DNA Computation

by

Martyn Amos

Thesis

Submitted to the University of Warwick

for the degree of

Doctor of Philosophy

Department of Computer Science

September 1997

Contents

List of Tables

List of Figures

Acknowledgments

First and foremost, I would like to thank my supervisor, **Prof Alan Gibbons**. Over the past few years, Alan has become a friend as well as a supervisor, and I owe much to his wisdom, generosity, patience and humour. I hope that this thesis represents the beginning of a long and fruitful relationship.

Without **my family**, I would never have been in a position to undertake a Ph.D. I thank them for their love, support (both emotional and financial) and for their frequent understanding.

The work presented in this thesis is partly the result of inter-disciplinary collaboration with colleagues in the Department of Biological Sciences at the University of Warwick. **Dr David Hodgson** was originally called upon to assess the viability of our embryonic research program. I am sure that he then had no idea what an important member of the team he would become. Dave's knowledge of the realities of molecular biology often prevented us from going up blind alleyways, and his supervision of the experimental work was vital to the project. On that note, I also thank **Steve Wilson** for his excellent work in the laboratory, **Andrew Easton** for his supervision of Steve, and **Gerald Owenson** for carrying on where Steve left off.

No thesis is written without being influenced by the thoughts of others. The

following colleagues from the international community provided rich discussions. corrected my mistakes and suggested lines of enquiry: **Len Adleman (Southern California), Paul E. Dunne (Liverpool), Chris Tofts (Leeds) and Erik Winfree (Caltech). Somasundaram Ravindran (John Moore's, Liverpool)** provided guidance and friendship in the early stages of my Ph.D.

Several people have kindly provided resources that greatly helped in the writing of this thesis. Figures 2.2 and 2.4 are used with the permission of the Access Excellence forum (http://www.gene.com:80/ae/AB/index.html). sponsored by **Genentech, Inc.** **J.H.M. Dassen** maintains an excellent on-line bibliography (http://www.wi.leidenuniv.nl/~jdassen/dna.html), and **Mark Hadley** provided the Warwick LaTeXstyle file used in the production of this thesis.

Over the course of my Ph.D work, the following friends provided caring support: **Katina Glancy, Will Barton, John Garrigan** and **all on UMR.** My partner, **Katie Deacon**, deserves a special mention for her unselfish love and support.

The work of this thesis was supported by an EPSRC Research Studentship, and by grants from the University of Warwick Research and Teaching Innovations Sub-Committee and the BBSRC/EPSRC Bioinformatics Program.

Declarations

This thesis is submitted to the University of Warwick in support of my application for admission to the degree of Doctor of Philosophy. No part of it has been submitted in support of an application for another degree or qualification of this or any other institution of learning. Parts of the thesis appeared in the following refereed papers in which my own work was that of a full pro-rata contributor:

1. Martyn Amos, Alan Gibbons and David Hodgson, "Error-resistant implementation of DNA computations", In *Proceedings 2nd Annual Meeting on DNA Based Computers*, Princeton, NJ. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. Providence, RI: American Mathematical Society; 1996 (in press)

2. Alan Gibbons, Martyn Amos and David Hodgson, "Models of DNA Computation", In *Proceedings Mathematical Foundations of Computer Science (MFCS)*, Cracow, Poland, 1996. Lecture Notes in Computer Science 1113. Penczek and Szalas (Eds.) pp. 18–36, Springer-Verlag

3. Alan Gibbons, Martyn Amos and David Hodgson, "DNA Computing". *Current Opinion in Biotechnology*, Vol. 8, No. 1, February 1996, pp. 103–106

4. Martyn Amos, Alan Gibbons and Paul E. Dunne, "The Complexity and Viability of DNA Computations". In *Proceedings of the International Conference on Bio-computing and Emergent Computation (BCEC97)*, Skövde, Sweden, September 1-2, 1997, Lundh, Olsson and Narayanan (Eds.) pp. 165–173. World Scientific

5. Martyn Amos and Alan Gibbons, "Practical Implementation of DNA Computations", Invited paper in *Proceedings of the First International Conference on Unconventional Models of Computation (UMC)*, Auckland, New Zealand, 5-11 January, 1998, Springer-Verlag (to appear)

Unrefereed papers were also presented as follows:

1. Martyn Amos," A New Model of DNA Computation", *12th British Colloquium on Theoretical Computer Science (BCTCS)*, University of Kent, UK, 1-4 Apr. 1996 (Abstract in *Bulletin of the EATCS*, Vol. 61, Feb. 1997)

2. Martyn Amos and Alan Gibbons, "The Complexity and Viability of DNA Computations" (invited paper); *13th British Colloquium on Theoretical Computer Science (BCTCS)*, Sheffield, UK, 23-26th March 1997 (Abstract to appear in *Bulletin of the EATCS*)

The following paper has been submitted for review:

1. Alan Gibbons, Martyn Amos and Paul E. Dunne, "Counting the Cost of DNA Computation", submitted to *Theoretical Computer Science*

M. Amos Date: 16 September, 1997

Abstract

This is the first ever doctoral thesis in the field of DNA computation. The field has its roots in the late 1950s, when the Nobel laureate Richard Feynman first introduced the concept of computing at a molecular level. Feynman's visionary idea was only realised in 1994, when Leonard Adleman performed the first ever truly molecular-level computation using DNA combined with the tools and techniques of molecular biology. Since Adleman reported the results of his seminal experiment, there has been a flurry of interest in the idea of using DNA to perform computations. The potential benefits of using this particular molecule are enormous: by harnessing the massive inherent parallelism of performing concurrent operations on trillions of strands, we may one day be able to compress the power of today's supercomputer into a single test tube. However, if we compare the development of DNA-based computers to that of their silicon counterparts, it is clear that molecular computers are still in their infancy. Current work in this area is concerned mainly with abstract models of computation and simple proof-of-principle experiments. The goal of this thesis is to present our contribution to the field, placing it in the context of the existing body of work. Our new results concern a general model of DNA computation, an error-resistant implementation of the model, experimental investigation of the implementation and an assessment of the complexity and viability of DNA computations. We begin by recounting the historical background to the search for the structure of DNA. By providing a detailed description of this molecule and the operations we may perform on it, we lay down the foundations for subsequent chapters. We then describe the basic models of DNA computation that have been proposed to date. In particular, we describe our *parallel filtering* model, which is the first to provide a general framework for the elegant expression of algorithms for NP-complete problems. The implementation of such abstract models is crucial to their success. Previous experiments that have been carried out suffer from their reliance on various error-prone laboratory techniques. We show for the first time how one particular operation, hybridisation extraction, may be replaced by an error-resistant *enzymatic separation* technique. We also describe a novel solution read-out procedure that utilizes cloning, and is sufficiently general to allow it to be used in any experimental implementation. The results of preliminary tests of these techniques are then reported. Several important conclusions are to be drawn from

these investigations, and we report these in the hope that they will provide useful experimental guidance in the future. The final contribution of this thesis is a rigorous consideration of the complexity and viability of DNA computations. We argue that existing analyses of models of DNA computation are flawed and unrealistic. In order to obtain more realistic measures of the time and space complexity of DNA computations we describe a new *strong* model, and reassess previously described algorithms within it. We review the search for "killer applications": applications of DNA computing that will establish the superiority of this paradigm within a certain domain. We conclude the thesis with a description of several open problems in the field of DNA computation.

Chapter 1

Introduction

" Where a calculator on the ENIAC is equipped with 18,000 vacuum tubes and weighs 30 tons, computers in the future may have only 1,000 vacuum tubes and perhaps weigh 1 1/2 tons." So said *Popular Mechanics* in 1949 [8]. Today, in the age of smart cards and wearable PCs, this statement is striking because it falls so short of reality. In fifty years from now, who would be prepared to predict how close to the levels of molecular miniaturisation described in Feynman's visionary paper [26] we will have come?

Huge advances in miniaturization have been made since the days of room-sized computers, yet the underlying computational model (the Von Neumann architecture) has remained the same. Today's supercomputers still employ the kind of sequential logic used by the mechanical dinosaurs of the 1930s. Some researchers are now looking beyond these boundaries and are investigating entirely new media and computational models. These include quantum, optical and DNA-based computers. It is the last development that this thesis concentrates on.

The idea that living cells and molecular complexes can be viewed as potential

machinic components dates back to the late 1950s, when Richard Feynman delivered his famous paper describing "sub-microscopic" computers. More recently, several papers [2, 7, 52] (also see [5, 36, 64]) have advocated the realisation of massively parallel computation using the techniques and chemistry of molecular biology. The development of existing silicon-based computers was only made possible by the invention of the transistor, which facilitated for the first time electronic manipulation of silicon. We may draw an interesting parallel between this historical precedent and the development of molecular-scale computers. Although the concept dates back to the late 1950s, only now do we have at our disposal the tools and techniques of molecular biology required to construct the prototype molecular computers. In [2], Adleman described how a computationally intractable problem, known as the *directed Hamiltonian Path Problem* (HPP) might be solved using molecular methods. Recall that the HPP involves finding a path through a graph that visits each vertex exactly once. Adleman's method employs a simple, massively parallel random search. The algorithm is not executed on a traditional, silicon-based computer, but instead employs the "test-tube" technology of genetic engineering. By representing information as sequences of bases in DNA molecules, Adleman shows how existing DNA-manipulation techniques may be used to quickly detect and amplify desirable solutions to a given problem.

How can we combine a flask of DNA with biological tools to solve a hard mathematical problem? Adleman's experiment proceeds as follows. The first stage created a flask of DNA molecules, each molecule encoding a potential solution to the problem. With reference to the HPP, for example, each strand encoded a path (not necessarily Hamiltonian) through the graph. Given every DNA molecule that encodes a path of length n, for a graph with n vertices, we can be sure that every

possible solution is present, some legal, but most illegal. Once the entire solution space was present in a flask the DNA computer really came into its own. Adleman used a small set of biological tools to "sift" out DNA that encoded illegal solutions. These are those paths that do not visit every vertex, or paths that visit a particular vertex more than once. At the end of the sifting process, he was left only with strands that encoded legal solutions.

Of course, for DNA computers, each individual operation, for example, extracting DNA strands, can take minutes or even hours to perform. This cost of a computational step, when compared to that of supercomputers capable of executing a trillion operations a second, looks unimpressive. However, the real power of DNA computers lies in their inherent parallelism – each operation is performed not on one single DNA strand, but on every strand in the flask *simultaneously*. The fastest supercomputers in existence today are capable of executing around a trillion operations a second. DNA computers have the potential to execute more than a *thousand* trillion operations per second, as well as being a billion times more energy-efficient and requiring a trillionth of the space needed by existing storage media. Nature has information compression down to a fine art – over forty 1 Mb floppy discs are required to store the genome of a single fruit fly [74].

The rest of this thesis is organized as follows.

- In chapter 2 we explain the structure of the DNA molecule and describe a variety of laboratory techniques for its manipulation. This provides an important foundation for the work presented in subsequent chapters.

- In chapter 3 we review the models of DNA computing. A summary of this chapter appears in [35]. In particular, we describe the novel *parallel filtering*

model, proposed by us in [7]. Although the earlier papers of Adleman [2] and Lipton [52] motivated such work, our parallel filtering model is the first to provide an elegant, general framework for the expression of algorithms for various NP-complete problems. In chapter 2 we propose a basic taxonomy of models of DNA computation, and describe an archetypal model within each category. We then describe in detail the common features of the so-called *filtering* models. This provides a foundation for a description of the operations within the parallel filtering model. In contrast to earlier work, where only a single algorithm is generally detailed, we describe several algorithms for NP-complete problems within the parallel filtering model. We also describe in detail an example of a *constructive* model, due to Ogihara and Ray [61], within which we may describe the simulation of Boolean circuits using DNA manipulation techniques.

- In chapter 4 we consider various issues arising from the implementation of theoretical models of DNA computing. In particular, we explain in detail the implementation of the models described in chapter 3. This implementation is also described in [7, 34]. We highlight several important problems with existing implementations, and describe an implementation of the parallel filtering model. We argue that the parallel filtering model provides a greater degree of error-resistance than those previously proposed.

- In chapter 5 we describe experimental investigations of the implementation described in chapter 4. Although these experiments are still at a preliminary stage, they have already highlighted several important factors to be taken into consideration when designing and implementing models of DNA computation.

Our investigations are far more rigorous than those previously described, since we perform numerous control and optimisation experiments. We describe the lessons to be drawn from such experiments, and suggest several potential techniques for reducing errors in future empirical studies. In particular, we show how reliance on error-prone techniques such as PCR and hybridisation extraction may be obviated. We also describe a novel technique for the read-out of the final result of a DNA computation. This technique is sufficiently general to allow it to be included in the implementation of any theoretical model.

- In chapter 6 we consider the complexity and viability of DNA computations. Such issues have, to date, been largely underestimated in the literature. We argue that existing analyses of models of DNA computation are flawed and unrealistic. In order to obtain more realistic measures of the time and space complexity of DNA computations we describe a new *strong* model, and re-assess previously described algorithms within it. We review the search for "killer applications": applications of DNA computing that will establish the superiority of this paradigm within a certain domain. A summary of this chapter appeared in [6].

- In chapter 7 we summarise this thesis, give some concluding remarks and suggest several open problems in the field of DNA computation.

Chapter 2

DNA structure and manipulation

2.1 Introduction

Ever since ancient Greek times, man has suspected that the features of one generation are passed on to the next. It was not until Mendel's work on garden peas was recognised (see [38, 75]) that scientists accepted that both parents contribute material that determines the characteristics of their offspring. In the early 20th century, it was discovered that *chromosomes* make up this material. Chemical analysis of chromosomes revealed that they are composed of both *protein* and *deoxyribonucleic acid*, or *DNA*. The question was, which substance carries the genetic information? For many years, scientists favoured protein, because of its greater complexity relative to that of DNA. Nobody believed that a molecule as simple as DNA, composed of only four subunits (compared to 20 for protein) could carry complex genetic information.

It was not until the early 1950s that most biologists accepted the evidence showing that it is in fact DNA that carries the genetic code. However, the physical structure of the molecule and the hereditary mechanism was still far from clear.

In 1951, the biologist James Watson moved to Cambridge to work with a physicist, Francis Crick. Using data collected by Rosalind Franklin and Maurice Wilkins at King's College, London, they began to decipher the structure of DNA. They worked with models made out of wire and sheet metal in attempt to construct something that fitted the available data. Once satisfied with their model, they published the paper [78] (also see [77]) that would eventually earn them (and Wilkins) the Nobel Prize for Physiology or Medicine in 1962.

2.2 The structure and manipulation of DNA

DNA (deoxyribonucleic acid) [1, 76] encodes the genetic information of cellular organisms. It consists of *polymer chains*, commonly referred to as DNA *strands*. Each strand may be viewed as a chain of *nucleotides*, or *bases*, attached to a sugar-phosphate "backbone". An n-letter sequence of consecutive bases is known as an n-mer or an *oligonucleotide* of length n.

The four DNA nucleotides are adenine, guanine, cytosine and thymine, commonly abbreviated to A, G, C and T respectively. A schematic representation of the structure of each nucleotide is depicted in figure 2.1.

Each strand has, according to chemical convention, a $5'$ and a $3'$ end, thus any single strand has a natural orientation. This orientation (and, therefore, the notation used) is due to fact that one end of the single strand has a free (i.e., unattached to another nucleotide) $5'$ phosphate group, and the other has a free $3'$ deoxyribose hydroxl group. The classical double helix of DNA (figure 2.2) is formed

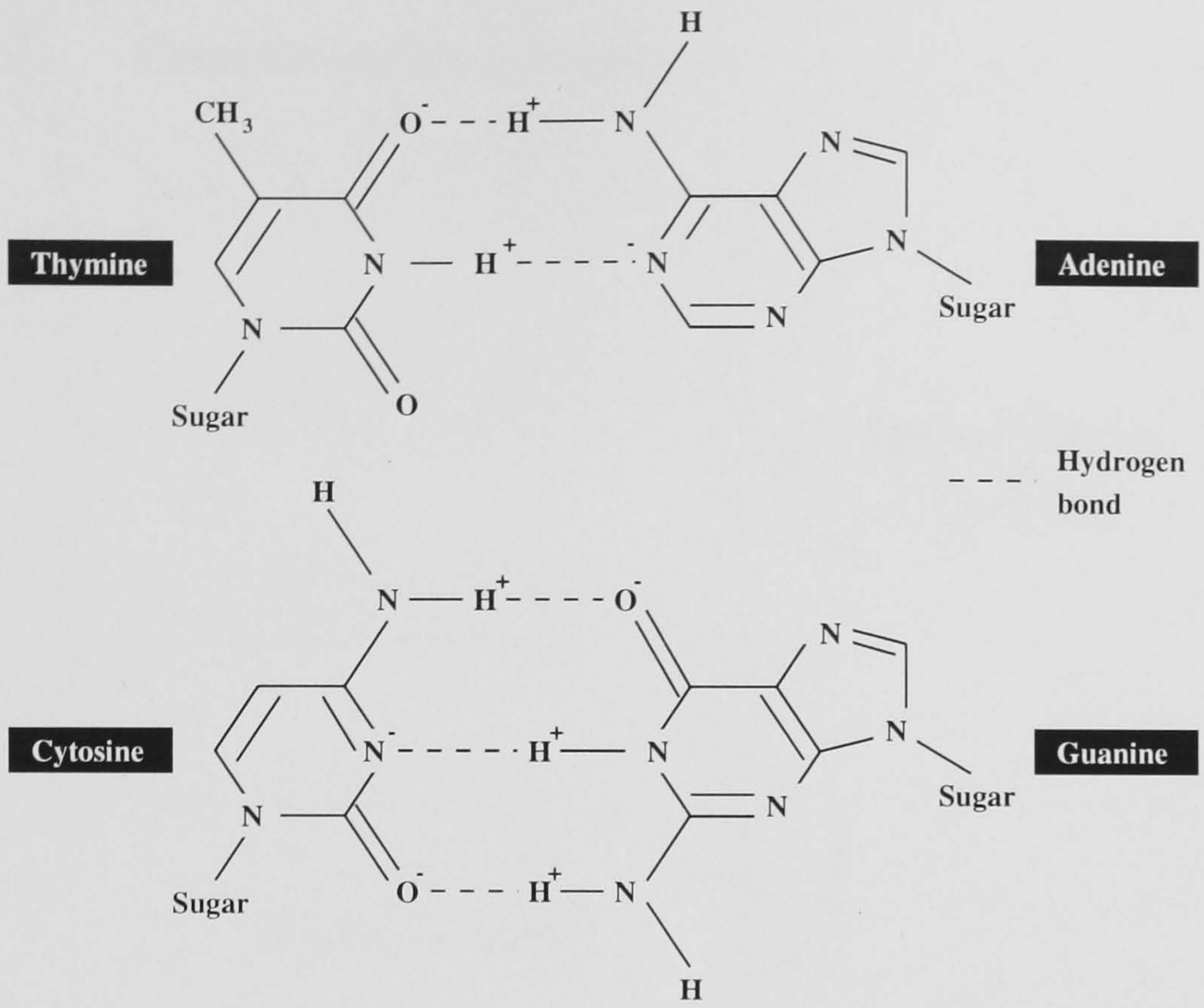

Figure 2.1: Structure of adenine, guanine, thymine and cytosine

when two separate strands bond. Bonding occurs by the pairwise attraction of bases; A bonds with T and G bonds with C. The pairs (A,T) and (G,C) are therefore known as *complementary* base pairs. The two pairs of bases form *hydrogen bonds* between each other, two bonds between A and T, and three between G and C (figure 2.3).

