Survey on Killer Applications, Recent Advancements and Basic Arithmetic Operations in *DNA Computing Paradigm*

A dissertation
submitted in partial fulfillment
of the requirements for
M.Tech.(Computer Science) degree
from
Indian Statistical Institute, Kolkata
by
Janardan Mishra

under the supervision of

Professor Rana Barua

Indian Statistical Institute
Kolkata 700 035.

July 2001

Indian Statistical Institute
203, Barrackpore Trunk Road
Kolkata 700035

Certificate of Approval

This is to certify that this thesis titled This is to Certify that this thesis entitled *Survey* **on Killer Applications, Recent Advancements and Basic Arithmetic Operations in** *DNA Computing Paradigm* submitted by *Janardan Mishra*, towards the partial fulfillment of the requirement for *degree of Master of Technology in Computer Science* from the *Indian Statistical Institute, Kolkata,* embodiies the work done under my supervision.

[illegible]

Rana Barua
Professor,
Stat - Math Unit,
Indian Statistical Institute,
Kolkata-700 035.

Acknowledgement

My sincerest gratitude from the deepest corners of heart and intellect goes to *Prof. Rana Barua* for his guidance, to the pin point advice, encouragement and highly helpful criticism throughout the course of this dissertation which actually is the base that I could at all complete my work.

There are some moments which become parcels of one's life - the moments of my discussion regarding the various topics , to start with the initial phases of learning this new topic , to final stage of discussions on recursive DNA arithmetic and preparing a draft and the final report - are a few of that sort. The times of his extreme cooperation whether that may be ,to say, sitting for hours listening me with newer ending enthusiasm or leaving his terminal on a slight request, so that I can take print-outs, will always remind me something which I don't know if at all I will receive in the rest of my life..

I would also like to express my gratefulness to S. Pyne for his helpful suggestions, inspirations, and above all bearing me with my all silly doubts pertaining his work on 'inverted paradigm', during this whole course of dissertation work.

My felicitation also goes to my teachers, scientists and staff members of the institute for providing me all the help during the period I worked.

Janardan Mishra
Indian Statistical Institute, Kolkata
July, 2001.

Contents

List of Figures

Chapter 1

Introduction to DNA Computing and More..

After a long journey of more than half a century of semiconductor technology based electronic computing devices it seems that limiting saturation point is not far away when electronic computing era may start touching its natural physical limits due the difficulties in fabrication technology like lithography bandwidth. Still hard problems possibly will remain insurmountable even then. A new ray of hope has come from the experimental demonstration by Leonard Adleman [Adle94] by solving an NP complete problem, namely the Hamiltonian Path Problem (HPP), using none other but the DNA (deoxirebonuclic acid) strands, the nature's choice as carriers' of the definition of life.

In fact even before anyone could think of performing actual experiment demonstrating direct application of DNA like large organic compounds as computing tools R.P. Feynman in his visionary lecture [Feynman61] envisioned molecular level computing. But the road was not easy, since to perform computing in a controlled fashion, structures (tools) must be stable and suitable to encode information, and DNA nano-technology achieved successes in this type of controls only later. With the help of these DNA nano-technology tools in 1994 for the first time Adleman carried out an biological experiment for encoding and solving HPP using DNA strands and standard DNA tool-kit operations. This opened a totally new vista in the field of theoretical as well as practical computing, most importantly because Adleman had demonstrated solving not something which conventional computing can easily work with but a well known hard (in fact NP complete) problem of solving HPP. Though the instance of the problem was not large enough (a digraph with 7 vertices with known Hamiltonian path), still that was a real ground breaking idea because it opened a totally new vista of computing paradigm with lots of potential.

The most *prima falsi* reason that DNA, RNA (Rebonuclic acid) and other peptide molecules have been worked upon more than any other possible alternative chemical compounds is the rapid and mature advancements in the area of gene and cell Biotechnology and no mention to say that these are the technologies to work with DNA and RNA molecules. Moreover the stable and robust structure is what makes it extremely feasible to effectively work with these organic molecules.

Thus to start with, in a nut shell DNA computing is - *"computing with DNA strands"* which definitely contrasts with conventionally known computational biology (which is the study of computational tools for solving biological problems (especially related to molecular biology, see for e.g. [Waterman95]))

1.1 Structure

DNA molecule is a composite organic compound [AKL86] with crystalline (gives robustness) and amorphic (storage for information) sturcture(See Figure 1.1 and 1.2). Usually a long DNA polymer strand consists of mono structures called deoxyrebonucleotides (or simply nucleotide or dnt). Each nucleotide basically contains a *deoxyribose* sugar , a chain of 5 carbon atoms (denoted) as 1' 2' 3' 4' and 5'. The 3' carbon atom is associated with a *hydroxyl group(OH)* with one open vacancy for chemical bounding, and a *phosphate group(P)* which is attached with 5' atom. The 1' atom is connected with any one of the four *nitrogenous bases* (or simply bases): Adinine(A), Cytocine(C), Guanine(G), and Thymine(T). These bases characterize the whole of DNA[1] nucleotide. Say for example if in a dnt 1' carbon atom is attached with Adinine or Guanine base then the dnt is simply referred as A or G. These four nitrogenous bases are chemically classified in two groups - *purines (A, G)* and *pirymidines (C, T).* (see figure 1.1)

1.2 Bonding - Giving Single and Double Strands

Chemically two main types of stable bonds possible between any two dnts are:

1.2.1 Phosphodiaster Bond

This covalent bond[2] is formed between the OH group at 3' carbon atom on one dnt and P group at 5' atom of other dnt. The main property of this phosphodiaster bond is its high strength, which gives rise to stable long DNA strands (human genome is supposed to comprise of a sequence of nearly 30, 000 such dnts!) with long half-life period. Another important issue due to this bond is that it gives a relative *direction*

[1] An another important polymer compound - RNA has its monomer, which differs from DNA monomer only by having *Urasil(U)* base instead of Thymine base (T) and 2' carbon atom of its ribose sugar is attached with hydroxyl group (OH) rather than an hydrogen atom (H).

[2] A covelent bond is formed by sharing of electrons in outermost shell by participating atoms.

to each DNA single strand (ssDNA or simplex) such that a ssDNA is referred to have 5'-3' direction such that the dnt with free P group is at the left end and dnt with free OH group is right end. As an example 5'- GCAA- 3' strand indicates that this ssDNA has free P group at 5' atom with (leftmost) dnt with Guanine base and (rightmost) dnt with Adinine base has OH group free at its 3' atom (because of this understanding a 5' - 3' strand is also referred to as simply 5'-), similarly 3'- GCAA -5' (or simply 3'- GACC)indicates that leftmost dnt with Guanine base has free OH group and rightmost dnt with Adinine base has P group free for further bonding. Some times in a relatively long chain these 5' and 3' ends may form a phosphodiaster bond between themselves giving rise to a *circular* strand.

1.2.2 Hydrogen Bond

This bond which is ionic[3] in nature is formed between the purine and pirymidine bases of two dnts abiding by the rule that **A** couples with **T** and **C**couples with **G** (and vice-versa). This rule is popularly known as *Watson and Crick complementary principle* and (A,T) and (C, G) are referred to as complementary base pairs. This bond is relatively weak in nature and because of that, easy to get formed if the dnts with complementary bases are present in close proximity under reactively moderate temperature (about 38^o C).

Most useful application of this hydrogen bond comes while forming DNA double strands (dsDNA or duplex) from complementary ssDNAs (Note that two ssDNAs are complementary when both the strands have reverse directions 5'- and 3'- (upper and lower)and corresponding dnts are complements of each other.)If the strands are not fully complement to each other then they may form partial bonding sometimes giving rise to hanging double strnds.

It may be noted sometimes that dsDNA is referred to by its upper strand which by convention is one having 5'- direction.

In case of RNA C and G are complementary and U complements (forms hydrogen bonding with) both A and G.

Length of ssDNA is expressed as the number of monomers or dnts present in the strand and written 'n mer' if the strand consists of a chain of n dnts. length of complete dsDNA is expressed in terms of numnber of complementary base pairs (bps). Partial double strands can involve both. Short nucleotides of length about 20 to 30 (mer or bp) are usually called Oligonucleotides(or oligos).

[3]An ionic bond is formed when one atom (called *donor*) donates electrons from its outer shell to second atom (called *acceptor*.

1.3 DNA tool kit *(Recombinant)* Operations

Controlled DNA manipulation is known as Recombinant DNA [Brown93]. A DNA tool-kit is a set of basic operations to do this in molecular biology laboratories in a routinic fashion. One of the inevitable agents in these operations are - *enzymes* - the chemicals that cause all of the transformation to happen in a cell, especially in genes or DNA strands. Some of the basic operations are:

Synthesis

Oligonucleotides may be synthesized to order by synthesizer by supplying it with the four nucleotide bases in solution, which are combined according to a sequence entered by the user. The instrument makes millions of copies of the required oligonucletide (oligo) and places them in solution in a small vial.

Anneal

This is probably the most basic operation used in DNA computing. Single stranded complementary DNA will spontaneously form a double strand of DNA when suspended in solution. This occurs through the hydrogen bonds that arise when complementary base pairs are brought into proximity. This is also called *hybridization.*

Melt

The inverse of annealing is *melting.* That is, the separation of double stranded DNA into single stranded DNA. As the name implies, this can be done by raising the temperature beyond the point where the longest double strands of DNA are stable. Since the hydrogen bonds between the strands are significantly weaker than the covalent bonds between adjacent nucleotides, heating separates the two strands without breaking apart any of the sequences of nucleotides.*Melting* is a bit of a misnomer because the same effect can be achieved by washing the double stranded DNA in doubly distilled water. The low salt content also de stabilizes the hydrogen bonds between the strands of DNA and thereby separates the two strands. Heating can be selectively used to melt apart short double stranded sequences while leaving longer double stranded sequences intact.

Ligate

Often invoked after an annealing operation, ligation concatenates strands of DNA. While it is possible to use some ligase enzymes to concatenate free double stranded DNA, it is dramatically more efficient to allow single strands to anneal together, connecting up a series of single strand fragments, and then use ligase to seal the covalent bonds between adjacent fragments.

Polymerase Extension

Polymerase enzymes attach dnts to the 3' end of a short strand that is annealed to a longer strand. It then extends the 3' side of the shorter strand so as to build the complementary sequence to the longer strand. The short strand here is called primer sequence and the longer one is the template sequence.

Most of the polymerase enzymes do require only 3' end of the primer sequence to be open for adding new prescribed nucleotide but some polymerases like *terminal transferage* don't require any template for extension so they usually add ssDNA tails to both sides of a dsDNA.

On the other hand the extension of a single strand proceeds in 3' - 5' direction by fixing the 5' end to a rigid support.

Nuclease Shortening

The enzymes used to shorten a DNA strand (single or double)are known as *nucleases.* These are of two types - based upon whether strand is being cut from the end (exonuclease) or from the mid (endonuclease). Exonucleases remove one nt at a time from the end of DNA strand. Different exonucleases differ the way they cut (5' end or 3' end, single strand or the double, one end or both the ends).For e.g. *Exonuclease III* is a 3'- nuclease cutting at 3' end and leaving overhanging 5' ends, *Bal31* works on both the ends.

On the other hand *endonucleases* , which cut the DNA strand at the middle are as well somewhat specific - S1 cuts only ssDNA *(anywhere)*, *DNase1* works upon both ssDNA and dsDNA and at any point (not site specific). There is a special and useful class of endonucleases which are very much site specific, popularly known as **restriction enzymes** (cut a strand of DNA at a specific subsequence). There are over 2300 different known restriction enzymes that are specific to more than 200 different subsequences. These subsequences are usually on the order of 4 to 8 nucleotides. Some restriction enzymes will only cleave single stranded DNA, while others will only cleave double stranded DNA. Similarly, methylation of the Cytosine nucleotides on a strand of DNA interferes with the activity of some restriction enzymes, but not others (see fig. below for EcoRI restriction Activity). Furthermore, some restriction enzymes will cleave a wide range of subsequences of DNA. Again some restriction enzymes leave hanging single ends (staggered cut)while others cut straight away (blunt cut). That is, the specificity of activity varies across different enzymes. (See Appendix B for some more often used restriction enzymes with their restriction sites and cutting points). It may be noted that high temperatures deactivate most restriction enzymes except for those that have evolved in thermophillic archaebacteria.

Separate by Length

Gel electrophoresis is an important technique for sorting DNA strands by size [Brown93]. Electrophoresis is the movement of charged molecules in an electric field. Since DNA molecules carry negative charge, when placed in an electrical field they tend to migrate towards the positive pole. The rate of migration of a molecule in an aqueous solution depends on its shape and electrical charge. Since DNA molecules have the same charge per unit length, they all migrate at the same speed in an aqueous solution. However, if electrophoresis is carried out in a gel (usually made of agarose, polyacrylamide or a combination of the two) the migration rate of a molecule is also affected by its size (migration rate of a strand is inversely proportional to the logarithm of its molecular weight [OP94]).This is due to the fact that the gel is a dense network of pores through which the molecules must travel. Once the gel has been run, it is necessary to visualize the results. This is achieved by staining the DNA with the fluorescent dye ethidium bromide and then viewing the gel under ultraviolet light (the gel may be photographed for convenience). Each lane in the gel corresponds to one particular sample of DNA (Usually the term tube in used in abstract models [AmosTh97]). Therefore several tubes are run on the same gel for the purposes of comparison. One marker lane also remains present containing various DNA fragments of known length, for the purposes of calibration.

These basic manipulations can be combined into higher level manipulations.(See Appendix A)

1.4 Why DNA computing?

Though not a single experiment has been conducted so far, which can really put DNA computing at an edge over conventional computing devices, still from the theoretical point of view and due to some salient properties of of DNA molecule some clear advantages of DNA computing can be related as:

Speed. This should fall out from the massive parallelism of the approach, amortized across the slow serial operations. in fact the excitement of DNA computing lies mainly mainly its capability of massive parallel searches.

Energy efficiency. Since the molecules actually release energy when they anneal together, there is some hope that computations could be carried out using very little energy.

Information density. Packing 10^{20} strands of data into a liter of volume would give us an information density at least five orders of magnitude better than current hard drive technology.

As for example in Adleman's experiment [Adle94] the number of operations per second was up to 1.2×10^8 . This is approximately $1,200,000$ times faster than the fastest supercomputer. While existing supercomputers execute 10^9 operations per Joule, the energy efficiency of a DNA computer could be 2×10^{19} operations per Joule, that is, a DNA computer could be about 10^{10} times more energy efficient [Adle94] . Finally, according to [Adle94], storing information in molecules of DNA could allow for an information density of approximately 1 bit per cubic nanometer, while existing storage media store information at a density of approximately 1 bit per $10^{12} nm^3$. similarly as estimated in [Bea96], a single DNA memory could hold more words than all the computer memories ever made.

1.5 Some Disadvantages with DNA computing

As compared to conventional computing (keeping super computing devices in mind)where DNA computing has some definite advantages, it has some inherent difficulties as well, apart from currently present errors or impreciseness (dealt in next section).

Slow. algorithms proposed so far use really slow molecular-biological operations. Each primitive operation takes hours when you run them with a small test tube of DNA. Scale up to the vast amounts of DNA we're talking about, and they may slow down dramatically.

Hydrolysis. the DNA molecules can fracture. They have a life time of about six months, after that they break - meaning A DNA computer, essentially needs overhauling after six months - a costly affair!

Unreliable. every operation in DNA computing is random. The components in the DNA computer are probabilistic. Because there are some noisy components, the computing sometimes is not reliable. If a tiny subcircuit is supposed to give the answer "1," it may yield that answer 90 percent of the time and "0" the rest of the time.

Not transmittable. the model of the DNA computer is conceived as a highly parallel computer, with each DNA molecule acting as a separate process or. In a standard multiprocessor a Connection-buses transmit information from one processor to the next. But the problem of transmitting information from one molecule to another in a DNA computer has yet to be solved. Current DNA algorithms compute successfully without passing any information, but this limits their flexibility.

Not practical. DNA computing is not a here and now practical technology as yet rather than being a pie-in-the-sky research project.

No generality. till yet even though having universal DNA computing models no precise standardization is there to work out efficiently any problem (computable). Only some concrete algorithms are there for solving some concrete problems with some constraints on it.

1.6 Errors, Pitfalls,and Perils in DNA Computing with Possible Solutions

One of the major problem with the current state of DNA computing is the imprecise nature of the results due to errors associated with most of the tool-kit operations. Severity of the condition is such that when Kaplen and others ([Kaplan95]) tried to rework with the Adleman's experiment with negative control they got confusing answers. Moreover by now no one has experimentally demonstrated solving any problem with even moderately large instance size using DNA computing. Still it doesn't undermine the potential of using DNA, which nature is using million of years as genetic carriers.

Synthesis of a DNA strand can sometimes result in the strand annealing to itself and creating a hairpin structure. Even the seemingly straightforward mixing operation can sometimes pose problems: if DNA is not handled gently, the sheer forces from pouring and mixing will fragment it. Also of concern for this operation is the amount of DNA which remains stuck to the walls of the tubes, pumps, pipette tips, etc., and is thus **lost** from the computation.

Hybridization has also to be carefully monitored because the thermodynamic parameters necessary for annealing to occur are sequence dependent. This is important because, depending on the conditions under which the DNA reactions occur, two oligonucleotides can hybridize without exact matching between their base pairs. Hybridization stringency refers to the number of complementary base pairs that have to match for DNA oligonucleotides to bond. It depends on reaction conditions like salt concentration, temperature, relative percentage of A's and T 's to G's and C's, duration of the reactions and it increases with temperature. One *solution* for increasing hybridization stringency is the choice of good encodings for the input information of the problem, [DMRGFS97]. Another solution proposed in [Baum96] to avoid self annealing and mismatches is encoding using specially chosen sequences as spacers that separate the information bits.

Amplifcation by PCR is used with the assumption that by maintaining a surplus of primer to template one can avoid undesired templatetemplate interactions. As pointed out in [KGL96], this assumption is not necessarily valid. Indeed, experimental evidence points to the possibility of the creation of complex structures like folded DNA, complexes between several strands and incorrect *ligation* products. This might further affect the accuracy of using the *gel electrophoresis* technique for separation of strands by length. Indeed, in the presence of complex structures, the mobility of the strands will not depend only on their length, as desired, but also on the DNA conformation shape. As a *possible solution*, the use of singlestranded gels for analysis is recommended in [KGL96]. Moreover, by keeping concentrations low, heteroduplex doublestrands with mismatches in formation and template template interactions can be minimized.

