviewed. An overview on Adleman Experiment is also included in the paper.

Keywords: *DNA computing, biomolecular-computer, design methodology, hamiltonian path problem*

Cite This Article: Shyam Nandan Kumar, "A Proper Approach on DNA Based Computer." *American Journal of Nanomaterials*, vol. 3, no. 1 (2015): 1-14. doi: 10.12691/ajn-3-1-1.

1. Introduction

In recent decades the word ''computer'' has become synonymous with an electronic computing machine due to the overwhelming success of this particular paradigm. Indeed, as we build faster and faster electronic computers, we are beginning to reach physical limits, beyond which our current technology cannot venture. Due to the limitation of traditional silicon based computer, user requirement is not fulfilling. DNA computing is a brand new research area which receives more and more attentions from both biologists and computer scientists.

DNA computing is a form of computing which uses DNA, biochemistry and molecular biology, instead of the traditional silicon-based computer technologies. This field was initially developed by Leonard Adleman of the University of Southern California, in 1994 [1]. Adleman demonstrated a proof-of-concept use of DNA as a form of computation which solved the seven-point Hamiltonian path problem(the traveling salesman problem), whose solution required finding a path from start to end going through all the points (cities) only once. As the number of cities increases, the problem becomes more difficult until its solution is beyond analytical analysis altogether, at which point it requires brute force search methods. TSPs with a large number of cities quickly become computationally expensive, making them impractical to solve on even the latest super-computer. Adleman's demonstration only involves seven cities, making it in some sense a trivial problem that can easily be solved by inspection. Nevertheless, his work is significant for a number of reasons. Since the initial Adleman experiments, advances have been made and various Turing machines have been proven to be constructible [2,3]. Adelman's experiment is regarded as the first example of true nanotechnology.

DNA computing is fundamentally similar to parallel computing in that it takes advantage of the many different molecules of DNA to try many different possibilities at once [4]. In the cell, DNA is modified biochemically by a variety of enzymes, which are tiny protein machines that read and process DNA according to nature's design. There is a wide variety and number of these "operational" proteins, which manipulate DNA on the molecular level. For example, there are enzymes that cut DNA and enzymes that paste it back together. Other enzymes function as copiers and others as repair units. Molecular biology, Biochemistry, and Biotechnology have developed techniques that allow us to perform many of these cellular functions in the test tube. It's this cellular machinery, along with some synthetic chemistry, that makes up the palette of operations available for computation. Just like a CPU has a basic suite of operations like addition, bit-shifting, logical operators (AND, OR, NOT NOR), etc. that allow it to perform even the most complex calculations, DNA has cutting, copying, pasting, repairing, and many others. And note that in the test tube; enzymes do not function sequentially, working on one DNA at a time. Rather, many copies of the enzyme can work on many DNA molecules simultaneously. This is the power of DNA computing, that it can work in a massively parallel fashion.

In 2002, researchers from the Weizmann Institute of Science in Rehovot, Israel, unveiled a programmable molecular computing machine composed of enzymes and DNA molecules instead of silicon microchips [5]. In January 2013, researchers were able to store a JPEG photograph, a set of Shakespearean sonnets, and an audio file of Martin Luther King, Jr.'s speech "I Have a Dream" on DNA digital data storage [6].

2. Conventional vs. DNA Computing

Unlike conventional computers, DNA computers perform calculations parallel to other calculations. Conventional computers operate linearly, taking on tasks one at a time. It is parallel computing that allows DNA to solve complex mathematical problems in hours, whereas it might take electrical computers hundreds of years to complete them. Transistor-based computers typically handle operations in a sequential manner. Increasing performance of silicon computing means faster clock cycles (and larger data paths), where the emphasis is on the speed of the CPU and not on the size of the memory. Better performance cannot be achieved by only doubling the clock speed or doubling the RAM of the system. While in DNA based computing, performance improves due to memory capacity and parallel processing.

Molecular computing is massively parallel computing, by taking advantage of the computational power of molecules, usually biological molecules. It is another way to surmount certain limitations of traditional silicon-based computing. We can remove size problem of traditional computers by making the processors as small as a molecules. This technique makes circuits thousand times smaller than traditional silicon based computers. Cells (living parts) of organisms are ingredients for computation. These provide the basic idea of computing, as these tiny parts are complete machines and perform all the processing for the organisms activities. It can overcome on two major limitations of silicon-based traditional computers: storage capacity and processing speed. In this technique, 560enzymes and amino acids of DNA are used to solve particular problem.

DNA performs ***parallel operations*** while conventional, silicon-based computers typically handle operations sequentially. A modern CPU basically repeats the same “fetch and execute (fetches an instruction and the appropriate data from main memory and executes it) cycle” over and over again. This process is repeated many, many times in a row, and really, really fast. Typically, increasing performance of silicon computing means faster clock cycles, placing emphasis on the speed of the CPU and not on the size of the memory. Oppositely, the power of DNA computing comes from its memory capacity and parallel processing. In other words, DNA loses its appeal if forced to behave sequentially.

DNA's key advantage is that it will make computers smaller than any computer that has come before them, while at the same time holding more data. One pound of DNA has the capacity to store more information than all the electronic computers ever built; and the computing power of a teardrop-sized DNA computer, using the DNA logic gates, will be more powerful than the world's most powerful supercomputer. More than 10 trillion DNA molecules can fit into an area no larger than 1 cubic centimeter (0.06 cubic inches). With this small amount of DNA, a computer would be able to hold 10 terabytes of data, and perform 10 trillion calculations at a time. By adding more DNA, more calculations could be performed. DNA computers have the potential to take computing to new levels, picking up where Moore's Law leaves off. There are several advantages to using DNA instead of silicon:

- DNA computers are many times smaller than today's computers.
- The large supply of DNA makes it a cheap resource.
- As long as there are cellular organisms, there will always be a supply of DNA.
- Unlike the toxic materials used to make traditional microprocessors, DNA biochips can be made cleanly.
- Based on parallel data processing
- DNA Computer has Big Data storage capacity.
- Low power consumption

For certain specialized problems, DNA computers are faster and smaller than any other computer built so far. Furthermore, particular mathematical computations have been demonstrated to work on a DNA computer. As an example, DNA molecules have been utilized to tackle the assignment problem [7].

With a conventional computer, one method would be to set up a search tree, measure each complete branch sequentially, and keep the shortest one. A very laborious method would be to generate all possible paths and then search the entire list! With this algorithm, generating the entire list of routes for a 20-city problem could theoretically take 45 million GBytes of memory. This would take a 100 MIPS (million instructions per second) computer two years just to generate all paths (assuming one instruction cycle to generate each city in every path). However, using DNA computing, this method becomes feasible because 10^{15} is just a nanomole of material, a relatively small number for biochemistry. Also, routes no longer have to be searched through sequentially: operations can be done all in parallel (Adams).

3. DNA Sequencing

By DNA Sequencing we calculate the precise order of nucleotides within a DNA molecule. DNA is a long polymer made from repeating units called nucleotides [8,9,10]. It includes any method or technology that is used to determine the order of the four bases—adenine(A), guanine(G), cytosine(C), and thymine(T)—in a strand of DNA. Figure 1. shows DNA Helix. The DNA double helix is stabilized primarily by two forces: hydrogen bonds between nucleotides and base-stacking interactions among aromatic nucleobases. DNA is well-suited for biological information storage. The DNA backbone is resistant to cleavage, and both strands of the double-stranded structure store the same biological information. Biological information is replicated as the two strands are separated.

DNA sequencing may be used to determine the sequence of individual genes, larger genetic regions (i.e. clusters of genes or operons), full chromosomes or entire genomes. Sequencing provides the order of individual nucleotides in DNA or RNA (commonly represented as A, C, G, T, and U) isolated from cells of animals, plants, bacteria, archaea, or virtually any other source of genetic information. The advent of rapid DNA sequencing methods has greatly accelerated biological and medical research and discovery. To encode information using DNA, one can choose an encoding scheme mapping the original alphabet onto strings over {A, C, G, T}, and proceed to synthesize the obtained information-encoding strings as DNA single strands. A can chemically bind to an opposing T on another single strand, while C can

similarly bind to G. Bases that can thus bind are called Watson=Crick (W=C) complementary, and two DNA single strands with opposite orientation and with W=C-complementary bases at each position can bind to each other to form a DNA double strand in a process called base pairing. Knowledge of DNA sequences has become indispensable for basic biological research, and in numerous applied fields such as diagnostic, biotechnology, forensic biology, virology and biological systematics. Each purine base is the Watson-Crick complement of a unique pyrimidine base (and vice versa) — adenine and thymine form a complementary pair, as do guanine and cytosine. We describe this using the notation A = T, T = A, C = G, G = C. The chemical ties between the two WC pairs are different — C and G pair through three hydrogen bonds, while A and T pair through two hydrogen bonds

A DNA sequence that totally describes a processor on decoding is bound to be complicated. Such complicated sequences will be difficult to work with during hybridization phase unless they are well formed and ordered. Thus a good encoding scheme must be biologically realistic, yet well ordered. By placing related nodes near each other, we allow Localization of Reactions wherein related characteristics are changed simultaneously. This bio inspired node placement scheme provides a good platform to build simple hybridization rules.