In what follows we adopt the following convention: if x denotes an oligonu-

Figure 2.2: Structure of double-stranded DNA

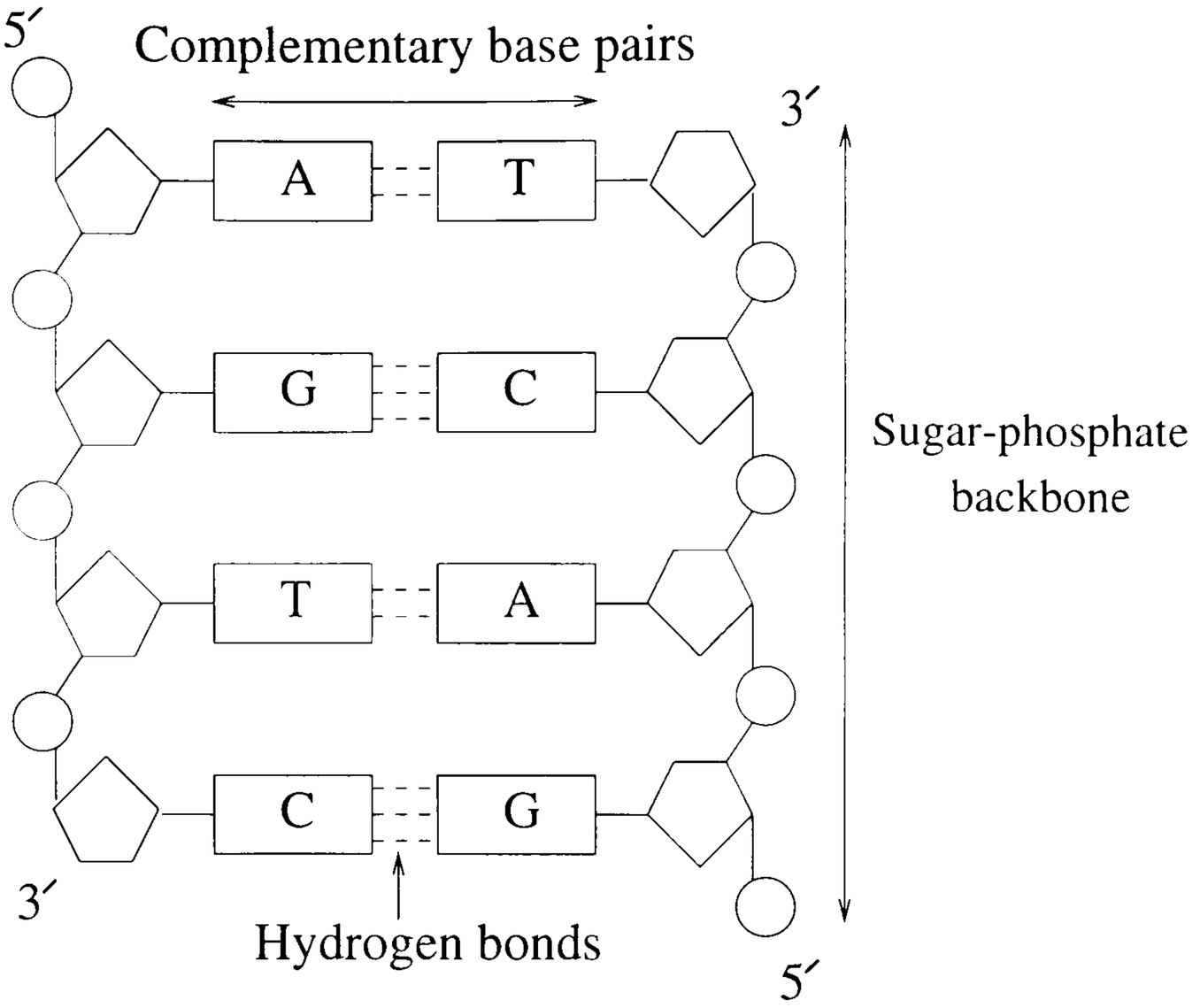

Figure 2.3: Detailed structure of double-stranded DNA

cleotide, then $\overline{x}$ denotes the complement of x. The bonding process, known as *annealing*, is fundamental to our implementation. A strand will only anneal to its complement if they have opposite polarities. Therefore, one strand of the double helix extends from $5'$ to $3'$, and the other from $3'$ to $5'$, as depicted in figure 2.2.

2.3 Operations on DNA

All models of DNA computation apply a specific sequence of biological operations to a set of strands. These operations are all commonly used by molecular biologists. Note that some operations are specific to certain models of DNA computation.

2.3.1 Synthesis

Oligonucleotides may be synthesised to order by a machine the size of a microwave oven. The synthesiser is supplied 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 and places them in solution in a small vial.

2.3.2 Denaturing, annealing and ligation

Double-stranded DNA may be dissolved into single strands (or *denatured*) by heating the solution to a temperature determined by the composition of the strand [17]. Heating breaks the hydrogen bonds between complementary strands (figure 2.4). Since a $G - C$ pair is joined by three hydrogen bonds, the temperature required to break it is slightly higher than that for an $A - T$ pair, joined by only two hydrogen bonds. This factor must be taken into account when designing sequences to represent computational elements.

Annealing is the reverse of melting, whereby a solution of single strands is cooled, allowing complementary strands to bind together (figure 2.4).

In double-stranded DNA. if one of the single strands contains a discontinuity (i.e., one nucleotide is not bonded to its neighbour) then this may be repaired by DNA *ligase* [18]. This allows us to create a unified strand from several bound together by their respective complements.

2.3.3 Hybridisation separation

Separation by hybridisation is an operation central to early models of DNA computation, and involves the extraction from a test tube of any *single* strands containing a

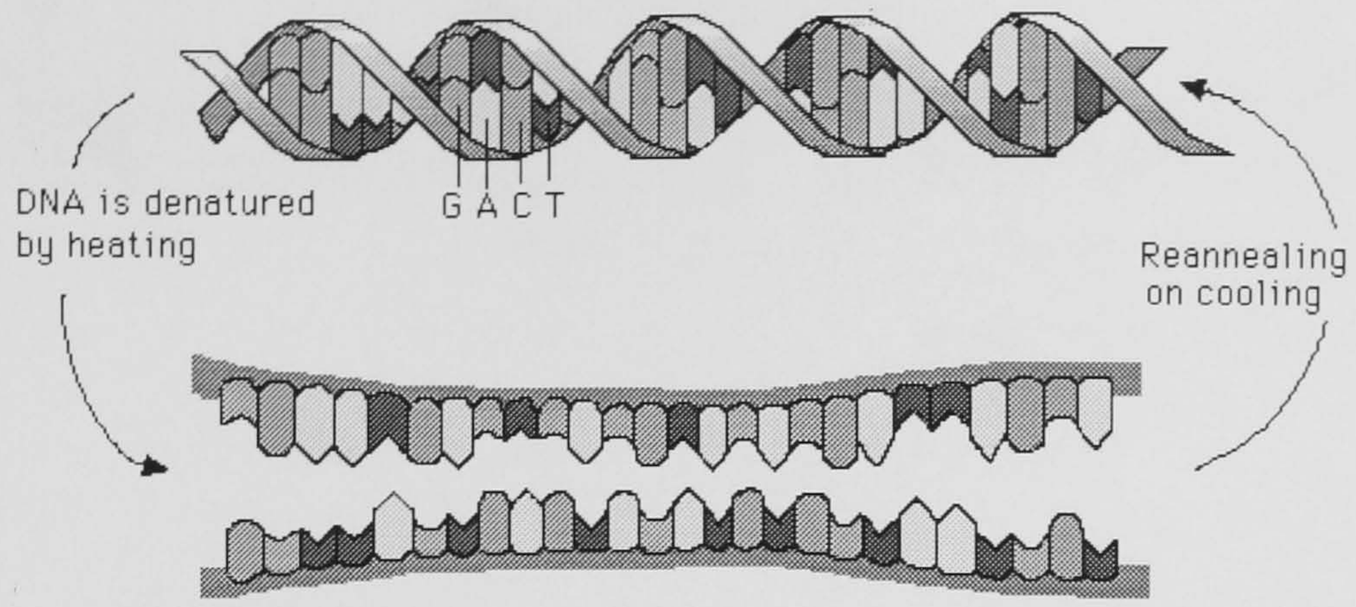

Figure 2.4: DNA melting and annealing

specific short sequence (e.g., extract all strands containing the sequence $TAGACT$). If we want to extract single strands containing the sequence x we first create many copies of its complement, $\overline{x}$. We attach to these oligonucleotides a biotin molecule[1] which bind in turn to a fixed matrix. 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. However, we describe problems with hybridisation extraction in section 4.4.

2.3.4 Gel electrophoresis

Gel electrophoresis is an important technique for sorting DNA strands by size [18]. 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

[1]This process is referred to as "biotinylation".

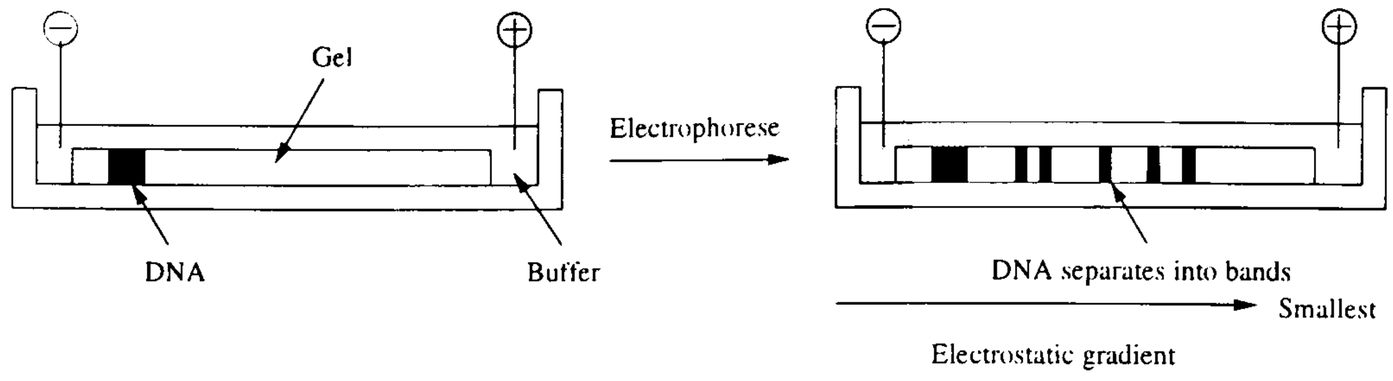

Figure 2.5: Gel electrophoresis process

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 $size$[2]. This is due to the fact that the gel is a dense network of pores through which the molecules must travel. Smaller molecules therefore migrate faster through the gel, thus sorting them according to size.

A simplified representation of gel electrophoresis is depicted in figure 2.5. The DNA is placed in a well cut out of the gel, and a charge applied.

Once the gel has been run (usually overnight), it is necessary to visualise the results. This is achieved by staining the DNA with the fluorescent dye ethidium bromide and then viewing the gel under ultraviolet light. At this stage the gel is usually photographed for convenience.

One such photograph is depicted in figure 2.6. Gels are interpreted as follows: each *lane* (1-7 in our example) corresponds to one particular sample of DNA (we use the term *tube* in our abstract model). We can therefore run several tubes on the same gel for the purposes of comparison. Lane 7 is known as the *marker lane*; this contains various DNA fragments of known length, for the purposes of calibration.

[2]Migration rate of a strand is inversely proportional to the logarithm of its molecular weight [62].

DNA fragments of the same length cluster to form visible horizontal *bands*, the longest fragments forming bands at the top of the picture, and the shortest at the bottom. The brightness of a particular band depends on the amount of DNA of the corresponding length present in the sample. Larger concentrations of DNA absorb more dye, and therefore appear brighter. One advantage of this technique is its sensitivity - as little as 0.05μg of DNA in one band can be detected as visible fluorescence.

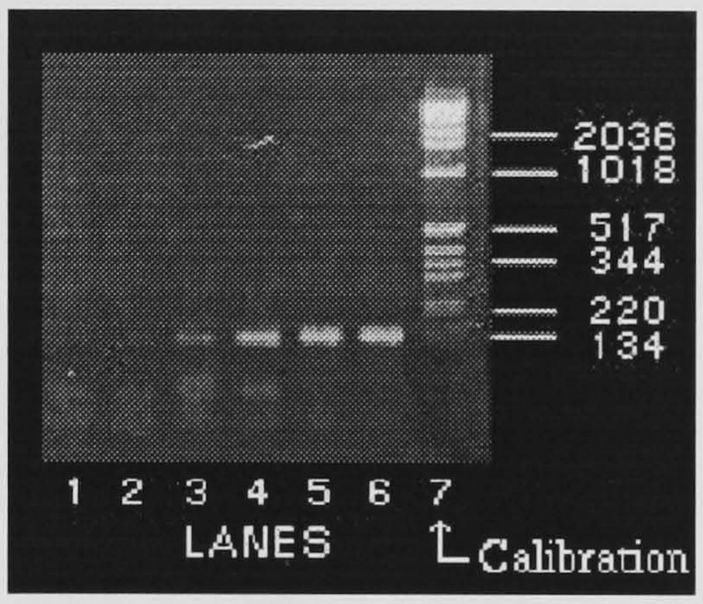

Figure 2.6: Gel electrophoresis photograph

The size of fragments at various bands is shown to the right of the marker lane, and is measured in *base pairs* (b.p.). In our example, the largest band resolvable by the gel is 2036 b.p. long, and the shortest 134 b.p. Moving right to left (tracks 6-1) is a series of PCR reactions which were set up with progressively diluted target DNA (134 b.p.) to establish the sensitivity of a reaction. The dilution of each tube is evident from the fading of the bands, which eventually disappear in lane 1.

2.3.5 Primer extension and PCR

The DNA *polymerases* perform several functions, including the repair and duplication of DNA. Given a short *primer* oligonucleotide, p in the presence of nucleotide

```
5' A T A G A G T T 3'
          | | |          (a)
3'        T C A   5'

5' A T A G A G T T 3'
   | | | | | | | |       (b)
3' T A T C T C A   5'
```

Figure 2.7: (a) Primer anneals to longer template (b) Polymerase extends primer in the $5'$ to $3'$ direction

triphosphates, the polymerase extends p if and only if p is bound to a longer *template* oligonucleotide, t. For example, in figure 2.7(a)), p is the oligonucleotide TCA which is bound to t, $ATAGAGTT$. In the presence of the polymerase, p is extended by a complementary strand of bases to the $3'$ end of t (figure 2.7(b)).

Another useful method of manipulating DNA is the *Polymerase Chain Reaction*, or PCR [59, 60]. PCR is a process that quickly amplifies the amount of DNA in a given solution. Each cycle of the reaction doubles the quantity of each strand, giving an exponential growth in the number of strands

2.3.6 Restriction enzymes

Restriction endonucleases [79, page 33] (often referred to as *restriction enzymes*) recognise a specific sequence of DNA, known as a *restriction site*. Any *double-stranded* DNA that contains the restriction site within its sequence is cut by the enzyme at that point[3] For example, the double-stranded DNA in figure 2.8(a) is cut by restriction enzyme *Sau*3AI, which recognises the restriction site $GATC$. The resulting DNA is depicted in figure 2.8(b). The resulting cleavage generates either

[3]In reality, only certain enzymes cut specifically at the restriction site, but we take this factor into account when selecting an enzyme.

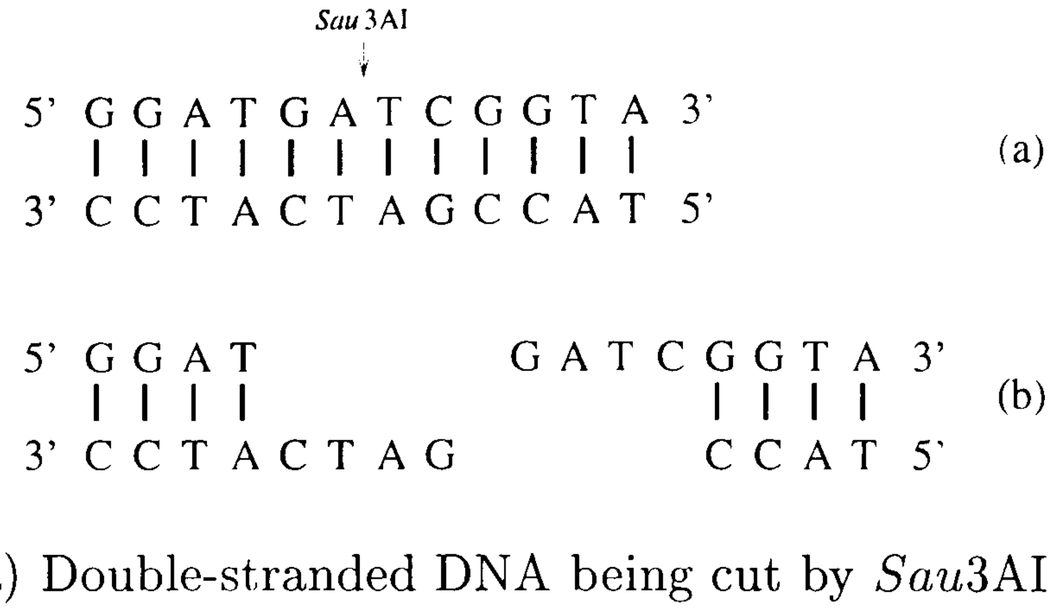

Figure 2.8: (a) Double-stranded DNA being cut by *Sau*3AI (b) The result

"blunt" or "sticky" (cohesive) ends, a feature that we will utilise later.

2.3.7 Cloning

Once the structure of the DNA molecule was elucidated and the processes of transcription and translation were understood, molecular biologists were frustrated by the lack of suitable experimental techniques that would facilitate more detailed examination of the genetic material. However, in the early 1970s, several techniques were developed that allowed previously impossible experiments to be carried out (see [19, 62]). These techniques quickly led to the first ever successful cloning experiments [45, 55].

Cloning is generally defined as "...the production of multiple identical copies of a single gene. cell, virus or organism." [67]. In the context of molecular computation, cloning therefore allows us to obtain multiple copies of specific strands of DNA. This is achieved as follows.

1. The specific sequence is inserted in a circular DNA molecule. known as a *vector*, producing a *recombinant DNA molecule.* This is performed by cleaving both the double-stranded vector DNA and the target strand with the *same* restriction enzyme(s). Since the vector is double-stranded, restriction with

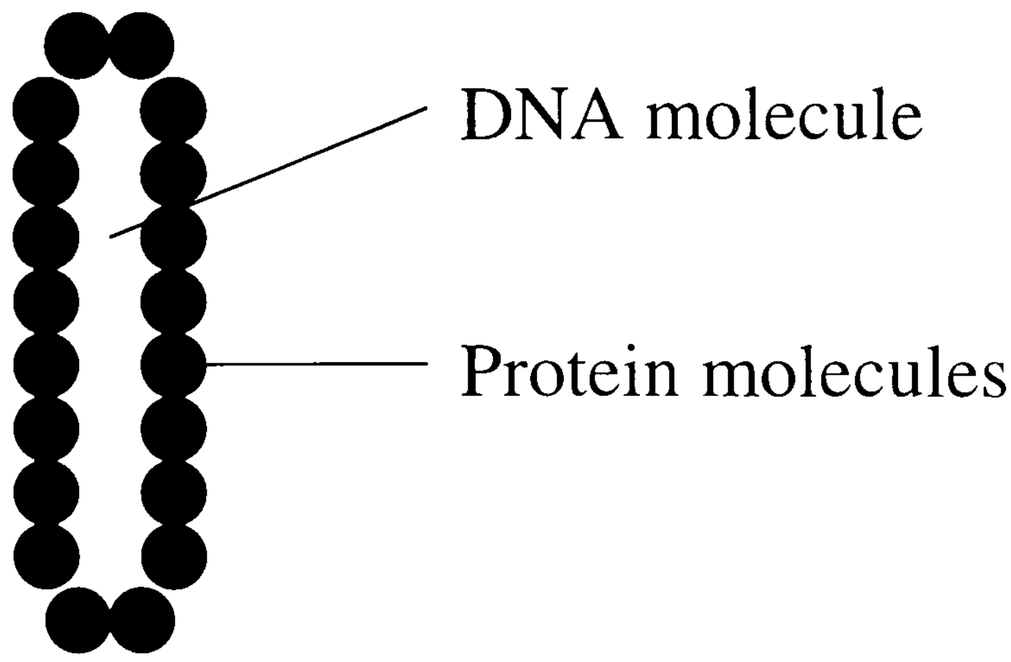

Figure 2.9: Schematic representation of the M13 phage structure

suitable enzymes produces two short single-stranded regions at either end of the molecule (referred to as *"sticky"* ends. The same also applies to the target strand. The insertion process is depicted in figure 2.10. The vector and target are both subjected to restriction, then a population of target strands is introduced to the solution containing the vector. The sticky ends of the target bind with the sticky ends of the vector, integrating the target into the vector. After ligation, new double-stranded molecules are present, each containing the new target sequence.