Separation of strands by length and extraction of strands containing a given pattern can also be inefficient,and this might pose problems with scaleup of the test tube approach. An alternative methodology has been proposed [LGCCLS96]: the set of oligonucleotides is initially attached to a surface (of glass, silicon, gold, or beads). They are then subjected to biooperations such as marking, unmarking and destruction, in order to obtain the desired solution. This method greatly reduces losses of DNA molecules that occur during extraction by affinity purification. Its drawbacks are that it relies on marking and unmarking which, in turn, assume specificity and discrimination of singlebase mismatches. While these processes have proved reliable when using 15mer sequences, they become more difficult for shorter or longer polynucleotide strands. Another problem is that the scale of the computation is restricted by the twodimensional nature of the surfacebased approach: one cannot reach too high an information storing density. *Extraction* of those strands that *contain some given pattern* is also not efficient enough, and may at times inadvertently retain strands that do not contain the specified sequence. While the error rate is reasonable in case only a few extractions are needed, if the number of extractions is in the hundreds or thousands, problems arise even if 95% efficiency of extraction is assumed. Indeed, the probability of obtaining a strand encoding the solution, while at the same time obtaining no strands encoding illegal solutions is quite low. As another possible solution, in [AGH96] the operation remove was proposed as a replacement for extract. The compound operation remove removes from a set of strands all strings that contain at least one occurrence of a given sequence. The operation is achieved by first marking all the strands that contain the given sequence as a substring and then destroying the marked strands. The advantage of the method is that the restriction enzymes used for the remove operation have a far lower error rate than extraction. One of the drawbacks is that, although the initial tube might contain multiple copies of each strand, after many remove operations the volume of material may be depleted below an acceptable empirical level. This difficulty can be avoided by periodic amplification by PCR.

Cutting of a DNA strand by a restriction endonuclease is also referred to as digestion of the DNA by that enzyme. The process may sometimes produce partial digestion products. One must test all protocols for the effectiveness of the restriction enzyme used, and it is often necessary to find means to remove undigested material. Similarly, the accuracy of ligation is high, but not perfect.

A *ligase* may ligate the wrong molecule to a sticky end, if it bears a close resemblance to the target molecule. Detection and sequencing conventionally require enzymatic reactions and gel electrophoresis that are expensive and laborious processes. A possible solution to these drawbacks is using a technique that achieves sequencing by hybridization, offering a onestep automated process for reading the output, [Mir96]. In this method, target strands are hybridized to a complete set of oligonucleotides synthesized on a solid surface (for example an array containing all the possible 8mers) and then the target is reconstructed from the hybridization data obtained. However, to avoid errors arising from selfannealing, a restricted genetic alphabet is recommended with this method, using only two of the four bases. In this way, the test tube contents would be resistant to intramolecular reactions but not to intermolecular reactions.

Besides the accuracy of biooperations, another peril of the implementation of DNA computations is the fact that the size of the problem influences the concentration of reactants, and this, in turn, has an effect on the rate of production and quality of final reaction products. In [KMRS96], an analysis of Adleman's experiment showed that an exponential loss of concentration occurs even on sparse digraphs, and that this loss of concentration can become a significant consideration for problems not much larger than those solved by Adleman. For volume decreasing DNA algorithms, an error resistant solution seems to be the repeated application of PCR to the intermediate products, [BDSL96]. However, this cannot always be the solution, as not all algorithms are volume decreasing. Indeed, as pointed out in [Hart95], one barrier to scalable DNA computation is the weight of DNA. In some cases, ([ARRW96]), to achieve the desirable error rate, approximately 23 Earth masses of DNA were needed. Clearly, this is not an acceptable situation, and a combination of algorithm transformations (changing algorithms by using intelligent space and time trade offs might be required to reduce the amount of DNA) [ARRW96] ,[Adle96]. This section, which discussed implementation techniques and the associated error rates, indicates that many substantial engineering challenges to constructing a DNA computer remain at almost all stages. However, it can be pointed out that the issues of actively monitoring and adjusting the concentration of reactants, as well as fault tolerance, are all addressed by biological systems in nature: cells need to control the concentrations of various compounds, to arrange for rare molecules to react, and they need to deal with undesirable byproducts of their own activity. Some recent approaches based upon in vitro techniques ,as a step in this direction, is in [KMRS96] a mechanism is suggested, based on membranes that separate volumes (vesicles) and on active systems that transport selected chemicals across membranes (see section 2.2.5).

⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆

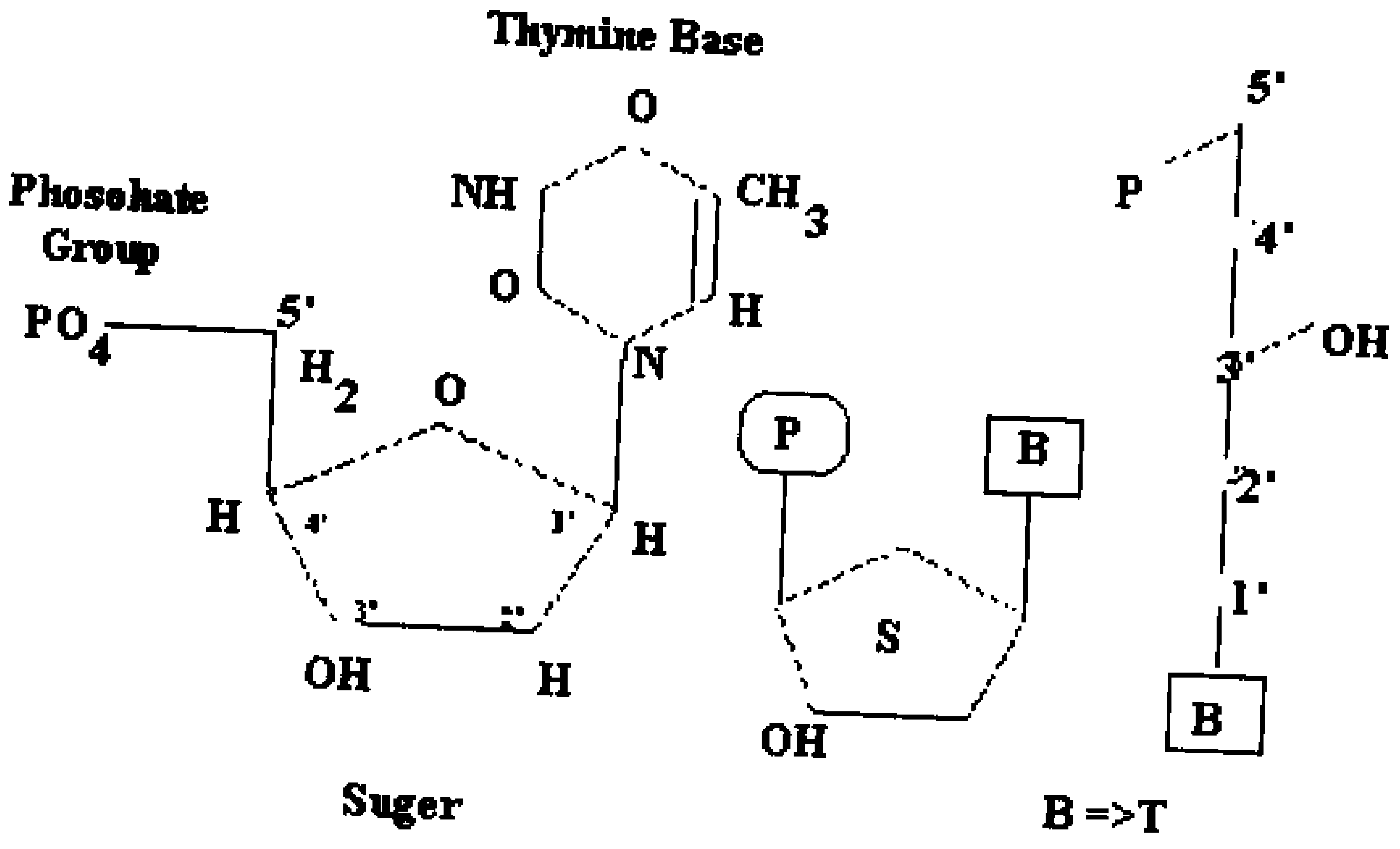


Figure 1.1: Chemical And Schematic Diagram of a DNA nucleotide with Thymine Base and Schematic Diagrams

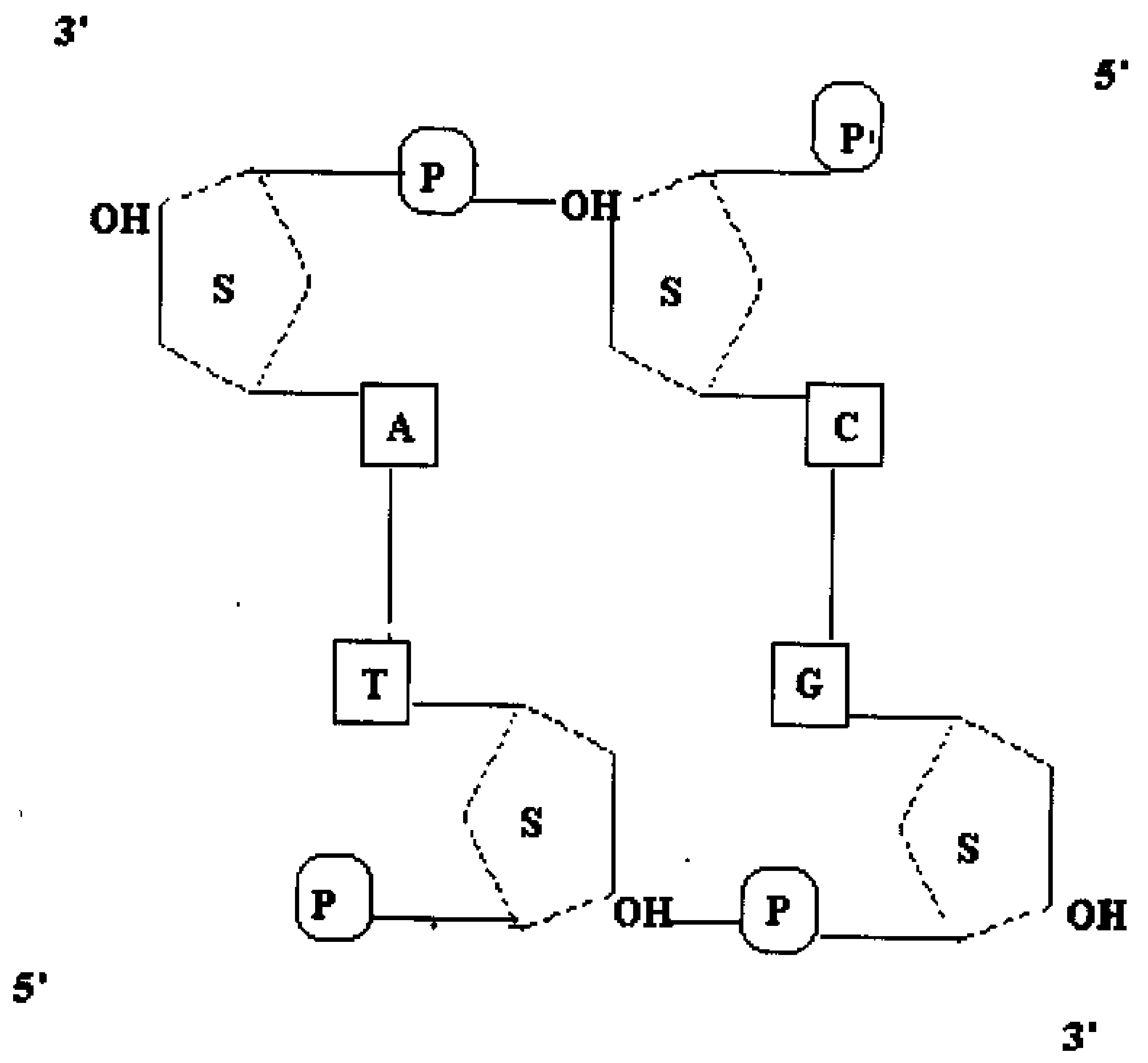


Figure 1.2: **Schematic Diagram of Structure of a DNA double Strand**

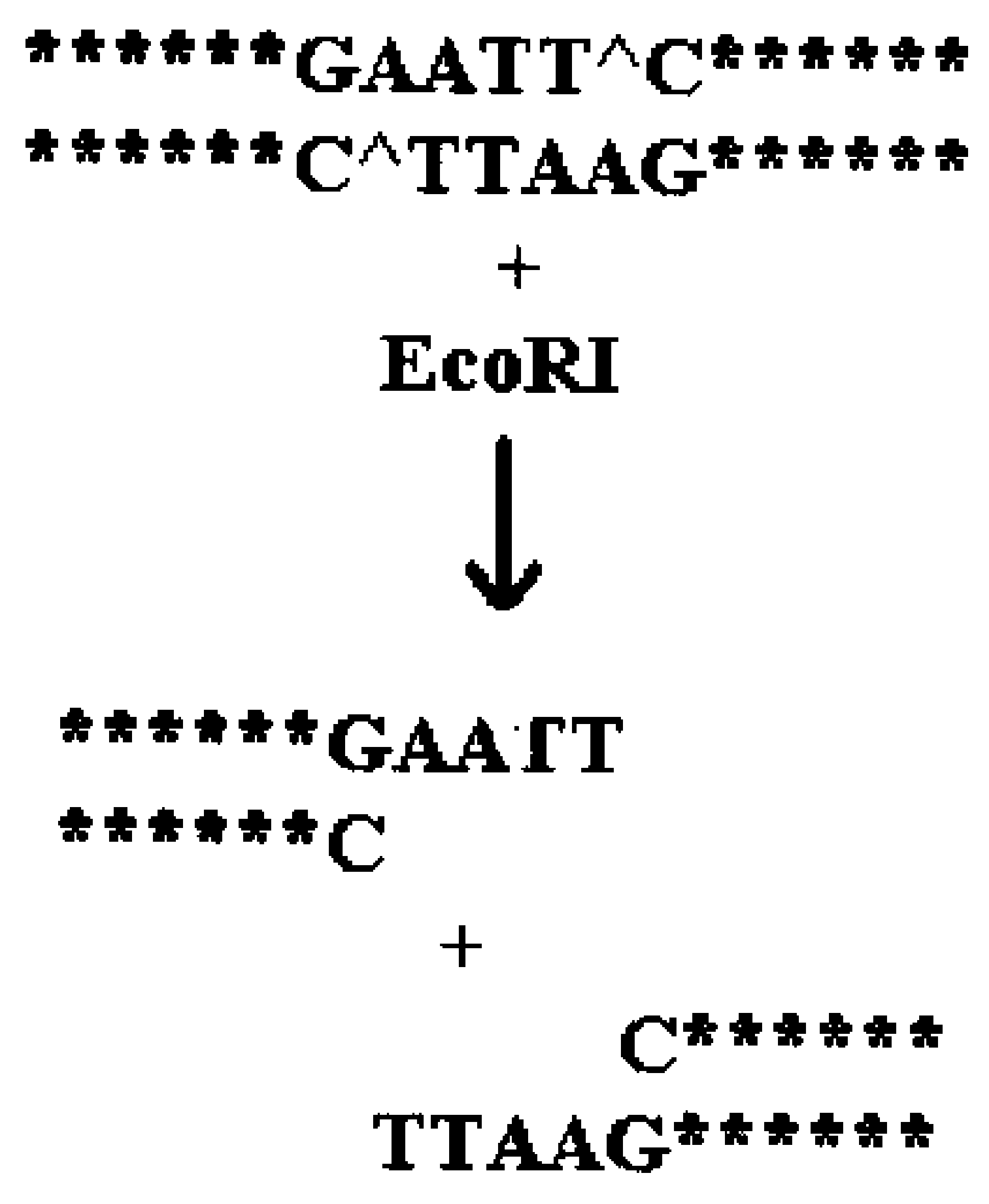


Figure 1.3: **Restriction Enzymetic Activity of EcoRI**

Chapter 2

Solving Problems with DNA Strands

HPP was the first problem to be solved using DNA strands. After that for several other problems, solutions have been proposed ranging from 3 SAT (a conjunctive boolean formula satisfiability problem with clauses of size 3), TSP (Travelling Salseman' Problem), Maxclique, 3 Coloring of planer graphs, Dynamic Programming, problem of evaluating transitive closure of a boolean matrix, Circuit (bounded fan in) evaluation and others. All these solutions are more or less based upon the massive parallelism [1], which allows to explore several structures (solutions or intermediate results) together, and Watson Crick complementarity, which ensures that these structures bind together to form desired solutions of a processing step.

Adleman's Solution: General idea behind the Adleman's solution of HPP was to encode vertices and edges of the graph in suitable way (as simplexes) to generate in the frist step all possible paths in the graph (as duplexes)[2] and then in second step discard all those paths which are not the correct solutions [3] using substrand extraction technique(see chap 1) for each vertex. Adleman encoded vertices and edges as 20 mer oligos such that no two oligos representing vertices were same and any edge (u,v) directed from vertex u to v was encoded by taking complement of second half of oligo for vertex u and first half of oligo for vertex v (thus preserving directionality of edges). Since in Adleman's case graph had a solution path so after filtering phase test tube contained DNA duplexes for these correct paths.

[1]Typical weight of a DNA molecule is around 550 Ds (dioptons : weight of a Hydrogen Atom) so generally in a test tube about 2^{70} DNA molecules can fit together.

[2]Solution Generation phase.

[3]Filtering or Processing phase

2.1 Working Models

After the Adleman's solution (and corresponding *filtering model*) several other models, charactrized by the kind of basic operations they allowed and their limitations, were proposed. Since DNA is the underlying computational substrate of all models described, its natural to assume that all abstract models operate on strings over a four letter alphabet,$\Sigma = \{AC, G, T\}$. A test tube T can be regarded as a finite multi - set of strings over Σ^*. The operation set within a model is also constrained by the availability of various molecular manipulation techniques. The implementation of abstract operations will largely determine the success or failure of a model. Most of the models described in this chapter use abstract operations common to the others, such as set union. However, even though models may utilize similar operations (e.g., removal of a string from a set), the chosen implementation method may differ from model to model. Details of implementation may also impact in various ways ([GAH97]):

1. The volume of DNA required (analogous to space in complexity theoretical terms) to perform the computation may vary by exponential factors.
2. Each operation takes a certain amount of time to implement in the laboratory, and so the sequence of operations performed determines the overall time complexity of the algorithm. Thus, the techniques chosen have a direct bearing on the efficiency of a DNA based algorithm. In addition, the time taken to construct the initial set of strings and readout the final solution may be very timeconsuming and must also be taken into account.
3. Each laboratory technique has associated with it a nonzero error rate. Some techniques are far more error prone than others, so the choice of laboratory techniques directly affects the probability of success of a DNA based algorithm.

Some of the more popular models are:

2.1.1 Adleman's Restricted Model

As described in [Adle96] the basic set of operation are

- *merge*$(T_1, T_2, \cdot, T_n)$. Make union of tubes $T_1, T_2, \cdot, T_n$ by poring contents of all tubes together.
- *separate(T,s)*. Extract all the strands from tube T containing *s* as a substring. Let the resultant tubes be +(T,s)(containing s) and - (T,s) (rest not containing s).
- *detect(T)*. Check if tube T is empty?