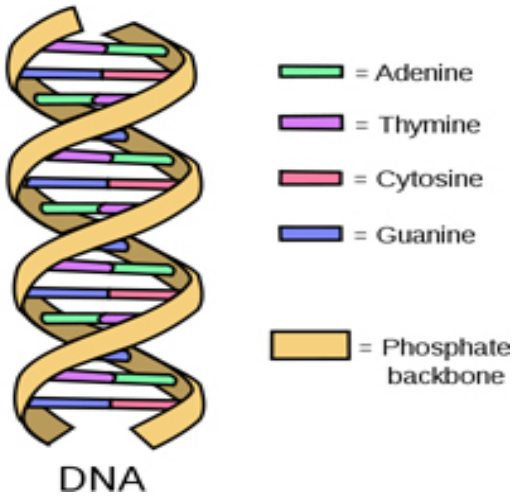

Figure 1. DNA Double Helix

A computation will consists of a succession of bio-operations (Kari, 1997), such as cutting and pasting DNA strands, separating DNA sequences by length, extracting DNA sequences containing a given pattern, or making copies of DNA strands. The DNA strands representing the output of the computation can then be read out and decoded. The rapid speed of sequencing attained with modern DNA sequencing technology has been instrumental in the sequencing of complete DNA sequences, or genomes of numerous types and species of life, including the human genome and other complete DNA sequences of many animal, plants, and microbial species. DNA computing is based on the idea that molecular biology processes can be used to perform arithmetic and logic operations on information encoded as DNA strands.

For a collection of interacting entities, one measure commonly used for assessing the system's property is the free energy. The stability and form of a secondary configuration is usually governed by this energy, the general rule-ofthumb being that a secondary structure minimizes the free energy associated with a DNA sequence. The free energy of a secondary structure is determined by the energy of its constituent pairings, and consequently, its loops. Nevertheless, some simple dynamic programming techniques can be used to approximately determine base pairings in a secondary structure of a oligonucleotide DNA sequence. Among these techniques, the ***Nussinov-Jacobson (NJ)*** folding algorithm is one of the simplest and most widely used schemes.

4. DNA Based Computing Device

DNA Based computing devices can be formed by several well-known ways which have unique feature. Most of these build the basic logic gates (AND, OR, NOT) associated with digital logic from a DNA basis. Some of the different bases include DNAzymes, ***deoxyoligonucleotides***, enzymes, DNA tiling, and polymerase chain reaction. The switches are based on deoxyribozymes that use oligonucleotides as both input and output.

4.1. Enzymes

Many enzymes contain highly active molecule which can be used for DNA based computing component. Enzyme based DNA computers are usually of the form of a simple Turing machine; there is analogous hardware, in the form of an enzyme, and software, in the form of DNA [11]. Benenson, Shapiro and colleagues have demonstrated a DNA computer using the FokI enzyme [12] and expanded on their work by going on to show automata that diagnose and react to prostate cancer: under expression of the genes PPAP2B and GSTP1 and an over expression of PIM1 and HPN [13]. Their automata evaluated the expression of each gene, one gene at a time, and on positive diagnosis then released a single strand DNA molecule (ssDNA) that is an antisense for MDM2. MDM2 is a repressor of protein 53, which itself is a tumor suppressor [14]. On negative diagnosis it was decided to release a suppressor of the positive diagnosis drug instead of doing nothing. A limitation of this implementation is that two separate automata are required, one to administer each drug. The entire process of evaluation until drug release took around an hour to complete. This method also requires transition molecules as well as the FokI enzyme to be present. The requirement for the FokI enzyme limits application in vivo, at least for use in "cells of higher organisms" [15]. It should also be pointed out that the 'software' molecules can be reused in this case.

4.2. DNAzymes

DNAzyme, which is also known as ***deoxyribozyme***, is a Catalytic DNA which catalyzes a reaction when interacting with the appropriate input, such as a matching ***oligonucleotide***. These DNAzymes are used to build logic gates analogous to digital logic in silicon; however, DNAzymes are limited to 1-, 2-, and 3-input gates with no current implementation for evaluating statements in series.

The DNAzyme logic gate changes its structure when it binds to a matching oligonucleotide and the ***fluorogenic***

substrate it is bonded to is cleaved free. While other materials can be used, most models use a fluorescence-based substrate because it is very easy to detect, even at the single molecule limit [16]. The amount of fluorescence can then be measured to tell whether or not a reaction took place. The DNAzyme that changes is then "used," and cannot initiate any more reactions. Because of this, these reactions take place in a device such as a continuous stirred-tank reactor, where old product is removed and new molecules added.

Two commonly used DNAzymes are named E6 and 8-17. These are popular because they allow cleaving of a substrate in any arbitrary location [17]. Stojanovic and MacDonald have used the E6 DNAzymes to build the MAYA I [18] and MAYA II [19] machines, respectively; Stojanovic has also demonstrated logic gates using the 8-17 DNAzyme [20]. While these DNAzymes have been demonstrated to be useful for constructing logic gates, they are limited by the need for a metal cofactor to function, such as Zn^{2+} or Mn^{2+}, and thus are not useful in vivo [16,21].

A design called a stem loop, consisting of a single strand of DNA which has a loop at an end, are a dynamic structure that opens and closes when a piece of DNA bonds to the loop part. This effect has been exploited to create several logic gates. These logic gates have been used to create the computers MAYA I and MAYA II which can play tic-tac-toe to some extent [22].

4.3. Toehold Exchange

DNA computers have also been constructed using the concept of toehold exchange. In this system, an input DNA strand binds to a sticky end, or toehold, on another DNA molecule, which allows it to displace another strand segment from the molecule. This allows the creation of modular logic components such as AND, OR, and NOT gates and signal amplifiers, which can be linked into arbitrarily large computers. This class of DNA computers does not require enzymes or any chemical capability of the DNA [23].

5. Basic Component of DNA-Based Computer

The entire DNA strand for a particular processing unit is split into 2 parts: Active Component and Passive Component. The Active Component participates in the hybridization. This component represents the architectural details as well as the Instruction Set details. It is created in the encoding stage and expanded in the hybridization stage i.e., the basic architectural details such as 128 bit instruction length is added to the DNA string in the encoding stage while finer aspects such as the usage of a Carry Save Adder (CSA) is added to the string during recombination stage. The Passive Component is formed in the mutation stage. It consists of the Finite State Machine (FSM) description, the Netlist and basic Placement details of the processor. The netlist defines the connectivity across various components of an electronic design while placement is the process of assigning exact locations to various components in the chip's core area. The active components of various processors react with each other during hybridization while the passive components are formed during mutation. They do not take part actively during hybridization. But, the FSM details of the two processors are inherited by the offspring based on its instruction set. These inherited FSMs are then used as guidelines to form the actual FSM of the offspring. Some major component of DNA based Computer are:

5.1. DNA Field-Effect Transistor

DNA field-effect transistor (DNAFET) is a field-effect transistor which uses the field-effect due to the partial charges of DNA molecules to function as a biosensor. Biosensor works as an analytical device, used for the detection of an analyte, which combines a biological component with a physicochemical detector. A biosensor typically consists of a bio-recognition component, biotransducer component, and electronic system which include a signal amplifier, processor, and display. Transducers and electronics can be combined, e.g., in CMOS-based microsensor systems [24,25]. Biosensors can be classified by their biotransducer type. The most common types of biotransducers used in biosensors are 1) electrochemical biosensors, 2) optical biosensors, 3) electronic biosensors, 4) piezoelectric biosensors, 5) gravimetric biosensors, 6) pyroelectric biosensors.

An important part in a biosensor is to attach the biological elements (small molecules/protein/cells) to the surface of the sensor (be it metal, polymer or glass). The simplest way is to functionalize the surface in order to coat it with the biological elements. This can be done by polylysine, aminosilane, epoxysilane or nitrocellulose in the case of silicon chips/silica glass. Subsequently the bound biological agent may be for example fixed by Layer by layer deposition of alternatively charged polymer coatings.

In electronics, a transistor controls the flow of electrons along a circuit. Similarly, in biologics, a ***Transcriptor*** (DNA based Transistor) controls the flow of a specific protein, RNA polymerase, as it travels along a strand of DNA. Transcriptor-based gates alone do not constitute a computer, but they are the third and final component of a biological computer that could operate within individual living cells.

Arrays of DNAFETs can be used for detecting single nucleotide polymorphisms (causing many hereditary diseases) and for DNA sequencing. Their main advantage compared to optical detection methods in common use today is that they do not require labeling of molecules. Furthermore they work continuously and (near) real-time. DNAFETs are highly selective since only specific binding modulates charge transport. The structure of DNAFETs is similar to that of MOSFETs with the exception of the gate structure which, in DNAFETs, is replaced by a layer of immobilized ssDNA (single-stranded DNA) molecules which act as surface receptors. When complementary DNA strands hybridize to the receptors, the charge distribution near the surface changes, which in turn modulates current transport through the semiconductor transducer. Transducer converts a signal in one form of energy to another form of energy. A sensor is used to detect a parameter in one form and report it in another form of energy, often an electrical signal.