In what follows, we use the *M13 bacteriophage* as the cloning vector. Specifically, we use the M13mp18 vector, which is a 7,249 b.p. long derivative of M13 constructed by Messing *et al.* [81].

Bacteriophages (or *phages*, as they are commonly known) are viruses that infect bacteria. The structure of a phage is very simple, usually consisting of a single-stranded DNA molecule surrounded by a sheath of protein molecules (the *capsid*) (figure 2.9).

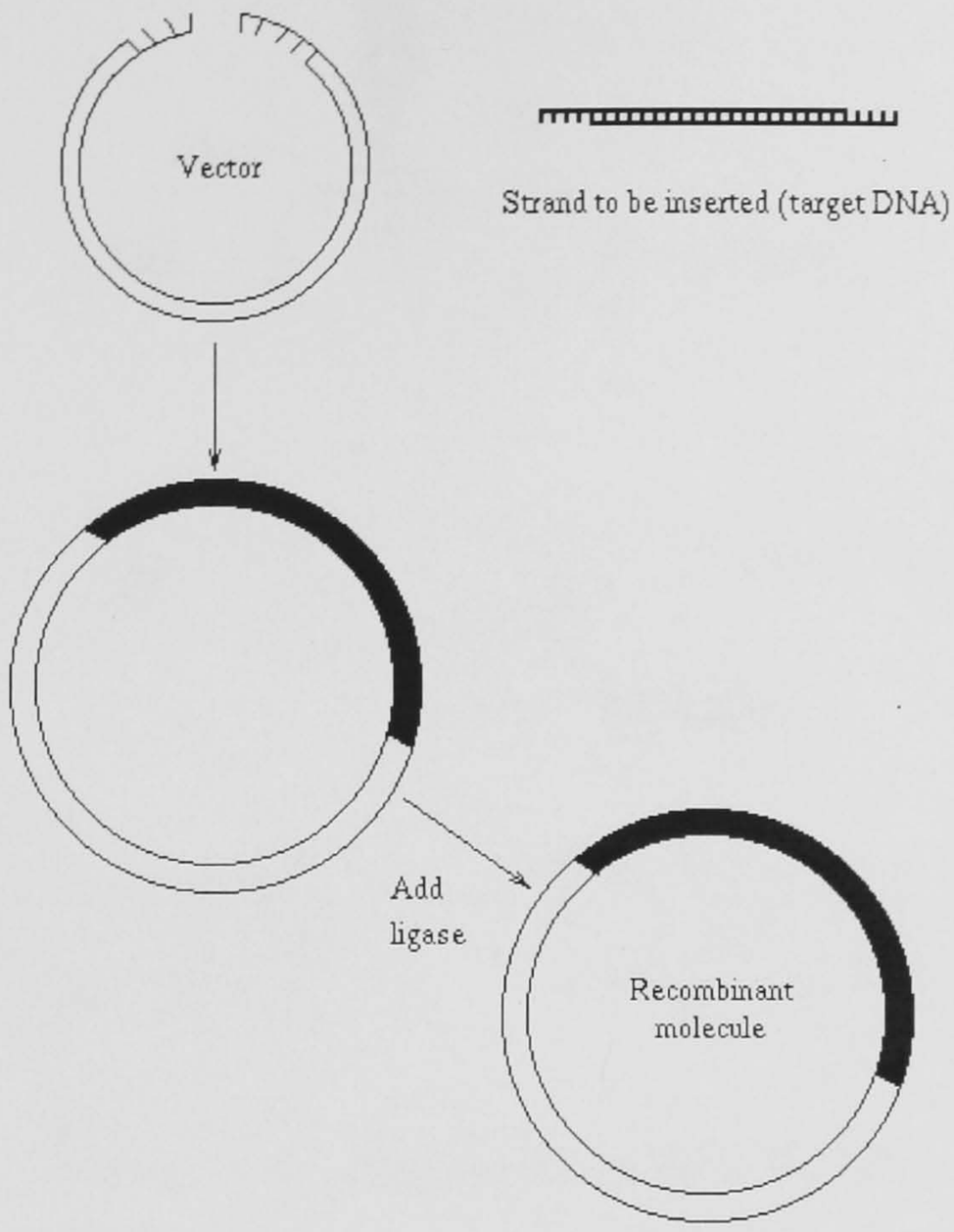

Figure 2.10: Insertion of target strand into vector DNA

2. The vector act as a *vehicle*, transporting the sequence into a *host* cell (usually a bacterium, such as *E. coli*. In order for this to occur, the bacteria must be made *competent*. Since the vectors are relatively heavy molecules, they cannot be introduced into a bacterial cell easily. However, subjecting *E. coli* to a variety of hot and cold "shocks" (in the presence of calcium, amongst other chemicals) allows the vector molecules to move through the cell membrane. The process of introducing exogenous DNA into cells is referred to as *transformation*. One problem with transformation is that it is a rather inefficient process; the best we can hope for is that around 5% of the bacterial cells will take up the vector. In order to improve this situation, we may use a technique known as

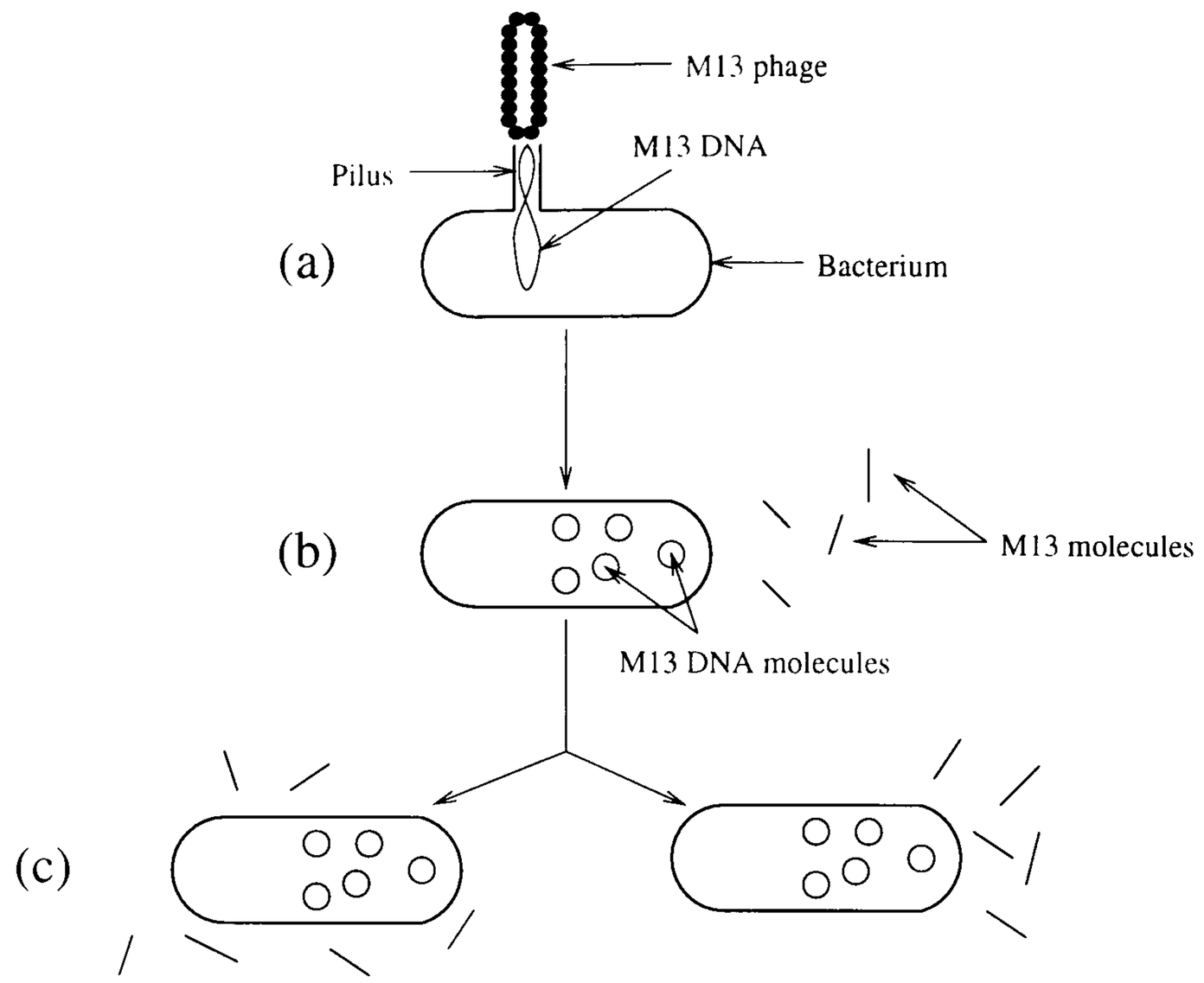

Figure 2.11: M13 phage infection cycle

electroporation. A high voltage pulse is passed through the solution containing the vectors and bacteria, causing the cell membranes to become permeable. This increases the probability of vector uptake.

3. The vector multiplies within the cell, producing numerous copies of itself (including the inserted sequence).

The infection cycle of M13 proceeds as follows. The phage attaches to a *pilus* (an appendage on the surface of the cell) and injects its DNA into the bacterium (figure 2.11(a)). The M13 DNA is not integrated into the DNA of the bacterium, but is still replicated within the cell. In addition, new phages are continually assembled within and released from the cell (figure 2.11(b)), which go on and infect other bacteria (figure 2.11(c)). When sufficient copies of the specific sequence have been

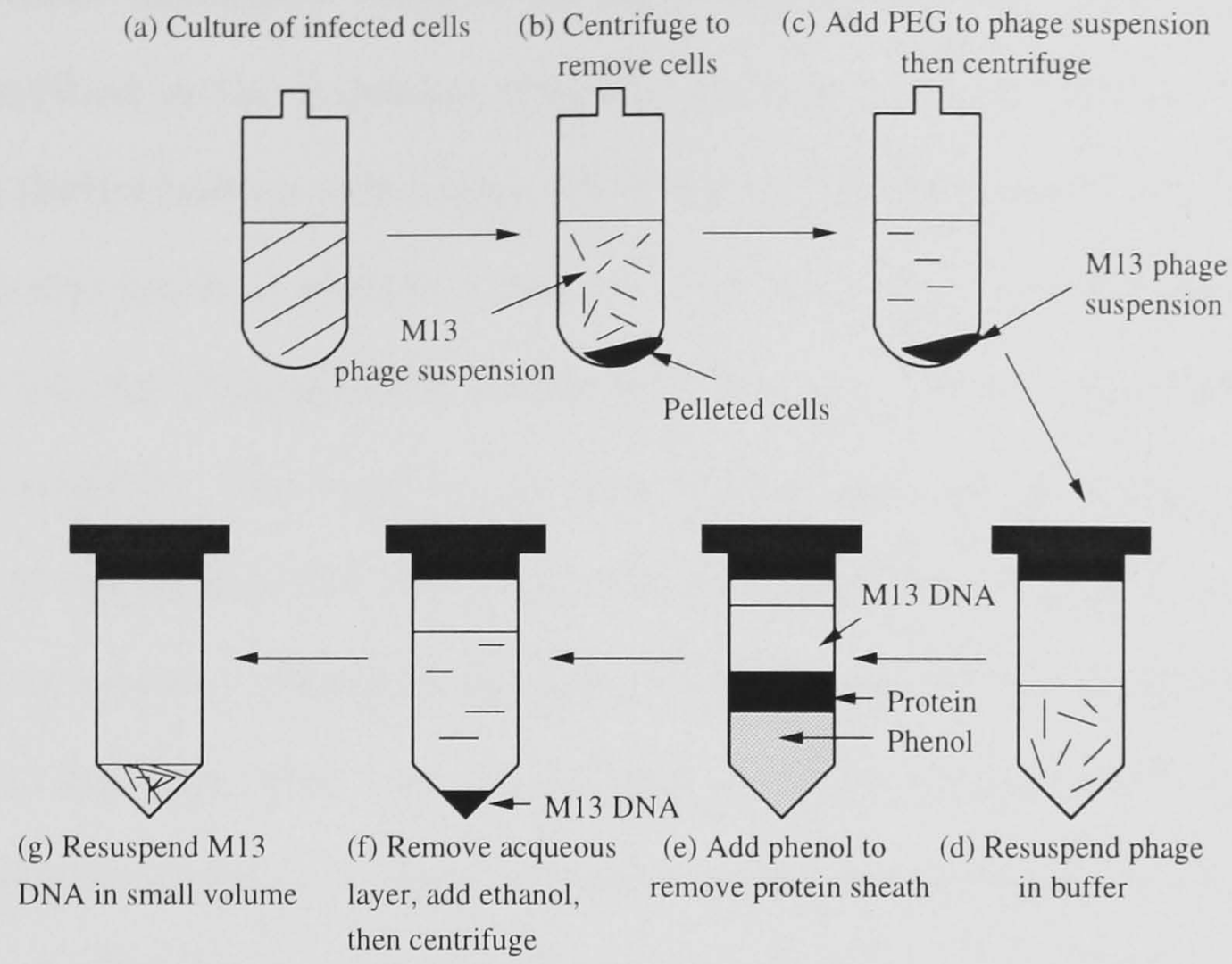

Figure 2.12: Preparation of M13 DNA from infected culture of bacteria

made, the single-stranded M13 DNA may be retrieved from the medium. The process by which this is achieved is depicted in figure 2.12 (also see [57]). Once a sufficient volume of infected culture has been obtained we centrifuge it to pellet the bacteria (i.e., separate the bacteria from the phage particles). We then precipitate the phage particles with polyethylene glycol (PEG), add phenol to strip off the protein coats and then precipitate the resulting DNA using ethanol.

2.4 Summary

We described here the basic structure of DNA and the methods by which it may be manipulated in the laboratory. These techniques owe their origin to, and are being constantly improved by the wide interests of molecular biologists working in modern areas such as the Human Genome project and genetic engineering. In chapter 4 we

show how these techniques allow us to implement the various DNA computational models described in the following chapter. Adleman used a small subset of these techniques (hybridisation extraction, PCR and gel electrophoresis) in [2]. Although other molecules (such as proteins) may be used as a computational substrate in the future, the benefit of using DNA is that this wide range of manipulation techniques is already available. The same is not true for proteins. In this chapter we extend the range of operations available, and introduce the use of primer extension and restriction to destroy unwanted strands, first detailed by us in [7]. In chapter 5 we show for the first time how the cloning procedure described in section 2.3.7 may be used to construct a final read-out (solution sequencing) procedure. This technique is sufficiently general to allow it to be incorporated into the experimental implementation of any abstract model of DNA computation. In the next chapter we describe several such models.

Chapter 3

Models of DNA computation

The major purpose of this chapter is to describe our "parallel filtering" model of computation which abstracts and generalises the type of computation exemplified by Adleman's inspirational experiment [2]. A detailed description of this model appeared in [7]. Our model is placed in the general context of other models which have appeared in the literature either at the same time as or since our model was developed. Our review of these models appears in *Current Opinion in Biotechnology* [35].

3.1 Introduction

We may describe abstract models of computation without necessarily considering their implementation. In [32], for example, for the sake of emphasising what is inherently parallelisable within problems, the authors disregard constraints of implementation. However, in what follows we are naturally constrained by what is feasible in the intended mode of implementation; in this case, what is possible in the molecular biology laboratory. We consider models that all operate upon sets

of *strings*. It is generally the case that a DNA computation starts and ends with a single set of strings. An algorithm is composed of a sequence of operations upon one or more set of strings. At the end of the algorithm's execution, a solution to the given problem is encoded as a string in the final set. We use the term *computational substrate* to describe the substance that is acted upon by the implementation of a model. In chapter 3 we explain the structure of the DNA molecule in detail. Since DNA is the underlying computational substrate of all models described, as we shall see, we may naturally assume that all abstract models operate on strings over a four-letter alphabet, $\{A, G, C, T\}$. Of course, 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 utilise 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 impact in various ways:

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 read-out the final solution may be very time-consuming and must also be taken into account.

3. Each laboratory technique has associated with it a non-zero 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.

In this chapter we describe various abstract models of DNA computation. We introduce a basic taxonomy of these models, and describe an archetypal example from each category. We begin by describing early work of Adleman and Lipton, which provides a foundation for later models. We then describe our generalisation of their models, showing how DNA-based algorithms may be applied to a variety of NP-complete problems. We then describe the work of Reif and others, who show that the addition of an extra *splicing* operation to the basic sets defined by Adleman and others establishes the Turing-completeness of models of DNA computation. We then describe a third model which emulates Boolean Circuits, due to Ogihara and Ray [61]. We conclude the chapter with a description of the rather more speculative self-assembly model.

3.2 Models of DNA computation

The models are characterised by the nature of the operations within them, and fall into three natural categories:

- Filtering
- Splicing
- Constructive

The major models are listed in table 3.1 according to the category in which they lie.

Filtering	*Splicing*	*Constructive*
Adleman [2]	Reif [66]	Ogihara/Ray [61]
Lipton [52]	Freund/Kari/Păun [21]	Baum/Boneh [11]
Amos/Gibbons/Hodgson [7]		Guarnieri/Fliss/Bancroft [39]
Karp/Kenyon/Waarts [48]		Winfree/Yang/Seeman [80]
Liu/Guo *et al.* [53]		
Roweis/Winfree *et al.* [69]		

Table 3.1: A taxonomy of models of DNA computation

3.3 Filtering models

In all filtering models (motivated by Adleman [2] and contemporaneously generalised by Lipton [52] and Amos *et al.* [7]), a computation consists of a sequence of operations on finite multi-sets of strings. It is normally the case that a computation begins and terminates with a single multi-set. Within the computation, by applying legal operations of a model, several multi-sets may exist at the same time. We define operations on multi-sets shortly but first consider the nature of an *initial set*.

An initial multi-set consists of strings which are typically of length $O(n)$ where n is the problem size. As a subset, the initial multi-set should include all possible solutions (each encoded by a string) to the problem to be solved. The point here is that the superset is supposed, in any implementation of the model, to be relatively easy to generate as a starting point for a computation. The computation then proceeds by filtering out strings which cannot be a solution.

For example, if the problem is to generate a permutation of the integers $1 \ldots n$ then the initial multi-set might include all strings of the form $p_1 i_1 p_2 i_2 \ldots p_n i_n$ where each i_k may be any of the integers in the range $[1 \ldots n]$ and p_k encodes the information "position k". Here, as will be typical for many computations, the multi-set has cardinality which is exponential in the problem size. For our example of finding a permutation, we should filter out all strings in which the same integer

appears in at least two locations p_k. Any of the remaining strings is then a legal solution to the problem.

We now describe the important features of the various filtering models.

3.3.1 Adleman

Adleman [2] provided the impetus for recent work through his experimental solution to the Hamiltonian Path Problem. This solution, however, was not expressed within a formal model of computation, and is therefore described later in section 4.3. In [52], Lipton considered Adleman's specific model and showed how it can encompass solutions to one other NP-complete problem. Here we summarise the operations within Adleman's subsequent *unrestricted* model [3]. All operations are performed on sets of strings over some alphabet α.