Based upon this simple set of operation graph 3 - Coloring problem's solution can be formulated as following:

Let T contains all possible colorings of graph G = (V,E)with n vertices $\{v_1, v_2, ..., v_n\}$on 3 colors *r, b, and g*. Let c_i to denote that vertex v_i is colored with color $c \in \{r, b, g\}$.

```
for i = 1 to n
 {
        $T_r := +(T, r_i) \quad and \quad T_{bg} := -(T, r_i)$
        $T_b := +(T_{bg}, b_i) \quad and \quad T_g := -(T_{bg}, b_i)$

  for all j such that $(v_i \to v_j) \in E$
   {
        $T_r := -(T_r, r_j)$
        $T_b := -(T_b, b_j)$
        $T_g := -(T_g, g_j)$
   }

        $T := merge(T_r, T_b T_g)$
        $detect(T)$
 }
```

comments

As can be seen restricted model of Adleman is simple enough for using only 3 basic operations. And it's powerful to solve graph 3 - Color abilityy problem (NP complete) in $O(n^2)$bio steps. So in principle it can solve any NP complete problem in polynomial time. But the main problem of the model is that it lacks strong expression power, that is, to encode any other problem it might be very difficut to express only in these terms and requires exponentially increasing number of DNA strands in solution generation phase. Moreover it's not univershal in nature and infact addition of splicing operation makes it universal[Rowis96]. Again the Adleman's implementation of saperate is based upon complementary annealings, which is not an error free (see sec. 1.5). The main significance of this model seems to be in promoting other models based upon it, like Liptons, Amos' etc. As proved in [Winfree95]the class of functions which the restricted model can invert (compute) in given time is exactly those computed by the branching programs of the same size.

That is, $RestrictedModel \sim BranchingPrograms$

2.1.2 Unrestricted Model

This model adds one more operation to the restricted model:

- *Amplify(T).* Make two copies of tube T as $T_1 and T_2$ using PCR.

Comments This model is more powerful than the restricted one since the class of functions which the unrestricted model can invert (compute) in given time is exactly those computed by the nondeterministic branching programs of the same size [Winfree95].

That is, $UnrestrictedModel \sim NondeterministicBranchingPrograms$

2.1.3 Lipton's Model

Lipton in [Lip94] describes solution to SAT problem based upon the Adleman's model. He proposes that after generating all 2^npossible assignments for a n variable SAT problem, consisting of k clauses in conjunctive (normal) form , in the processing or filtering phase each and every clause is scanned from the first for its every litral. And all the strands which do not set the value true are filtered out. Thus at last the tube contains the final satisfying set of assignments , if at all.

Comments

Lipton's solution to SAT problem solves the problem in O(km) bio steps, where m is largest number of variables present in any clause but using exponential number of strands. Other feature of his technique is that the filtering phase not only generates the satisfying assignment but gives approximation for the number of satisfying assignments also, thus solving more than mere NP complete problem.

2.1.4 Amos' Model

Martin Amos, Alan Gibson and others describe in [AGH96] a less error prone filtering based model. This model differs from the earlier ones, the way substrand extraction operation is performed - they construct the strands of type - $p_1 i_1 p_2 i_2 ... p_n i_n$, where $i_1, i_2, ..$ are the value strands (defined according to the problem structure) and $p_1, p_2, ..$ are position markers, which contain different restriction sites in them. Here in filtering phase the strands containing wrong value strands are destroyed by cutting them with restriction enzymes. For e.g. they propose that the graph 3 - Coloring problem for a graph on n vertices can be solved as follows:

1. generate all possible DNA double strands of type - $p_1 c_1 p_2 c_2 ... p_n c_n$, where each i_j takes 3 different values r, g ,b and p_j work as position marker at position j ranging from 1 to n. Thus the tube contains all possible 3_n colorings.

2. for every vertex remove (destroy) all the strands which do not color it with color $c \in \{r, g, b\}$ or color its neighbors with color c (in parallel for all 3 colors)

3. select the tube to check if the tube contains any strings (final solutions) or not.

In Amos' model the r*removal* of all those strands which contain any of the given subatrands is carried out in parallel by adding the required restriction enzyme (SauMI) in the tube. similarly they propose as a basic operation parallel *copy* opearation ,which develops several of copies of same tube They also propose solutions to other hard problems like *Subgraph Isomorphism, HPP, Maximum Independent Set* and describe in their model the initial stages of implementation for graph 3 - coloring problem ([AmosTh97])

Comments

Amos model like previous models is simple enough and error probability is less due to their restriction enzyme [1] implementation of operation *remove*, which in effect actually achieves the same results as the extract operation in Adleman's models.

One of the serious problem with their model is that implementation of remove operation is limited by the number of available diffrent restriction enzymes with different restriction sites, which at present don't go beyond 200 mark, that is, no more than 200 different values can be considered in their model. Another limitation of increasing exponential volume as noted earlier is as well present with their model.

2.2 Theoretical Models

Main aim behind developing the above discussed working models was to show better feasibility in respect to error rates, choice of operations requiring different times, for solving some specific hard problems. That way these models were not particular about claims of universality in computation. On the other hand some other models were proposed which fundamentally aimed the underlying universal computing ability by having a selected choice of operations. Some of these models were inspired by language theoretic developments, for e.g., 'twin - shuffle' languages, splicing systems, L systems, Watson - Crick Autometa (see [PRS98]). While others were proved universal by exhibiting their turing machine simulation [Bea96, Rot96], or other universal system (see for example Agihara and Ray's Boolean circuit simulation, and cellular Automata simulation by Winfree's model of self - assembly of DNA tilings [Winfree96]).

[1]see ref([15] p 9) in ([AmosTh97]) for the fact that restriction enzymes cut the site with 100% success .

2.2.1 Ogihara and Ray's Boolean circuit model

Boolean circuits are an important Turingequivalent model of parallel computation (see [Harrison65]). An n input *bounded fan in* Boolean circuit may be viewed as a directed, acyclic graph, S, with two types of node: n input nodes with in degree (i.e., input lines) zero, and gate nodes with maximum indegree two. Each input node is associated with a unique Boolean variable x_i from the input set $X = (x_1, x_2, :::, x_n)$. Each gate node, g_i is associated with some Boolean function $f_i \in \xi$ where ξ is set of universal logic function[1]. S has in addition, one unique output node, s, with out degree zero. The *size* of a circuit S is the number of gates in S; and its depth, d, is the number of gates in the longest directed path connecting an input vertex to an output gate. In [OG96], Ogihara and Ray describe the simulation of Boolean circuits within a model of DNA computation. The basic structure operated upon is a tube, U , which contains strings representing the results of the output of each gate at a particular depth. The initial tube contains strands encoding the values of each of the inputs X. Strands $ds_i(1)$ $(ds_i(0))$ are synthesized for the output 1(0) of each gate g_i in S. Gates are scanned (in parallel) starting from the first level (input gates) to the output level, if gate $g_i$is OR gate ($\vee$), then all the strands which set either of its inputs to 1 are appended with its output strand for 1 $(ds_i(1))$ and rest strands are appended with $ds_i(0)$. Similarly in case of AND ($\wedge$) gate as well.

Comments As is the case with most of the theoretical models things are not yet practical. Here it can be easily seen that simulation though correct yet misses implementation practicality - scanning each gate and deciding what to add where is as time consuming as obtaining the answer right away. Moreover as pointed out in ([AmosTh97])that in more realistic strict model this mechanism of simulating the circuit takes bio - steps of the order of size of the circuit not as the depth of the circuit as required by procedure above, in their model.
One of the important contributions of Ogihara and Ray's constructive model is in direction of reducing volume of the tube by giving an algorithm requiring only liner amount of different strands with the size of the circuit. Their technique has been recently used by Yoshid and Suyams [YoSh00] for solving 3 - SAT problem using $2^{.58n}$ volume. Similarly S. Pyne [Pyne01] also proposes an "inverted model" for solving 3- coloring problem on random graphs using constant number of strands O(1) on average with polynomial number of bio - steps.

2.2.2 Sticker Model

In [Rowis96] Rowis and others introduce the socalled sticker model. Unlike previous models, the sticker model has a memory that can be both read and written to, and employs reusable DNA. Each string is composed of k bits,each encoded by a substring of some defined length (about 20 to 30 mer and bp). The sticker model

[1]Set of logic functions which can express all other functions, for e.g., NAND, or $\{\vee, \wedge, \neg\}$

represents a 0 as a sequence of ssDNA and a 1 as a sequence of dsDNA. The initial and basic structure is a single fixed length strand representing a string of 0's. Bit positions are encoded by unique oligos.

Comments The problem of clearing a single bit position without clearing all the bit positions has not yet been solved and remains a limitation of the sticker model at present. one possibe but poor way in order to set bit i to 1 is that each string is annealed to the complement of the sequence representing bit i. However, if we wish to clear a bit (i.e., set it to 0) we must remove the annealed strand. This can only be done by heating the solution in which the strands are suspended, resulting in all hydrogen bonds being broken and all bits being cleared.

2.2.3 Splicing Models

One of the reasons that working models are not expected to be universal in nature is that they don't support string rewriting (editing) operations. This makes it difficult to see how the transition from one state of the Turing Machine (if simulation is being done of Turing machine or any equivalent of it) to another state may be achieved using DNA. However as it is provedone further operation, called splicing operation, can provide full Turing computability .

Splicing operation Let S and T be two strings over the alphabet V. Then the splice operation consists of cutting S and T at specific positions (restriction sites)and concatenating the resulting prefix of S with the suffix of T and concatenating the prefix of T with the suffix of S .

Formally, a splicing rule is a string of the form r = $u_1\#u_2\$u_3\#u_4$ where $\#$ and $\$$ are two special marker symbols not in V , and $u_i \in V^\star (1 \leq i \leq 4)$. For such a rule r, applying it to two strings x, y results in a string z such that
$(x, y, z \in V^\star)$
$(x, y) \vdash_r z \Longleftrightarrow x = x_1u_1u_2x_2$, $y = y_1u_3u_4y_2$, and $z = x_1u_2u_4y_2$, for some
$x_1, x_2, y_1, y_2 \in V^\star$.

Splicing Systems

Based upon the the above rule there are some of the restricted Splicing systems can be defined as

H scheme. An H scheme is a pair σ = (V, R) where V is an alphabet and R $\subseteq V^\star\#V^\star\$V^\star\#V^\star$ is a set of splicing rules.
An H system σ = (V; R) is used as a unary operator on languages. Applying σonce to a language $L \subseteq V^\star$ yields
$\sigma(L) = \{z \in V^\star \mid (x, y) \vdash_r z$; for some $x, y \in L, r \in R\}$

This can be used to study a single application of an H scheme. It can be extended to iterated application $\sigma_\star$ as follows -

$$\sigma^0(L) = L;$$
$$\sigma^{i+1}(L) = \sigma^i(L) \cup \sigma(\sigma^i(L)) i \geq 0,$$
$$\sigma_\star(L) = \cup_{i \geq 0} \sigma^i(L)$$

An H system, though normally viewed as an operator, can be likened to productions in the grammars of classical Formal Language theory. Classical grammars are 'complete' devices for generating languages. They specify an alphabet, a starting point, rules for combining generated strings into new ones, and terminal symbols out of which the output strings may consist. Associated with them is one interpretation - the language generated by the grammar. Following this analogy closely, the concept of **extended H system** is defined as a quadruple:

$$\mu = (V, T, A, R)$$

where V is an alphabet,$T \in V$is the terminal alphabet, $A \in V^\star$ is the set of axioms, and R $\subseteq V^\star \# V^\star \$ V^\star \# V^\star$is the set of splicing rules. For such an extended H system μ= (V; T; A; R), an underlying H system is σ= (V; R) .

The language generated by μ is defined as follows:

$$L(\mu) = \sigma^\star(A) \cup T^\star$$

Variations of Splicing Systems

Similar to traditional grammars' computational/generative tools - the form of the productions ,within extended H systems, there are two generative tools - the classes of languages from which A, the set of axioms, and R, the set of splicing rules, are taken. E.g. [HPP96] shows that when both A and R are finite, extended H systems can produce regular languages; and that when A is kept finite, but R regular, the recursively enumerable languages can be produced. Some of the more obvious variations of splicing systems are in which an H system is applied only once, and splicing systems in which all symbols are terminal. Splicing systems on multi - sets of strings, are of practical interest, since they can accurately model the fact that strands are consumed in a splicing operation, as are splicing systems on circular strings, studied in e.g.,[YK97], which can model the behavior of circular strands. Both of these variations can achieve universal computation for A and R from simple families in the Chomsky hierarchy. There are even generalizations of splicing to graphs and other non - string- like structures.

Practical Splicing Systems

While splicing systems are interesting in themselves as abstract models in Formal Language theory, they are of special interest mostly for their original purpose to model the languages of double strands of DNA generated under the domain of restriction enzymes and ligases.

Implementation Considerations. There are a number of aspects in which splicing systems abstract away from practical biochemical limitations that become important again, when considering practical implementation. First, in a practical model, the amount of initial strands and the number of different restriction enzymes is finite, so both the initial set and the set of axioms in a corresponding model will have to be finite. Secondly, in practice, DNA strands are consumed in splicing when strands w and z are generated from strands x and y, x and y are no longer available. This requirement is quite strict and may not be demanded in full in most cases, the model still works when we assume a large, but finite, supply of all strands involved. Thirdly, the length of a recognition site of a restriction enzyme is limited to 6 to 8 bases - restriction enzymes cannot recognize arbitrarily long sequences.

Candidate models for universal computation based on splicing When we take into account the requirements just formulated, there are a few splicing system models that are practical, and capable of universal computation. One is splicing systems based on multi - sets, as introduced in [DG89],with universal computational power. Another is that of splicing systems for circular strings, as studied in [YK97]. Lastly, [Pi96] proved the existence of a universal (for a given alphabet) multi - set splicing system with finite axioms.

Difficulties

Splicing Systems at first sight appear to be an attractive model for developing practical Molecular Computation. However, there are as yet several severe problems that hinder their applicability in this way. *Only one type of chemistry.* Splicing Systems were explicitly developed as models for DNA re combination. There are several other chemistries on which practical Molecular Computation might be based, like RNA editing, or the 'weird' DNA complexes used by Winfree. Focusing on Splicing Systems as the theoretical model for Molecular Computation would be voluntarily blinding oneself to the other possibilities, for some of which theoretical models to study their computational power may still have to be developed.
Unrealistic splicing. We have seen several barriers to directly implementing splicing systems. For example, restriction enzymes are capable of recognizing only rather short (6 to 8 base pairs) sequences, while splicing rules can recognize finite, but arbitrarily long, subwords. Any potentially practical splicing system will have to use only a few restriction enzymes, since they are quite expensive, and function optimally under diverse reaction circumstances.
Finiteness. All practical systems have to be finite. Incorporating this finiteness directly into splicing systems (by using multi - sets, a finite number of axioms and a finite number of splicing rules) easily results in models that are not capable of universal computation. In splicing systems, the number of tools that can be tuned to produce a finite system is larger, and the tools are less well understood than memory. The theory of Splicing Systems can enhance into approximations that may produce

practical models. For instance, if a universal splicing system with a small number of splicing rules, but which requires a regular set of axioms, an approximation would be to use a large but finite subset from these axioms, and see how much practical computing power is lost.

2.2.4 Contextual Insertion/Deletion systems

Generalizations of contextual insertion/deletion of words were discussed in [Kath96]. Authers study closure properties of the Chomsky families under the defined operations, contextual insclosed and delclosed languages and decidability of existence of solutions to equations involving these operations. Moreover, they prove that every Turing machine can be simulated by a system based entirely on contextual insertions and deletions. Since contextual insertion and deletion are can be thought of relizable in DNA computing by manipulations of restriction enzymetic activities with PCR and some other tools from molecular biotechnology, therefore theirat least establishes the universality of these operations and gives hints regarding the degradation of generative power under more realistic restricted conditions.

Given a pair of words (x, y)[1] called a *context*, the (x, y)contextual insertion of a word v into a word u is performed as follows. For each occurrence of xy as a subword in u, we include in the result of the contextual insertion the words obtained by inserting v into u, between x and y. The (x, y)contextual deletion operation is defined in a similar way. The contextual insertion operation is a generalization of the concatenation and insertion operations on strings and languages - words can be inserted into a string only if certain contexts are present. More precisely, given a set of contexts we put the condition that insertion of a word can be performed only between a pair of words in the context set. Analogously, contextual deletion allows erasing of a word only if the word is situated between a pair of words in the context set.

Insertion and deletion schemes

An insertion scheme INS is a pair INS = (X, I) where X is an alphabet with more than one symbols and I $\subseteq X^\star \times X^\star \times X^\star$, I $\neq \phi$. The elements of I are denoted by $(x, z, y)_I withx, y, z \in X^\star$ and are called the contextual insertion rules of the scheme. For every word u $\in X^\star$, let

$$conxt\_ins_I(u) = \{v \in X^\star \mid v \in u \leftarrow_{(x,y)} z, (x, z, y)_I \in I\} \ where$$
$$u \leftarrow_{(x,y)} v = \{u_1 xvyu_2 \mid u_1, u_2 \in X^\star, u = u_1 xyu_2\}$$

the (x,y)- $contxualinsertionofv \in X^\star intou \in X^\star$.

[1]under the same DNA alphabet {A,C,G,T}

Informally, in a contextual insertion rule (x, z, y), the pair (x, y) represents the context of insertion while z is the word to be inserted. If L $\in X^\star$ and I is fixed, then

$$conxt\_ins_I(L) = \{cins_I(u) \mid u \in L\}$$

.

A deletion scheme DEL is a pair DEL = (X, D) with D$\subseteq X^\star \times X^\star \times X^\star, D \neq \phi$. The elements of D are denoted by
$(x, z, y)_D withx, y, z \in X^\star$ and are called the contextual deletion rules of the scheme. For every word u $\in X^\star$, let

$$conxt\_del_I(u) = \{v \in X^\star \mid v \in u \rightarrow_{(x,y)} z, (x, z, y)_D \in D\} \ where$$
$$u \rightarrow_{(x,y)} v = \{u_1xyu_2 \mid u_1, u_2 \in X^\star, u = u_1xvyu_2\}$$

the (x,y) - *contxual deletion* of v $\in X^\star$ from u $\in X^\star$. Informally, in a contextual deletion rule (x, z, y), the pair (x, y) represents the context of deletion while z is the word to be deleted. If L $\in X^\star$ and I is fixed, then

$$conxt\_del_D(L) = \{conxt\_del_D(u) \mid u \in L\}.$$

An *insdel system* ID is a quintuple, ID = (X, T, I, D, w) (X, I) is an insertion scheme, (X, D) is a deletion scheme, I, D are finite, T $\subseteq$ X is the terminal alphabet, and w $\in X^+$is a fixed word called the axiom of the insdel system.
If u$\in X^\star$ and $v \in conxt\_ins_I(u) \bigcup conxt\_del_D(u)$, then v is said to be directly ID-derived from u and this derivation is denoted by u $\Rightarrow$ v. The sequence of direct derivations -

$$u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k; k \geq 1$$

is denoted by $u_1 \Rightarrow^\star u_k$ and u_k is said to be derived from u_1.

The *language* $L_g(ID)$ *generated* by the insdel system ID is the set -

$$L_g(ID) = \{v \in T^\star \mid w \Rightarrow^\star \ v \ where \ w \ is \ the \ axiom \ \}$$

and analogously the *language*$L_a(ID)$ *accepted by* the insdel system can be defined as

$$L_a(ID) = \{v \in T^\star \mid w \Rightarrow^\star \ v \ where \ w \ is \ the \ axiom \ \}$$

Result

If a language is acceptable by a Turing machine TM, then there exists an insdel system ID accepting the same language. (see proof in [Kath96])

2.2.5 P Systems - Computing with Membranes

Introduced in [Paun98] Membrane[1] Computing structures are inspired by biochemistry (from the way the living cells process chemicals and energy) and which proposes cell - like computing models of a distributed parallel type. Basically, in the regions defined by a membrane structure one places multi sets of objects which can evolve according to given evolution rules. The rules are used in a maximally parallel manner, nondeterministically choosing them and the objects to which they are applied. The objects (described by symbols from a given alphabet) can interact and can pass (selectively) through membranes, while the membranes can be dissolved or can be divided. By such operations evolution happens through transitions from a configuration of a system to another configuration, hence we get computations. A result is associated with a halting computation in the form of the number of objects present in the last configuration within a specified membrane. One of the important variants of such computing structures are P systems, which can compute all recursively enumerable sets of natural numbers. When an enhanced parallelism is provided, by means of membrane division (and, in certain variants where one works with string - objects, by means of object replication), NP - complete problems can be solved in linear time (of course, making use of an exponential space).