5.2. DNA-Based Storage System

In DNA Computer, data are stored in base sequence of DNA and these data will be digital. It could be used to record data in a living cell, without using silicon chips like we do in computers. This could allow us to track cell divisions to study cell processes like development, ageing and the changes that occur in cancer. This technology uses artificial DNA made using commercially available oligonucleotide synthesis machines for storage and DNA sequencing machines for retrieval. The researchers used a simple code where bits were mapped one-to-one with bases, which had the shortcoming that it led to long runs of the same base, the sequencing of which is error-prone. This research result showed that besides its other functions, DNA can also be another type of storage medium such as hard drives and magnetic tapes.

This type of storage system is more compact than current magnetic tape or hard drive storage systems due to the data density of the DNA. It also has the capability for longevity, as long as the DNA is held in cold, dry and dark conditions, as is shown by the study of woolly mammoth DNA from up to 60,000 years ago, and for resistance to obsolescence, as DNA is a universal and fundamental data storage mechanism in biology. These features have led to researchers involved in their development to call this method of data storage "apocalypse-proof" because "after a hypothetical global disaster, future generations might eventually find the stores and be able to read them" [24]. It is, however, a slow process, as the DNA needs to be sequenced in order to retrieve the data, and so the method is intended for uses with a low access rate such as long-term archival of large amounts of scientific data [26,27].

DNA is directional, and now a team, led by Jerome Bonnet at Stanford University have found a way to encode information using enzymes that can flip small pieces of DNA between their normal direction, which acts as a 0, and the opposite direction, which acts like a 1. This means it can store one bit of data. And the process used is reliable enough to write and rewrite information again and again.

On August 16, 2012, the journal Science published research by George Church and colleagues at Harvard University, in which DNA was encoded with digital information that included an HTML draft of a 53,400 word book written by the lead researcher, eleven JPG images and one JavaScript program. Multiple copies for redundancy were added and 5.5 petabits can be stored in each cubic millimeter of DNA [28].

To encode information using DNA, one can choose an encoding scheme mapping the original alphabet onto strings over {A, C, G, T}, and proceed to synthesize the obtained information-encoding strings as DNA single strands. A computation will consists of a succession of bio-operations (Kari, 1997), such as cutting and pasting DNA strands, separating DNA sequences by length, extracting DNA sequences containing a given pattern, or making copies of DNA strands. The DNA strands representing the output of the computation can then be read out and decoded.

5.3. DNA Microarray

DNA Microarrays are small, solid supports onto which the sequences from thousands of different genes are immobilized, or attached, at fixed locations. The supports themselves are usually glass microscope slides, the size of two side-by-side pinky fingers, but can also be silicon chips or nylon membranes. The DNA is printed, spotted, or actually synthesized directly onto the support. With the aid of a computer, the amount of mRNA bound to the spots on the microarray is precisely measured, generating a profile of gene expression in the cell. It is important that the gene sequences in a microarray are attached to their support in an orderly or fixed way, because a researcher uses the location of each spot in the array to identify a particular gene sequence. The spots themselves can be DNA, cDNA, or ***oligonucleotides***.

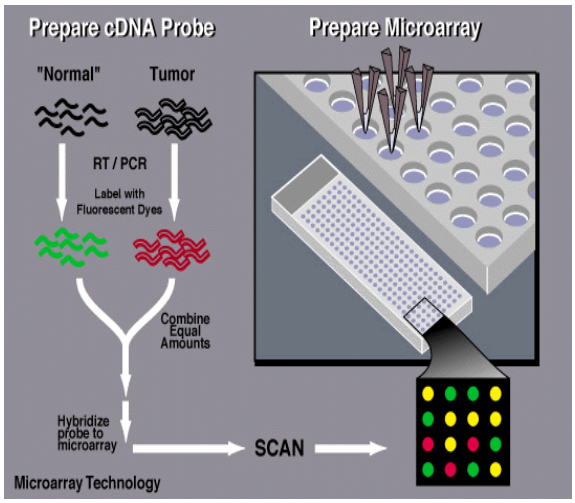

Figure 2. DNA Microarray Generation

When a gene is activated, cellular machinery begins to copy certain segments of that gene. The resulting product is known as messenger RNA (mRNA), which is the body's template for creating proteins. The mRNA produced by the cell is complementary, and therefore will bind to the original portion of the DNA strand from which it was copied. To determine which genes are turned on and which are turned off in a given cell, the messenger RNA molecules present in that cell is first collected. After that we label each mRNA molecule by using a reverse transcriptase enzyme (RT) that generates a complementary cDNA to the mRNA. During that process fluorescent nucleotides are attached to the cDNA. The tumor and the normal samples are labeled with different fluorescent dyes. Next, the labeled cDNAs onto a DNA microarray slide are placed as shown in Figure 2. The labeled cDNAs that represent mRNAs in the cell will then ***hybridize*** – or bind – to their synthetic complementary DNAs attached on the microarray slide, leaving its fluorescent tag. A special scanner is used to measure the fluorescent intensity for each spot/areas on the microarray slide.

There are two major application forms for the DNA microarray technology:

1. Identification of sequence (gene / gene mutation)
2. Determination of expression level (abundance) of genes of one sample or comparing gene transcription in two or more different kinds of cells

6. The Adleman Experiment

There is no better way to understand how something works than by going through an example step by step. So

let's solve our own directed Hamiltonian Path problem, using the DNA methods demonstrated by Adleman. The concepts are the same but the example has been simplified to make it easier to follow and present.

Suppose that a person 'X' lives in Los Angeles (LA), and he want to visit four cities: Dallas, Chicago, Miami, and New York (NY), with New York being the final destination as shown in Figure 3. The airline he takes, has a specific set of connecting flights that restrict which routes he can take (i.e. there is a flight from Los Angeles to Chicago, but no flight from Miami to Chicago). What should his itinerary be if he wants to visit each city only once?

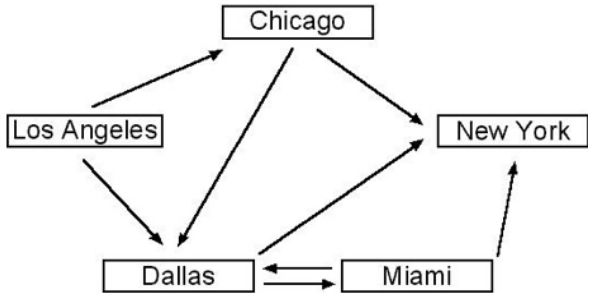

Figure 3. Hamiltonian path problem

It should take you only a moment to see that there is only one route. Starting from Los Angeles you need to fly to Chicago, Dallas, Miami and then to New York. Any other choice of cities will force you to miss a destination, visit a city twice, or not make it to New York. For this example you obviously don't need the help of a computer to find a solution. For six, seven, or even eight cities, the problem is still manageable. However, as the number of cities increases, the problem quickly gets out of hand. Assuming a random distribution of connecting routes, the number of itineraries you need to check increases exponentially. Pretty soon you will run out of pen and paper listing all the possible routes, and it becomes a problem for a computer.

The method Adleman used to solve this problem is basically the shotgun approach mentioned previously. He first generated all the possible itineraries and then selected the correct itinerary. This is the advantage of DNA. It's small and there are combinatorial techniques that can quickly generate many different data strings. Since the enzymes work on many DNA molecules at once, the selection process is massively parallel. Specifically, the method based on Adleman's experiment would be as follows:

- Generate all possible routes.
- Select itineraries that start with the proper city and end with the final city.
- Select itineraries with the correct number of cities.
- Select itineraries that contain each city only once.

All of the above steps can be accomplished with standard molecular biology techniques.

6.1. Technique for Generating Routes

Strategy: Encode city names in short DNA sequences. Encode itineraries by connecting the city sequences for which routes exist.

DNA can simply be treated as a string of data. For example, each city can be represented by a "word" of six bases:

Table 1. Code for Generating Routes

City Name	DNA Sequences
Los Angeles	GCTACG
Chicago	CTAGTA
Dallas	TCGTAC
Miami	CTACGG
New York	ATGCCG

The entire itinerary can be encoded by simply stringing together these DNA sequences that represent specific cities. For example, the route from L.A -> Chicago -> Dallas -> Miami -> New York would simply be GCTACGCTAGTATCGTACCTACGGATGCCG, or equivalently it could be represented in double stranded form with its complement sequence.

Synthesizing short single stranded DNA is now a routine process, so encoding the city names is straightforward. The molecules can be made by a machine called a DNA synthesizer or even custom ordered from a third party. Itineraries can then be produced from the city encodings by linking them together in proper order. To accomplish this you can take advantage of the fact that DNA hybridizes with its complimentary sequence. For example, you can encode the routes between cities by encoding the compliment of the second half (last three letters) of the departure city and the first half (first three letters) of the arrival city. For example the route between Miami (CTACGG) and NY (ATGCCG) can be made by taking the second half of the coding for Miami (CGG) and the first half of the coding for NY (ATG). This gives CGGATG. By taking the complement of this you get, GCCTAC, which not only uniquely represents the route from Miami to NY, but will connect the DNA representing Miami and NY by hybridizing itself to the second half of the code representing Miami (...CGG) and the first half of the code representing NY (ATG…). Figure 4. shows the idea of route encoding using DNA Sequence.