- $separate(T, S)$. Given a set T and a substring S, create two new sets $+(T, S)$ and $-(T, S)$, where $+(T, S)$ is all strings in T containing S, and $-(T, S)$ is all strings in T *not* containing S.

- $merge(T_1, T_2, \ldots, T_n)$. Given set $T_1, T_2, \ldots, T_n$, create $\cup(T_1, T_2, \ldots, T_n) = T_1 \cup T_2 \cup \ldots T_n$.

- $detect(T)$. Given a set T, return *true* if T is non-empty, otherwise return $false$.

For example, given $\alpha = \{A, B, C\}$, the following algorithm only returns *true* if the initial multi-set contains a string composed entirely of 'A's:

Input(T)

$T \leftarrow -(T, B)$

$T \leftarrow -(T, C)$

Output(detect(T))

In [3] Adleman describes an algorithm for the *3-vertex-colourability* problem. In order to obtain a proper colouring of a graph $G = (V, E)$ colours are assigned to the vertices in such a way that no two adjacent vertices are similarly coloured. The problem of whether 3 colours are sufficient to achieve such a colouring for an arbitrary graph is NP-complete [33]. We now describe Adleman's algorithm in detail.

The initial set, T, consists of strings of the form $c_1, c_2, \ldots, c_n$, where $c_i \in \{r_i, g_i, b_i\}$ and $n = |V|$. Thus each string represents one possible (not necessarily proper) colouring of the given graph. We assume that all possible colourings are represented in T. The algorithm proceeds as follows:

(1) Input(T)

(2) **for** $i = 1$ to n **do begin**

(3) $\quad T_r \leftarrow +(T, r_i)$ **and** $T_{bg} \leftarrow -(T, r_i)$

(4) $\quad T_b \leftarrow +(T_{bg}, b_i)$ **and** $T_g \leftarrow -(T_{bg}, b_i)$

(5) $\quad$ **for all** j such that $< i, j > \in E$ **do begin**

(6) $\quad\quad T_r \leftarrow -(T_r, r_j)$

(7) $\quad\quad T_g \leftarrow -(T_g, g_j)$

(8) $\quad\quad T_b \leftarrow -(T_b, b_j)$

(9) $\quad$ **end for**

(10) $T \leftarrow merge(T_r, T_g, T_b)$

(11) **end for**

(12) Output(detect(T))

At Step 1 we input all possible colourings of the graph. Then, for each vertex $v_i \in V$ we perform the following steps: split T into three sets, T_r, T_g, T_b, where T_r contains only strings containing r_i, T_g contains only strings containing g_i and T_b contains only strings containing b_i (Steps 3-4). Then, for each edge $< i, j > \in E$, we remove from these sets any strings containing $c_i = c_j$ (i.e., those strings encoding colourings where adjacent vertices i and j are coloured the same) (Steps 5-9). Then, these sets are merged, forming the new set T (Step 10), and the algorithm proceeds to the next vertex (Step 11). After the colouring constraints for each vertex have been satisfied, we perform a detection (Step 12). If T is non-empty then any string in T encodes a proper 3-vertex-colouring of G.

3.3.2 Lipton

Lipton [52] described a solution to another NP-complete problem, namely the so-called $satisfiability$ problem (SAT). SAT may be phrased as follows; given a finite set $V = \{v_1, v_2, \ldots, v_n\}$ of logical variables, we define a *literal* to be a variable, v_i, or its complement, $\overline{v_i}$. If v_i is *true* then $\overline{v_i}$ is $false$, and vice-versa. We define a *clause*, C_j, to be a set of literals $\{v_1^j, v_2^j \ldots, v_l^j\}$. An instance, I, of SAT consists of a set of clauses. The problem is to assign a Boolean value to each variable in V such that at least one variable in each clause has the value *true*. If this is the case we may say that I has been $satisfied$.

Although Lipton does not explicitly define his operation set in [52], his solu-

tion may be phrased in terms of the operations described by Adleman in [3]. Lipton employs the *merge*, *separate* and *detect* operations described above. The initial set T contains many strings, each encoding a single n-bit sequence. All possible n-bit sequences are represented in T. The algorithm proceeds as follows:

(1) Create initial set, T

(2) For each clause do begin

(3) For each literal v_i do begin

(4) if v_i=x_j extract from T strings encoding v_i=1 else

extract from T strings encoding v_i=0

(5) End for

(6) Create new set T by merging extracted strings

(7) End for

(8) If T non-empty then I is satisfiable

The pseudo-code algorithm may be expressed more formally thus:

(1) Input(T)

(2) **for** $a = 1$ to $|I|$ **do begin**

(3) **for** $b = 1$ to $|C_a|$ **do begin**

(4) **if** v_b^a=x_j **then** $T_b \leftarrow +(T, v_b^a = 1)$

else $T_b \leftarrow +(T, v_b^a = 0)$

(5) **end for**

(6) $T \leftarrow merge(T_1, T_2, \ldots, T_b)$

(7) **end for**

(8) Output(detect(T))

Step 1 generates all possible n-bit strings. Then, for each clause $C_a = \{v_1^a, v_2^a, \ldots, v_l^a\}$ (Step 2) we perform the following steps. For each literal v_b^a (Step 3) we operate as follows: If v_b^a computes the positive form then we extract from T all strings encoding 1 at position v_b^a, placing these strings in T_b. If v_b^a computes the negative form we extract from T all strings encoding 0 at position v_b^a, placing these strings in T_b (Step 4). After l iterations, we have satisfied every variable in clause C_a. We then create a new set T from the union of sets $T_1, T_2, \ldots, T_b$ (Step 6) and repeat these steps for clause $C_a + 1$ (Step 7).

If any strings remain in T after all clauses have been operated upon then I is satisfiable (Step 8).

3.3.3 Amos, Gibbons and Hodgson

A detailed description of our parallel filtering model appears in [7]. This model was the first to provide a formal framework for the description of DNA algorithms for any problem in the complexity class NP. Lipton claims some generalisation of Adleman's style of computation in [52], but it is difficult to see how algorithms for different problems may be *elegantly* and *universally* expressed within his model. Lipton effectively uses the same operations as Adleman, but does not explicitly describe the operation set. In addition, he describes only one algorithm (3SAT), whereas in subsequent sections we show how our model provides a natural description for any NP-complete problem through many examples.

As stated in section 3.3. within our model all computations start with the construction of the initial set of strings. Here we define the basic legal operations on sets within the model. Our choice is determined by what we know can be

effectively implemented by very precise and complete chemical reactions within the DNA implementation. The operation set defined here provides the power we claim for the model but, of course, it might be augmented by additional operations in the future to allow greater conciseness of computation. The main difference between the parallel filtering model and those previously proposed lies in the implementation of the removal of strings. All other models propose *separation* steps, where strings are conserved, and may be utilised later in the computation. Within the parallel filtering model, however, strings that are removed are discarded, and play no further part in the computation.

- $remove(U, \{S_i\})$. This operation removes from the set U. in parallel, any string which contains at least one occurrence of any of the substrings S_i.

- $union(\{U_i\}, U)$. This operation, in parallel, creates the set U which is the set union of the sets U_i.

- $copy(U, \{U_i\})$. In parallel, this operation produces a number of copies, U_i, of the set U.

- $select(U)$. This operation selects an element of U at random, if U is the empty set then *empty* is returned.

From the point of view of establishing the parallel time-complexities of algorithms within the model, these basic set operations will be assumed to take *constant-time*. However, this assumption is re-evaluated in chapter 6.

3.3.4 A first algorithm

We now provide our first algorithmic description within the model. The problem solved is that of generating the set of all permutations of the integers 1 to n. The

initial set and the filtering out of strings which are not permutations were essentially described earlier. Although not NP-complete, the problem does of course have exponential-sized input and output.

The algorithmic description below introduces a format that we utilise elsewhere. The particular device of copying a set (as in copy$(U, \{U_1, U_2, \ldots, U_n\})$) followed by parallel *remove* operations (as in remove$(U_i, \{p_j \neq i, p_k i\})$) is a very useful compound operation as we shall see in several later algorithmic descriptions. Indeed, it is precisely this use of *Parallel Filtering* that is at the core of most algorithms with in the model.

- **Problem: Permutations**

  Generate the set P_n of all permutations of the integers $\{1, 2, \ldots, n\}$.

- **Solution**

- *Input*: The input set U consists of all strings of the form $p_1 i_1 p_2 i_2 \ldots p_n i_n$ where, for all j, p_j uniquely encodes "position j" and each i_j is in $\{1, 2, \ldots, n\}$. Thus each string consists of n integers with (possibly) many occurrences of the same integer.

- *Algorithm*

  for $j = 1$ to $n - 1$ **do**

  begin

  copy$(U, \{U_1, U_2, \ldots, U_n\})$

  for i=1, 2, $\ldots$, n and all $k > j$

  in parallel do remove$(U_i, \{p_j \neq i, p_k i\})$

  union$(\{U_1, U_2, \ldots, U_n\}, U)$

end

$P_n \leftarrow U$

- *Complexity*: $O(n)$ parallel-time.

After the jth iteration of the **for** loop, the computation ensures that in the surviving strings the integer i_j is not duplicated at positions $k > j$ in the string. The integer i_j may be any in the set $\{1, 2, \ldots, n\}$ (which one it is depends in which of the sets U_i the containing string survived). At the end of the computation each of the surviving strings contains exactly one occurence of each integer in the set $\{1, 2, \ldots, n\}$ and so represents one of the possible permutations. Given the specified input, it is easy to see that P_n will be the set of all permutations of the first n natural numbers. As we shall see, production of the set P_n can be a useful sub-procedure for other computations.

3.3.5 Algorithms for a selection of NP-complete problems.

We now describe a number of algorithms for graph-theoretic NP-complete problems (see [33], for example). Problems in the complexity class NP seem to have a natural expression and ease of solution within the model. We describe linear-time solutions although, of course, there is frequently an implication of an exponential number of processors available to execute any of the basic operations in unit time.

The 3-vertex-colourability problem.

- **Problem: Three colouring**

  Given a graph $G = (V, E)$, find a 3-vertex-colouring if one exists, otherwise return the value *empty*.

- **Solution**

  - *Input*: The input set U consists of all strings of the form $p_1c_1p_2c_2 \ldots p_nc_n$ where $n = |V|$ is the number of vertices in the graph. Here, for all i, p_i uniquely encodes "position i" and each c_i is any one of the "colours" 1, 2 or 3. Each such string represents one possible assignment of colours to the vertices of the graph in which, for each i, colour c_i is assigned to vertex i.

  - *Algorithm*

    for $j = 1$ to n **do**

    begin

    copy$(U, \{U_1, U_2, U_3\})$

    for i=1, 2 and 3, and all k such that $(j, k) \in E$

    in parallel do remove$(U_i, \{p_j \neq i, p_k i\})$

    union$(\{U_1, U_2, U_3\}, U)$

    end

    select(U)

  - *Complexity*: $O(n)$ parallel time.

After the jth iteration of the **for** loop, the computation ensures that in the remaining strings vertex j (although it may be coloured 1, 2 or 3 depending on which of the sets U_i it survived in) has no adjacent vertices that are similarly coloured. Thus, when the algorithm terminates, U only encodes legal colourings if any exist. Indeed, every legal colouring will be represented in U.

The Hamiltonian Path problem.

A Hamiltonian path between any two vertices u, v of a graph is a path that passes through every vertex in $V - \{u, v\}$ precisely once [33].

- **Problem: Hamiltonian path**

  Given a graph $G = (V, E)$ with n vertices, determine whether G contains a Hamiltonian path.

- **Solution**

  - *Input*: The input set U is the set P_n of all permutations of the integers from 1 to n as output from **Problem: Permutations**. An integer i at position p_k in such a permutation is interpreted as follows: the string represents a candidate solution to the problem in which vertex i is visited at step k.

  - *Algorithm*

    for $2 \leq i \leq n-1$ and j, k such that $(j, k) \notin E$

    in parallel do remove $(U, \{jp_ik\})$

select(U)

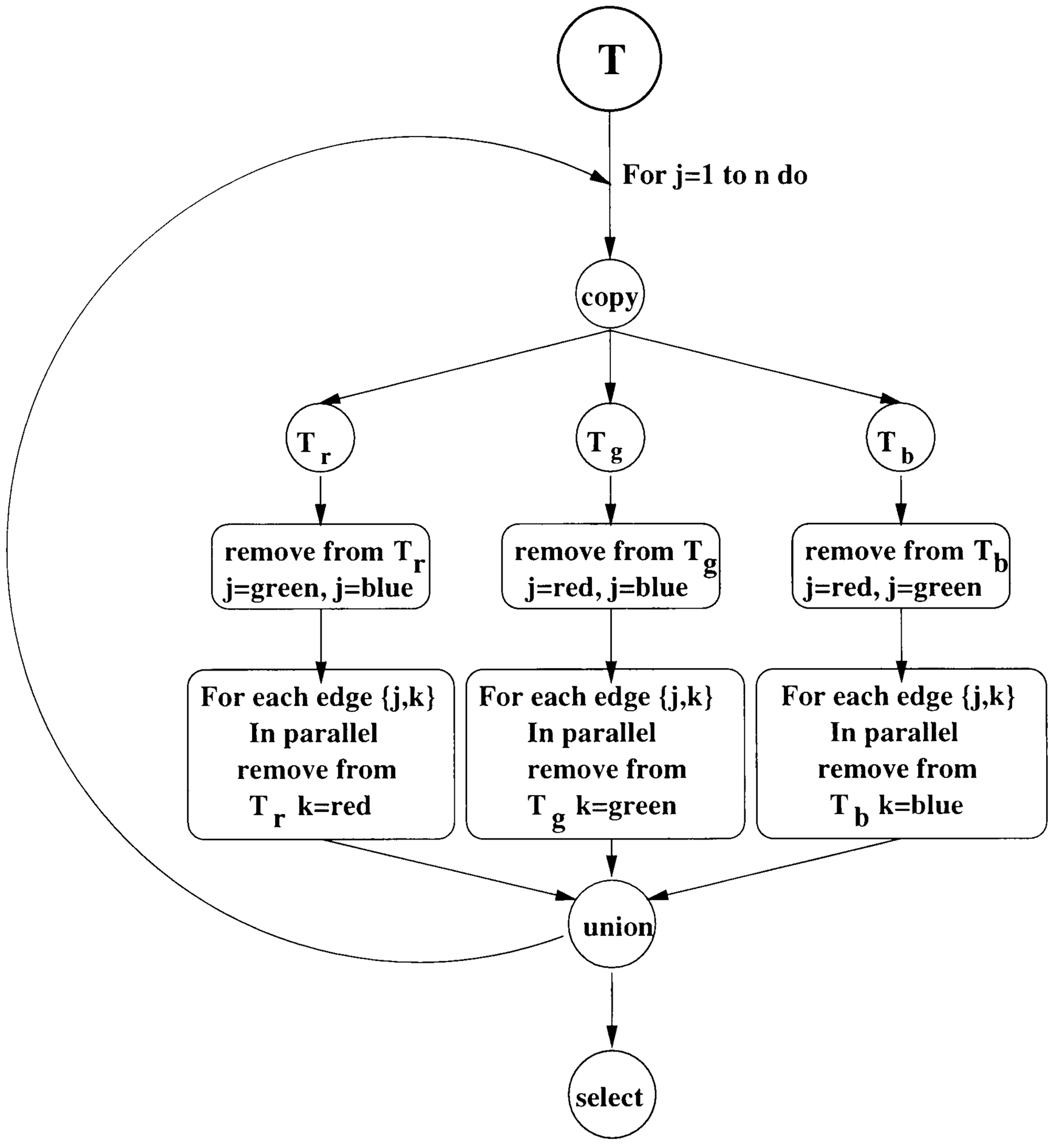

Figure 3.1: 3-colouring algorithm flowchart

- *Complexity*: Constant parallel time given P_n.

In surviving strings there is an edge of the graph for each consecutive pair of vertices in the string. Since the string is also a permutation of the vertex set it must also be a Hamiltonian path. Of course, U will contain every legal solution to the problem.

The Subgraph isomorphism problem.

Given two graphs G_1 and G_2 the following algorithm determines whether G_2 is a subgraph of G_1.

- **Problem: Subgraph isomorphism**

  Is $G_2 = (V_2, E_2)$ a subgraph of $G_1 = (V_1, E_1)$? By $\{v_1, v_2, \ldots, v_s\}$ we denote the vertex set of G_1, similarly the vertex set of G_2 is $\{u_1, u_2, \ldots, u_t\}$ where, without loss of generality, we take $t \leq s$.

- **Solution**

  - *Input*: The input set U is the set P_s of permutations output from the Permutations algorithm. For $1 \leq j \leq t$ an element $p_1 i_1 p_2 i_2 \ldots p_s i_s$ of P_s is interpreted as associating vertex $p_j \in \{u_1, u_2, \ldots, u_t\}$ with vertex $i_j \in \{v_1, v_2, \ldots, v_s\}$. The algorithm is designed to remove any element which maps vertices in V_1 to vertices in V_2 in a way which does not reflect the requirement that if $(p_s, p_t) \in E_1$ then $(i_s. i_t) \in E_2$.

- *Algorithm*

  for j=1 to $t-1$ **do**

  begin

  copy$(U, \{U_1, U_2, \ldots, U_t\})$

  for all $l, j < l \leq t$ such that $(p_j, p_l) \in E_2$ and $(ij, i_l) \notin E_1$

  in parallel do remove$(U_j, \{p_l i_l\})$

  union$(\{U_1, U_2, \ldots, U_t\}, U)$

  end

  select(U)

- *Complexity*: O($|V_s|$) parallel time.

for any remaining strings, the first t pairs $p_l i_l$ represent a one-to-one association of the vertices of G_1 with the vertices of G_2 indicating the subgraph of G_1 which is isomorphic to G_2. If select(U) returns the value *empty* then G_2 is not a subgraph of G_1.

The Maximum clique and maximum independent set problems.

A clique K_i is the complete graph on i vertices [33]. The problem of finding a maximum independent set is closely related to the maximum clique problem.

- **Problem: Maximum clique**

  Given a graph $G = (V, E)$ determine the largest i such that K_i is a subgraph of G. Here K_i is the complete graph on i vertices.

- **Solution**

  - In parallel run the subgraph isomorphism algorithm for pairs of graphs

(G, K_i) for $2 \leq i \leq n$. The largest value of i for which a non- *empty* result is obtained solves the problem.

- *Complexity*: $O(|V|)$ parallel time.

A maximum independent set is a subset of vertices of a graph such that no two members of the set are adjacent [33].

- **Problem: Maximum independent set**

Given a graph $G = (V, E)$ determine the largest i such that there is a set of i vertices in which no pair are adjacent.

- **Solution**

Run the maximum clique algorithm on the *complement* of G.

- *Complexity*: $O(|V|)$ parallel time.

The above examples fully illustrate the way in which the NP-complete problems have a natural mode of expression within the model. The mode of solution fully emulates the definition of membership of NP: that instances of problems have candidate solutions that are polynomial time verifiable and that there are generally an exponential number of candidates.

3.3.6 Sticker model

In [69] the authors introduce the so-called *sticker model*. Unlike previous models, the sticker model has a memory that can be both read and written to, and employs reusable DNA. At present, this model is of theoretical interest, since considerable practical difficulties exist in its implementation. Each string is composed of k bits, each encoded by a substring of some defined length. In order to set bit i to 1, we

anneal to each string the complement of the sequence representing bit i. However, if we wish to clear a bit (i.e., set it to zero) 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.