No experiment implementing such systems in biochemical media have been done, but there have been several attempts to simulate/implement P systems on usual electronic computers. One of the potaintial natural simulations of P systems are DNA controlled structures where DNA computing can play an active role. Another direction is hinted in [Paun98] by condidering a variant, with the objects being strings over a given alphabet. The evolution rules are now based on string transformations. they investigate the case when either the rewriting operation from Chomsky grammars (with respect to contextfree productions) or the splicing operation from H systems investigated in the DNA computing is used. In both cases, characterizations of recursively enumerable languages are obtained by very simple supercell systems[2] : with three membranes in the rewriting case and four in the splicing case.

[1]A membrane is a construct which works like a delimiter for a group of objects in a selective fashion. For e.g. biology and chemistry membranes keep together certain chemicals and leave to pass other chemicals, in a selective manner, sometimes only in one direction.

[2]a supercell system is a membrane structure with objects in its membranes, with specified evolution rules for objects, and with given input output prescriptions. Any object, alone or together with one more object, evolves, can be transformed in other objects, can pass through one membrane, and can dissolve the membrane in which it is placed. All objects evolve at the same time, in parallel; in turn, all membranes are active in parallel. The evolution rules are hierarchical by a priority relation, given in the form of a partial order relation; always, the rule with the highest priority among the applicable rules is actually applied. If the objects evolve alone, the system is said to be noncooperative; if there are rules which specify the evolutions of several objects at the same time, then the system is cooperative; an intermediate case is that where there are certain objects (we call them catalysts), specified in advance, which do not evolve alone, but appear together with other objects in evolution rules and they are not modified by the use of the rules.

2.2.6 Bearver Model

In [Bea95, Bea96], Donald Beaver designed a deterministic Turing Machine consisting of a single DNA strand. At each step of the computation that we are going to describe, a DNA molecule encodes a configuration of the Turing Machine: the content of the tape, the current state and the head position. Each state transition requires O(1) laboratory steps to be carried out. Finally, such method can be extended to perform the computation of a nondeterministic Turing Machine.

Context Sensitive Substitution Beaver proposes a specific form of site - directed mutagenesis - a small modification in a strand whose location is determined by a specific base sequence - to implement this substitution. The replacement of a substring α X β by α Y β in a string Lα X βR, in which α, β X, Y are finite strings over alphabet Σ = {A,C,G,T } and where neither α, nor β occur elsewhere in the string.

To implement, double strands containing $\alpha X \beta$ are converted to single strands. Strands of $\alpha Y \beta$ are added, whose α and β parts anneal to their complements $\overline{\alpha}$ and $\overline{\beta}$. The result, except the X and Y sequences that cannot bind, is then made double - stranded via PCR. This results in double strands that are properly aligned except for the part between α and β where substitution is to take place. These strands are melted into single strands again; primers that anneal to starts of L and ends of R are added, and PCR is performed. Now tube contains two types of strands: double - strands that encode for LαYβR that are to be kept, and single strands LαXβR, which need to be removed. These undesired single strands are destroyed by cutting with S1 nuclease. Thus the substitution is completed.

Turing Mechine Simulation

Let M = (Q, Σ, δ, s_0) be a deterministic Turing Machine where Q is the state set (containing the halt state h), Σ is a finite alphabet, $\delta$is the state transition function, and s_0 is the start state. First, states and symbols are codified using the alphabet of the four nucleotides: Σ = {A, C, T, G }. Since Q and Σ are finite sets, such encoding is obviously possible. We denote with the string $\alpha q v \beta$ the configuration of the Turing Machine M that is in the state q $\in$ Q, whose head points to the beginning of a string $v \; in \Sigma^*$ that has the string α on its left end the string β on its right. Let S(n) be the polynomial length of M 's tape. The entire configuration of M is encoded in a DNA single strand in the following way: the presence of the substrand u = E(i, v_i) indicates that the symbol $v_i \in \Sigma$ is in the i^{th} position of the tape. In the same way, the presence of the substrand v = E(j, q) indicates that M is in the state q $\in$ Q and that the head points at the j^{th} position of the tape. Thus, for example, the configuration $v_1 v_2 \ldots v_{i-1} q v_i v_{i+1} \cdots v_{S(n)}$ is codified by the DNA strand E(1, v_1)E(2, v_2)$\cdots$ E(i - 1, v_{i-1})E(i, q)E(i + 1, v_{i+1})$\cdots$ E(S(n), $v_{S(n)}$) where each E(k,v) or E(k; q_h) with $k \in \{1; 2; : : : ; S(n)\}$; $\sigma \in \Sigma$; $q_h \in$ Q is an opportune random substrand.

In order to perform a state transition, once a DNA single strands encodes a configuration, the idea is of substituting the portion of the DNA strand that is worked upon by the transition.let us suppose that the Turing Machine is in the state q, the head position is i where the symbol $v_i$is pointed. That is, the encoding of the configuration contains the substrand

$$E(v_{i-2}, i-2)E(v_{i-1}, i-1)E(q, i)E(v_i, i)E(v_{i+1}, i+1)$$

Moreover, the state transition that must be performed is

$$\delta(q, vi) = (q', v', \ left)$$

where $q, q' \in Qand v_{i-2}, v_{i-1}, v_i, v_{i+1}, v' \in \Sigma$. The substitution that performs this state transition is L$\alpha X \beta R \rightarrow L\alpha Y \beta$R, where

$$\begin{aligned} \alpha &= E(v_{i-2}; i-2), \\ X &= E(v_{i-1}; i-1)E(q; i)E(v_i; i), \\ \beta &= E(v_{i+1}; i+1), \\ Y &= E(q'; i-1)E(v_{i-1}; i-1)E(q'; i). \end{aligned}$$

while L and R are respectively the left and the right side of (the encoding of) the rest of the configuration. In this way a new configuration is encoded by the DNA strand that contains the new substrand

$$E(v_{i-2}, i-2)E(q', i-1)E(q', i)E(v_{i-1}, i-1)E(v_{i+1}, i+1)$$

In order to give the input to the Turing Machine, a initial testtube $T^0$is created. Let us suppose that the start state is q^0 and the tape contains the word x as an input, the initial configuration would be q^0x followed by $S(\| X) - \| x$ number of zeros. and its encoding should be the content of T^0 . Moreover, at the end of the computation the final state must be recognized (we can suppose that there is only one halt state) and a result (accept or reject of the input) must be given. We can suppose that the input has been accepted if and only if the symbol '1' is in the first position of the tape at the end of the computation (and vice versa if the first symbol is '0', then the input is rejected). Thus, the biological operation

Detect(Extract(T, E(halt, 1)E(1, 1)))

realizes the termination. It extracts all the strands that contain the substrand E(halt; 1)E(1; 1), that are the configurations of the Turing Machine that reached the halt state and accepted the input (an analogous extraction with E(0; 1) instead of E(1; 1) would extract final configurations where the input is rejected). If at least one of such configurations exists (that is, the detect operations finds there are DNA strands that have been extracted), the computation terminates and the opportune result is given.

Finally, Beaver suggests to extend his method to simulate nondeterministic Turing Machine computations: the strands (that encode configurations) can be amplified at each step reaching a sufficient number to perform all the possible state transitions. That is, if the current state is q and the pointed symbol is oe, the strands are amplified and separated in $| \delta(q, v) |$ different testtubes. In this way, in each testtube an opportune state transition is performed. Beaver asserts [Bea95] *such unconstrained parallelism allows to perform any PSPACE computation, far beyond NP* and recognizes that molecular massive parallelism is not exponential parallelism. This is caused by the practical restrictions that are introduced (for the solution of NPcomplete problems require an exponential number of molecules, because each candidate for the solution needs at least one molecule to be encoded).

Comments

There are several problems with Beaver's implementation of the substitution operator, noted in [Bea95]. Unintended complexes can form when the $\overline{\alpha}$ sequence of an $\alpha Y \beta$ sequence anneals to a different strand than its $\overline{\beta}$ sibling. This may be prevented by temporary attachment of the strands on which substitution is to take place to solid support. Another solution might be to use circular strands, causing unintended complexes to have improper length, and filtering them out using gel electrophoresis. For useful Turing machines, the length of a tape is unpredictable, and this trick cannot be used (unless one is willing to give up on potential unlimited tape length, and a priori choose a maximum length). PCR is involved twice in every substitution, although it is slow, expensive and error - prone ([KGL96]). The digestion of single stranded DNA is done with S1 nuclease, which can work on double stranded DNA too, and which requires reaction circumstances that can destroy information stored in DNA.

2.2.7 Rothemund Model

In [Rot96], a DNA restriction enzyme implementation of a Turing Machine is suggested. Restriction enzymes are employed in nature by bacteria to cut DNA double strands at specific substrands called restriction sites. Many of them cut DNA in two pieces leaving sticky ends on the two new strands (see sec. 1.3 for details). There are enzyme like EcoRI, which belong to the class II endonucleases having the properties of this type of enzymes that the two single strands that compose the restriction site are identical: since each DNA strand has a polarity, and since two complementary single strands are anti parallel, if read for example, the restriction site of the enzyme EcoRI (see Appendix B) from the 5' end to the 3' end (or vice versa), that they are the same strand: GAATTC. Consequently, also *the two sticky ends generated by the cut are identical.* This symmetry permits that a sticky end of a DNA strand cut by the enzyme can anneal to a sticky end of another DNA strand that has been cut by the same enzyme. This property of the class II endonucleases enzyme is the ground of Rothemund's idea of implementing a Turing Machine. In few words, any instantaneous description of the Turing Machine is encoded in a DNA strand and the state transition is realized cutting the strand at an opportune substrand (recognized by the enzyme associated with the transition) and inserting a new (opportune) substrand between the two sticky ends to obtain the new instantaneous description. To be more precise, the elements of the Turing Machine are distinguished in constants and variables. The transition table is constant, while the instantaneous description (also ID from now on) is variable. Each ID is encoded in a single circular DNA molecule, called plasmid. The state transition table - – that is the constant information—is encoded permanently by oligonucleotides. These strands are opportunely chosen and each one of them realizes—with few chemical steps—the correspondent transition and it modifies the plasmid into a new one representing the new ID. This is obtained in the following way : the plasmid encodes the symbols of the tape concatenating double stranded fragments that encode each element. The current state and the head position are also encoded in the plasmid. They both appear near the encoding of the pointed symbol and they consist in subsequences that contain a restriction site. Each state has its site and—consequently—its enzyme. The site (and then the enzyme) also depends on the symbol that the head is reading; thus, there is a different site for each entry in the transition table. The table is codified by a set of double strands, one for each transition of the Turing Machine. Each one of these oligos contains the encodings of the new symbol, the one of the new state and that of the move (to the left or to the right) that the Turing Machine must realize to perform the state transition associated with that precise oligo. Finally, the oligo has two sticky ends that allow it to insert exactly in the site cut by the enzyme associated with old state, with the old pointed symbol and—thus—with the performing state transition. In fact, this substitution of that fragment of the plasmid results in a new plasmid that encodes the desired new instantaneous description, including the encoding of the new state and the new head position. This operation is realized—as we said—in few chemical steps that, first, cut the old plasmid and detach the fragment that must be removed and, then, circularize the DNA molecule with DNA ligase after the new fragment is opportunely inserted.

There is a difference with Beaver's proposal because here the encoding of the new elements are already in the molecules that encode the transition table (that Beaver did not encode) and they must not be synthesized at each transition. In Rothemund's method, the table is encoded once at the beginning of the computation and each entry is associated with commercially available restriction enzymes.

Rothemund, as an example, designed the implementation of the BB3 Turing Machine

, that is a machine with 3 states and an alphabet of 2 symbols. Since the product of these two number is 6, just 6 different enzymes are needed because this is the number of the entries of the transition table. Rothemund asserts that his implementation of a Turing Machine is possible for machine in which this product is up to 60, since this is the number of the commercially available enzymes of the opportune class that are sufficiently different not to generate errors during the simulation of the Turing Machine's execution. Finally, Rothemund shows how his method—that uses only commercially available reagents and performs operations that are routine for molecular biologists— can be applied to implement a Universal Turing Machine (UTM), since Minsky designed a UTM with 7 states and 4 symbols (and $4\times 7 = 28 < 60$). The chemical expression of a universal model of computation improves all the demonstrations of the computational capacity of a hypothetic molecular computer.

Comment Rothemund's scheme is specified mostly in great detail. It works entirely with dsDNA, which is more stable than ssDNA. Reasonable estimates are given that it scales up to at least the scale of the smallest known universal Turing machine. This scale-up is in the size of the transition table

There are some problems to Rothemund's scheme. It does not describe how to generate the initial tapes. For the 'Busy Beaver' running example, the initial tape is blank; it is presumably relatively easy to sequence the strand encoding for instantaneous description of the Turing machine at the start of its computation, and then make sufficient copies of it (e.g. via PCR) to compensate for the practical problems discussed below. However, a universal Turing machine requires an initial tape with the encoding of the Turing machine to be simulated and its input; the initial strand can be much longer than that for the Busy Beaver case, and is therefore more difficult to generate. The Busy Beaver example is a deterministic one. One can therefore indeed simply mix the transition oligos with the DNA Turing tapes. Rothemund does not explain if his scheme is suitable for (or can be modified to suit) non - deterministic Turing machines: simply adding all transition nucleotides could result in interference in the reactions involving transition nucleotides that represent the different choices possible at the current time - step, while sequentially adding them (either in a strict order, or in random order) results in taking the same choice for all machines with the same status instead of having part of the machines take one choice, and the other machines take the other choices. It appears likely, though, that Beaver's suggestion to add random bits to the initial tape can be applied to Rothemund's model too; the 'fork using PCR' approach may work too, but is an additional source of errors. The scheme requires unfortunately many different kinds of restriction enzymes, some of which may have poor performance or imperfect specificity of restriction. Failed restrictions can result in defective tapes. By incorporating suitable labels, these defective tapes can be removed directly after the step in which they have been generated, thus preventing their interference with subsequent steps. Incorrect restrictions can

[1]The Busy Beaver problem for a Turing Machine with N states (that is BBN) consists in designing a Turing Machine with N states and a twosymbol alphabet {Black,White} that writes the greatest possible number of Black symbols before halting

occur, but do so only very infrequently, when the restriction is performed under the recommended reaction circumstances. The number of different kinds of restriction enzymes increases with the transition table size. Thus, Turing machines with large transition tables cannot be implemented with this scheme directly; one has to resort to simulation via a small universal Turing machine. The scheme can be used to implement the smallest known universal Turing machine, but simulations by this Turing machine are very inefficient. This is not a true problem, since Rothemund's goal was a proof of concept: universal computation is possible with Molecular Computation. Also, the ligations involved may fail or even ligate mismatching pairs of sticky ends and ligation may occur between two tapes, instead of between parts of one tape. Suitable enzymes exist that can remove precisely the products of such undesired reactions. Lastly, attachment of the DNA strands to solid support can be used to simplify the removal of reagents after the step in which they were needed is finished, and the tapes are kept separately, preventing them from ligating together. The palindromic property of the type II restriction endonucleases enzymes can take to undesired events during this implementation of a Turing Machine. For example, since the two sticky ends generated by the cut are identical, a new annealing between the two cut strands could happen. Besides - for the same reason - having control of the orientation of the fragment that must be inserted is not possible. For these and other reasons use of *IIS restriction endonucleases* preferred. These enzymes cut the strand near the restriction site at an opportune distance (these enzymes are called *nonphilindropic*), thus avoiding undesired events.

2.2.8 Erik Winfree - Model of Self Assembly of DNA tilings

Winfree ([Winfree96]) has developed a model, capable of universal computation, is based on a very different model capable of universal computation: *cellular automata* (CA). CA ([CAF]) consists of a collection of cells (e.g. a two dimensional array), each containing a symbol from a finite alphabet, and a set of transition rules that are applied in parallel to all cells at fixed time intervals. The transition of a cell's content from one 'time tick' to another is determined by its current content and the contents of a finite number of 'neighbours' to it (e.g. for a two dimensional matrix, the cells directly left, right, above and below it). There exist even one - dimensional CA that are universal: for each Turing machine (including a universal one), one can construct a cellular automaton whose initial array contents correspond to the initial tape contents of the Turing machine, and whose transition rules simulate both the transition table, the cell replacement and the head movement of the Turing machine. A universal cellular automata(UCA) thus has a fixed set of transition rules, and is programmed by providing the initial array contents.

One of the variations of UCA that is relevant to Winfree's model is blocked CA (BCA), a one - dimensional variation on partitioning CA. In this model, the transition rule is formulated for pairs of cells. There are two possible partitions, ways of dividing the cells into pairs of neighbours, in the array of a one - dimensional cellular automaton $(\cdots c_n c_{n+1} c_{n+2} c_{n+3} c_{n+4} \cdots$ can be paired like$\cdots (c_n c_{n+1})(c_{n+2} c_{n+3})(c_{n+4}\cdot$ or like $\cdots c_n (c_{n+1} c_{n+2})(c_{n+3} c_{n+4}) \cdots)$. During successive time steps the two possible partitions of cells in pairs are strictly alternated in the application of the transition rule. Blocked cellular automata is universal due to a construction analogous to that for normal one - dimensional cellular automata.

Simulation of BCA Winfree simulates a universal blocked cellular automaton by designing small units of DNA in such a way that they self - assemble into two - dimensional complexes according to the rules of the automaton in a hybridization reaction. In these complexes, a slice in one direction corresponds to the state of the whole automaton at a certain point in time, while a slice in the perpendicular direction shows the contents of one cell during the whole development of the automaton.

Comments

Winfree's approach seems to be one of the potential candidates for the futurestic developments in DNA computing because it utilizes the sofisticated structures mainly specific to DNA molecules. It is conceptually much simpler than Beaver's and Rothemund's models, using only one basic reaction (hybridization), in a straight - forward simulation, requiring no external processing (it is 'one - pot'). This illustrates the necessity of studying the many different models of computation for their suitability for implementation using molecular computational hardware, and of the search for a model of computation natural to molecular computation. Also, it shows that the asynchronous nature of parallelism in biochemical reactions does not necessarily preclude approaches based on synchronous parallelism.

Though it depends on an unusual DNA structure, whose behavior in practice has not been fully tested as yet. Some authors ([SZDC95]) discuss the large gap between theory and practice of constructing unusual DNA structures, including the tendency of DNA to form double stranded helices, difficulty in control and the importance of studying the actual three - dimensional structure instead of relying on two - dimensional models. The problem in achieving substantial yield of desired results is less acute here, since the building blocks are simple enough to produce in sufficient quantities, and the complex structure forms as a result of the self - assembly of the building blocks.

It may even be possible to use a similar self - assembling system to simulate Turing machines, although such a system would probably require many more, likely complex, building blocks, and would not use the parallelism that is natural to cellular automata.