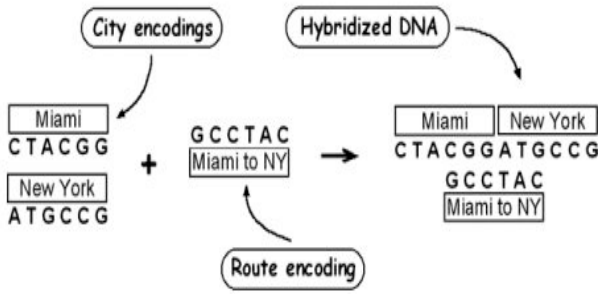

Figure 4. Route Generation in the form of DNA Sequence

Random itineraries can be made by mixing city encodings with the route encodings as shown in Figure 5. Finally, the DNA strands can be connected together by an enzyme called ligase. What we are left with are strands of DNA representing itineraries with a random number of cities and random set of routes. For example:

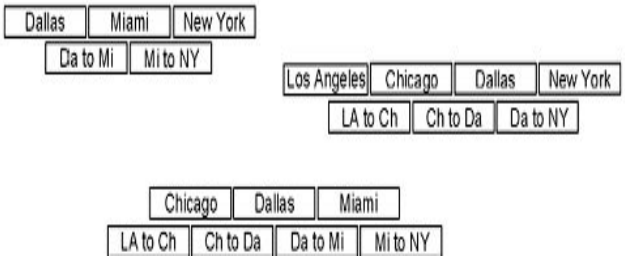

Figure 5. Hybrid DNA Sequence Encoding for route generation

We can be confident that we have all possible combinations including the correct one by using an excess of DNA encodings, say 10^{13} copies of each city and each route between cities. Remember DNA is a highly compact data format, so numbers are on our side.

6.2. Itineraries Selection: Start and End with Correct Cities

Strategy: Selectively copy and amplify only the section of the DNA that starts with LA and ends with NY by using the Polymerase Chain Reaction.

After generating the routes, we now have a test tube full of various lengths of DNA that encode possible routes between cities. What we want are routes that start with LA and end with NY. To accomplish this we can use a technique called Polymerase Chain Reaction (PCR), which allows you to produce many copies of a specific sequence of DNA. PCR is an iterative process that cycle through a series of copying events using an enzyme called polymerase. Polymerase will copy a section of single stranded DNA starting at the position of a primer, a short piece of DNA complimentary to one end of a section of the DNA that you're interested in. By selecting primers that flank the section of DNA you want to amplify, the polymerase preferentially amplifies the DNA between these primers, doubling the amount of DNA containing this sequence. After many iterations of PCR, the DNA you're working on is amplified exponentially. So to selectively amplify the itineraries that start and stop with our cities of interest, we use primers that are complimentary to LA and NY. What we end up with after PCR is a test tube full of double stranded DNA of various lengths, encoding itineraries that start with LA and end with NY.

6.3. Itineraries Selection: Based on Correct Number of Cities

Strategy: Sort the DNA by length and select the DNA whose length corresponds to 5 cities.

Our test tube is now filled with DNA encoded itineraries that start with LA and end with NY, where the number of cities in between LA and NY varies. We now want to select those itineraries that are five cities long. To accomplish this we can use a technique called Gel Electrophoresis as shown in Figure 6, which is a common procedure used to resolve the size of DNA. The basic principle behind Gel Electrophoresis is to force DNA through a gel matrix by using an electric field. DNA is a negatively charged molecule under most conditions, so if placed in an electric field it will be attracted to the positive potential. However since the charge density of DNA is constant (charge per length) long pieces of DNA move as fast as short pieces when suspended in a fluid. This is why you use a gel matrix. The gel is made up of a polymer that forms a meshwork of linked strands. The DNA now is forced to thread its way through the tiny spaces between these strands, which slows down the DNA at different rates depending on its length. What we typically end up with after running a gel is a series of DNA bands, with each band corresponding to a certain length. We can then simply cut out the band of interest to isolate DNA of a specific length. Since we know that each city is encoded with 6 base pairs of DNA, knowing the length of the itinerary gives us the number of cities. In this case we would isolate the DNA that was 30 base pairs long (5 cities times 6 base pairs).

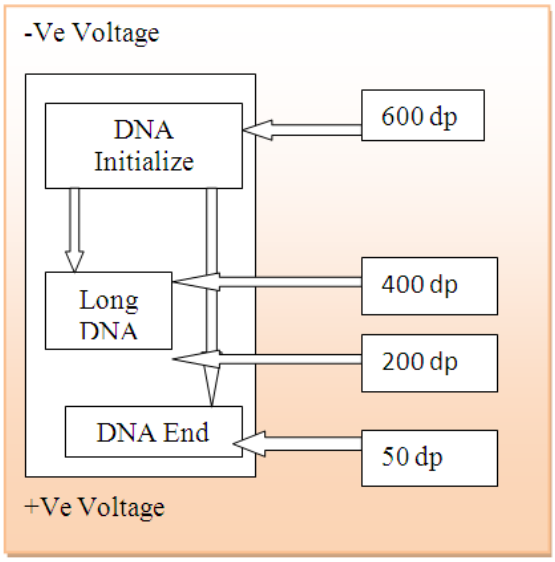

Figure 6. Gel Electrophoresis of DNA

6.4. Itineraries Selection: Have a Complete Set of Cities

Strategy: Successively filter the DNA molecules by city, one city at a time. Since the DNA we start with contains five cities, we will be left with strands that encode each city once.

DNA containing a specific sequence can be purified from a sample of mixed DNA by a technique called affinity purification, as shown in Figure 7. This is accomplished by attaching the compliment of the sequence in question to a substrate like a magnetic bead. The beads are then mixed with the DNA. DNA, which contains the sequence you're after then hybridizes with the complement sequence on the beads. These beads can then be retrieved and the DNA isolated.

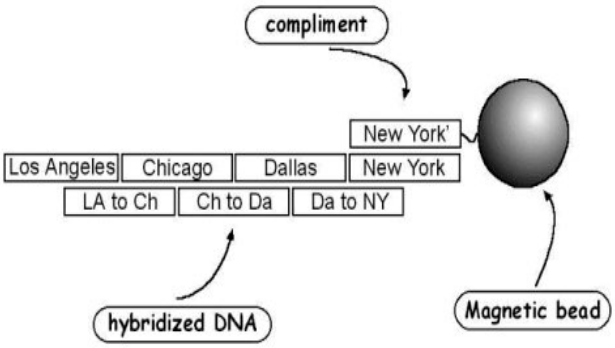

Figure 7. Affinity Purification of DNA

So we now affinity purifies five times, using a different city complement for each run. For example, for the first run we use L.A.'-beads (where the ' indicates compliment strand) to fish out DNA sequences which contain the encoding for L.A. (which should be the entire DNA because of step 3), the next run we use Dallas'-beads, and then Chicago'-beads, Miami'-beads, and finally NY'-beads. The order isn't important. If an itinerary is missing a city,

then it will not be "fished out" during one of the runs and will be removed from the candidate pool. What we are left with the itineraries that start in LA, visit each city once, and end in NY. This is exactly what we are looking for. If the answer exists we would retrieve it at this step.

Adleman's experiment solved a seven city problem, but there are two major shortcomings preventing a large scaling up of his computation. The complexity of the traveling salesman problem simply doesn't disappear when applying a different method of solution - it still increases exponentially. For Adleman's method, what scales exponentially is not the computing time, but rather the amount of DNA. Unfortunately this places some hard restrictions on the number of cities that can be solved; after the Adleman article was published, more than a few people have pointed out that using his method to solve a 200 city HP problem would take an amount of DNA that weighed more than the earth. Another factor that places limits on his method is the error rate for each operation. Since these operations are not deterministic but stochastically driven (we are doing chemistry here), each step contains statistical errors, limiting the number of iterations you can do successively before the probability of producing an error becomes greater than producing the correct result. For example an error rate of 1% is fine for 10 iterations, giving less than 10% error, but after 100 iterations this error grows to 63%.

7. Nussinov-Jacobson(*NJ*) Algorithm

The NJ algorithm provides some guidelines for DNA code design. The NJ algorithm is based on the assumption that in a DNA sequence $q_1q_2 \ldots q_n$, the energy of interaction, $\alpha(q_i, q_j)$, between the pair of bases (q_i, q_j) is independent of all other base pairs. The interaction energies $\alpha(q_i, q_j)$ are negative quantities whose values usually depend on the actual choice of the base pair (q_i, q_j). Let $E_{i,j}$ denote the minimum free energy of the subsequence $q_i \ldots q_j$. The independence assumption allows us to compute the minimum free energy of the sequence $q_1q_2 \ldots q_n$ through the recursion

$$E_{i,j} = min\{E_{i+1,j-1} + \alpha(q_i, q_j), E_{i,k-1} + E_{k,j}, i < k \leq j,$$

where $E_{i,i} = E_{i,i-1} = 0$ for $i = 1, 2, ..., n$. The value of $E_{1,n}$ is the minimum free energy of a secondary structure of $q_1q_2 \ldots q_n$. Note that $E_{1,n} \leq 0$. A large negative value for the free energy, $E_{1,n}$, of a sequence is a good indicator of the presence of a secondary structure in the physical DNA sequence.