3.4 Splicing models

Since any instance of any problem in the complexity class NP may be expressed in terms of an instance of any NP-complete problem, it follows that the multi-set operations described earlier at least implicitly provide sufficient computational power to solve any problem in NP. We do not believe they provide the full algorithmic computational power of a Turing Machine. Without the availability of string editing operations, it is difficult to see how the transition from one state of the Turing Machine to another may be achieved using DNA. However, as several authors have recently described, one further operation, the so-called *splicing* operation, will provide full Turing computability. Here we provide an overview of the various splicing models proposed.

Let S and T be two strings over the alphabet α. Then the *splice* operation consists of cutting S and T at specific positions and concatenating the resulting prefix of S with the suffix of T and concatenating the prefix of T with the suffix of S (figure 3.2). This operation is similar to the *crossover operation* employed by genetic algorithms [37, 50].

Splicing systems date back to 1987, with the publication of Tom Head's seminal paper [43] (see also [44]). In [21], the authors show that the generative power of *finite extended splicing systems* is equal to that of Turing Machines. See [22] for a review of splicing systems.

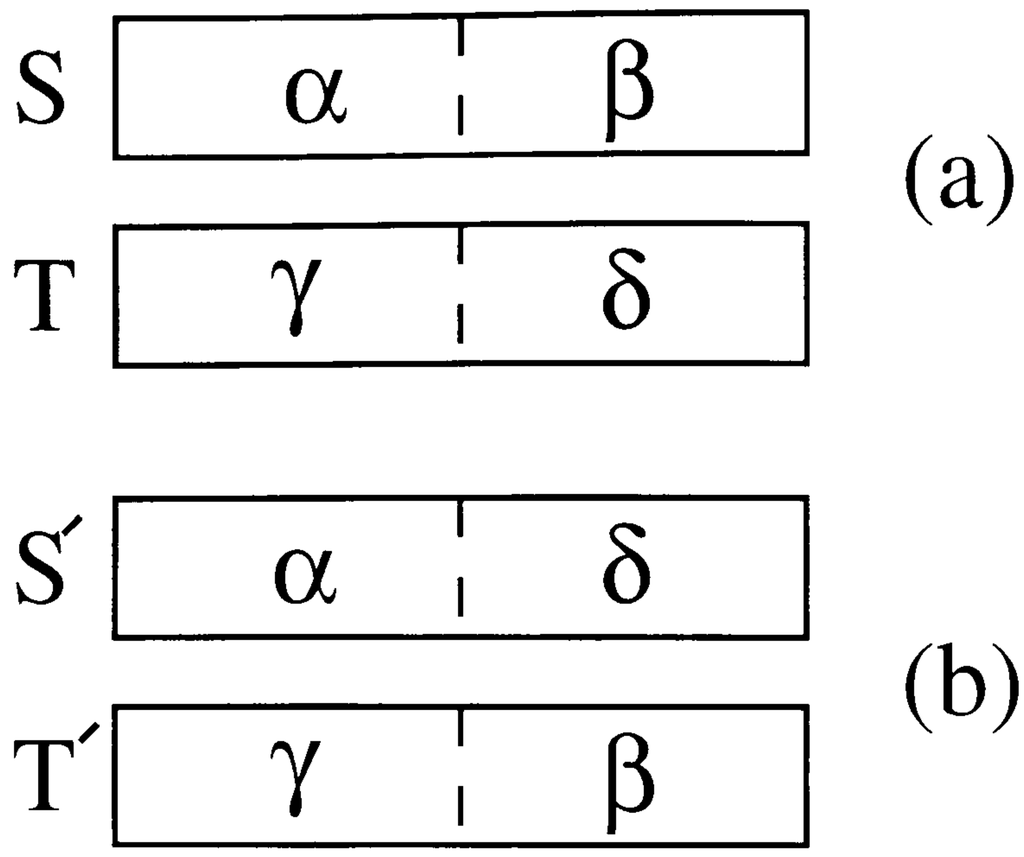

Figure 3.2: (a) Two strings, S and T (b) The result of $splice(S, T)$

3.4.1 Reif's PAM model

In [66], Reif within his so-called *Parallel Associative Memory Model* describes a *Parallel Associative Matching* (PA-Match) operation. The essential constituent of the PA-Match operation is a restricted form of the splicing operation which we denote here by *Rsplice*, and describe as follows. If $S = S_1S_2$ and $T = T_1T_2$, then the result of $Rsplice(S, T)$ is the string S_1T_2 *provided* $S_2 = T_1$, but has no value if $S_2 \neq T_1$.

Leading results of Reif [66], made possible by his PA-Match operation, concern the simulation of nondeterministic Turing Machines and the simulation of Parallel Random Access Machines (specifically CREW PRAMs). We can capture the spirit of his Turing Machine simulation through the *Rsplice* operation as follows. The initial tube in the simulation consists of all strings of the form S_iS_j where S_i and S_j are encodings of *configurations* of the simulated nondeterministic Turing Machines and such that S_j follows from S_i after one (of possibly many) machine

cycle. By a configuration here we mean an instantaneous description of the Turing Machine capturing the contents of the tape, the machine state and which tape square is being scanned. If the *Rsplice* operation is now performed between all pairs of initial strings, the tube will contain strings $S_k S_l$ where S_l follows from S_k after two machine cycles. Similarly, after t repetitions of this operation the tube will contain strings $S_m S_n$ where S_n follows from S_m after 2^t machine cycles. Clearly, if the simulated Turing Machine runs in time T, then after $O(\log T)$ operations the simulation will produce a tube containing strings $S_o S_f$ where S_o encodes the initial configuration and S_f a final configuration. Other papers have also addressed the problem of Turing Machine simulation (see [14, 68, 13], for example. and [22] for a review of some of these models).

3.5 Constructive models

An important area of enquiry that is outstanding is the quest for algorithms which proceed through polynomial-sized sets of strings (i.e., volumes of DNA). It is clear that the naive, filtering approach will not sustain such algorithms, since they rely upon the existence of an exponential-sized initial set of strings. If such algorithms are to be realized then they must be phrased within a *constructive* model of DNA computation. Such algorithms will gradually construct solutions to the given problem rather than isolating them from a large initial multi-set.

3.5.1 Ogihara and Ray's Boolean circuit model

Boolean circuits are an important Turing-equivalent model of parallel computation (see [25, 41]). 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-

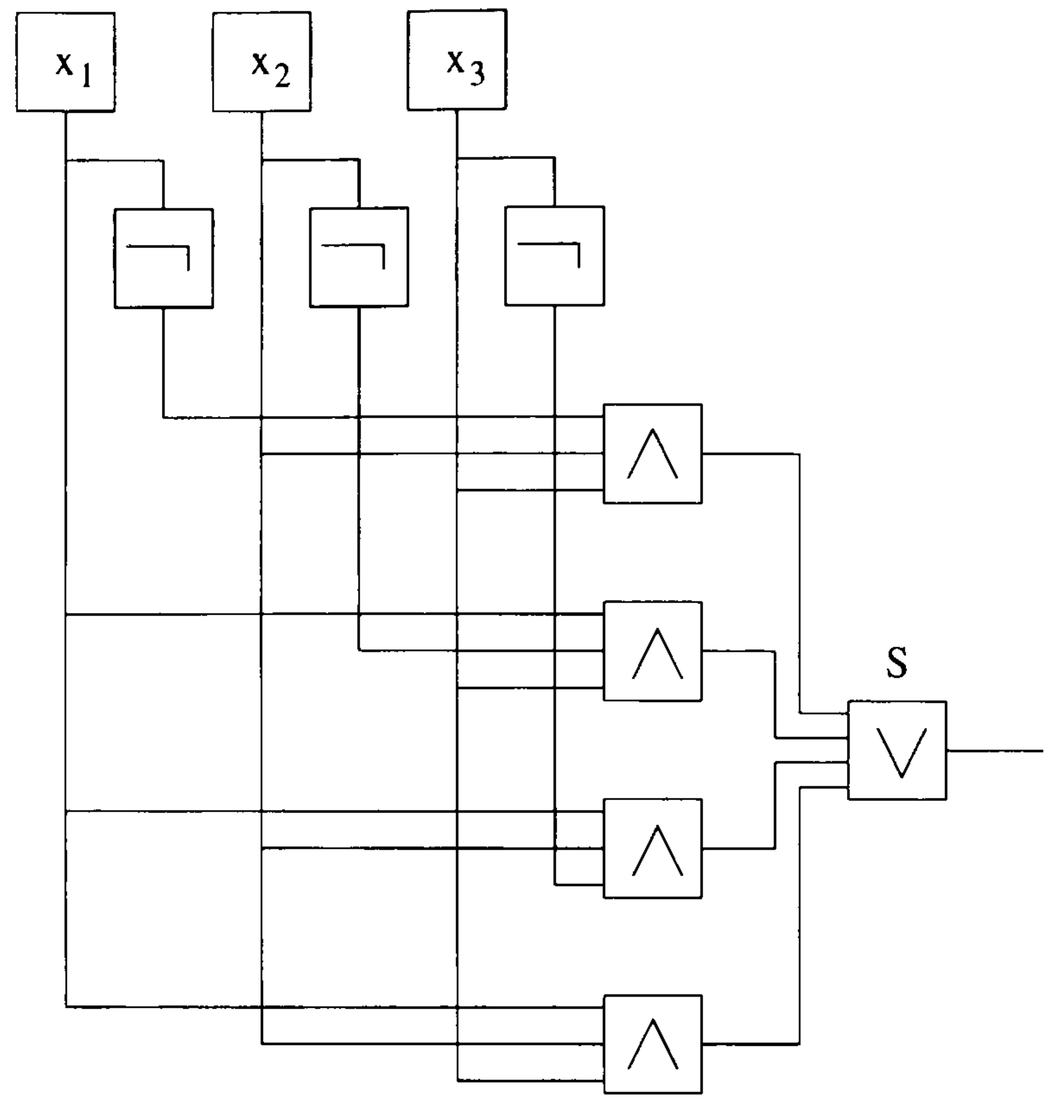

Figure 3.3: Boolean circuit for the three-input majority function

degree (i.e., input lines) zero, and *gate* nodes with maximum in-degree two. Each input node is associated with a unique Boolean variable x_i from the input set $X_n = (x_1, x_2, \ldots, x_n)$. Each gate node, g_i is associated with some Boolean function $f_i \in \Omega$. We refer to Ω as the circuit *basis*. A *complete* basis is a set of functions that are able to express all possible Boolean functions. It is well-known that the NAND function provides a complete basis by itself, but for the moment we consider the common basis, according to which $\Omega = \{\wedge, \vee, \neg\}$ In addition, S has some unique *output* node, s, with out-degree zero. An example Boolean circuit for the three-input *majority function* is depicted in figure 3.3.

The two standard complexity measures for Boolean circuits are *size* and *depth*: the size of a circuit, S, m, is the number of gates in S; its depth, d, is the number of gates in the *longest* directed path connecting an input vertex to an

output gate. The circuit depicted in figure 3.3 has size 8 and depth 3.

In [61], 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_n.

In what follows, $\Omega = \{\wedge, \vee\}$. For each i, $1 \leq i \leq m$, a string $\sigma[i]$ is fixed. The presence of $\sigma[i]$ in U signifies that g_i evaluates to 1. The absence of $\sigma[i]$ in U signifies that g_i evaluates to 0. The initial tube, U. (i.e., a tube representing the inputs X_n is created as follows:

for each gate x_i **do**

if $x_i = 1$ **then** $U \leftarrow U \cup \sigma[i]$

end for

The simulation of gates at level $k > 0$ proceeds as follows. We denote by i_1 and i_2 the indices of the gates that supply the inputs for g_i.

(1) Input(U)

(2) **for** $k = 1$ to d **do**

(3) **for** each gate g_i at level k **in parallel do**

(4) **if** g_i computes $\vee$ then do

(5) **if** $(\sigma[i_1] \in U)$ **or** $(\sigma[i_2] \in U)$ **then** $U \leftarrow U \cup \sigma[i]$

(6) **else if** g_i computes $\wedge$ then do

(7) **if** $(\sigma[i_1] \in U)$ **and** $(\sigma[i_2] \in U)$ **then** $U \leftarrow U \cup \sigma[i]$

(8) **end for**

(9) **end for**

At step (1) the initial tube, U, is created. Then, for each circuit level $k > 0$ (step 2), the gates at level k are simulated in parallel (steps 3-8). The simulation of each gate g_i at level k is achieved as follows. If g_i is an $\vee$-gate, the string $\sigma[i]$ is made present[1] in U (i.e., g_i evaluates to 1) if either of the strings representing the inputs to g_i is present in U (step 5). If g_i is an $\wedge$-gate, the string $\sigma[i]$ is made present in U (i.e., g_i evaluates to 1) if *both* of the strings representing the inputs to g_i are present in U (step 7). The simulation then proceeds to the next level.

At the termination of the computation, Ogihara and Ray analyse the contents of U to determine the output of the circuit. This analysis is described in more detail in section 4.7, where we provide full details of the model's biological implementation.

3.5.2 Winfree, Yang and Seeman

Of even greater speculation is the use of DNA self-assembly as a computational model. The construction of unusual DNA molecules has a long history [56, 72, 73]. Such molecules include knots [24], Holliday junctions [29], double cross-overs [30] and octahedra [82]. The construction of such objects relies upon the creation of branched DNA molecules known as junctions (figure 3.4). Since these objects do not occur naturally, they must be engineered in the laboratory. In order to determine the specific sequences to assign to components of branches (i.e. single strands of DNA) a technique known as sequence symmetry minimisation [71] is used. This assigns sequences to various components such that when placed in solution they hybridise to form the desired structure.

In [80], Winfree *et al.* propose as a computational tool the tendency of DNA

[1]The process by which this is achieved is described in detail in section 4.7.

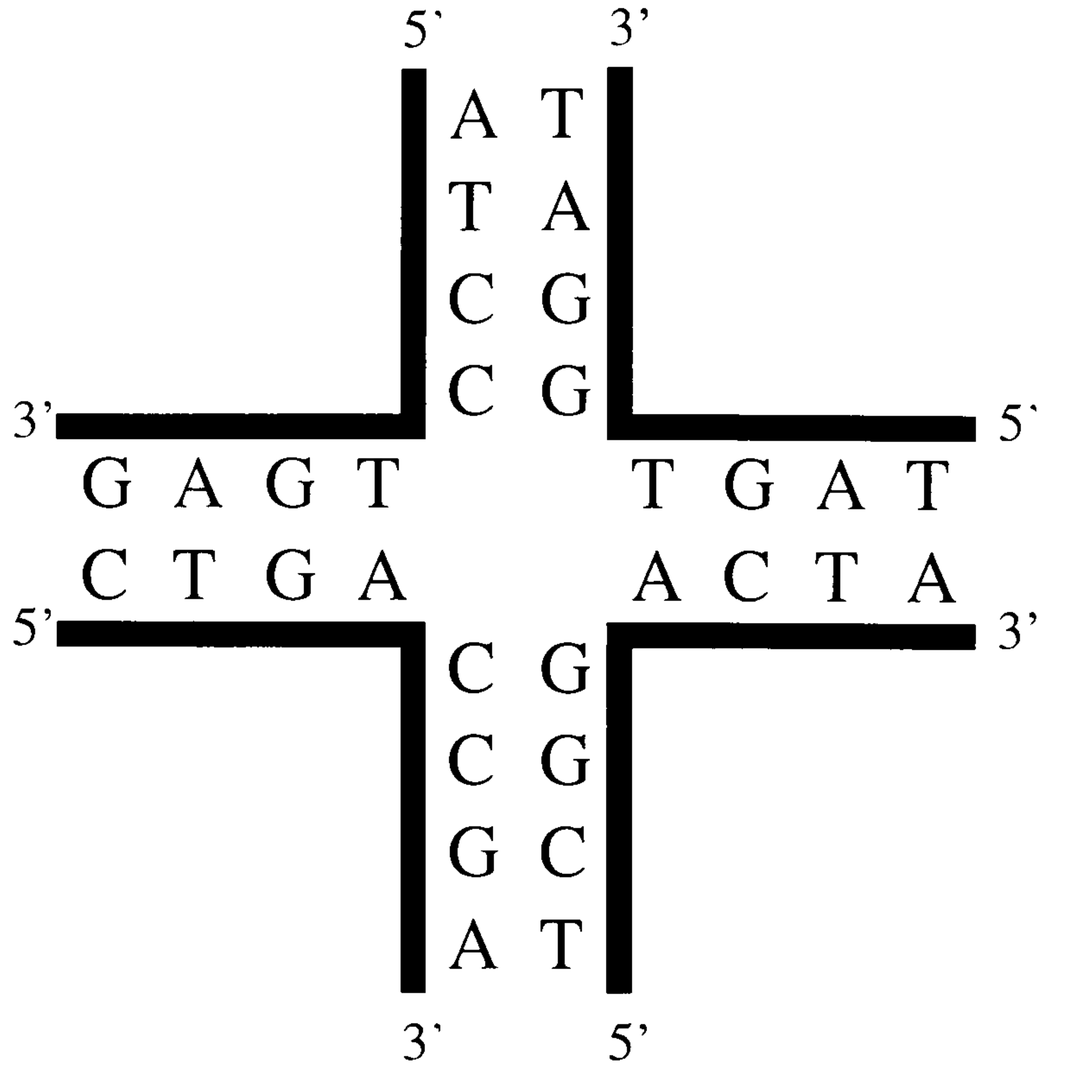

Figure 3.4: DNA branched junction

structures to self-assemble. They show that the self-assembly of DNA molecules into two dimensional sheets or three dimensional solids is a powerful model, capable of universal computation by virtue of the fact that it simulates a one-dimensional cellular automaton. Although of great theoretical interest, experimental investigations of the power of self-assembly are still at a very preliminary stage. In [80], Winfree *et al.* show that, in principle, the two dimensional self-assembly model is experimentally implementable.

3.6 Summary

In this chapter we described our parallel filtering model [7], which was essentially the first model following Adleman's seminal experiment in which solutions to the NP-complete problems could be elegantly expressed. We also observed that an extension

of the model to acquire full Turing computability is easily achieved through the addition of the splicing operation. We also placed our own work in the general context of the literature on abstract models of DNA computation. Most of this has appeared since our work, or is at best contemporaneous.

Although several theoretical models of DNA computation have been described in this chapter, their potential for physical realisation is unclear. Out of all of the models previously described, only that of Adleman [2] has been successfully implemented in the laboratory. The difficulty of physical implementation varies greatly between models, although the models that use standard laboratory techniques (i.e., filtering models and some of the constructive models) may be easier than others to implement. The realisability of more speculative models, such as the splicing and self-assembly models is still an open problem. We consider implementation issues in the following chapter.

Chapter 4

Implementation issues

4.1 Introduction

This chapter provides an introduction to the laboratory implementation of abstract models of DNA computation. We concentrate in particular on a full description of two filtering models (Adleman's and parallel filtering). and on one constructive model (Ogihara and Ray's Boolean circuit model). We highlight the practical implementation problems inherent in all models, and suggest possible ways to alleviate these.

4.2 Initial set construction within filtering models

As stated in section 3.3, all filtering models use the same basic method for generating the initial set of strands. An essential difficulty in all filtering models is that initial multi-sets generally have a cardinality which is exponential in the problem size. It would be too costly in time. therefore, to generate these individually. What we do in practice is to construct an initial solution, or *tube*, containing a polynomial number

of distinct strands. The design of these strands ensures that the exponentially large initial multi-sets of our model will be generated automatically. The following paragraph describes this process in detail.