2.2.9 Reif's Parallel Associative Memory Model

In [Rei95], Reif suggested his Parallel Associative Memory Model (PAM model), in order to define operations on DNA strands that can be implemented with the aim of calculating with DNA and how to compute with DNA implementing something different from a brute force approach (requiring increasing exponential volume) to a problem. Thus giving a formalism that expresses the molecular parallelism. The operation that characterizes the PAM Model is the *PAMatch* operation. Let E(x) be the encoding of x)[1] and let E(x,y) be the one of the pair (x, y). The PAMatch operation is defined as:

$$E(x,y) \bowtie E(w,z) = \{E(x,z) \text{ if } y = w, .\} \perp \textit{otherwise}$$

The operation is applied to entire testtubes with the following meaning:

$$T \bowtie T_{\prime} = \{x \bowtie x_{\prime} \mid x \in T, x_{\prime} \in T_{\prime}\}$$

where x and $x_{\prime}$ are DNA molecules. In addition to this special operation, the PAM Model uses the well known operations of union, extraction and detect. To evaluate an algorithm, Reif suggests to consider the number of required steps (as a time measure), the maximum number of strands that appear at a time in the testtube, and the length of the longest strand in the testtube. Using O(t) PAMatch operations and O(s log s) operation (that are not the PAMatch), Reif shows how to:

- Simulate the behavior of a nondeterministic Turing Machine with space bound o(s) and time bound $2^{O(s)}$.
- Simulate a CREW PRAM with M memory cells, P processors and time bound D. Where s = O(log(PM)) and t = D + s.
- Find an assignment of values to n boolean variables that satisfies a circuit constructible in s space with unbounded fanout and depth D with t = D + s.
- Do List and Tree Contraction for L sized inputs constructible in s space. Here t = log L, each of the used tubes contains O(B+L= log L) aggregates and it is assumed that there is a pre computed tube of B aggregates.

comment

PAM Model is a high level one. Thus, it allows to express algorithm that should be implementable at a lower level. In the case of DNA computing, the lowest level involves DNA molecules. Reif formalized this aspect defining another model, the Recombinant DNA Model (RDNA). It is shown how the operations of the PAM Model are implemented by those of the RDNA Model, that is a lower level one. It is

[1]Recall that underlying alphabet set is $\Sigma = \{A, C, G, T\}$.

an abstraction of the recombinant DNA techniques, available in a molecular biology lab, that are at the bottom of any DNA computation. Finally, it is showed with a low slowdown the probability of errors can be reduced.

****

Chapter 3

Killer Applications in DNA Computing

In this chapter we try to search for some proposed ambitious applications of DNA computing with the main aid of either theory e.g., applications in cryptography or sophisticated DNA manipulation techniques like self assembly or surface based techniques. A brief but succinct survey of this nature can be found in [Reif98].

Killer applications are possible because a DNA strand may need at most 1,000 bps to encode desired information state [1] and so a litre of DNA solution in water encodes the state of approximately 10^{18} distinct processors giving the overall potential for about 10^{15} to 10^{16} operations per second, which is 1,000 tera - ops (something far beyond even the most ambitious supercomputers processing speed). While this number is very large, it is finite, so there is a finite constant upper limit to the enhanced power of computation using DNA computing within moderate volume. Nevertheless, the size of this constant is so large that it way well be advantageous in certain key applications, as compared to conventional (macroscopic) computation method and there are a wide variety of problems (up to moderate sizes) that may benefit from the massive parallelism and nanoscale miniaturization available to DNA computing. Some of the most compelling applications are the following:

1. **NP search problems**. These are a class of computational problems apparently requiring a large combinatorial search for their solution, but requiring modest work to verify a correct solution. NP search problems may be solved by DNA computing by (i) assembling a large number of potential solutions to the search problem, where each potential solution is encoded on a distinct strand of DNA, and (ii) then performing recombinant DNA operations which separate out the correct solutions of the problem. DNA computing has been proposed for the following NP search problems:

[1]if the information is changeable than the strand is equivalent to a processor state

proposed for the following NP search problems:
(i) *Hamiltonian path.* In addition to Adleman's [Adle94] first solution,Fu et al [FBZ98] suggest further improvements his HPP DNA computing experiment.
(ii) *SAT.* SAT is the problem of finding variable assignments that satisfy a Boolean formula. Lipton [Lip94] proposed use of DNA computing for finding satisfying inputs to a Boolean expression, and this approach was generalized in [BDSL95] to solve the SAT problem. Also Eng [Eng98] proposed in vivo DNA computing methods for SAT.
(iii)*Graph coloring.* Jonoska and Karl [JK96].
(iv) *Shortest common super string problem.* Gloor et al [GKG98].
(v) *Integer factorization.* Beaver [Bea94].
(vi) *Breaking the DES cryptosystem.* [BDL95] and [ARRW96].

2. **SurfaceBased NP search.** Eng, and Serridge [ES97] give a surfacebase DNA computing algorithm for *minimal set cover.* Liu et al [LFW98] give an experimental demonstration of surface based DNA computing using a one word approach to solve a SAT problem. Eng [Eng98] proposes in vivo DNA computing methods for the NP complete problem of satisfiability of Boolean formula in 3CNF form.

3. **NP search using RNA.** Cukras, Faulhammer, Lipton, and Landweber [CFL98] gave an experimental demonstration of a DNA computing method for the solution of a class of SAT problems (derived from the knights problem in Chess), that appears likely to scale to at least moderate number of Boolean variables (say 18 to 24). Their method was also significant due to their use of RNA rather than DNA and their development of a powerful evolutionary method for doing the combinatorial search to optimize their DNA word codes.

4. **Whiplash PCR** Hagiya and Arita [HA97] use DNA computing method that uses the end segments of DNA strands to do editing and processing within the interior of the strand. Hagiya and Arita [HA97] showed that Whiplash PCR can be used for SAT problems for a class of Boolean formulas known as μ - formulas, and Winfree [Winfree98] extended these techniques to solve general SAT problems. Sakamoto et al [SKK98] describe how to do finite state transitions using Whiplash PCR, using a graduated scale of melting temperatures to reduce the number of laboratory steps, and also describes implementations of these methods.

5. **Decreasing the Volume Used in NP search.** In all these methods, the number of bio - steps grows as a polynomial function of the size of the input, but the volume grows exponentially with the input. For exact solutions of NP complete problems, we may benefit from a more general type of computation than simply brute force search. The molecular computation needs to be general enough to implement sophisticated heuristics, which may result in a smaller

search space and volume. For example, Ogihara and Ray [OR97] proposed a DNA computing method for decreasing the volume providing a smaller constant base of the exponential growth rate required to solve the SAT problem . The difficulty with many of these approaches for NP search is that they initially generate a very large volume containing all possible solutions. An alternative heuristic approach of iteratively refining the solution space. to solve NP search problems has been suggested by Hagiya and Arita [HA97](also see [YoSh00] for better result as described before in sec 2.2), and may in practice give a significant decrease in the volume.

6. **Combinatorial Chemistry as NP Searches.** Combinatorial chemistry techniques (also known as diversity techniques) have been used by biochemists to do combinatorial searches for biological objects with special properties. These techniques were very similar to the use of massive parallelism in DNA computing to solve NP search problems. Generally,they use recombinant DNA techniques to first construct a large pool of random sequences and then choose elements with specific properties from within the pool.Also, Bartel and Szostak [BS91] constructed a large pool of random sequences and then isolated new ribozymes. Also, Eigen and The disciplines of combinatorial chemistry and DNA computing may benefit by combining some of their techniques. For example, the search space of combinatorial chemistry might be decreased by sophisticated heuristics used in NP search methods.

7. **Huge Associative Memories.** DNA computing has the potential to provide huge memories. Each individual strand of DNA can encode binary information. A small volume can contain a vast number of molecules: DNA solution in one liter of water can encode 10^7 to 10^8 terabytes, and we can perform massively parallel associative searches on these memories. Baum [Baum95] proposed a parallel memory where DNA strands are used to store memory words, and provided a method for doing associative memory searches using complementary matching. Lipton [Lip98 96] describes the use of web data bases and associative search within them to do cryptoanalysis. This idea for associative memory can be extended to allow us to execute operations in parallel, that is to do concurrent word searches. From this follows the concept of a data base molecular computer using DNA computing. The time and volume efficiency of associative memory searches can be improved by the use of microflow device technology (Gehani and Reif [GR98]) to segragate pools (microTest Tubes) of DNA strands to be searched, and to apply the searches in parallel for each pool.

8. **Massively Parallel Machines.** DNA computing also has the potential to supply massive computational power. DNA computing can be used as a parallel machine where each processor's state is encoded by a DNA strand. DNA computing can perform massively parallel computations by executing recombinant DNA operations that act on all the DNA molecules at the same time.

These recombinant DNA operations may be performed to execute massively parallel local memory read/write, logical operations and also further basic operations on words such as parallel arithmetic. DNA in weak solution in one liter of water can encode the state of about 10^{18} processors, and since certain recombinant DNA operations can take many minutes, the overall potential for a massively parallel DNA computing machines is about 1,000 teraops. (This assumes the parallel machine uses local rather than global shared memory. To allow such a parallel machine to use global shared memory, we need to do massively parallel message (DNA strand) routing. In Reif's [Rei95] DNA computing simulation of a PRAM with shared memory required volume growing at least quadratically with size of the storage of the PRAM, but Gehani and Reif [GR98] describe a microflow device technology that can do the massively parallel message routing with a substantial decrease in the volume.

9. **Other Algorithmic Applications of DNA computing.** DNA computing may also be used to speed up computations that would require polynomial time on conventional machines: Beigel and Fu [BF97] discuss approximation algorithm for NP search problems, Baum and Boneh discuss DNA computing methods for executing dynamic programming algorithms, and Oliver [O96] discusses DNA computing methods for matrix multiplication.

10. **Neural Network Learning and Image Recognition.** Mills, Yurke, and Platzman [MYP98] propose a rather innovative DNA computing system for errortolerant learning in a neural network, which is intended to be used for associative matching of images. They use a DNA computing method for matrix multiplication (Oliver [O96]) to implement the inner products required for neural network training and evaluation, and their proposed DNA computing system also makes innovative use of DNA chips for I/O.

11. **DNA Nanofabrication and Selfassembly.** DNA computing techniques combined with Seeman's DNA nanofabrication techniques may allow for the selfassembly of DNA tiles into lattices in 2 and 3 dimensions and the construction of complex nanostructures that encode computations.

12. **Biological Applications: Processing of Natural DNA.** The field of DNA computing has restricted its attention mostly to applications which are computational problems, e.g., NP search problems.DNA computing techniques may also be used in problems that are not implicitly digital in nature, for example the processing of natural (biologically derived) DNA. These techniques may be used to provide improved methods for the sequencing and fingerprinting of natural DNA, and the solution of other biomedical problems. The results of processing natural DNA can be used to form wet data bases with recoded DNA in solution, and DNA computing can be used to do fast searches and data base operations on these wet databases. However, DNA computing techniques might be ideally suited to solve problems in molecular biology which inherently

involve natural DNA, that is DNA that is biologically derived (as opposed to artificially synthesized DNA which is coded over a given word alphabet). A class of problems, including sequencing, finger printing and mutation detection may well be the killer applications of DNA computing due to the basic involvement of DNA in these problems. An experimental demonstration, at moderate scale, of a DNA computing method for solving a significant problem in molecular biology with natural DNA inputs, will be a major milestone in DNA computing.
Recoding DNA. One interesting approach to use DNA computing to solve problems concerning natural DNA is to allow natural DNA to be recoded. The natural DNA is recoded as sequences of encoded nmers. This recoding allows the DNA to be then operated in a purely digital manner. The processing of recoded DNA can then be done by the usual DNA computing techniques.
DNA Sequencing. One possible application considered by [LL97] is DNA sequencing by hybridization. Redundant recoding of nmers may be used to reduce errors due to incomplete hybridization. These redundant encodings would be constructed and attached to the nmers using known DNA computing methods, yielding an encoded array of nmers providing the DNA sequence information (also Boneh and Lipton [BL95] have a quite distinct divide and conquer approach to DNA sequencing).
Further Processing of Recoded DNA. Once natural DNA is recoded, general DNA computing methods may be used to speed up many other key applications in biology and medicine [SM97], such as fingerprinting and mutation detection. Recoded natural DNA derived from many sources can be used to assemble large wet data bases containing DNA that encodes data of biological interest, without the problem inherent in I/O to an electronic medium. DNA computing, with its huge memory capacity, has a considerable advantage over conventional technologies for storing such biological data bases. Once the wet data bases are assembled then we can do further processing using DNA computing techniques, for example fast associative searches ([Baum96])can be done in these wet data bases.

13. **Approximate Counting of DNA.** Faulhammer, Lipton, and Landweber [FLL98] give a DNA computing method for estimating the number of DNA strands within a test tube. This may make possible to even solve problems beyond NP, e.g., counting approximately the number of possible satisfying solutions of a SAT formula.

●●●●●●●●●●●

Chapter 4

Recent Experimental Advancement in DNA Nano Technologies and Paradigms

Aim of this chapter is to have a cursory look at the more significant experimental advancements in DNA manipulation nano techniques and thus established new DNA computing paradigms, so that to have a bit of idea that if at all some of the troubling aspects like longer time periods and errors associated with usual DNA recombinant technology provided tool -kit operatrions can be overcome.

4.1 New Technologies

4.1.1 Recombinant DNA Technology

Biotechnology has developed a large set of procedures for modifying DNA, known collectively known as recombinant DNA (as described in detail in chap 1 and in appendix B). Many of those recombinant DNA operations which where once considered highly sophisticated are now routine, and many have been automated by robotic systems. As a further byproduct of the industrialization of the biotechnology field, many of the constraints (such as timing, pH, and solution concentration, contamination etc.) critical to the successful execution of recombinant DNA techniques for conventional biological and medical applications (but not necessarily for all DNA computing applications), are now quite well understood, both theoretically and in practice.

4.1.2 Alternative Recombinant DNA Methodologies

The most pervasive enabling biotechnology for DNA computing is solution - based recombinant DNA, that is the recombinant DNA operations are done on test tubes with DNA in solution. However, there are a number of alternative enabling biotechnologies, that allow similar and sometimes enhanced capabilities.

1. **Solid Support DNA computing**

   An example of an alternative recombinant DNA methodology is the solid support of individual DNA, for example by surface attachments. In solid support, the DNA strands are affixed to supports of some sort. In surface - based chemistry, surface attachments are used to affix DNA strands to organic compounds on the surface of a container. This can allow for more control of recombinant DNA operations, since this insures (i) that distinct DNA strands so immobilized can not interact, and also (ii) allows reagents and complementary DNA to have easy access to the DNA, and (iii) allows for easy removal of reagents and secondary by - products. Also, handling of samples is simpler and more readily automated. Surface - based chemistry has been used in protein sequencing, DNA synthesis,and peptide synthesis. Surface attachment methods can also be used for optical read - out (e.g., via fluorescent tagging of specific DNA words) on 2D arrays. A possible drawback of surface attachment technology, in comparison to solution - based recombinant DNA techniques, is a reduction on the total number of DNA strands that can be used.

2. **Automation and Miniaturization of DNA computing**

   Some of the current limits of DNA computing stem from the labor intensive nature of the laboratory work, the error rates, and the large volumes needed for certain bio - molecular reactions to occur (e.g., for searching and associative matching in wet data bases). [GR98] propose the use of MEMS micro - flow device technology for DNA computing which may provide several advantages: it would allow automation of the laboratory work, parallel execution of the steps of a DNA computing algorithm (for improved speed and reliability), and for transport of fluids and DNA among multiple micro - test tubes. [GR98] provide a model for micro - flow based bio - molecular computation (MF - DNA computing) which uses abstractions of both the recombinant DNA (RDNA) technology as well as of the micro - flow technology, and takes into account both of their limitations (e.g., concentration limitations for reactants in RDNA, and the geometric limitations of the MEMS device fabrication technology). [GR98] also give a time and volume efficient MF - DNA computing architecture for routing DNA strands among multiple micro - test tubes (this gives a substantial decrease in the volume required for the PRAM simulation of [Rei95]).

4.2 New Experimental Paradigms for DNA computing

4.2.1 General - purpose Molecular Computers using DNA computing

DNA computing machines using molecular parallelism and providing large memories, are being constructed at Wisconsin, [LTCSC97] and USC [Adle96]. In both projects, a large number of DNA strands are used, where each DNA strand stores multiple memory words. Both these machines will be capable of performing, in parallel, certain classes of simple operations on words within the DNA molecules used as memory. Both projects developed error - resistant word designs.Successful prototyping at moderate scale of either of these machines will be a major experimental milestone in DNA computing. The Wisconsin project is employing a surface to immobilize the DNA strands which correspond to the solution space of a NP search problem. Since they are all on the same surface, all DNA strands are operated in a Single Instruction Multiple Data (SIMD) fashion. Their operations on words are restricted to mark, unmark, and destroy operations, which suffice for certain NP search problems. A key challenge in their approach is to provide scaling to a sufficiently large number of DNA strands within the constraints of surface attachment technology. In contrast, the USC project uses a combination of solution - based and solid support methods, which are used to improve the efficiency of the separation operations. In this method, the computation is done without formation and breaking of covalent bonds. Their operations on words include the Boolean logic operations. All DNA strands within a given test tube are operated on in a SIMD fashion. However, their approach allows splitting of the solution space into separate test tubes, and thus potentially allows for DNA strands to be operated on in a very limited Multiple Instruction Multiple Data (MIMD) fashion, where the number of distinct instructions executed at the same time is limited to the number of test tubes used in parallel. A key challenge in their approach, and the major focus of their effort, is to provide for efficient error - resistant separations.

4.2.2 The Local Assembly Paradigm

The local parallelism (LP - DNA computing) paradigm for DNA computing allows operations to be executed in parallel on a given molecule (in contrast to the parallelism where operations are executed in parallel on a large number of distinct molecules but execute sequentially within any given molecule). Before we describe these local assembly techniques, we first discuss DNA nano - assembly techniques, and some previously known tiling results, which provided the intellectual foundations for local assembly.

4.2.3 DNA Nano - Fabrication Techniques

Feynman [Feynman61] proposed nano - fabrication of structures of molecular size. Seeman nano - fabricated in DNA ([SZDC95]): 2D polygons, including interlinked squares, and 3D polyhedra, including a cube and a truncated octahedron. Seeman's constructions used for basic constructive components:
*DNA junctions*i.e., immobile and partially mobile DNA n - armed branched junctions
DNA knots: i.e., ssDNA knots rings,
DNA crossover molecules: i.e., double helix(DX) molecules,the octahedron used solid - support, to avoid interaction between constructed molecules. (See Figure ??)