The NJ algorithm can be described in terms of free-energy tables. In a free-energy table, the entry at position (i, j) (the top left position being (1,1)), contains the value of $E_{i,j}$. The table is filled out by initializing the entries on the main diagonal and on the first lower sub-diagonal of the matrix to zero, and calculating the energy levels according to the recursion in (1). The calculations proceed successively through the upper diagonals: entries at positions (1, 2),(2, 3), ...,(n − 1, n) are calculated first, followed by entries at positions (1, 3),(2, 4), ...,(n−2, n), and so on. Note that the entry at (i, j), j > i, depends on α(i, j) and the entries at (i, l), l = i, . . . , j−1, (l, j), l = i+1, . . . , n−1, and (i + 1, j − 1). The complexity of the NJ algorithm is $O(n^3)$, since each of the $O(n^2)$ entries requires O(n) computations. The minimum-energy secondary structure itself can be found by the backtracking algorithm which retraces the steps of the NJ algorithm. From a DNA code design point of view, it would be of considerable interest to determine a set of amenable properties that oligonucleotide sequences should possess so as to either facilitate testing for secondary structure, or exhibit a very low probability for forming such a structure. The overall complexity of computing the free-energy tables of a DNA codeword $q_1q_2 \ldots q_n$ and all of its cyclic shifts is $O(n^3)$.

8. Design Methodology

Conventional microprocessor design automation involves optimization at various phases to minimize gate count, power, chip area and to maximize performance. Many tools are available to automate these processes. Evolutionary algorithms are applied towards these optimization processes. This is due to the fact that evolution is an optimization process by which the phenotype of a population gets optimized over successive generations. The fact that evolution has produced complex organisms such as humans stands testimony to its success as an optimization process. Natural evolution involves DNA based processes. But modeling this natural evolution at the DNA level with realistic encoding and recombination processes has not been attempted to automate microprocessor design [29]. In fact, microprocessors, which are less complex than humans, can be evolved naturally without human intervention if their characteristics (phenotype) can be mapped onto the DNA domain through a biologically realistic encoding process. Then automating microprocessor design process would reduce to combining different microprocessor DNAs to produce an offspring and simulating their working environment thereby incorporating random variation and selection which form the two major steps in natural evolution.

DNA computing focuses on the use of massive ***parallelism***, or the allocation of tiny portions of a computing task to many different processing elements. The structure of the DNA allows the elements of the problem to be represented in a form that is analogous to the binary code structure. Trillions of unique strands of DNA are able to represent all of the possible solutions to the problem. Some scientists predict a future where our bodies are patrolled by tiny DNA computers that monitor our well-being and release the right drugs to repair damaged or unhealthy tissue. "The speed of any computer, biological or not, is determined by two factors: (i) how many parallel processes it has; (ii) how many steps each one can perform per unit time. The exciting point about biology is that the first of these factors can be very large: recall that a small amount of water contains about 1022 molecules. Thus, biological computations could potentially have vastly more parallelism than conventional ones." DNA computers can perform 2×10^{19} (irreversible) operations per joule. Existing supercomputers aren't very energy-efficient, executing a maximum of 109 operations per joule. Here, the energy could be very valuable in future. So, this character of DNA computers can be very important. DNA uses ***Adenosine Triphosphate (ATP)*** as

fuel to allow ligation or as a means to heat the stand to cause dissociation.

Figure 8 shows processes of computation phase of DNA Computer. Parameter Trees are built by using DNA based source input. The DNA sequences of the processors are then obtained using a specific, nature inspired Encoding scheme. Hybridization between various sequences is driven by certain Methodology. Certain components of a processor like Carry Save Adder, Baugh-Wooley Multiplier etc are pre-encoded and stored in the DNA Pool. These components can then be added to the offspring processor based on the user specifications. Mutation involves the extraction of the overall netlist of the processor. Graphical Decoding is done to fix the layout level map of the processor and incorporates Embryogenesis. An optimizer attempts to improve the time and space requirements of a program. There are many ways in which code can be optimized, but most are expensive in terms of time and space to implement.

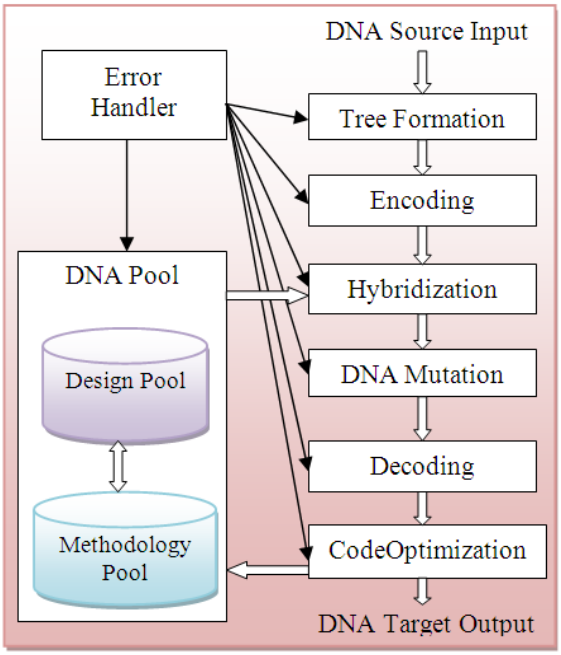

Figure 8. Six Phases of Computation Process for DNA Computer

8.1. Tree Formation

The first phase of the design methodology involves mapping the characteristics of a processor onto a Directed Acyclic Graph (parameter tree) where each node represents a characteristic and the edge weight between two nodes quantifies the dependency between the nodes. To build the parameter tree, various characteristics that define a processor must be obtained. Then various relationships among these characteristics must be studied. This can be done by collecting a large number of processor specifications and observing the trend among various parameters. For example, if most of the processors show that increasing the cache levels increases the power consumption tremendously, a high edge weight can be associated between these two parameters. A simple data mining algorithm can automate this process. The Node Function as shown in Figure 9, of a particular node is a mathematical relation connecting the Node Data of the node to the Node Data of its parents and children. Figure 8 shows a simple functional dependence. The child node Maximum Number of Instruction per Clock-cycle (I), dependents on three of its parent's node: the No. of Cores(C), the Number of Data Paths per core (D) and the Number of Processors(P). "I" is just the direct product of these values. The node function is very important because it defines the actual mathematical relations between nodes. Thus node values can be fixed up and validated during hybridization. For example, if the user specifies a "I" of 18, then we can fix the No. of Cores, No. of Processors and No. of Data Paths per core as any combination of 3, 3, 2 (3x3x2=18). This could be one of the methodologies during hybridization. The edges in the figure have weights Wc, Wd and Wp. These weights signify the level of dependence between two nodes: higher the weight, higher the dependence. For example, Cache Power is most affected by Voltage. These weights play a vital role while traversing the tree.

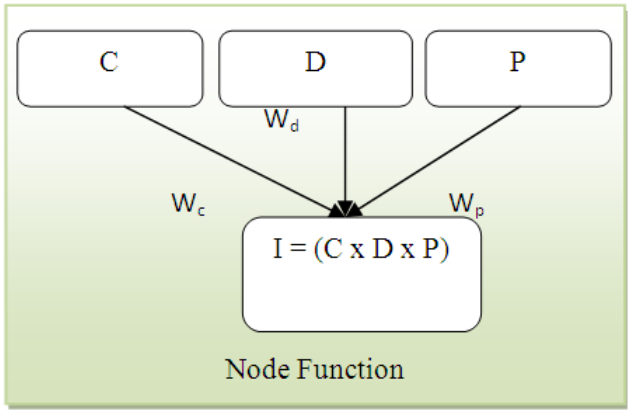

Figure 9. Node Function

For optimal performance, we use supervised learning algorithms of Neural Network. By applying this process to non-connected nodes, inferences can be drawn about the way they interact indirectly. For example, instruction size and type of full adder used in the ALU don't have any obvious relationship. But if hidden relations do exist between them, then they could be found out using this method. The node function can be stored in standard formats like post-fix notations with variables as pointers to other nodes. The node function is mostly architecture independent. But, sometimes, certain architectures may change the function to an extent. For example, the inclusion of Hyper-threading may change the equation governing Level 1 Cache Size. In natural DNA sequences, related genes are placed close to each other. So a Depth First Traversal (DFT) is performed on the parameter tree to build a linear sequence of nodes. Since DFT visits a node that is most strongly related to the node being processed, it provides a list where related nodes are placed together. By specifying directions on the edges, the DFT is restricted in exploring only the children of the node being processed. When a node has multiple parents and has a stronger connection to one of its unexplored parents than any of its children, it would be obviously better to place that parent node next to the node being processed.