Consider an initial set of all elements of the form $p_1k_1.p_2k_2.....p_nk_n$. This may be constructed as follows. We generate an oligonucleotide (commonly abbreviated to *oligo*) uniquely encoding each possible subsequence p_ik_i where $1 \leq i \leq n$ and $1 \leq k_i \leq k$. Embedded within the sequence representing p_i is our chosen restriction site. There are thus a polynomial number, nk, of distinct oligos of this form. The task now is how to combine these to form the desired initial multi-set. This is achieved as follows. For each pair $(p_ik_i, p_{i+1}k_{i+1})$ we construct an oligo which is the concatenation of the complement of the second half of the oligo representing p_ik_i and the complement of the first half of the oligo representing $p_{i+1}k_{i+1}$. We also construct oligos that are the complement of the first half of the oligo representing p_1k_1 and the last half of the oligo representing p_nk_n. There is therefore a total of $2nk+1$ oligos in solution.

The effect of adding these new oligos is that double-stranded DNA will be formed in the tube one strand in each will be an element of the desired initial set. The new oligos have, through annealing, acted as "splints" to join the first oligos in the desired sequences. These splints may then be removed from solution (assuming that they are biotinylated).

4.3 Adleman's implementation

Adleman utilised the incredible storage capacity of DNA to implement a brute-force algorithm for the directed Hamiltonian Path Problem (HPP). Recall that the HPP involves finding a path through a graph that visits each vertex exactly once. The

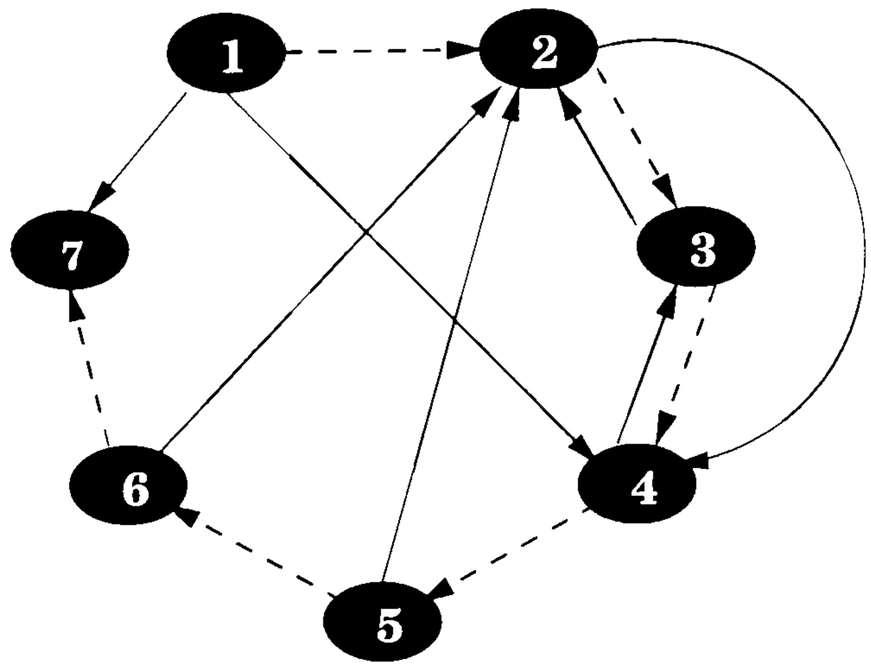

Figure 4.1: Instance of the HPP solved by Adleman

instance of the HPP that Adleman solved is depicted in figure 4.1, with the unique Hamiltonian Path (HP) highlighted by a dashed line.

Adleman's approach was simple:

1. Generate strands encoding random paths such that the Hamiltonian Path (HP) is represented with high probability. The quantities of DNA used far exceeded those necessary for the small graph under consideration, so it is likely that *many* strands encoding the HP were present.

2. Remove all strands that do not encode the HP.

3. Check the remaining strands encode a solution to the HPP.

The individual steps were implemented as follows:

Stage 1: Each vertex and edge was assigned a distinct 20-mer sequence of DNA. This implies that strands encoding a HP were of length 140 b.p. Sequences representing edges act as 'splints' between strands representing their endpoints. In formal terms, the sequence associated with an edge $i \to j$ is the $3'$ 10-mer of the sequence representing v_i followed by the $5'$ 10-mer of the sequence representing v_j.

These oligonucleotides were then combined to form strands encoding random paths through the graph. Fixed amounts (50 pmol) of each oligonucleotide were mixed together in a single ligation reaction. At the end of this reaction, it is assumed that a strand representing the HP is present with high probability. This approach solves the problem of generating an exponential number of different paths using a polynomial number of initial oligonucleotides.

Stage 2: PCR was first used to massively amplify the population of oligonucleotides encoding paths starting at v_1 and ending at v_7. Next, strands that do not encode paths containing exactly n visits were removed. The product of the PCR amplification was run on an agarose gel to isolate strands of length 140 b.p. A series of affinity purification steps was then used to isolate strands encoding paths that visited each vertex exactly once.

Stage 3: Graduated PCR was used to identify the unique HP that this problem instance provides. For an n-vertex graph, we run $n-1$ PCR reactions, with the strand representing v_1 as the left primer and the complement of the strand representing v_i as the right primer in the ith lane. The presence of molecules encoding the unique HP depicted in figure 4.1 should produce bands of length 40, 60, 80, 100, 120 and 140 b.p. in lanes 1-6 respectively. This is exactly what Adleman observed. The graduated PCR approach is depicted in figure 4.2.

Adleman's experiment was remarkable in that it was the first to demonstrate in the laboratory the feasibility of DNA computing. However, we note that it was performed on a single problem instance with just one HP. No control experiments were performed for cases without Hamiltonian Paths. The final detection step is problematic, due to the reliance on the error-prone PCR procedure. In addition, the use of affinity purification is also error-prone, which may mean that the experiment

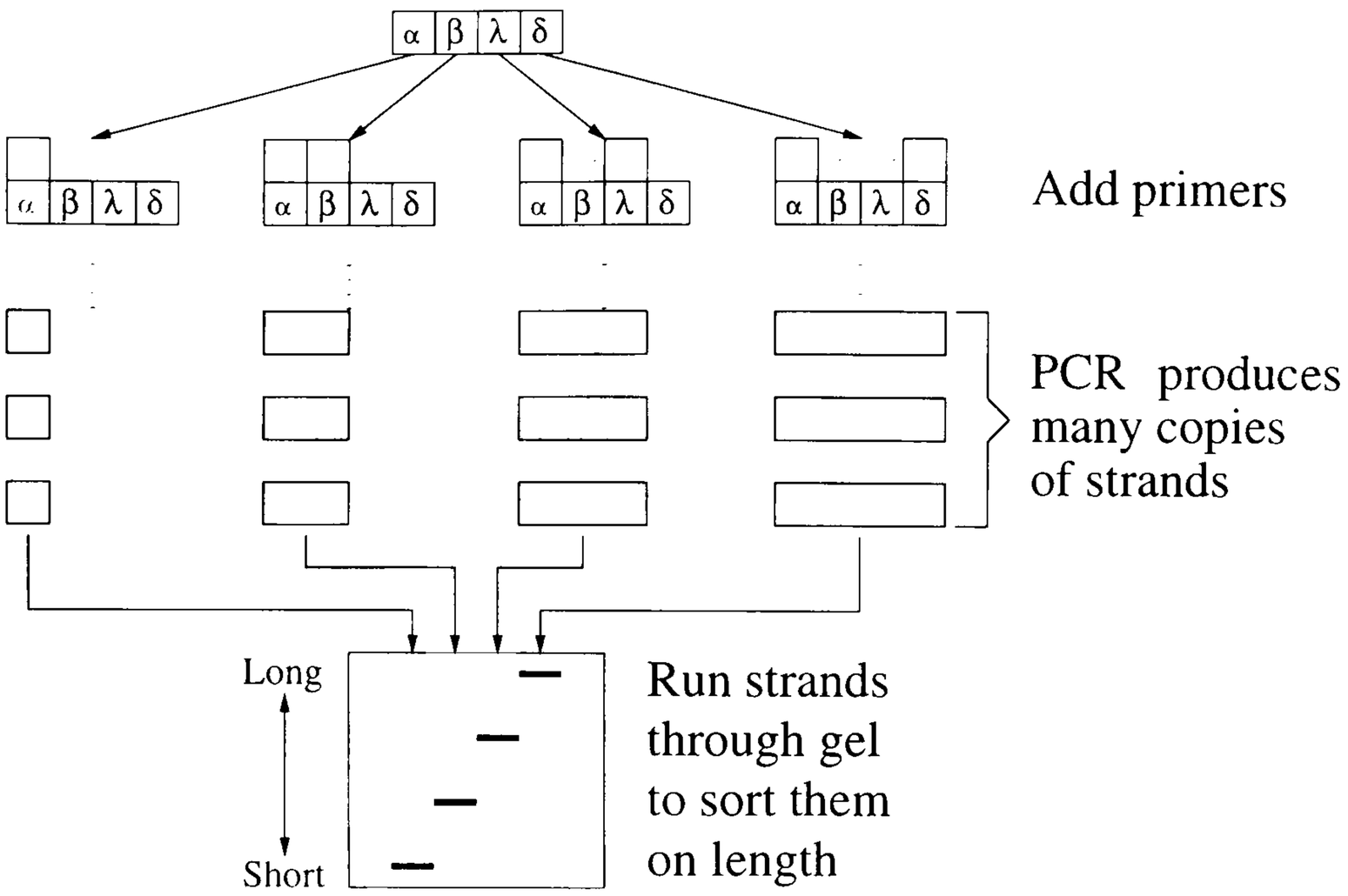

Figure 4.2: Graduated PCR

will not successfully scale up. We consider these issues in later sections.

4.4 Evaluation of Adleman's implementation

We describe later how the various multi-set operations described in the previous section may be realised thorough standard DNA manipulation techniques. However, it is convenient at this point to emphasise two impediments to effective computation by this means. The first hampers the problem size that might be effectively handled, and the second casts doubt on the potential for biochemical success of the precise implementations that have been proposed.

Naturally, the strings making up the multi-sets are encoded in strands of DNA in all the proposed implementations. Consider for a moment what volume of DNA would be required for a typical NP-complete problem. The algorithms

mentioned earlier require just a polynomial number of DNA manipulation steps. For the NP-complete problems there is an immediate implication that an exponential number of parallel operations would be required within the computation. This in turn implies that the tube of DNA must contain a number of strands which is exponential in the problem size. Despite the molecular dimensions of the strands, for only moderate problem sizes (say, n $\sim$ 20 for the Hamiltonian Path problem) the required volume of DNA would make the experiments impractical. As Hartmanis points out in [42], if Adleman's experiment were scaled up to 200 vertices the weight of DNA required would exceed that of the Earth. Mac Dónaill also presents an analysis of the scalability of DNA computations in [23], as do Linial and Linial [51], Lo *et al.* [54] and Bunow [20].

We note that [9] have described DNA algorithms which reduce the problem just outlined, however, the "exponential curse" is inherent in the NP-complete problems. There is the hope, as yet unrealised (despite the claims of [11]) that for problems in the complexity class P (i.e. those which can be solved in sequential polynomial-time) there may be DNA computations which only employ polynomial-sized volumes of DNA.

Now we consider the potential for biochemical success that was mentioned earlier. It is a common feature of *all the early proposed implementations* that the biological operations to be used are assumed to be error-free. An operation central to and frequently employed in most models is the *extraction* of DNA strands containing a certain sequence (known as *removal by DNA hybridisation*). The most important problem with this method is that it is not 100% specific[1], and may at times inadvertently remove strands that *do not* contain the specified sequence. Adle-

[1]The actual specificity depends on the concentration of the reactants.

man did not encounter problems with extraction because only a few operations were required. However, for a large problem instance, the number of extractions required may run into hundreds, or even thousands. For example, a particular DNA-based algorithm may rely upon repeated "sifting" of a "soup" containing many strands, some encoding legal solutions to the given problem, but most encoding illegal ones. At each stage, we may wish to extract only strands that satisfy certain criteria (i.e., they contain a certain sequence). Only strands that satisfy the criteria at one stage go through to the next. At the end of the sifting process, we are hopefully left only with strands that encode legal solutions, since they satisfy all criteria. However, assuming 95% efficiency of the extraction process, after 100 extractions the probability of us being left with a soup containing (a) a strand encoding a legal solution, and (b) no strands encoding illegal solutions is about 0.006. Repetitive extraction will not guarantee 100% efficiency, since it is impossible to achieve the conditions whereby only correct hybridisation occurs. Furthermore, as the length of the DNA strands being used increases, so does the probability of incorrect hybridisation.

These criticisms have been borne out by recent attempts [46] to repeat Adleman's experiment. The researchers performed Adleman's experiment twice; once on the original graph as a positive control, and again on a graph containing no Hamiltonian path as a negative control. The results obtained were inconclusive. The researchers state that *"At this time we have carried out every step of Adleman's experiment, but have not gotten (sic) an unambiguous final result."*

Although attempts have been made to reduce errors by (1) simulation of highly reliable purification using a sequence of imperfect operations [48], and (2) application of PCR at various stages of the computation [16], it is clear that reliance on affinity purification must be minimised, or, ideally, removed entirely. In [7], we

describe one possible error-resistant model of DNA computation that removes the need for affinity purification within the main body of the computation. It is proposed that affinity purification be replaced by a new enzymatic removal technique.

In [49], Kurtz *et al.* consider the effect of problem size on the initial concentrations of reactants and analyse the subsequent probability of a correct solution being produced. They claim that, without periodic amplification of the working solution, the concentration of strands drops exponentially with time to "homeopathic levels". One proposal to reduce strand loss during computations is described in [53]. Rather than allowing strands to float free in solution, the authors describe a surface-based approach, whereby strands are immobilised by attachment to a surface (glass is used in the experiments described in [53], although gold and silicon are other possible candidates.) The attachment chemistry is described in detail in [40]. This model is similar to that described in [7], in that it involves selective destruction of specific strands, although in this case Exonuclease is used to destroy unmarked rather than marked strands. Preliminary experimental results suggest that strand loss is indeed reduced, although the scalability of this approach is questionable due to the two dimensional nature of the surface.

4.5 Implementation of the parallel filtering model

Here we describe how how the set operations within the Parallel Filtering Model described in section 3.3.3 may be implemented.

4.5.1 Remove

$remove(U, \{S_i\})$ is implemented as a composite operation, comprised of the following:

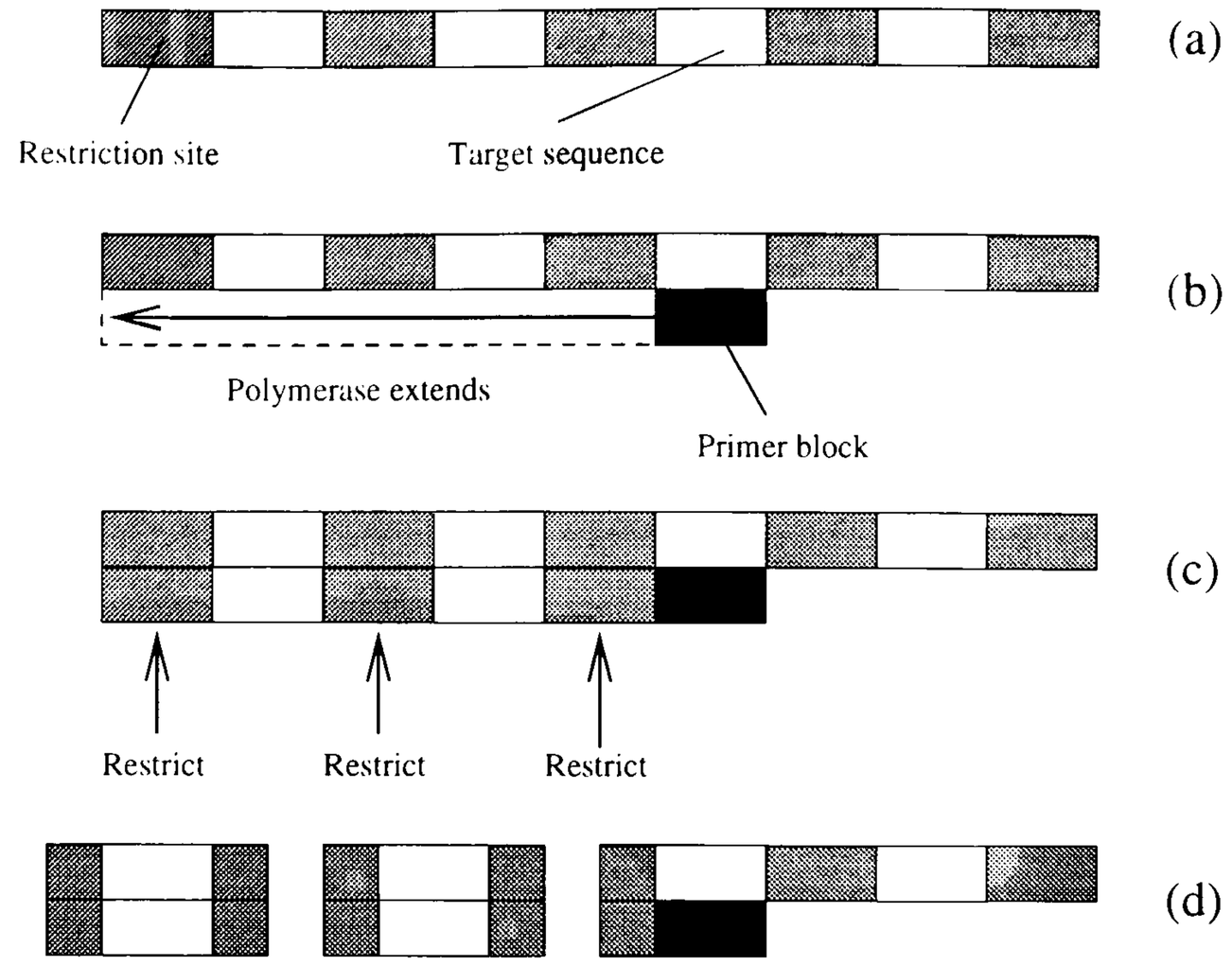

Figure 4.3: Implementation of *destroy*

- $mark(U, S)$. This operation marks all strings in the set U which contains at least one occurrence of the substring S.

- $destroy(U)$. This operation removes all marked strings from U.

$mark(U, S)$ is implemented by adding to U many copies of a primer corresponding to $\overline{S}$ (figure 4.3(b)). This primer only anneals to single strands containing the subsequence S. We then add DNA polymerase to extend the primers once they have annealed, making double-stranded only the single strands containing S (figure 4.3(b)).

We may then *destroy* strands containing S by adding the restriction enzyme *Sau*3AI. Double-stranded DNA (i.e. strands marked as containing S) is cut at the restriction sites embedded within, single strands remaining intact (figure 4.3(c)). We may then remove all intact strands by separating on length using gel

electrophoresis. However, this is not strictly necessary, and leaving the fragmented strands in solution will not affect the operation of the algorithm.

4.5.2 Union

We may obtain the *union* of two or more tubes by simply mixing their contents together, forming a single tube.

4.5.3 Copy

We obtain i "copies" of the set U by splitting U into i tubes of equal volume. We assume that, since the initial tube contains multiple copies of each candidate strand, each tube will also contain many copies.

4.5.4 Select

We can easily detect remaining *homogeneous* DNA using PCR and then sequence strands to reveal the encoded solution to the given problem. One problem with this method is that there are often multiple correct solutions left in the soup which must be sequenced using nested PCR. This technique is only useful when the final solution is known in advance. Also, the use of PCR may introduce an unacceptable level of error in the read-out procedure. A possible solution is to utilise *cloning*. We describe this approach in more detail in chapter 5.

Although the initial tube 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 (this may also be performed after *copy* operations).