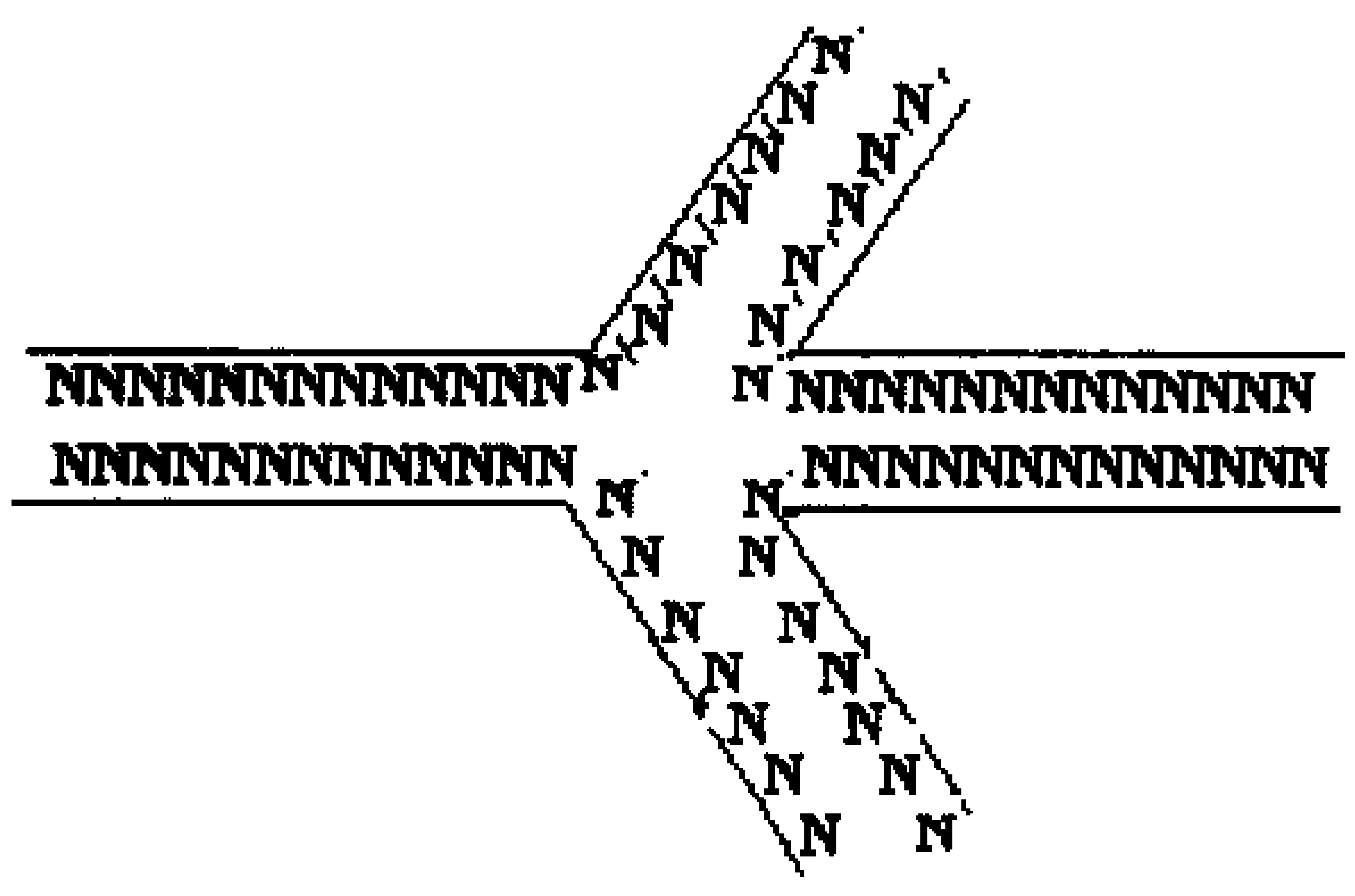


Figure 4.1: **Other Possible Graph like Structures of DNA Molecule**

4.2.4 More on Computation via Local Assembly

A class of tiling problems are defined as follows: we are given a finite set of tiles of unit size square tiles each with top and bottom sides labeled with symbols over a finite alphabet. These labels will be called pads. We also specify the initial placement of a specified subset of these tiles, and the borders of the region where tiles must be placed defining the extent of tiling. The problem is to place the tiles, chosen with replacement, in all these square regions within the specified borders, so that each pair of vertical abutting tiles have identical symbols on their contacting sides. Let the size of the tiling assembly be the number of tiles placed. Berger [B66] proved that given a finite set of tile types, the tiling problem is undecidable if the extent of tiling is infinite. As discussed in last chapter Winfree proposed idea is to do the tiling constructions by application of the DNA nano - fabrication techniques of Seeman et al [SZC94], which may be used for the construction of small DNA molecules that can function as square tiles with pads on the sides. The pads are ssDNA. As described in chapter 1, if two ssDNA are sticky (i.e., Watson - Crick complementary and $3'$ - $5'$ and $5'$ - $3'$ oriented in opposite directions), they may hybridize together at the appropriate conditions into doubly stranded DNA. The assembly of the tiles is due to this hybridization of pairs of matching sticky pads on the sides of the tiles. This innovative paradigm for DNA computing can be termed as unmediated self - assembly since the computations advance with no intervention by any controllers. The advantages of the unmediated DNA assembly idea of Winfree is potentially very significant for DNA computing since the computations advance with no intervention by any controllers, and require no thermal cycling. It is a considerable paradigm shift from distributed molecular parallelism, which requires the recombinant DNA steps (which implement the molecular parallelism) to be done in sequence.

To simulate a 1D parallel automata or a one tape Turing Machine, Winfree et al [Winfree98] proposed self - assembly of 2D arrays of DNA molecules, applying the recombinant DNA nano - fabrication techniques of Seeman et al [SZC94] in combination with the tiling techniques of Berger [B66]. Winfree et al [Winfree98] then provided further elaboration of this idea to solve a variety of computational problems using unmediated DNA self - assembly. For example, they propose the use of these unmediated DNA assembly techniques to directly solve the NP - complete directed Hamiltonian path problem, using a construction similar to the NP - completeness proof of [GJ79] for tiling of polynomial size extent. Winfree et al [WYS96] also provided an experimental test validating the preferential pairing of matching DNA tiles over partially non - matching DNA tiles. Winfree [Winfree98] made computer simulations of computing by self - assembly of DNA tiles.

Erik Winfree, et al [WLW98] recently experimentally constructed the first large (involving thousands of individual times) two dimensional arrays of DNA crystals by self - assembly of nearly identical DNA tiles. The tiles consisted of two double - crossovers (DX) which self - assemble into a periodic 2D lattice. They produced spectacular atomic force microscope(AFM) images of these tilings (by insertion of a hairpin sequence into one of the tiles they created 25 nm stripes in the lattice). They also verified the assembly by the use of "reporter" ssDNA sequences. This experiment provided strong evidence of the feasibility of large scaling self - assembly, but it was not in itself computational. LaBean, et al [LYR98] recently designed and experimentally tested in the lab a new DNA tile which is a rectangular shaped triple crossover molecule with sticky ends on each side that can match with other such tiles and with a "reporter" ssDNA sequence that runs through the tile from lower left to upper right, facilitating output of the tiling computation.

What remains is to experimentally demonstrate: (i) DNA self - assembly for a (non - trivial) computation, and (ii) DNA self - assembly of a (possibly non - computational) 3D tiling.

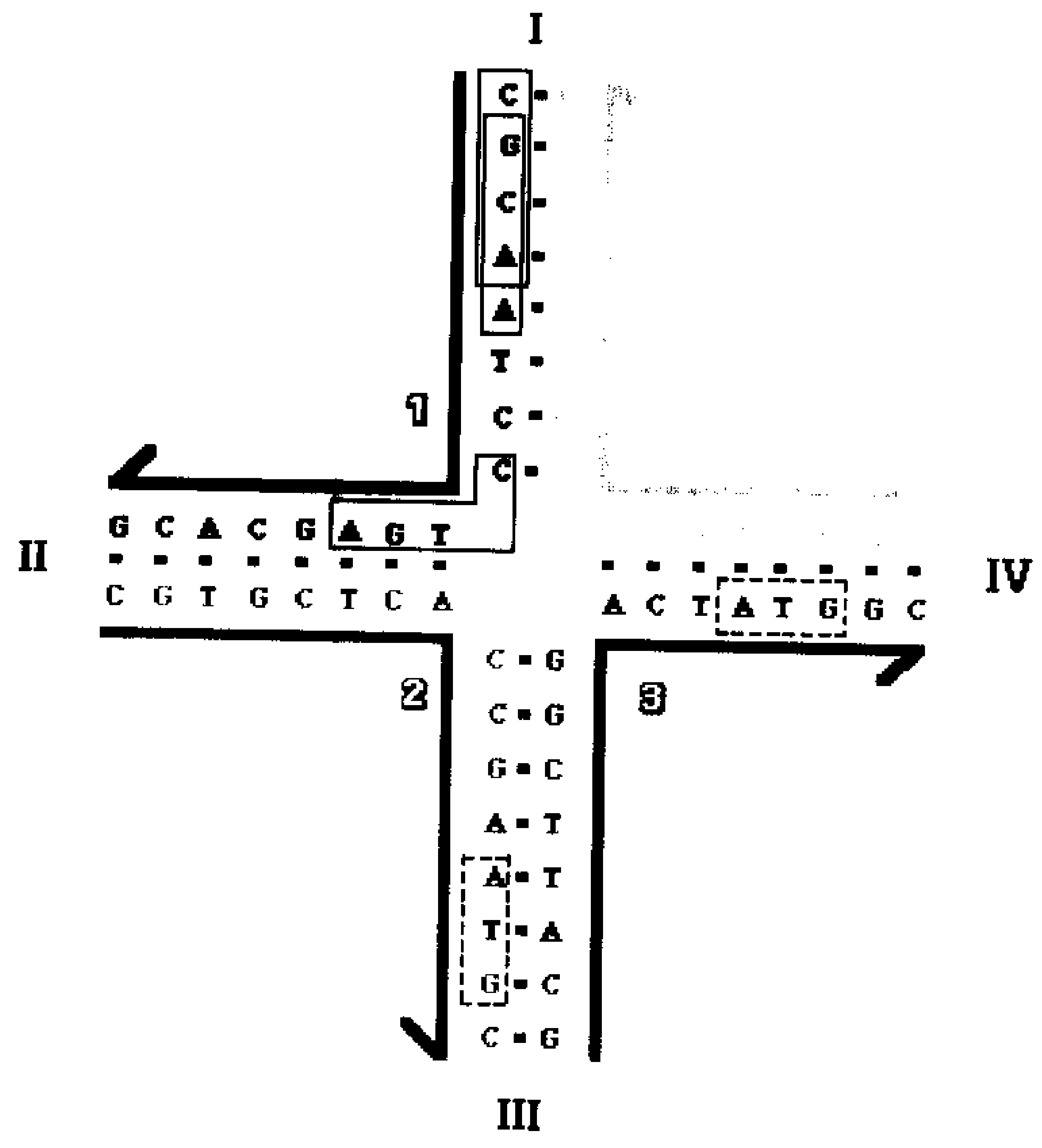
4.2.5 Assemblies of Small Size and Depth

To increase the likelihood of success of assembly, Reif [Reif97] proposed a step - wise assembly which provides control of the assembly in distinct steps. The total number of steps is bound by the depth of the assembly. Also, [Reif97] proposed the use of frames, which are rigid DNA nano - structures used to constrain the geometry of the assembly and to allow for the placement of input DNA strands on the boundaries of the tiling assembly. Using these assembly techniques, [Reif97] proposed LP - DNA computing methods to solve a number of fundamental problems that form the basis for the design of many parallel algorithms, for these decreased the size of the assembly to linear in the input size and and significantly decreased the number of time steps. For example, the prefix computation problem is the problem of applying an associative operation to all prefixes of a sequence of n inputs, and can be used to solve arithmetic problems such as integer addition, subtraction, multiplication by a constant number, finite state automata simulation, and to fingerprint (hash) a string. [Reif97] gave step - wise assembly algorithms,with linear assembly size and logarithmic time, for the prefix computation problem. [Reif97] gave DNA computing methods for perfect pair - wise exchange using a linear size assembly and constant assembly depth, and thus constant time. This allows one to execute normal parallel algorithms using DNA computing in logarithmic time. Also, this implies a method for parallel evaluation of a bounded degree Boolean circuit in time bounded by the circuit depth times a logarithmic factor. Previously, such operations had been implemented using Dynamic Programming - DNA computing techniques [Rei95] in similar time bounds but required a large amount of volume; in contrast the Linear Programming - DNA computing methods of [Reif97] require very modest volume. All of these LP - DNA computing algorithms of [Reif97] can also use DNA computing to simultaneously solve multiple problems with distinct inputs (e.g. do parallel arithmetic on multiple inputs, or determine satisfying inputs of a circuit), so they are an enhancement of the power of DNA computing. Jonoska et al [JKS98] describes techniques for solving the Hamiltonian path problem by self assembly of single strand DNA into three dimensional DNA structures representing a Hamiltonian path.

4.2.6 The Cellular Processor Paradigm

DNA computing may make use of micro organisms such as bacteria to do computation. A cellular processor is a microorganism such as a bacteria, which does computation via a re - engineered regulatory feedback system for cellular metabolism. The re - engineering involves the insertion of modified regulatory genes. whose DNA has been modified and engineered so that the cell can compute using regulatory feedback systems used in cellular metabolism.

⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆



I
II
III
IV
1
2
3
C G C A A T C C
G C A C G A G T
C G T G C T C A
A C T A T G G C
C C G A A T G C
G G C T T A C G

Chapter 5

Basic DNA Arithmetic: Recursive Implementation

In this chapter we describe a new efficient recursive DNA computing technique for working with basic arithmetic operations like addition and multiplication of binary numbers, based upon their set representation. The major significant features of our technique are:
(1) Its fully procedural in nature, that is, the structure of output is exactly similar with that of inputs. This way result can be further reused without any changes, making iterative as well as parallel operations feasible.
(2) Technique requires number of different DNA strands at most of the order of size of the binary number. That means it avoids the problem of increasing volume.
(3) Time required by the technique for addition is O($log_2 n$) and for multiplication it is O(n$log_2 n$), where n is the size of the binary numbers.

5.1 Related Work

5.1.1 Horizontal Chain Reaction Technique

One of the earliest attempts to perform arithmetic operations (addition of two positive binary numbers) using DNA are by Guarneiri and others [GFB96], utilizing the idea of encoding differently bit values 0 and 1 as ssDNA, based upon their positions and the operand in which they are actually appearing. This enabled them to propagate carry successfully as horizontal chain reaction using intermediate placeholders because of the presence of appropriate complementary substrands, which annealed together and PCR gave way to insert correct value of carry and way to further propagate it. Though their technique yields correctly result of addition of two given binary numbers but is highly nonprocedural in nature since the output strand is vastly different in structure from the input strands (which themselves are coded differently!) making it of little use in effect.

5.1.2 Truth - Table Mapping Technique

The later attempts were by Vineet Gupta and others [GPZ97] to perform logic and arithmetic operations using the fixed bit encoding of full corresponding truth tables. They construct the strands for bits in first operand (level one)and corresponding to each bit value (0 or 1) all possible bit values (00, 01, 10, 11)[1] in second operand(level two) such that in the next phase when first operand - strands are pored into the pot containing all possible second operand - strands (including the correct one)annealing results in correct in a structure which can be interpreted according to the type of the operation. In case of arithmatic operation in later stages of computation they add all possible intermediate results and successively propagate carry from the lowest weighted bit to highest weighted bit.

Though the encoding works well with logic operations but arithmetic operation are not so easy as the technique requires that all the possible intermediate results to be coded and added manually one by one during processing which in effect is a labor intensive and time consuming job.

Other later attempts are due to Z. Frank Qin and Mi Lu [QL98], which use substitution operation to insert results (by encoding all possible outputs of bit by bit operation along with second operand) in the operand strands. Though they propose to extend the radix of numbers from binary to any other higher radix like octal, decimal etc but this does not decrease the possible number of encoding of different intermediate results (which are exponential in number). Again in case of addition operation quite against to their claim and given example one step procedure does not seem to propagate the carry from the first bit to the last bit yielding the correct result. Moreover the cleansing operation which they claim making the output similar to first input, for reuse in further operations, is itself not an error resistant operation.

5.1.3 Generalised Techniques

Some generalized techniques are there due to Ogihara and Ray and by Amos and others [OG96, AD97], who present methods to realize any Boolean circuit (with bounded fan in) using DNA strands in constructive fashion. Here though the the operations are feasible but problem is of constructing large Boolean circuits for arithmetic operations, manually or automating the process, leaving the technique for theoretical importance only.

[1]In fact they use along with usual dnts other nucleotides like Uracil(U), 7 -deaza adenine(P) to achieve some additional complementary structures

5.1.4 Other Results

Other new suggestions to perform all basic arithmetic operations are by Atanasiu [Ata00] using P systems (as introduced earlier in sec. 2.2.5 and already having several of variants with universal computing power), in [Fri00] using splicing operation under general H systems by encoding operands and results (in base one in the same strand) with redundancy, then gradually by filtering the wrong ones out , and by Hubert and Schuler [HS01]

5.2 Recursive DNA Arithmetic

5.2.1 Underlying Mathematical Model

Let us suppose that the n -bit binary numbers to be operated upon are $A = A_n \cdots A_1$ and $B = B_n \cdots B_1$ where A_i , $B_i \in \{0,1\}$ $\forall 1 \leq i \leq n$. And that

$$X[A] = \{i: A_i = 1\}\, and\, X[B] = \{j: B_j = 1\};$$

which actually are the sets containing the position numbers where binary representation these number sets bit to 1. Next we define

$$Z^{i+} = \{z + i: z \in Z,\ z\, and\, i\, are\, integers\,\}\, and\, Z^{+} = Z^{1+}\ ,$$

$$X_1 \oplus X_2 \ = \ \{x: x \in X_1 \cup X_2 \ but\ x \notin X_1 \cap X2\}\ (symmetric\ difference),\ and$$

Val(X): outputs the binary number represented by set X (of positive integers) as for example Val(X[A]) = A and Val(X[B]) = B and Val (ϕ) = 0.
In terms of this symbolism we can state the abstract recursive procedures for addition and multiplication of the given two positive binary numbers as follows:

Add(A, B) = Val(RecursiveAdd(X[A], X[B])) Where

RecursiveAdd(Y, Z) = Y if Z = Φ

= Z if Y = ϕ

= RecursiveAdd((Y $\oplus$ Z), $(Y \cap Z)^{+}$) otherwise.

Since Add(A, B, C) = Add(Add(A, B), C), we have

Mul(A, B) = Add($\{$Val($X[A]^{j-1}$)$\}_{B_j=1}$) .

The Multiplication procedure actually realizes, using successive additions of left shifted A's, the following formula:

$$A \star B = \sum_{j=1, B_j=1}^{n} A \star 2^{j-1}$$

Where multiplication of a binary number with power of 2 is nothing but the left shifting of the number by that exponent, and the set containing all the positions where binary representation contains 1 is given by adding j's to each of these integers together.

Subtraction operation for integer arguments can be performed as 2's complement addition ([Hayes88] p 23).

Division. Once we can perform addition and subtraction operation then mapping of division operation in terms of these can be done using any of the standard digital arithmetic techniques ([Hayes88] p 250). For example here we will consider nonrestoring division ([Hayes88] p 253) which requires an average n additions or substractions.