8.2. Encoding Strategy

The process of creation of the actual DNA strands is carried out in the encoding phase. Variable Length Delimiter Based Encoding (VLDBE) system is used for

encoding. A delimiter is a special sequence which serves as a flag that represents the beginning of a node, a field, a number, a character, a symbol or any other data entity. The delimiter is set as ATCG+ where + is a wild card that can be substituted by any of the symbols. Each substitution signifies a different delimiter. In particular, AA signifies No Delimiter. This is very important to identify a data sequence which contains a substring 'ATCG'. For example if a data contains the string GAGCTATCGCGA, then this sequence would be transformed and encoded as GAGCTATCGAACGA. The 'AA' string signifies that the ATCG sequence was a data element rather than a part of a delimiter sequence. The 'AA' string is ignored when the sequence is decoded. Each symbol in a DNA sequence is known as a Dit, which is an acronym for a DNA digit. Table 2 lists DNA Sequence and their meaning during encoding process.

Table 2. DNA Sequence Meaning during Encoding

DNA Sequence	Meaning
ATCGAA	No Delimiter
ATCGAT	Node Begins
ATCGCT	Number Begins
ATCGCG	FSM Details Begins
ATCGCA	Child Node List Begins
ATCGCC	Netlist Begins
ATCGTG	Parents Node List Begins
ATCGTC	Node Name Begins
ATCGTT	Node ID Begins
ATCGTA	Node Function Begins
ATCGAG	Alpha-String Begins
ATCGAC	Data Field Begins
ATCGG+	Reserved for Future Use

The encoding process transforms information to a base 4 domain. The base 4 system allows us to define nature inspired hybridization rules. Moreover, if characteristics of a species can be related to each parameter of the processor, then the DNA strands defining those traits can be used. By using this trait based encoding and natural DNA recombination rules, evolution can be completely realistic. This encoding scheme for numbers involves a straight forward conversion to base 4 systems.

8.2.1. Coding Technique in DNA Computing

A DNA code is simply a set of (oriented) sequences over the alphabet Q = {A, C, G, T}. One design principle that arises out of a study of the ***NJ algorithm*** is that DNA codes should contain a cyclic structure. The key idea behind this principle is based on the observation that once the free-energy table, and consequently, the minimum free energy of a DNA sequence q have been computed, the corresponding computation for any cyclic shift of q becomes easy.

While testing DNA sequences for secondary structure is one aspect of the code design process, it is equally important to know how to design codewords that have a low tendency to form secondary structures. The obvious approach here would be to identify properties of the sequence of bases in an oligonucleotide that would encourage secondary structure formation, so that we could then try to construct codewords which do not have those properties. For example, it seems intuitively clear that if a sequence q has long, non-overlapping segments s_1 and s_2 such that $s_1 = s_2^{RC}$, then there is a good chance that q will fold to enable s_1 to bind with s_2^R thus forming a stable structure. Actually, we can slightly strengthen the above condition for folding by requiring that s1 and s_2 be spaced sufficiently far apart, since a DNA oligonucleotide usually does not make sharp turns, i.e., does not bend over small regions. In any case, the logic is that a sequence that avoids such a scenario should not fold.

For any pair of length-n words $p = p_1p_2 \ldots p_n$ and $q = q_1q_2 \ldots q_n$ over the alphabet Q, the ***Hamming distance*** $d_H(p, q)$ is defined as usual to be the number of positions i at which $p_i \neq q_i$. We further define the reverse Hamming distance between the words p and q to be $d^R{}_H(p, q) = d_H(p, q^R)$. Similarly, their reverse-complement Hamming distance is defined to be $d^{RC}{}_H(p, q) = d_H(p, q^{RC})$. For a DNA code C, we define its minimum (Hamming) distance, minimum reverse (Hamming) distance, and minimum reverse-complement (Hamming) distance in the obvious manner:

$$d_H(C) = min\, d_H(p, q) \text{ such that } p, q \in C,\ p \neq q,$$

$$d^R{}_H(C) = min d^R{}_H(p, q) \text{ such that } p, q \in C,$$

$$d^{RC}{}_H(C) = d^{RC}{}_H(p, q) \text{ such that } p, q \in C.$$

The above result shows that the complexity of testing a DNA code with M length-n codewords for secondary structure is reduced from $O(Mn^3)$ to $O(Mn^2)$, if the code is cyclic. It is also worth pointing out that a cyclic code structure can also simplify the actual production of the DNA sequences that form the code.

Table 3. Coding Table in DNA Computing

DNA Coding System(A, G, C, T)	Binary Coding System(00, 01, 10, 11)	Quartet Coding System(0, 1, 2, 3)	Decimal Coding System
AGT	000111	013	0 * 16 + 1 * 4 + 3 * 1 = 7
TAA	110000	300	3 * 16 + 0 * 4 + 0 * 1 = 48
TTT	111111	333	3 * 16 + 3 * 4 + 3 * 1 = 63
AGC	000110	012	0 * 1 + 1 * 1/4 + 2 * 1/16 = 0.375
CCA	101000	220	2 * 1 + 2 * 1/4 * 0 * 1/16 = 2.5
GTT	011111	133	1 * 1 + 3 * 1/4 + 3 * 1/16 = 1.9375

Another important code design consideration linked to the process of oligonucleotide ***hybridization*** pertains to the GC-content of sequences in a DNA code. The GC-content, $w_{GC}(q)$, of a DNA sequence $q = q_1q_2 \ldots q_n$ is defined to be the number of indices i such that $q_i \in \{G, C\}$. A DNA code, in which all codewords have the same GC-content, w, is called a constant GC-content code. The constant GC-content requirement assures similar thermodynamic characteristics for all codewords, and is introduced in order to ensure that all hybridization operations take place in parallel, i.e., roughly at the same time. The GC-content is usually required to be in the range of 30–50% of the length of the code.

DNA code Construction Methodology:

8.2.1.1. Binary Mapping

The problem of constructing DNA codes with some of the properties desirable for DNA computing can be made

into a binary code design problem by mapping the DNA alphabet onto the set of length-two binary words as shown in Table 3. The mapping is chosen so that the first bit of the binary image of a base uniquely determines the complementary pair to which it belongs.

Let q be a DNA sequence. The sequence b(q) obtained by applying coordinatewise to q the mapping given in Table 3., will be called the binary image of q. If b(q) = $b_0b_1b_2 \ldots b_{2n-1}$, then the subsequence e(q) = $b_0b_2 \ldots b_{2n-2}$ will be referred to as the even subsequence of b(q), and o(q) = $b_1b_3 \ldots b_{2n-1}$ will be called the odd subsequence of b(q). Given a DNA code C, we define its even component E(C) = {e(p) : p ∈ C}, and its odd component O(C) = o(p) : p ∈ C}. It is clear from the choice of the binary mapping that the GC-content of a DNA sequence q is equal to the Hamming weight of the binary sequence e(q). Consequently, a DNA code C is a constant GC-content code if and only if it's even component, E(C), is a constant-weight code.

Let B be a binary code consisting of M codewords of length n and minimum distance d_{min}, such that $c \in B$ implies that $c \in B$. For $w > 0$, consider the constant-weight subcode $B_w = \{u \in B : w_H(u) = w\}$, where $w_H(\cdot)$ denotes Hamming weight. Choose $w > 0$ such that $n \geq 2w + |d_{min}/2|$, and consider a DNA code, C_w, with the following choice for its even and odd components:

$$E_w = \{ab' : a,b \in B_w\}, O = \left\{ab^{RC} : a,b \in B, a <_{lex} b\right\},$$

where $<_{lex}$ denotes lexicographic ordering. The $a <_{lex} b$ in the definition of O ensures that if $ab^{RC} \in O$, then $ba^{RC} \in^{No} O$, so that distinct codewords in O cannot be reverse-complements of each other.

Therefore, the DNA code $C = U^{wmax}_{w=dmin} C_w$, with $wmax = (n - |d_{min}/2|)/2$, has $1/2\ M(M-1) \sum^{wmax}_{w=dmin} |Aw|^2$ codewords of length $2n$, and satisfies $d_H(B) \geq d_{min}$ and $d^{RC}{}_H(B) \geq d_{min}$.

8.2.1.2. DNA Codes from Generalized Hadamard Matrices

A generalized Hadamard matrix $H \equiv H(n, \mathbb{Q}_m)$ is an $n \times n$ square matrix with entries taken from the set of mth roots of unity, $\mathbb{Q}m = \{e^{-2\pi il/m}, l = 0, ..., m-1\}$, that satisfies $HH* = nI$. Here, I denote the identity matrix of order n, while $*$ stands for complex-conjugation. We will only concern ourselves with the case $m = p$ for some prime p. A necessary condition for the existence of generalized Hadamard matrices $H(n, \mathbb{Q}_p)$ is that $p|n$. The exponent matrix, $E(n, Z_p)$, of $H(n, \mathbb{Q}_p)$ is the $n \times n$ matrix with entries in $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$, obtained by replacing each entry $(e^{-2\pi i})^l$ in $H(n, \mathbb{Q}p)$ by the exponent l.