4.6 Advantages of our implementation

As we have shown, algorithms within our model perform successive "filtering" operations, keeping *good* strands (i.e., strands encoding a legal solution to the given problem) and destroying *bad* strands (i.e., those that do not). So long as the operations work correctly, the final set of strands will only consist of good solutions. However, as we have already stated, errors can take place. If either good strands are accidentally destroyed or bad strands are left to survive through to the final set then the algorithm will fail. The main advantage of our model is that it doesn't repeatedly use the notoriously error-prone separation by DNA hybridisation method to extract strands containing a certain subsequence. Restriction enzymes are *guaranteed* [15, page 9] [2] to cut any double-stranded DNA containing the appropriate restriction site, whereas hybridisation separation is never 100% efficient. Instead of extracting *most* strands containing a certain subsequence we simply destroy them with high probability, without harming those strands that *do not* contain the subsequence. Even if, in reality, restriction enzymes have a small non-zero error rate associated with them, we believe that it is far lower than that of hybridisation separation. Another advantage of our model is that it minimises physical manipulation of tubes during a computation. Biological operations such as pipetting, filtering and extraction all lose a certain amount of material along the way. As the number of operations increases, the material loss rises and the probability of successful computation decays. Our implementation uses relatively benign physical manipulation, and avoids certain "lossy" operations.

[2] "New England Biolabs provides a color-coded 10X NEBuffer with each restriction endonuclease to ensure optimal (100%) activity."

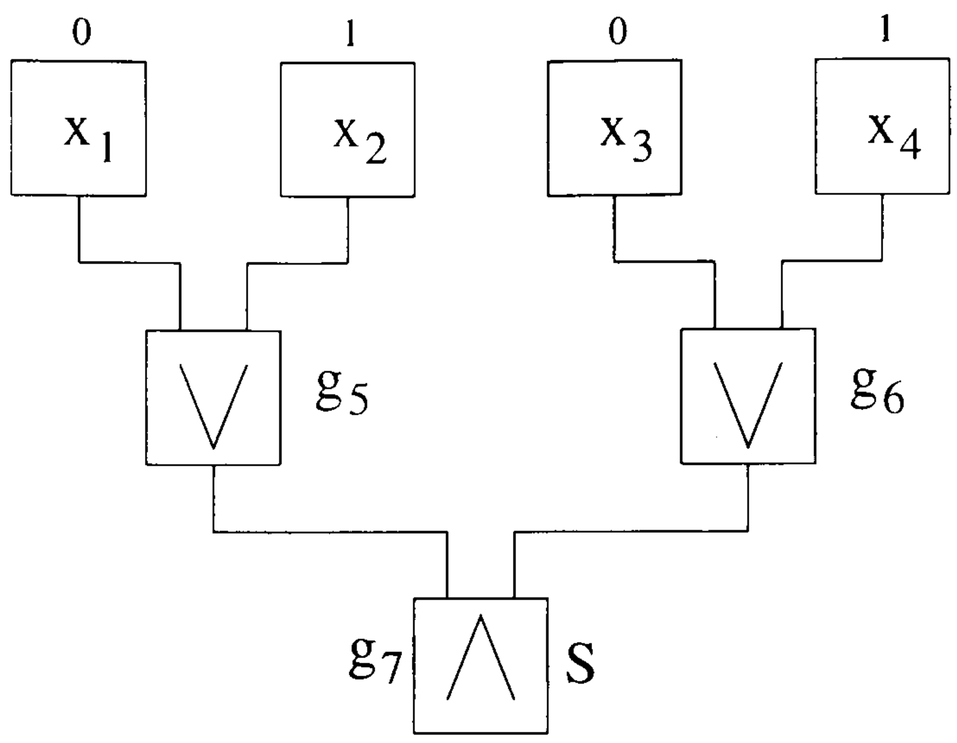

Figure 4.4: Boolean circuit simulated by Ogihara and Ray

4.7 Ogihara and Ray's implementation

In this section we describe the laboratory implementation of the abstract Boolean circuit model described in section 3.5.1. The circuit simulated in [61] is depicted in figure 4.4.

Each gate g_i is assigned a sequence of DNA, $\sigma[i]$, of length $\mathcal{L}$. beginning with a specific restriction site, $\mathcal{E}$. Each edge $i \to j$ is assigned a sequence $e_{i,j}$ that is the concatenation of the complement of the 3′ $\mathcal{L}/2$-mer of $\sigma[i]$ and the complement of the 5′ $\mathcal{L}/2$-mer of $\sigma[j]$. In this way, $e_{i,j}$ acts as a "splint" between g_i and g_j iff both $\sigma[i]$ and $\sigma[j]$ are present. However, we later highlight a case where this strand design does not hold.

The simulation of gates at level 0 (i.e., the construction of the initial tube) proceeds as follows. Begin with a tube of solution containing no DNA. For each input gate x_i that evaluates to 1, pour into the tube a population of strands representing $\sigma[i]$. If x_i evaluates to 0, do not add $\sigma[i]$ to the tube.

We now consider the simulation of gates at level $k > 0$. The simulation of

$\vee$-gates differs from that $\wedge$-gates, so we first consider the case of a $\vee$-gate, g_j. at level k.

First, pour into the tube strands representing $\sigma[j]$. Then, for each edge $i \rightarrow j$ pour into the tube strands representing $e_{i,j}$. Allow ligation to occur. If strands representing either of the inputs to g_j are present, the edge strands will cause strands of length $2\mathcal{L}$ to be formed, consisting of the concatenation of the sequence representing the g_i with the sequence representing g_j. This process is depicted in figure 4.5(a). These strands are then separated out by running the solution on a polyacrylamide gel. These strands are then cut with a restriction enzyme recognising sequence $\mathcal{E}$. This step leaves in solution strands of length $\mathcal{L}$ that correspond to g_j (i.e., g_j evaluates to 1).

We now consider the simulation of a $\wedge$-gate, g_j at level k. Again, pour into the tube strands representing $\sigma[j]$. Then, for all edges $i \rightarrow j$ pour into the tube strands representing $e_{i,j}$. Allow ligation to occur. If strands representing *both* of the inputs to g_j are present, the edge strands will cause strands of length $3\mathcal{L}$ to be formed, consisting of the concatenation of the sequence representing the first input to g_j, the sequence representing g_j and the sequence representing the second input to g_j (see figure 4.5(b)). Note that this splinting only occurs if the polarity of the edge splints is designed carefully, and we consider this in the next section. The strands of length $3\mathcal{L}$ are again separated out by a gel and cut with the appropriate restriction enzyme. This step leaves in solution strands of length $\mathcal{L}$ that correspond to g_j (i.e., g_j evaluates to 1).

If $k = d$, we omit the gel electrophoresis and restriction stages, and simply run the solution on a gel. If the output gate, s, is an $\vee$-gate, the output of the circuit is 1 iff length $2\mathcal{L}$ strands exist in solution. If s is an $\wedge$-gate, the output of

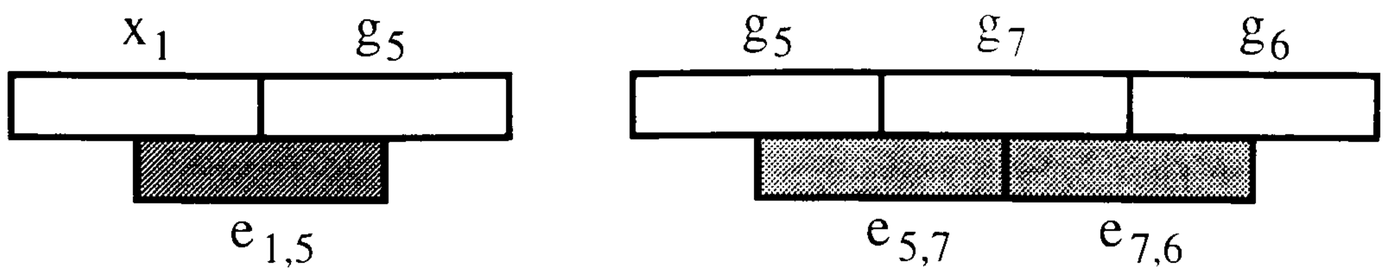

Figure 4.5: (a) Splinting for ∨-gate (b) Splinting for ∧-gate

the circuit is 1 iff length $3\mathcal{L}$ strands exist in solution.

4.7.1 Experimental results obtained

In [61], Ogihara and Ray report attempts to simulate the circuit previously described using the techniques of molecular biology. Their implementation is as described in the previous section. The results obtained were ambiguous. although Ogihara and Ray claim to have identified the ambiguity as being caused by pipetting error.

4.7.2 Evaluation of Ogihara and Ray's implementation

We consider the time complexity of Ogihara and Ray's model in chapter 6, but here we concentrate on one particular aspect of the model's practical implementation. The major problem lies in the design of strands to represent edges in the given circuit. As we have seen from the small example depicted in figure 4.4, careful thought must go into the design of edges incident to ∧-gates if correct splinting is to occur. It is not clear that there exists a method for correct sequence design in the general case. This problem could hamper efficient and error-resistant simulation of Boolean circuits within this model.

4.8 Summary

In this chapter we showed how various abstract models of DNA computation may be realised in the laboratory. By describing Adleman's inspirational experiment, we highlighted several important flaws in his approach. The implementation of our parallel filtering model described in this chapter provides a greater degree of error-resistance than those previously proposed, including Adleman's. By replacing the hybridisation separation technique with our own novel enzymatic removal procedure, we removed a major cause of errors from the implementation of DNA computations. We then described Ogihara and Ray's implementation of their Boolean circuit model, and considered its applicability in the general case.

Chapter 5

Experimental results

5.1 Introduction

In this chapter we describe the results of an experimental implementation of the parallel filtering model. In particular, we concentrate on testing the efficiency of the implementation of the *remove* operation, which is central to our model. We test this operation in the context of an implementation of the 3COL algorithm described in chapter 2. Although we have not yet advanced to the stage of fully implementing this algorithm, the results obtained are promising. More importantly, we describe the implications of our results for future experimental investigations of DNA-based algorithms. The major contribution of this chapter is to highlight potential problems with such implementations. We describe several impediments to efficient and error-resistant implementation of models of DNA computation, and suggest possible ways to alleviate these. In particular, we concentrate on the problem of *final read-out*, that is, the method of obtaining a final solution at the end of the execution of a DNA-based algorithm. We argue that this problem has been largely ignored in the

literature, and suggest a general read-out strategy (the *cloning* method) that may be employed by any implementation, regardless of the abstract model.

5.2 Experimental objectives

The primary objectives of the experiments detailed in this chapter are as follows:

1. To first ascertain optimal experimental conditions.

2. To test the implementation of the *remove* operation. We do this by performing a sequence of *removal* experiments, comprised of primer annealing, primer extension and restriction.

3. To test the error-resistance of a *sequence* of removal operations that would occur during an actual algorithmic implementation.

5.3 Experimental overview

Before attempting to solve an instance of 3COL, it is first necessary to ensure that we can execute repeated removal of strings containing a target sequence. Only then can we be confident of our claims for the efficiency of the removal method. In what follows, we assume that the problem instance to be solved is the graph, $G = (V. E)$ depicted in figure 5.1. Note that this graph is *not* 3-vertex-colourable, since it is important to demonstrate a lack of false positive results before attempting to implement the algorithm on a graph that *is* 3-vertex-colourable. This approach differs markedly from that of Adleman [2], who only carried out a single experiment on a rather contrived graph with a known solution.

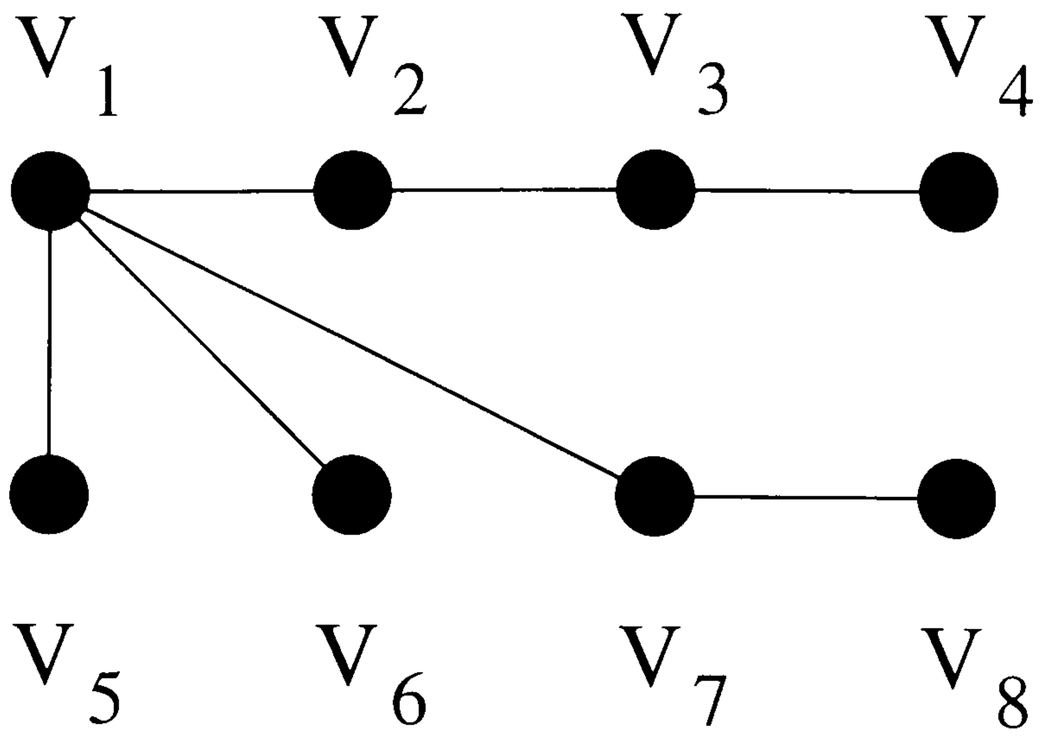

Figure 5.1: 3COL instance to be solved

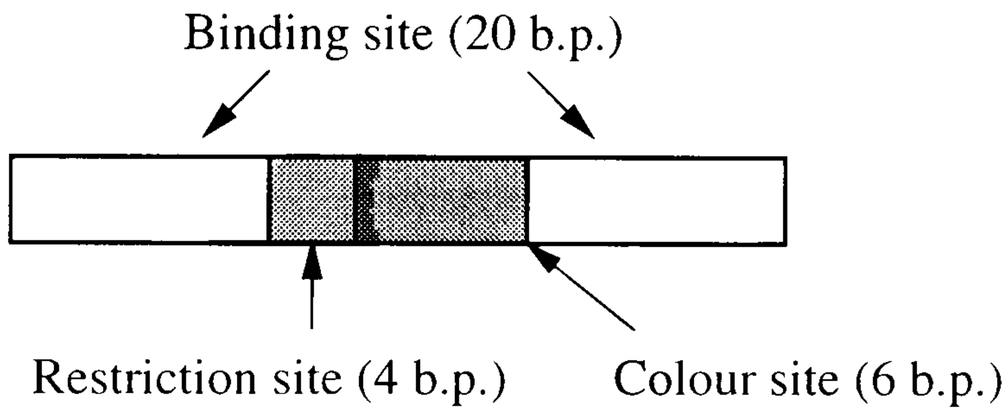

Figure 5.2: Oligonucleotide structure

5.4 Encoding colourings in DNA

We construct an initial library of (not necessarily legal) colourings in the following manner. For each vertex $v_i \in V$ we synthesise a *single* oligonucleotide (or *oligo*) to represent each of v_i=red, v_i=green and v_i=blue. The structure of these strands is depicted in figure 5.2.

With reference to figure 5.2, each oligo (apart from those representing v_1 and v_8) is composed of the following:

1. a unique 20-base binding site, within which is embedded a 4-base restriction

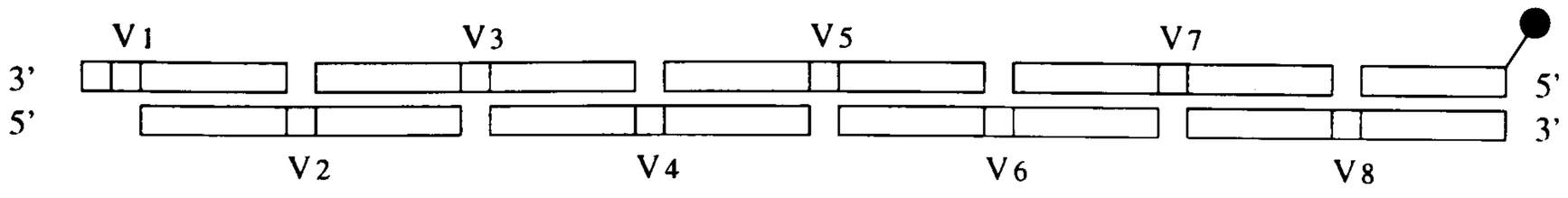

Figure 5.3: Library construction

site, $GATC$

2. a 6-base colour identification site. The sequences chosen to represent "red", "green" and "blue" are $AAAAAA$, $GGGGGG$ and $CCCCCC$ respectively

3. a unique 20-base binding site

- Oligos representing v_1 and v_8 are 34 bases long (8-base binding section, 6-base colour section and 20-base binding section). The initial sequences chosen to represent each vertex colouring are listed in table 5.1. Note that restriction sites are underlined (e.g., $\underline{GATC}$) and colour sequences are depicted in **bold**. The melting temperature (T_m) of each strand is specified in the final column.

All sequences described in this thesis were originally designed by hand. This process is laborious and prone to error, and we anticipate that one component of future work will be concerned with the implementation of a software package to automate the sequence design process. The sequences were then checked with the Microgenie [65] package to check for common subsequences and hairpin loops.

We now describe how the initial library is constructed from these oligos. In order to reduce to a minimum the number of oligos to be synthesised, we reject the *splinting* method described in [2, 52] in favour of an *overlapping* approach.

Sequences representing *odd*-numbered vertices run in the $3' \rightarrow 5'$ direction. Sequences representing *even*-numbered vertices run in the $5' \rightarrow 3'$ direction (see

Colouring	*Sequence*	T_m
$v_1 = red$	GCTCTGCT**AAAAAA**TCTTGATTTCACAGCATGGT	74.1
$v_1 = green$	GCTCTGCT**GGGGGG**TCTTGATTTCACAGCATGGT	83.1
$v_1 = blue$	GCTCTGCT**CCCCCC**TCTTGATTTCACAGCATGGT	82.5
$v_2 = red$	CGTCATAGGATCACCATGC**TTTTTTT**ACCATGCTGTGAAATCAAGA	81.5
$v_2 = green$	CGTCATAGGATCACCATGCT**CCCCCC**ACCATGCTGTGAAATCAAGA	88.4
$v_2 = blue$	CGTCATAGGATCACCATGCT**GGGGGG**ACCATGCTGTGAAATCAAGA	88.4
$v_3 = red$	AGCATGGTGATCCTATGACG**AAAAAA**TGCTGCTAAGACGAAGAGTT	80.9
$v_3 = green$	AGCATGGTGATCCTATGACG**GGGGGG**TGCTGCTAAGACGAAGAGTT	86.6
$v_3 = blue$	AGCATGGTGATCCTATGACG**CCCCCC**TGCTGCTAAGACGAAGAGTT	86.7
$v_4 = red$	GTAGGTGTGATCCAGTGGTT**TTTTTT**AACTCTTCGTCTTAGCAGCA	79.2
$v_4 = green$	GTAGGTGTGATCCAGTGGTT**CCCCCC**AACTCTTCGTCTTAGCAGCA	86.0
$v_4 = blue$	GTAGGTGTGATCCAGTGGTT**GGGGGG**AACTCTTCGTCTTAGCAGCA	86.0
$v_5 = red$	AACCACTGGATCACACCTAC**AAAAAA**GGTCTTCGGCGGCAATCTAC	83.7
$v_5 = green$	AACCACTGGATCACACCTAC**GGGGGG**GGTCTTCGGCGGCAATCTAC	89.9
$v_5 = blue$	AACCACTGGATCACACCTAC**CCCCCC**GGTCTTCGGCGGCAATCTAC	89.9
$v_6 = red$	GTAGGTGTGATCCAGTGGTT**TTTTTT**GTAGATTGCCGCCGAAGACC	83.8
$v_6 = green$	GTAGGTGTGATCCAGTGGTT**CCCCCC**GTAGATTGCCGCCGAAGACC	89.5
$v_6 = blue$	GTAGGTGTGATCCAGTGGTT**GGGGGG**GTAGATTGCCGCCGAAGACC	89.5
$v_7 = red$	AACCACTGGATCACACCTAC**AAAAAA**CACTGACAAGACCTTTGCTT	80.8
$v_7 = green$	AACCACTGGATCACACCTAC**GGGGGG**CACTGACAAGACCTTTGCTT	87.4
$v_7 = blue$	AACCACTGGATCACACCTAC**CCCCCC**CACTGACAAGACCTTTGCTT	86.8
$v_8 = red$	GCGGAATTCCTCTGCTGATC**TTTTTT**AAGCAAAGGTCTTGTCAGTG	81.9
$v_8 = green$	GCGGAATTCCTCTGCTGATC**CCCCCC**AAGCAAAGGTCTTGTCAGTG	89.1
$v_8 = blue$	GCGGAATTCCTCTGCTGATC**GGGGGG**AAGCAAAGGTCTTGTCAGTG	89.1

Table 5.1: Sequences chosen to represent vertex/colour combinations

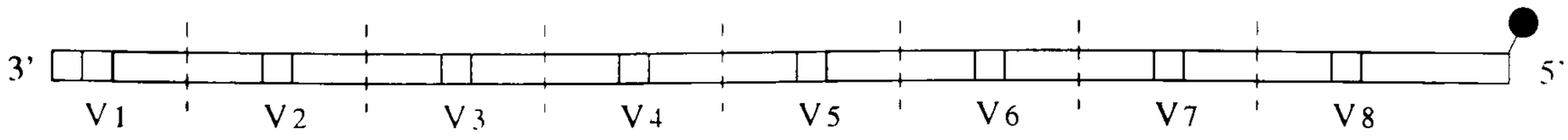

Figure 5.4: Structure of strands in initial library

figure 5.3). We now describe the structure of the binding sections of strands representing even-numbered vertices. The "left-hand" binding section of strands representing v_n (where n is even) is the complement of the "right-hand" binding section of strands representing v_{n-1}. Similarly, the "right-hand" binding section of strands representing v_n is the "left-hand" binding section of strands representing v_{n+1}.