5.2.2 DNA Algorithm

Since the procedure described above is recursive in nature and as can be seen easily in context of currently available DNA tool - kit operations and other high level operations as suggested in [BDSL95], that the most important operation to be realized is incrementing all the integers in the sets like X[A] by one. Actually whole of the coding of numbers and various other steps basically rest upon the ease with which this step can be realized. Keeping this point in mind we propose the following DNA algorithm:

DNA Encoding of Binary Numbers

Each binary number is represented as a test tube (a multi set of strings over S = {A, C, G, T}) of dsDNA encoding the integers (positions where bit is set to 1) from 1 to n, such that a dsDNA for integer 'i' is - (following the notations in [Bis98], discussed in appendix B) $ds_i = \updownarrow S_0(GAATTGC^5)^i GAATTC$ (Note that $\updownarrow$ GAATTC is restriction site for EcoRI),here S_0 may be any suitable 20 to 30 base - pair long dsDNA not containing GAATTA as a substrand.
For e.g.we may consider test tube T[A] representing binary number A as

$$T[A] = \{ds_i : i \in X[A]\},$$

similarly T[B] for B.

1. **Addition**

   Step0. Check whether any of T[A] or T[B] is empty. If yes then second tube contains the final result (which can be obtained by detecting the presence of all different strands using gel electrophoresis or extraction technique (for all the strands (for each integer i from 1 to n) one by one, followed by Test if Empty operation to check whether there is any strand in the tube as a result of extraction), else go to step 1. Initially T[A] and T[B] represent the test tubes encoding A and B respectively.

   Step1. Melt ds' in T[A] and T[B] to extract up - strands with respect to $\uparrow S_0$ from T[A] and down - strands with respect to $\downarrow S_0$ from T[B]. Now mix these two extracts so that complementary strands can get annealed to form stable ds'. As can be seen the resulted ds' in the tube are exactly those coming from T[A]$\cup$ T[B] and single strands are those coming from T[A]/T[B] (up - strands with $\uparrow S_0$) and from T[B]/T[A] (down strands with $\downarrow S_0$). Using standard DNA toolkit operations single strands can be separated from double strands and then using PCR the ssDNA can be complemented using S_0 as a primer. Let T[A] now contain these dsDNA obtained after PCR (i.e. T[A] $\oplus$ T[B]) and T[B] the annealed dsDNA (i.e. T[A]$\cup$T[B]).

   Step2.*(Increment by One)*. Add restriction enzyme EcoRI to T[B] to cut all
   its restriction site GAATTC the dsDNA at ~~their 3' end~~. This restriction enzyme activity leaves double strands with 5' hanging ends, of the form

   $$\updownarrow S_0(GAATTGC^5)^iG \uparrow AATT.$$

   Now down strands

   $$\downarrow AATTGC^5GAATTC$$

   and ligation enzyme is added to test tube which results in the strands of the form $\updownarrow S_0(GAATTGC^5)^iGAATT \downarrow GC^5GAATTA$. These strands can now be polymerized to form

   $$\updownarrow S_0(GAATTGC^5)^{i+1}GAATTA.$$

   Thus now T[B] contains strands representing (X[A]$\cup X[B])^+$.

   Step3. Go back to step0.

2. **Multiplication**

   Step1. $\forall j \in$ X[B]
   Construct test - tubes $T_j[A]$ similar to step2 above with the difference that for annealing add $\downarrow AATTGC^5(GAATTGC^5)^{j-1}$GAATTC instead of adding $\downarrow AATTGC^5GAATTC$ (add it when j = 2).

Step2. If the test tubes obtained in step1 are $T_{j_1}[A], T_{j_2}[A], T_{j_3}[A] \cdots$ etc do the following:
Step2.1 perform Addition operation (described above) with tubes $T_{j_1}[A], T_{j_2}[A]$. Let the result be kept in T.
Step2.2 Repeat the following $\forall\ j_i$ such that $i \geq 3$: perform Addition operation (above) with tubes T and $T_{j_i}[A]$ and keep the result back in T.

3. **Subtraction**
   The subtraction operation can be done utilizing ideas from the conventional digital arithmetic, that is to say as per 2's complement method. To perform A - B we do the following:

   Step1. Construct T[A] and T[B] and T, which consists of the ds_i for all i ∈ [1, n].

   Step2. Obtain T_1 = T ⊕ T[B] as described above with Addition operation or as a set extract operation described in [BDLS 98].
   Step3. perform addition operation with T[A] and T1 and keep the result in T1.
   Step4. Perform addition operation with T1 and T[N=1] (consist of only one type of dsDNA $\updownarrow S_0GAATTGC^5GAATTC$).

   Step5. Post - processing of the result by comparing it with 2^n gives the desired result.

4. **Division: NonRestoring Division**
   Let us suppose that number A is dividend and B is divisor.the result of division are quotient Q and shifted remainder finally in A. Let corresponding test tubes are T[A], T[B] , and T[Q], (initially T[Q],is empty) . Let Test tube T contains (as above in substraction procedure) the dsDNA strands ds_i for all $i \in [1, n]$. A DNA algorithm is performed as follows:

   Step0. Perform steps 1, 2, and 3 of the subtraction procedure above to get test tube representing 2's complement of B as Tube T[2'B].

   Repeat the following $\forall 1 \leq i \leq n-1$

   Step1. Perform last two steps (4 and 5) of substraction procedure (above) with tubes T[A], and T[2'B] (to get A - B).
   Step2. Perform increment by one operation on T[A] as described above in addition procedure's step1. **Step3**. If the result is positive (from the last

step5 of substraction procedure above), then
Step3.1 if i $\in$ X[Q] so add ds_i (already manufactured) to T[Q].
Step3.2 Perform step1.
Step4. Else if the result is negative (in step0 above) , then
Perform addition operation with tubes T[A], and T[B] (i.e. A + B).
Result 1. Perform step1.
2. If the result is negative, perform addition operation with tubes T[A], and T[B](i.e. A + B).
3. Quotient is X[Q] or T[Q] and shifted remainder (2^n R) is T[A] which can be read as describes in addition operation (step0).

5.2.3 Complexity Analysis

Time Complexity of Operations

Addition. As each level of recursion in addition operation involves fixed number of bio steps, therefore average time complexity of the procedure would be same as expected number of recursion levels in the abstract model. Since the procedure is simple enough thus even the following semi formal arguments can lead to the answer: Since the probability that at each position bit will be set to 1 (or 0) is 1/2, therefore expected number of 1s' in any randomly chosen binary number of length n are n/2. Similarly probability that at any position both the numbers (A, B, randomly chosen) set the bit to 1 is (1/2)*(1/2) = (1/4), therefore expected number of positions where both the numbers set bits to 1 are n/4, that is the expected size of X[A]$\cup$X[B] and consequently of (X[A]$\cup$X[B]$^+$). Similarly probability that at any position both the numbers set bits differently is (1/2)$\star$(1/2) + (1/2)$\star$(1/2) = (1/2) giving expected number of 1's in A$\oplus$B as n/2, which is same as the expected number of integers in X[A]$\oplus$X[B]. Now for 2nd level of recursion probability that position number i is present in both the sets from the 1st step is (1/2)$\star$(1/4) = (1/8) , therefore expected number of integers after set - intersection and shifting by one are n/8 while expected number of integers after symmetric difference of the sets are n/2 because probability that an integer is present in only one of the two sets is (1/2)*(1/4) + (1/2)*(3/4) = (1/2). Following the same argument it can be seen that after ith level of recursion expected size of set resulting after set intersection and shifting will be n/2^{i+1}. Therefore expected number of recursion levels will be $\lfloor log_2 n \rfloor$ - 1. Thus the average number of biosteps needed in the addition operation are O($log_2 n$). It's not difficult to see that worst case complexity is as well O($log_2 n$). In case when the final result is obtained by performing extraction (followed by 'Test if Empty') operation, total steps might increase to O(n + $log_2 n$) = O(n).

Multiplication. Since multiplication is nothing but repeated addition (at most $log_2 n$ times), as above, therefore average number of steps will be (n/2)* $log_2 n$ i.e. O(n$log_2 n$).

Subtraction. Similarly it can be seen that subtraction operation requires on average O($log_2 n$) bio steps. Since 2's complement of the 2nd operand to be subtracted can be obtained in at most O(1 $+log_2 2n$) steps, then rest addition operation requires O($log_2 n$) bio - steps.

Division Its clear from the division procedure that non restoring division technique above requires n additions or subtraction, amounting O(n$log_2 n$) number of bio - steps an average to perform complete devision operation with n bit long integer operands in base 2.

Volume Complexity

Since at any step of the above procedure we need only test tubes containing DNA strands representing integers from 1 to n, the space complexity (volume) is linear in the size of binary numbers i.e. O(n).

5.2.4 Error Analysis

Errors in DNA computing experiments are one of the main reasons of primary concern and active areas of current research [BDSL96, AWHOG98, AYT98, CW99]. One potential source of error in the above suggested algorithms seems to be partial annealing [YYSHTSMO99]. But as far as the correct and effective (procedural) enrealization of the abstract model is concerned it seems unlikely that any other encoding (instead of C^5) will fully overcome this error, though this does not mean anyway that it can't be reduced. Infact choice of C5 is chosen to make results of partial annealing less stable as compared to the correct matches . Other types of errors (like as loss of strands during separation of ss' and ds' from the tube) are actually experimental errors and can be minimized using only more sensitive and effective tools.

●●●●●●●●●●●

Appendix A

Composite Operations

Append

This adds a specific subsequence to the ends of all the strands. This can be done by annealing a short strand to a longer strand so that the short strand extends off the end of the longer strand. Then the complement of the subsequence that is extending off the end can be added to the longer strand either through the use of a polymerase enzyme, or through the introduction and annealing of that sequence, cemented by ligase.

Polymerase Chain Reaction or PCR

It's one of the most useful operation, which is used to amplify a known DNA sequence. Usually the strand is double, which is melted to form two complementary sequences then primer subsequences for both these strands are added together with a polymerase enzyme , which extends these sequences to form two full double strands as original one. This cycle can be repeated l times to get 2^l copies of the original double strand.

How to select a set of primers to use for PCR? Try to keep the primer 50% G - C give or take 15%. If overly G - C rich, add a string of As or Ts at 5' end; If overly A - T rich, do the same with Gs and Cs. Try to avoid Gs and Cs at 3' end of the primers. This may increase the chance of forming primer dimers. Avoid self - annealing regions within each primer([InGel90]). A good practice is to check the target DNA sequence if it is known for mis priming areas. A quick check scanning the sequence of vector for approximately 70% and above homology regions can help prevent obtaining multiple contaminating bands in PCR. Use of a computer program may help eliminate the use of a poorly designed pair of primers.

Substituting

substitute, insert or delete DNA sequences by using PCR site - - specific oligonucleotide mutagenesis (see [Kath96], [Kari97]). The process is a variation of PCR in which a change in the template can be induced by the process of primer modification. Namely, one can use a primer that is only partially complementary to a template fragment. (The modified primer should contain enough bases complementary to the template to make it anneal despite the mismatch.) After the primer is extended by the polymerase, the newly obtained strand consist of the complement of the template in which a few nucleotides have been substituted by other, desired ones.

Detecting and Reading

given the contents of a tube, say YES if it contains at least one DNA strand, and NO otherwise ([Adle94] [Adle96], [Kari97]). PCR may be used to amplify the result and then a process called sequencing is used to actually read the DNA strands in solution. The basic idea of the most widely used sequencing method is to use PCR and gel electrophoresis. Assume to have a homogeneous solution, that is, a solution containing mainly copies of the strand we wish to sequence, and very few other strands. For detection of the positions of As in the target strand, a blocking agent is used that prevents the templates from being extended beyond As during PCR. As a result of this modified PCR, a population of subsequences is obtained, each corresponding to a different occurrence of A in the original strand. Separating them by length using gel electrophoresis reveals the positions where A occurs in the strand. The process can then be repeated for each of C, G, and T , to yield the sequence of the strand. Recent methods use four different fluorescent dyes, one for each base, which allows all four bases to be processed simultaneously. As the fluorescent molecules pass a detector near the bottom of the gel, data are output directly to an electronic computer.

Subsequence Extraction

This operation involves the extraction from a test tube of all those single strands, which contain a specific short sequence (e.g., extract all strands containing the sequence 5' -AGACTG).To extract single strands containing the sequence x we first create many copies of its complement, $\bar{x}$. We attach to these oligonucleotides a biotin molecule (biotination)which bind in turn to a fixed matrix of avidin molecules at the cross points. If we pour the contents of the test tube over this matrix, strands containing x will anneal to the anchored complementary strands. Washing the matrix removes all strands that did not anneal, leaving only strands containing x. These may then be removed from the matrix. This opearation can be performed with specific types of sequences using other methods (e.g., restriction enzymes as in [AmosTh97]).

Mark

This operation tags strands so that they can be separated or otherwise operated upon selectively. Marking is commonly implemented by making a single strand into a double strand through annealing or the action of a polymerase. However, it can also mean appending a tag sequence to the end of a strand or even methylation of the DNA or (de)phosphorylation of the 5' ends of the strands. Unmark The complement of the marking operation. Unmark removes the marks on the strands.

Substitution

Substitute, insert or delete DNA sequences by using PCR site specific oligonucleotide mutagenesis ([BDSL95]). The process is a variation of PCR in whichachange in the template can be induced by the process of primer modification. Namely, one can use a primer that is only partially complementary to a template fragment. The modified primer should contain enough bases complementary to the template to make it anneal despite the mismatch. After the primer is extended by the polymerase, the newly obtained strand will consist of the complement of the template in which a few oligonucleotides have been substituted by other, desired ones.

Destroying

The marked strands by using exonucleases, or by cutting all the marked strands with a restriction enzyme and removing all the intact strands by gel electrophoresis.By using enzymes called exonucleases, either doublestranded or singlestranded DNA molecules may be selectively destroyed. The exonucleases chew up DNA molecules from the end inward, and exist with specificity to either singlestranded or doublestranded form.

⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆

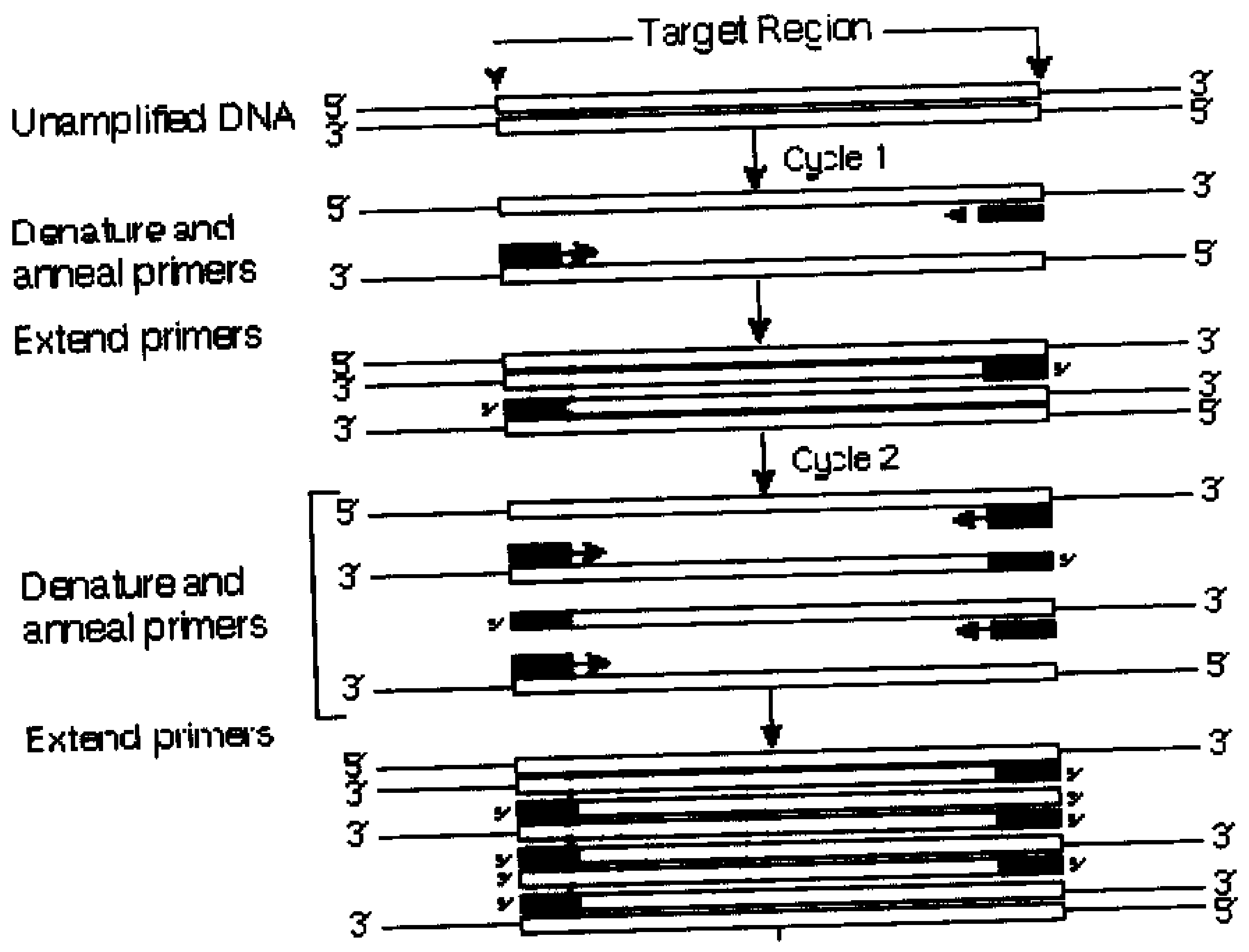


Figure A.1: **PCR Cycles**

Appendix B

Common Restriction Enzymes

SN.	Enzyme	Restriction Site	Result	Cut Type
1	EcoRI	$\updownarrow GAATTC$	$\updownarrow G \downarrow AATT$	sharp
2	XmaI,HpaH, MspI	$\updownarrow CCCGGG$	$\updownarrow C \downarrow CCGG$	sharp
3	PstI	$\updownarrow CTGCAG$	$\updownarrow C \uparrow TGCA$	sharp
4	SmaI	$\updownarrow CCCGGG$	$\updownarrow CCC$	blunt
5	BamHI	$\updownarrow GGATCC$	$\updownarrow G \downarrow GATC$	sharp
6	BalI	$\updownarrow TGGCCA$	$\updownarrow TGG$	blunt
7	HaeII	$\updownarrow (A/G)GCGC(C/T)$	$\updownarrow (A/G) \uparrow GCGC$	sharp
8	HaeIII	$\updownarrow CCCGGG$	$\updownarrow GG$	blunt
9	HindII	$\updownarrow GT(C/T)(A/G)AC$	$\updownarrow GT(C/T)$	blunt
10	HindIII	$\updownarrow AAGCTT$	$\updownarrow A \downarrow (AGCT)$	sharp
11	HpaI	$\updownarrow GTTAAC$	$\updownarrow GTT$	blunt
12	HpaII	$\updownarrow CCGG$	$\updownarrow C \downarrow CG$	sharp

Note: Conventions have been adopted from [BDSL95] abd [Bis98]. Where $\uparrow$x represent a ssDNA x with $5'$ - $3'$ direction, similarly $\downarrow$x represents the ssDNA x with $3'$ - $5'$ direction; this way $\uparrow$x and $\downarrow$x are complement to each other. $\updownarrow$x indicates dsDNA. A mix of both $\uparrow$ and $\downarrow$ indicates that each strand has structure directed by one arrow starting from that arrow towards right until either of right end or the other arrow are reached.