For any $k \in \mathbb{Z}^+$, there exist DNA codes D with $3^k - 1$ codewords of length $3^k - 1$, with constant GC-content equal to $3^k - 1$, $d_H(D) = 2 \cdot 3^k - 1$, $d^{RC}{}_H(D) \geq 3^k - 1$. And in which each codeword is a cyclic shift of a fixed generator codeword g.

DNA codes with constant GC-content can obviously be constructed from constant-composition codes over $\mathbb{Z}_p$ by mapping the symbols of $\mathbb{Z}_p$ to the symbols of the DNA alphabet, Q = {A, C, G, T}. DNA codes can be obtained from such generators by mapping {0, 1, 2, 3} onto {A, G, C, T}. Although all such mappings yield codes with (essentially) the same parameters, the actual choice of mapping has a strong influence on the secondary structure of the codewords.

8.3. Hybridization

In DNA Based Computer hybridization is an important phase. The DNA sequences are set to replicate and create trillions of new sequences based on the initial input sequences in a matter of seconds, called DNA hybridization. Hybridization depends on nature and function as shown in Figure 10. Hybridization between various sequences are driven by certain heuristics .When DNA pool updates, new value is used for hybridization. The microprocessor DNA sequences are combined to form an offspring. The recombination algorithm relies on the principle of Localization of Reaction according to which the probability of a node participating in recombination is determined by its proximity to a reacting node. The number of computational errors in a DNA system designed for solving an instance of a 3-SAT problem [31] were reduced by generating DNA sequences that avoid folding and undesired hybridization phenomena.

However, in DNA-based computation, only relatively short single-stranded DNA sequences, referred to as ***oligonucleotides***, are used. The computing process simply consists of allowing these oligonucleotide strands to self-assemble to form long DNA molecules via the process of hybridization. Hybridization is the process in which oligonucleotides with long regions of complementarity bond with each other. The astounding parallelism of biochemical reactions makes a DNA computer capable of parallel-processing information on an enormously large scale. However, despite its enormous potential, DNA-based computing is unlikely to completely replace electronic computing, due to the inherent unreliability of biochemical reactions, as well as the sheer speed and flexibility of silicon-based devices [30].

The process of self-assembly in DNA computing requires the oligonucleotide strands (***codewords***) participating in the computation to selectively hybridize in a manner compatible with the goals of the computation. If the codewords are not chosen appropriately, unwanted (non-selective) hybridization may occur. For many applications, even more detrimental is the fact that an oligonucleotide sequence may self-hybridize, i.e., fold back onto itself, forming a secondary structure which prevents the sequence from participating in the computation process altogether.

Hybridization between a pair of distinct DNA sequences is referred to as ***cross-hybridization***, to distinguish it from ***self-hybridization*** or sequence folding. The distance measures defined above come into play when evaluating cross-hybridization properties of DNA words under the assumption of a perfectly rigid DNA backbone. As an example, consider two DNA codewords 3'−AAGCTA− 5' and 3' − ATGCTA – 5' at Hamming distance one from each other. For such a pair of codewords, the reverse complement of the first codeword, namely 3' − TAGCTT – 5', will show a very large affinity to hybridize with the second codeword. In order to prevent

such a possibility, one could impose a minimum Hamming distance constraint, $d_H(C) \geq d_{min}$, for some sufficiently large value of d_{min}. On the other hand, in order to prevent unwanted hybridization between two DNA codewords, one could try to ensure that the reverse-complement distance between all codewords is larger than a prescribed threshold, i.e. $d^{RC}(C) \geq d^{RC}_{min}$. Indeed, if the reverse-complement distance between two codewords is small, as for example in the case of the DNA strands 3' – AAGCTA – 5' and 3' − TACCTT−5', then there is a good chance that the two strands will hybridize.

In DNA based Computer, Spread of Hybridization is a strictly decreasing function which gives the Probability of Hybridization of each node based its distance from the currently reacting node. String combination in DNA based computer is mainly performed by two ways: Dit-wise Hybridization and Node Hybridization.

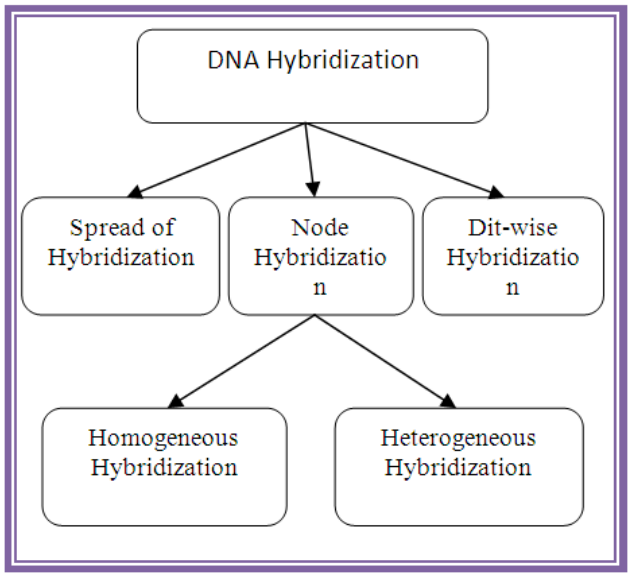

Figure 10. Hybridization Type in DNA Computer

The ***Dit-Wise Hybridization*** involves two strings reacting dit by dit. The biggest advantage of this method is the evolution of radical offspring. On the other hand this technique may produce large number of invalid strings before providing the required architecture. The ***Node Hybridization*** involves two strings which react node by node. The nodes can be identified using their delimiters. This overlapping can be either with similar node or with different node. ***Homogeneous Hybridization*** takes places between two similar nodes. Since these nodes represent the same characteristic of the processor, only the values of these nodes are affected. ***Heterogeneous Hybridization*** takes place between 2 dissimilar nodes. Since the nodes do not represent the same characteristic of the processor, the recombination between them results in either the loss of information of a node or the formation of a new node. In the former case, the node that is lost is reformed afterwards in the mutation stage and in the latter case the newly created node is stored as a special sequence. During the decoding phase, the user is informed about this new node. The user can then either discard this new node or revert back to the two nodes which combined to create them or incorporate this new node in the design of the machine. The heterogeneous hybridization occurs with a slightly smaller probability when compared to homogeneous hybridization. Moreover, even in the event of heterogeneous hybridization, the probability of discarding a node and reforming it at a later stage is higher than the probability of formation of a new node.

8.4. DNA Mutation Algorithm

Mutation in this methodology refers to the formation of DNA strings that represent the netlist and FSM information. Netlist generation and initial placement is carried out in the DNA domain. For this effect DNA based algorithms need to be devised. Conventional CAD algorithms for placement and routing cannot be directly applied without decoding the processor string. Instead, if the DNA counter parts of these algorithms are devised, then these processes can be carried out in the DNA domain itself. Apart from perfectly suiting a DNA computational model, aspects of developmental biology can be incorporated in the interconnect development during the decoding phase. The following section describes a novel methodology for creating such DNA algorithms from conventional algorithms. Figure 11. illustrates this methodology. The algorithms that are to be executed on the processor must be converted to the processor's instruction set. A Generic Compiler that takes in the processor's instruction set as an input and translates the Benchmarking Algorithms in terms of the processor's instructions is designed for this purpose. Based on the processor's architecture, the delay and the power consumed by each instruction are computed.

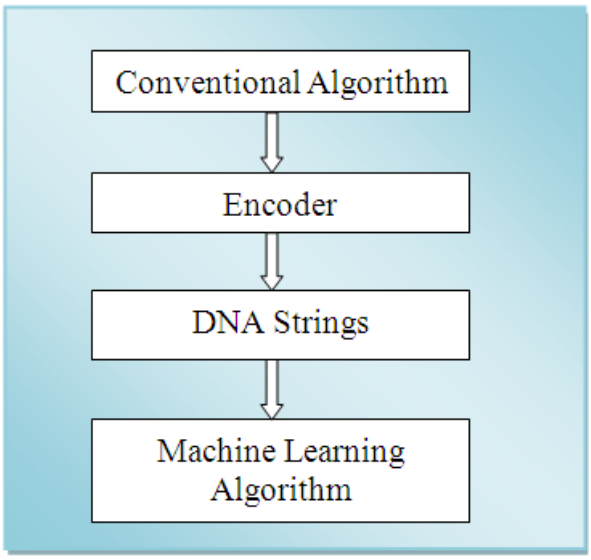

Figure 11. Technique for Creating DNA Based Algorithm

The problem parameters at each of each step of a Conventional Algorithm is tracked and converted into a DNA string using an Encoder. Thus, for every step, a set of DNA sequences are obtained. A Machine Learning Algorithm draws inferences from the changes in the problem parameters after each step and creates a corresponding DNA action step. These action steps constitute the DNA algorithm. Several examples are given to the system to enable it to learn. The offspring DNA obtained after the mutation process contains information about the connectivity of the modules, the netlist and the FSM of the offspring processor.