Bibliography

[AD97] Martyn Amos and Paul E. Dunne. "'DNA simulation of boolean circuits"'. Technical Report CTAG-97009, Department of Computer Science, University of Liverpool, UK, December 1997.

[Adle94] Adleman L. M. "Molecular computation of solutions to combinatorial problems", *Science*, 266:1021–1024.

[Adle96] Adleman L. M. "On constructing a molecular computer". [LB96], pp. 1-22.

[AGH96] M.Amos, A.Gibbons, D.Hodgson. "Errorresistant implementation of DNA computation". *2nd DIMACS workshop on DNA basedcomputers*, Princeton, 1996, pp. 87–101.

[AmosTh97] Martyn Amos."DNA Computation". *PhD thesis*, Department of Computer Science, University of Warwick, UK, September 1997, `http://www.csc.liv.ac.uk/ ctag/archive/th/amos-thesis.ps.gz`.

[AKL86] Roger L. P. Adams, John T. Knowler, and David P. Leader. "The Biochemistry of the Nucleic Acids ". *Chapman and Hall*, tenth edition, 1986.

[ARRW96] L.Adleman, P.Rothemund, S.Roweis, E.Winfree."On applying molecular computation to the Data Encryption Standard". *2nd DIMACS workshop on DNA basedcomputers*, Princeton, 1996, pp. 28–48.

[Ata00] A. Atanasiu. "'Arithmetic with membranes"'. In Workshop on Multiset Processing, Curtea de Arges, Romania, August 2000, pages 1–17

[AWHOG98] Martyn Amos, Steve Wilson, David A. Hodgson, Gerald Owenson, and Alan Gibbons. "'Practical implementation of DNA computations"'. In Calude et al., proceedings of the First International Conference on Unconventional Models of Computation, Auckland, New Zealand, Jan5–11 1998, pages 1–18.

[AYT98] Yohei Aoi, Tatsuo Yoshinobu, Katsuyuki Tanizawa, Kozo Kinoshita, and Hiroshi Iwasaki. "'Ligation errors in DNA computing"'. In Proceedings 4th DIMACS Workshop on DNA Based Computers, held at the University of Pennysylvania, Philadelphia, USA June 15 - June 19, 1998, pages 181–187

[B66] Berger, R. The Undecidability of the Domino Problem, Memoirs of the American Mathematical Society, 66, (1966).

[Baum95] Baum, E. B. "'How to build an associative memory vastly larger than the brain"'. Science, April 28, 1995.

[Baum96] Baum, E. B. "'DNA Sequences Useful for Computation"'. 2nd Annual DIMACS Meeting on DNA Based Computers, Princeton University, June 1996.

[BDSL95] Boneh, D., C. Dunworth, R. Lipton, J. Sgall. "'On the Computational Power of DNA"'. Princeton CS TechReport number CSTR49995 1995.

[BDSL96] D.Boneh, C.Dunworth, J.Sgall, R.Lipton. "Making DNA computers error resistant". *2nd DIMACS workshop on DNA basedcomputers*, Princeton, 1996, pp. 102 – 110.

[BDL95] Boneh, D., C. Dunworth, R. Lipton, "'Breaking DES Using a Molecular Computer"', Princeton CS TechReport number CSTR48995 (1995).

[Bea94] Beaver, D. "'Factoring: The DNA Solution"'. Advances in Cryptology, Asia Crypt94 Proceedings, Lecure Notes in Computer Science, (1994), Springer Verlag, (http:// www.cse.psu.edu/ beaver/ publications/ pubindex.html),

[Bea95] Donald Beaver. "A universal molecular computer". Condensed abstract (of [Bea96]) for DIMACS Workshop of April 4, 1995.

[Bea96] Donald Beaver. "Molecular Computing". *Technical Report* TR 95001, Penn State University, USA, January 1995.

[BF97] Beigel, R. and Bin Fu. "'Molecular Approximation Algorithm for NP Optimization Problems"'. 3rd DIMACS Meeting on DNA Based Computers, Univ. of Penns., (June, 1997).

[BL95] Boneh, D., and R. Lipton, "'A Divide and conquer approach to DNA sequencing"', Princeton University, 1996.

[Brown93] T. A. Brown, "Genetics: A Molecular Approach",*Champan And Hall*, 1993.

[BS91] Bartel, D. and J. Szostak. "'Isolation of new ribozymes from a large pool of random sequences"'. Science, 261, 1411–1418, (1991).

[Bis98] Somenath Biswas. "'A note on DNA representation of binary strings"'. In Computing with Bio - Molecules. Theory and Experiments, pages 153–157, 1998.

[CAF] Cross reference in [Das96].

[CFL98] Cukras, A., D. Faulhammer, R. Lipton, L. Landweber "'Chess games: A model for RNAbased computation"', 4th Int. Meeting on DNABased Computing, Baltimore, Penns., (June, 1998).

[CW99] Kevin Chen and Erik Winfree. "'Error correction in DNA computing: Misclassification and strand loss"'. In *Erik Winfree and David K. Gifford, editors. Proceedings 5th DIMACS Workshop on DNA Based Computers*, held at the Massachusetts Institute of Technology, Cambridge, MA, USA June 14 - June 15, 1999. American Mathematical Society, 1999, pages 49–63.

[Das96] J.H.M. Dassen."Molecular computation and splicing systems". Msc thesis, Leiden University, August 1996,`http://www.wi.LeidenUniv.nl/~jdassen/thesis.ps.gz`.

[DMRGFS97] R.Deaton, R.Murphy, J.Rose, M.Garzon, D.Franceschetti, S.Stevens. "A DNA based implementation of an evolutionary search for good encodings for DNA computation". Proceedings of 1997 IEEE International Conference on Evolutionary Computation, Indianapolis, pp. 267–271.

[Eng98] Eng, T. "'On solving a 3CNFSatisfiability with an in vivo algorithm"'. 4th Int. Meeting on DNABased Computing, Baltimore, Penns., (June, 1998).

[ES97] Eng, T., and B.M. Serridge. "'A SurfaceBased DNA Algorithm for Minimal Set Cover"', 3rd DIMACS Meeting on DNA Based Computers, Univ. of Penns., (June, 1997).

[FBZ98] Fu, B., R. Beigel, F. Zhou, "An O(2^n) volume molecular algorithm for Hamiltonian Path",*4th Int. Meeting on DNABased Computing*, Baltimore, Penns., (June, 1998).

[Feynman61] R. P. Feynman, "There's Plenty of Room at the Bottom", In D. Gilbert editor, *Miniaturization*, pp. 1021–1024, Reinhold 1961.

[FLL98] Faulhammer, D., R. Lipton, L. Landweber, Counting DNA: Estimating the complexity of a test tube of DNA, 4th Int. Meeting on DNABased Computing, Baltimore, Penns., (June, 1998).

[Fri00] P. Frisco. "Parallel arithmetic with splicing". Romanian Journal of Information Science and Technology (ROMJIST), 3(2):113–128, 2000.

[GAH97] Alan Gibbons, Martyn Amos, and David Hodgson. "DNA computing". *Current Opinion in Biotechnology*, 8:1997, pp.103–106.

[GKG98] Gloor, G., L. Kari, M. Gaasenbeek, S. Yu, "Towards a DNA solution to the shortest common superstring problem", 4th Int. Meeting on DNABased Computing, Baltimore, Penns., (June, 1998). Also, Proceedings of IEEE'98 International Joint Symposia on Intelligence and Systems, Rockville, MD, 140–145, (May 1998).

[GJ79] Garey, M. R., and D. S. "Johnson Computers and Intractability: A Guide to the Theory of NPCompleteness", W.H Freeman and Company, page 257, 1979.

[GPZ97] Vineet Gupta, Srinivasan Parthasarathy, and Mohammed J. Zaki. "Arithmetic and logic operations with DNA". In Proceedings of the 3rd DIMACS Workshop on DNA Based Computers, held at the University of Pennsylvania, June 23 - 25, 1997, pages 212-220.

[GFB96] F. Guarneiri, M, Fliss, C. Bancroft. "'Making DNA Add"'. Science,vol. 273,pp. 220 - 223, July 12, 1996.

[GR98] Gehani, A., J. Reif, "'Micro Flow BioMolecular Computation"', 4th Int. Meeting on DNABased Computing, Baltimore, Penns., (June, 1998). `http://www.cs.duke.edu/geha/public html/misc/biosystems.ps`

[HA97] Hagiya, M., and M. Arita. "'Towards Parallel Evaluation and Learning of Boolean μ Formulas with Molecules"', 3rd DIMACS Meeting on DNA Based Computers, Univ. of Penns., (June, 1997).

[Hart95] J.Hartmanis. "On the weight of computations". *Bulletin of the European Association of Theoretical Computer Science*, 55:1995,pp. 136–138.

[Harrison65] M.A. Harrison."Introduction to switching and automata theory". McGrawHill,1965.

[HS01] Hubert Hug and Rainer Schuler. "'DNA -based parallel computation of simple arithmetic"'. In DNA Computing, 7th international Workshop on DNA - Based Computers, DNA 2001, Tampa, U.S.A., 10 - 13 June 2001, pages 159–166.

[Hayes88] John P. Hayes. "'Computer Architecture and Organization"'. McGraw - Hill International Editions, 2nd ed. 1988, Singapore, ISBN 0 - 07 - 027366 - 9.

[HPP96] Thomas Head, Gheorghe Paun, and Dennis Pixton. "Generative Mechanisms Suggested by DNA Recombination", In Volume 2 of *Handbook of Formal Languages* by Rozenberg and Salomaa , October 1996, ISBN.

[InGel90] Innis, M. A., and D. H. Gelfand. "Optimization of PCRs". In *PCR Protocols : A guide to methods and applications.* Academic Press, New York 1990.

[JK96] Jonoska, N., and S.A. Karl, "'A Molecular Computation of the Road Coloring Problem"', 2nd Annual DIMACS Meeting on DNA Based Computers, Princeton University, June 1996.

[JKS98] Jonoska, N, S. Karl, M. Saito. "Three dimensional DNA structures in computing" 4th Int. Meeting on DNABased Computing, Baltimore, Penns., (June, 1998).

[Kath96] Lila Kari and Gabriel Thierrin."Contextual insertions/deletions and computability". *Information and Computation*, 131, 1(1996), pp 47-61.

[Kari97] L.Kari. "DNA computing the arrival of biological mathematics". *The Mathematical Intelligencer*, vol.19, 2(1997),pp 9-22.

[Kaplan95] P. Kaplan, G. cecchi, A. Libchaber. "'Molecular Computation: Adleman's experiment repeated"'.Technical Report, NEC Research Institute,1995.

[KGL96] P.Kaplan, G.Cecchi, A.Libchaber. "DNAbased molecular computation: template to template interactions in PCR". 2nd DIMACS workshop on DNA basedcomputers, Princeton, 1996, pp.159–171.

[KMRS96] S.Kurtz, S.Mahaney, J.Royer, J.Simon."Active transport in biological computing". *2nd DIMACS workshop on DNA basedcomputers*, Princeton, 1996, pp. 111–121.

[Lip94] Lipton, R. "'Speeding Up Computations via Molecular Biology,"' Princeton University Draft, (1994).

[LB96] R.J.Lipton, E.M.Baum. Eds.: "DNA Based Computers I", *Proceedings of a DIMACS Workshop*, Princeton, 1995, American Mathematical Society, 1996.

[LFW98] Liu, Q., A. Frutos, L. Wang, A. Thiel, S. Gillmor, T. Strother, A. Condon, R. Corn, M. Lagally, L. Smith. "'Progress towards demonstration of a surface based DNA computation: A one word approach to solve a model satisfiability problem"', 4th Int. Meeting on DNABased Computing, Baltimore, Penns., (June, 1998).

[LGCCL96] Liu, Q., Z. Guo, A.E. Condon, R.M. Corn, M.G. Lagally, and L.M. Smith. "'A SurfaceBased Approach to DNA Computation"'. Proc. 2nd Annual Princeton Meeting on DNABased Computing, June 1996.

[Lip98 96] Lipton, R., "'A memory based attack on cryptosystems with application to DNA computing"', 4th Int. Meeting on DNABased Computing, Baltimore, Penns., June, 1998.

[LL97] Landweber, L.F. and R. Lipton. "'DNA 2 DNA Computations: A Potential 'Killer App'?"'. 3nd Annual DIMACS Meeting on DNA Based Computers, University of Pens., (June 1997).

[LTCSC97] Liu, Q., A.J. Thiel, A.G. Frutos, R.M. Corn, L.M. Smith, "'SurfaceBased DNA Computation: Hybridize and Destruction"', 3rd DIMACS Meeting on DNA Based Computers, Univ. of Penns., (June, 1997).

[LGCCLS96] Q.Liu, Z.Guo, A.Condon, R.Corn, M.Lagally, L.Smith."A surface-based approach to DNA computation". *2nd DIMACS workshop on DNA basedcomputers*, Princeton, 1996, PP 206–216.

[LYR98] Thomas H. LaBean, Hao Yan, John H. Reif and Nadrian Seeman. "'Construction and Analysis of a DNA Triple Crossover Molecule"', November 1998.

[Mir96] K.Mir. "A restricted genetic alphabet for DNA computing". *2nd DIMACS workshop on DNA basedcomputers*, Princeton, 1996, pp. 128 – 130.

[MYP98] Mills, A., B. Yurke, P. Platzman. "'Errortolerant massive DNA neuralnetwork computation"'. 4th Int. Meeting on DNA Based Computing, Baltimore, Penns., (June, 1998).

[O96] Oliver, J.S. "'Computation With DNAMatrix Multiplication"', 2nd Annual DIMACS Meeting on DNA Based Computers, Princeton, June, 1996.

[OP94] R. Old and S. Primrose. "Principles of gene manipulation, an introduction to genetic engineering". *Blackwell Scientific Publications*, fifth edition, 1994.

[OG96] Mitsunori Ogihara and Animesh Ray. "Simulating Boolean circuits on a DNA computer". Technical Report 631, University of Rochester, August 1996.

[OR97] Ogihara, M., and A. Ray, "'Breadth first search 3SAT algorithms for DNA computer"', Technical Report TR629, Department of Computer Science, University of Rochester, (July 1996).

[Paun98] Gheorghe Paun. "Computing with Membranes".*Turku Centre for Computer Science TUCS Technical Report* No 208,November 1998, ISBN 952120303X ISSN 12391891.

[Pi96] Pixton, D. "'Regularity of splicing languages"'. Discrete Applied Math., 69, 101–124, (1996).

[PRS98] Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa. "DNA Computing: NewComputing Paradigms".*Springer Verlag*, Berlin, Heidelberg, New York., September 1998, ISBN 3-540-64196-3.

[Pyne01] S. Pyne. "Solving NP - Complete Problems with tractable number of DNA Strands on Average - Part I". Indian Statistical Institute, Calcutta, Technical Report No. - SMU/04/01, June 2001.

[Reif97] Reif, J.H. "'Local Parallel Biomolecular Computation"'. at `http://www.cs.duke.edu/~reif/paper/Assembly.ps`

[QL98] Z. F. Qiu and M. Lu. "'Arithmetic and logic operations for dna computers"'. Second IASTED International Conference on Parallel and Distributed Computing and Networks, pages 481–486, December 1998, Brisbane, Australia.

[Reif98] John .H. Reif. "Alternative Computational Models: A Comparison of Biomolecular and Quantum Computation",18th *Invited paper, International Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS98)*, December, 1998.

[Rot96] Paul Rothemund. "A DNA and restriction enzyme implementation of Turing Machine". In *'DNA based computer'*,American Mathematical Society, ISBN 08218005185, 1996.

[Rei95] Reif, J., "'Parallel Molecular Computation: Models and Simulations"', Seventh Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA95), ACM, Santa Barbara, 213–223, June 1995.

[Rowis96] Sam Roweis, Erik Winfree, Richard Burgoyne, Nickolas Chelyapov, Myron Goodman, Paul Rothemund, and Leonard Adleman. "'A sticker based architecture for DNA computation".In *Proceedings of the Second Annual Meeting on DNA Based Computers, DIMACS : Series in Discrete Mathematics and Theoretical Computer Science.*, June 1996.

[SKK98] Sakamoto, K., D. Kiga, K. Komiya, H. Gouzu, S. Yokoyama, S. Ikeda, H. Sugiyama, M. Hagiya. "'State transitions by molecules"'. 4th Int. Meeting on DNABased Computing, Baltimore, Penns., (June, 1998).

[SM97] Setubal, J., and J. Meidanis. "'Introduction to Computational Molecular Biology"'. PWS Pub. Co., Chapt 9, (1997).

[SZDC95] Seeman, N. C., Y. Zhang, S.M. Du, and J. Chen"' Construction of DNA polyhedra and knots through symmetry "'*Minimization, Supramolecular Stereochemistry*, J. S. Siegel, ed.,pp. 27–32, (1995).

[SZC94] Seeman, N. C., Y. Zhang, and J. Chen. "'DNA nanoconstructions"'. J. Vac. Sci. Technol., 12:4, 1895–1905, (1994).

[Waterman95] T. A. Waterman, "Introduction to Computational Biology Maps, Sequences and Genomes",*Champan And Hall*, first ed. 1995.

[Winfree95] Erik Winfree. "Complexity of restricted and unrestricted models of molecular computation". At , `http://dope.caltech.edu/winfree/Papers/models.ps.gz`.

[Winfree96] Erik Winfree, Xiaoping Yang, and Nadrian C. Seeman."Universal computation via self - assembly of DNA: Some theory and experiments". In *Proceedings of the Second Annual Meeting on DNA Based Computers, held at Princeton University, June 10 - 12, 1996*.pp 172–190, `ftp://hope.caltech.edu/pub/winfree/DIMACS/self - assem.ps`. Draft.

[Winfree98] Erik Winfree."'Whiplash PCR for O(1) computing"', 4th Int. Meeting on DNABased Computing, Baltimore, Penns., (June, 1998).

[WLW98] Erik Winfree, Furong Liu, Lisa A. Wenzler, Nadrian C. Seeman. "'Design and SelfAssembly of Two Dimensional DNA Crystals"', Nature 394: 539–544, 1998. (1998).

[WYS96] Eric Winfree, , X. Yang, N.C. Seeman. "'Universal Computation via Self-assembly of DNA: Some Theory and Experiments"' 2nd Annual DIMACS Meeting on DNA Based Computers, Princeton, June, 1996.

[YK97] Yokomori, T., and S. Kobayashi. "'On the Power of Circular Splicing Systems and DNA Computability."' ICEC'97 Special Session on DNA Based Computation, Indiana, April, 1997.

[YoSh00] Hiroshi Yoshida, Akira Suyama, "Solution to 3- SAT by Breadth first-Search", In*DAMACS, Series in Discrete Mathematics and Theoreticcal computer Science"* American Methamatical Society, Rhode Island, Vol 54, pp 9–22, 2000.

[YYSHTSMO99] Masahito Yamamoto, Jin Yamashita, Toshikazu Shiba, Takuo Hirayama, Shigeharu Takiya, Keiji Suzuki, Masanobu Munekata, and Azuma Ohuchi. "A study on the hybridization process in DNA computing". Erik Winfree and David K. Gifford, editors. Proceedings 5th DIMACS Workshop on DNA Based Computers, held at the Massachusetts Institute of Technology, Cambridge, MA, USA June 14 - June 15, 1999. American Mathematical Society, 1999, pages 101–110.