8.5. Decoding and Code Optimization

In Mutation Phase, the placement of netlist is not optimized completely. Decoding phase involves decoding the offspring DNA sequence to the conventional domain and optimizing the placement of the modules and the netlist. The offspring DNA sequence is read till a delimiter is encountered. The sequence between two node delimiters is interpreted as a node and is added to the parameter tree. The sequence inside the node is further decoded with the help of the delimiters to obtain its dependencies. In this manner the parameter tree is constructed. When an unknown node is encountered (if the node name does not match with a stored list of node names), the node is stored separately for further investigation. The details of the connectivity of the modules are also extracted from the offspring DNA sequence. The main advantage of evolution is that it is need based. The environment of a species plays a major role in the selection process and also may cause mutation of the DNA to make the species better suited to their environment.

An optimizer attempts to improve the time and space requirements of a program. There are many ways in which code can be optimized, but most are expensive in terms of time and space to implement. Common optimizations include: removing redundant data, removing unreachable sections of code, unfolding loops, identifying common sub-expressions, etc.

8.6. DNA Pool and Error Handler

DNA Pool is responsible for storing the component and their functions methodology. These components can then be added to the offspring processor based on the user specifications. There are two different DNA pools: the Design Pool and the Methodology Pool. The basic methodologies involved during hybridization vary from natural splicing to Boolean and binary operations. These are stored in the Methodology Pool along with their associated Degree of Belief (DoB). The heuristics are selected as per their degree of belief; higher the DoB, greater is the probability for that methodology being used for hybridization.

Design Pool stores the complete design of various processing component as well as several different functional units while the methodology pool stores various heuristics used in the hybridization and their respective effectiveness value. Architectural details, instruction set and FSM design of a microprocessor are stored as DNA sequences in the design pool. The processor strings are stored in an indexed array format for ease of retrieval. Each processor has an associated Potency Factor. The potency factor is a measure of a processor's capability in producing an offspring with a specific characteristic (metrics). This is not a simple numerical value, rather it is a vector. The vector (of n metrics) stores the potency of the processor to produce various types of processors. The n metrics are decided by the user based on his opinion of a processor's goodness. For example, a user might want to use power and performance as two important metrics while another user may use chip area, which ultimately decides cost of a processor, as an important metric.

Each of the six phases of, computation process of DNA computer, can encounter errors. On detecting an error the Error Handler must: report the error in a helpful way, correct the error if possible, and continue processing (if possible) after the error to look for further errors.

9. Conclusion and Future Work

DNA computing (biomolecular computing) is a fast developing interdisciplinary area. Research and development in this area concerns theory, experiments, and applications of DNA computing. This paper presented a novel Design Methodology for DNA Based Computer as per user specifications. Tree formation phase is based on the components of DNA computing. Encoding and mutation is fully biological function, which works on DNA. Coding for DNA Computer is a progressive part. Current technologies that are available in DNA computing are also reviewed in this paper. Due to the highly parallel characteristics of DNA operations, the corresponding DNA algorithms scale well in the size of the problem. Also comparison between Silicon based processor and DNA based processor yields DNA based computer has optimal performance. Therefore DNA computing shows potential advantages in solving the hard problems.

In the future, I would like to work on optimal algorithms for DNA based Computer.

References

[1] Adleman, L. M. (1994). "*Molecular computation of solutions to combinatorial problems*". Science 266 (5187): 1021-1024. doi:10.1126/science.7973651. PMID 7973651.

[2] Boneh, D.; Dunworth, C.; Lipton, R. J.; Sgall, J. Í. (1996). "*On the computational power of DNA*". Discrete Applied Mathematics 71: 79-94.

[3] Lila Kari, Greg Gloor, Sheng Yu (January 2000). "*Using DNA to solve the Bounded Post Correspondence Problem*". Theoretical Computer Science 231 (2): 192-203.

[4] Lewin, D. I. (2002). "*DNA computing*". Computing in Science & Engineering 4 (3): 5-8.

[5] Lovgren, Stefan (2003-02-24). "*Computer Made from DNA and Enzymes*". National Geographic. Retrieved 2009-11-26.

[6] *".In Just a Few Drops, A Breakthrough in Computing*", New York Times, May 21, 1997.

[7] Shu, Jian-Jun; Wang, Q.-W.; Yong, K.-Y. (2011). "*DNA-based computing of strategic assignment problems*". Physical Review Letters 106 (18): 188702.

[8] Saenger, Wolfram (1984). "*Principles of Nucleic Acid Structure*". New York: Springer-Verlag.

[9] Alberts, Bruce; Johnson, Alexander; Lewis, Julian; Raff, Martin; Roberts, Keith; Walters, Peter (2002). "*Molecular Biology of the Cell*"; Fourth Edition. New York and London: Garland Science.

[10] Butler, John M. (2001). "*Forensic DNA Typing*". Elsevier. ISBN 978-0-12-147951-0. OCLC 223032110 45406517, PP-14-15.

[11] Shapiro, Ehud (1999-12-07)., "*A Mechanical Turing Machine: Blueprint for a Biomolecular Computer*," Weizmann Institute of Science. Retrieved 2009-08-13.

[12] Benenson, Y.; Paz-Elizur, T.; Adar, R.; Keinan, E.; Livneh, Z.; Shapiro, E. (2001). "*Programmable and autonomous computing machine made of biomolecules*". Nature 414 (6862): 430-434.

[13] Benenson, Y.; Gil, B.; Ben-Dor, U.; Adar, R.; Shapiro, E. (2004). "*An autonomous molecular computer for logical control of gene expression*". Nature 429 (6990): 423-429.

[14] Bond, G. L.; Hu, W.; Levine, A. J. (2005). "*MDM2 is a Central Node in the p53 Pathway: 12 Years and Counting*". Current Cancer Drug Targets 5 (1): 3-8.

[15] Kahan, M.; Gil, B.; Adar, R.; Shapiro, E. (2008). "*Towards molecular computers that operate in a biological environment*". Physica D: Nonlinear Phenomena 237 (9): 1165-1172.

[16] Weiss, S. (1999). "*Fluorescence Spectroscopy of Single Biomolecules*". Science 283 (5408): 1676-1683.

[17] Santoro, S. W.; Joyce, G. F. (1997). "*A general purpose RNA-cleaving DNA enzyme*". Proceedings of the National Academy of Sciences 94 (9): 4262-4266.

[18] Stojanovic, M. N.; Stefanovic, D. (2003). "*A deoxyribozyme-based molecular automaton*". Nature Biotechnology 21 (9): 1069-1074.

[19] MacDonald, J.; Li, Y.; Sutovic, M.; Lederman, H.; Pendri, K.; Lu, W.; Andrews, B. L.; Stefanovic, D.; Stojanovic, M. N. (2006). "*Medium Scale Integration of Molecular Logic Gates in an Automaton*". Nano Letters 6 (11): 2598-2603.

[20] Stojanovic, M. N.; Mitchell, T. E.; Stefanovic, D. (2002). "*Deoxyribozyme-Based Logic Gates*". Journal of the American Chemical Society 124 (14): 3555-3561.

[21] Cruz, R. P. G.; Withers, J. B.; Li, Y. (2004). "*Dinucleotide Junction Cleavage Versatility of 8-17 Deoxyribozyme*". Chemistry & Biology 11: 57-67.

[22] Darko Stefanovic's Group, "*Molecular Logic Gates and MAYA II*", a second-generation tic-tac-toe playing automaton.

[23] Seelig, G.; Soloveichik, D.; Zhang, D. Y.; Winfree, E. (8 December 2006). "*Enzyme-free nucleic acid logic circuits*". Science 314 (5805): 1585-1588.

[24] A. Hierlemann, O. Brand, C. Hagleitner, H. Baltes, "*Microfabrication techniques for chemical/biosensor"s*, Proceedings of the IEEE, 91 (6), 2003, 839-863

[25] A. Hierlemann, H. Baltes, "*CMOS-based chemical microsensors, The Analyst*", 128 (1), 2003, pp. 15-28

[26] Yong, E. (2013). "*Synthetic double-helix faithfully stores Shakespeare's sonnets*". Nature.

[27] Goldman, N.; Bertone, P.; Chen, S.; Dessimoz, C.; Leproust, E. M.; Sipos, B.; Birney, E. (2013). "*Towards practical, high-capacity, low-maintenance information storage in synthesized DNA*". Nature 494 (7435): 77-80.

[28] Church, G. M.; Gao, Y.; Kosuri, S. (2012). "*Next-Generation Digital Information Storage in DNA*". Science 337 (6102): 1628.

[29] David B. Fogel*: Evolutionary Computation: The Fossil Record*. Wiley-IEEE Press (May 1,1998).

[30] E. Winfree, "*DNA computing by self-assembly*," The Bridge, vol. 33, no. 4, pp. 31-38, 2003.

[31] R.S. Braich, N. Chelyapov, C. Johnson, P.W.K. Rothemund and L. Adleman, "*Solution of a 20-variable 3-SAT problem on a DNA computer*," Science, vol. 296, pp. 492-502, April 2002.