We also construct a single biotinylated oligo, corresponding to the complement of the "right-hand" binding section of strands representing v_8. This allows us to purify strands encoding colourings away from the splint strands. The use of hybridisation extraction does not cause a problem at this stage, since the process is only performed once, rather than repeatedly during the main body of a computation.

We then pour all oligos into solution. When all oligos have annealed we expect to obtain many double strands of the form depicted in figure 5.3. Vertex and colour sections each occupy a distinct subsection of each strand. It is clear from the structure of these strands that the sequence encoding a particular vertex/colour combination depends not only on the vertex in question, but on whether or not the vertex number is odd or even. For example, sections colouring v_1 "red" have the sequence $AAAAAA$, as expected. However, due to the overlapping nature of the strand construction technique, sections colouring v_2 have the sequence $TTTTTT$. This minor complication does not present a problem, and, knowing the sequence assigned to each vertex/colour combination it is a trivial task to derive the sequences of the appropriate primer. The primer sequences are listed in table 5.2.

Primer	*Sequence*	T_m
$v_1 = red$	GCTCTGCTAAAAAATCTT	51.1
$v_1 = green$	GCTCTGCTGGGGGGTCT	66.8
$v_1 = blue$	GCTCTGCTCCCCCCTCTT	65.8
$v_2 = red$	AGCATGGTAAAAAAAGCA	55.8
$v_2 = green$	AGCATGGTGGGGGGAGC	71.5
$v_2 = blue$	AGCATGGTCCCCCCAGCA	71.5
$v_3 = red$	CTATGACGAAAAAATGCT	52.6
$v_3 = green$	CTATGACGGGGGGGTGCT	67.2
$v_3 = blue$	CTATGACGCCCCCCTGCT	67.4
$v_4 = red$	GAAGAGTTAAAAAAAACC	47.7
$v_4 = green$	GAAGAGTTGGGGGGAACC	63.3
$v_4 = blue$	GAAGAGTTCCCCCCAACC	63.3
$v_5 = red$	ACACCTACAAAAAAGGTC	51.1
$v_5 = green$	ACACCTACGGGGGGGGTC	68.2
$v_5 = blue$	ACACCTACCCCCCCGGTC	49.3
$v_6 = red$	CAATCTACAAAAAAAACC	49.3
$v_6 = green$	CAATCTACGGGGGGAACC	64.0
$v_6 = blue$	CAATCTACCCCCCCAACC	63.9
$v_7 = red$	ACACCTACAAAAAACACT	48.7
$v_7 = green$	ACACCTACGGGGGGCACT	66.7
$v_7 = blue$	ACACCTACCCCCCCCACT	65.4
$v_8 = red$	CTTTGCTTAAAAAAGATC	48.9
$v_8 = green$	CTTTGCTTGGGGGGGAT	66.6
$v_8 = blue$	CTTTGCTTCCCCCCGAT	66.7

Table 5.2: Sequences of primers

5.5 Materials and methods

We now describe in detail the 32 experiments carried out during this particular phase of the project. A summary of these is given in table 5.3. The results of these experiments are listed in the next section - here we describe only the materials and methods used. We represent the order of experimental execution in figure 5.5. Each process box is labelled with the numbers of the experiments carried out at that stage. Note that the only cycle in the flowchart occurs while attempting to remove strands containing a certain sequence. This is necessitated by the need for control and optimisation experiments in order to establish optimal (or near-optimal) experimental conditions. Specific conditions are detailed in the experimental protocol in appendix A.

Experiment 1. Oligos and Reagents

All the tile oligos were resuspended to 100pmoles/μl in distilled water, then diluted to produce two mastermixes, the first containing all the oligos and the other containing only red oligos. The final concentration in each case was 2.5pmoles each oligo/μl. The primer oligos were resuspended to 100pmoles/μl stocks and also diluted to 30pmoles/μl PCR working stock. The biotinylated primer (v_8) was resuspended to 200pmoles/μl and stored in 20μl single use aliquots. All oligos were stored at -20°C. A 5xPolymerase/Ligase buffer was made up (based on the requirements for 2nd strand cDNA synthesis).

Experiment 2. Library construction

Two hybridisation reactions were set up, one containing a full set of colouring oligos and the other a control containing only red colouring oligos. The products were

No.	*Summary of experimental objective*
1	Resuspension of oligos
2	Creation of initial library
3	Amplification of initial library
4	Purification of product of Experiment 3
5	Check to ensure correct hybridisation
6	Test of specificity of Experiment 5
7-8	Cloning and sequencing to ensure correct library construction
9	Removal experiment 1
10	Amplification and purification of test library subset
11	Removal experiment 2
12	Removal experiment 3
13	Test of ability of *Sau*3A to digest dsDNA
14-15	Control experiments using *Mbo*I rather than *Sau*3A
16	Removal experiment 4
17	Removal experiment 5
18	Removal experiment 6
19	Removal experiment using Klenow (7)
20	Removal experiment 8
21	Removal experiment 9
22	Test of multiple removals
23-24	Oligo redesign
25	New initial library construction
26-29	New initial library amplification
30-31	PCR control experiments for new library
32	Removal experiment 10

Table 5.3: Summary of experimental objectives

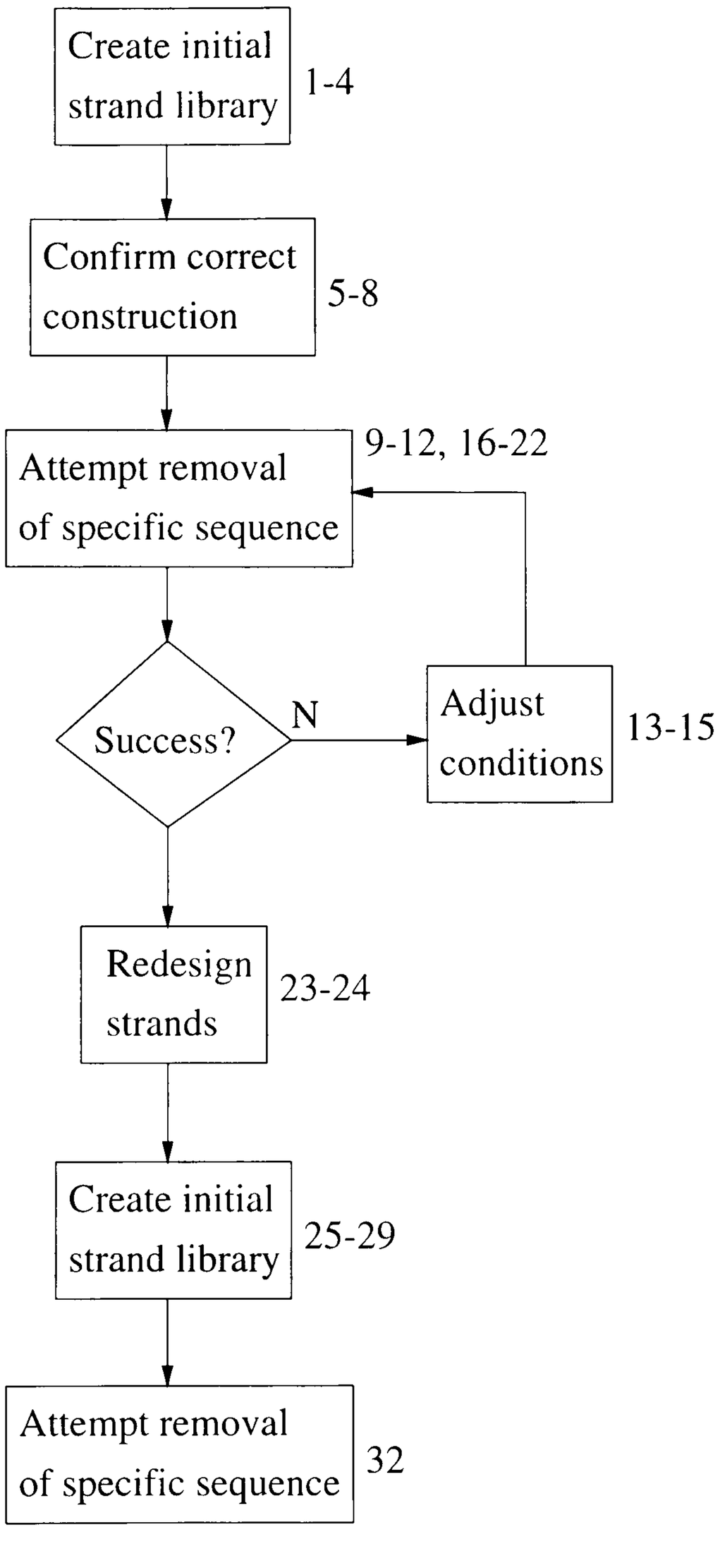

Figure 5.5: Flowchart depicting experimental cycle

labelled ALL and RED

Experiment 3. Amplification of template ALL

Hybridisation product ALL was amplified by PCR between v_8 and a mixture of the three v_1 primers. The primary aim was to generate a working stock containing a mixed population of colourings. The amplification was also used to confirm that the oligos had annealed correctly and that the polymerase/ligase step had repaired the gaps between oligos to produce double-stranded tile chains of the correct length. PCR template concentration and MgCl2 concentrations were titrated in order to optimise PCR conditions.

Experiment 4. Purification of v_1-v_8 PCR product

The 10 best samples producing bands in Experiment 3 were pooled and gel purified on 2% agarose. The ~200 b.p. fragment was extracted using the Qiagen gel extraction kit. The PCR product was ligated into a T cloning vector and used to transform stored competent bacteria. Several thousand clones were produced. 12 were grown up, miniprepped and sequenced (Experiments 7-8).

Experiment 5. PCR between v_2 and v_8

The aim was to check that all three colourings of v_2 were present in the amplified chain produced in Experiment 3. This was done simply by using v_8 and either red, green or blue specific v_2 primers in a standard detection PCR reaction.

Experiment 6. PCR from RED template (from Experiment 2)

This control experiment was set up to check the specificity of the PCR detection

step (to make sure the colour-specific primers did not cross-react, giving false positive results). The RED hybridisation product from Experiment 2 was diluted 1/10 and colour-specific detection PCRs set up for v_1, v_2, v_3 and v_4.

Experiments 7-8. Sequencing of clones from Experiment 4

8 clones were sequenced using either universal or reverse primers.

Experiment 9. Exclusion experiment 1

This initial control experiment was designed to test the ability of the proposed *Taq*-based primer extension/*Sau*3A digestion method of excluding specific sequences from a mixed population of chains. Template prepared in Experiment 4 was bound to Dynabeads, denatured and then split into two. One half was treated with the intention of excluding all but one vertex colouring, the other was used as a control and taken through the procedure without adding any exclusion primers. Following *Sau*3A digestion, the surviving DNA was harvested from the beads by *Eco*R1 digestion. Detection PCR reactions were set up to detect specific vertex sequences within each of the samples. The untreated half was intended as a positive control for the detection PCR step.

Experiment 10. Amplification and purification of the RED template

A red-only template was prepared by PCR from the RED hybridisation product (Experiment 2) using $v_1 = red$ and v_8 primers. The PCR product was gel purified and extracted using a Qiagen gel extraction kit.

Experiment 11. Exclusion experiment 2

This was a repeat of Experiment 9 with various modifications intended to increase the stringency of the exclusion step (amount of template reduced. primer concentration increased, annealing temperature reduced, number of cycles increased). The RED control template was also taken through the procedure as a PCR specificity control.

Experiment 12. Exclusion experiment 3

Conditions were modified further to favour exclusion. An additional control was included to test the ability of the *Sau*3A to destroy double-stranded sequences. This sample was bound to the Dynabeads, washed, but not denatured, and then taken through the procedure as double-stranded template, which should have been completely destroyed by the exclusion procedure.

Experiment 13. *Sau*3A control

This control was used to assess the ability of *Sau*3A to digest double-stranded DNA. Serial dilutions were made from template ALL. 1μl of each dilution was digested using 10U *Sau*3A at 37°C for one hour, in a total volume of 10μl. 1μl of each digest, and undigested controls, were used as template in standard v_1-v_8 detection PCR reactions.

Experiments 14-15. Control experiments using Mbo1 to digest targeted sequences.

Experiment 13 was repeated using *Mbo*1 (a *Sau*3A isoschizomer which has the same specificity for double stranded DNA).

Experiment 16. Exclusion experiment 4

Experiment 12 was repeated, substituting *Mbo*1 for *Sau*3A.

Experiment 17. Exclusion experiment 5

Fresh v_1-v_8 template was prepared and purified, then used in an exclusion experiment as in Experiment 12. Digestion was carried out overnight using 50U enzyme.

Experiment 18. Exclusion experiment 6

Template prepared in Experiment 17 was diluted 1/10 and 1μl bound to Dynabeads. denatured and washed. A v_2 exclusion was set up, using *Taq* in the primer extension reaction as usual. The beads were resuspended every 10 cycles and a 3μl aliquot of beads removed at roughly 20 cycle intervals, up to a maximum of 85 cycles. *Mbo*1 digestion of each aliquot was carried out overnight at 37°C in a rotating oven (to keep the beads in suspension) using 50U (a huge excess) of enzyme. v_2 detection PCRs were set up from each of the digested samples.

Experiment 19. Exclusion experiment using Klenow

In this experiment Klenow was substituted for *Taq* in the primer extension step. The idea was to overcome the problem of the beads settling out while cycling in the PCR block, which may have accounted for the inefficiency of the reaction. To do this the tubes were incubated in a 37°C rotating oven for 3 hours. Detection PCR was carried out as normal and the products run out alongside the samples from Experiment 18.

Experiment 20. Exclusion experiment 8

This was a *Taq*-based exclusion, using modified cycling conditions. The ALL template from Experiment 17 was diluted 1/10, and bound to the dynabeads. denatured, washed, and split in to three. Three single exclusion reactions were set up, one each for the v_2 colours (in this way the three templates acted as PCR controls for each other).

Experiment 21. Exclusion experiment 9

Experiment 20 was repeated with modifications to primer extension conditions aiming to favour the annealing of red primers.

Experiment 22. Multiple tile exclusion

This was the first attempt to exclude more than one tile at a time. It was thought that expanding the experiment in this way may increase the efficiency of the exclusion reactions by reducing the overall level of template available at the detection PCR step.

Experiments 23-24 Oligo redesign

New oligos were designed to replace the v_3 oligos in the existing chain. By slotting in oligos with modified characteristics, it was hoped that the principle of the exclusion technique could be shown to work (even if it was not possible to execute a full algorithm). The basic design features of the new oligos were as follows:

- The regions of overlap with v_2 and v_4 were conserved so that the new oligos could be incorporated into the old chain.

- The colour signatures were increased from 6 to 10 nucleotides in length to

increase stability.

- The three colour sequences shared no base homology. i.e.. they were different at each individual base.

- All the oligos were checked for runs of bases, homologies, hairpins etc.

- Primer T_m were as close as possible to each other (so that primers could be used in combination under optimal conditions), and all around 60°C. This was achieved by maintaining a GC ratio of 50% for each primer.

The new sequences are described later in this section.

Experiment 25. New full-length chain preparation

Three new full-length chains were prepared containing either red green or blue new v_3 in a backbone of the original red tiles v_1, v_2, v_4, v_7 and v_8. By omitting the green and blue tiles in the backbone of the molecule we hoped to avoid any interactions between blue $CCCCCC$ and green $GGGGGG$ sequences, which could lead to the formation of hairpins in the full length molecule. The construction method was exactly as Experiment 2.

Experiments 26-29. New chain amplification

Each of the new chains were amplified and purified separately to produce both biotinylated and non-biotinylated products. The non-biotinylated products were cloned for sequencing, (unfortunately the sequencing reactions failed and there was not enough time to repeat them). The biotinylated products were used in the control and exclusion optimisation experiments.

Experiments 30-31. PCR control experiments for the new tile chains

The three new tile chains containing either the red, green or blue v_3 were assessed separately. Serial dilutions were made from each template (down to a dilution of 10-8). PCR reactions were then set up in order to determine the limit of detection for each template. For subsequent control and exclusion reactions these templates were used at a concentration ~10fold above the limit of detection. It was hoped that by balancing the initial amount of template used in the experiments, any partial exclusion (as seen previously) would be sufficient to reduce the level of template below the limit of PCR detection, giving a negative result (i.e., showing that exclusion had been successful). PCR conditions were optimised to ensure that there was no cross reaction between the new coloured sequences, and *Mbo*1 digestion times were assessed to ensure complete digestion of double-stranded DNA at these concentrations.

Experiment 32. Final exclusion experiment

Red, green and blue templates were mixed in roughly equal proportions (based on their PCR detection limit in Experiment 30.) The mixed template was bound to the Dynabeads and prepared in bulk before splitting into four tubes. Red, green and blue v_3 exclusions were set up in separate reactions, alongside a no exclusion control. Following *Mbo*b digestion, detection PCR reactions were set up to determine the relative levels of each v_3 in each of the exclusion samples and control.

5.6 Results obtained

Experiment 3. The major product of ~200 b.p. was detectable at all template concentrations, though very faint at 1/1000 dilution. MgCl2 concentration had very

little effect on PCR efficiency. Optimal conditions appeared to be at 2mM MgCl2 using the template diluted to 1/10. In all cases the product appeared slightly smaller than expected, though it was not clear if this was a gel artifact or a problem with the library oligo assembly.

Experiment 5. All three vertices were found to be represented in the chain population (assuming no cross reaction had occurred)

Experiment 6. The PCR products were faint but in each case appeared to be colour specific. There was also a step-wise reduction in the size of the PCR product from v_1 through to v_4, showing that the PCR reactions were vertex-specific and that the majority of colourings had assembled in the correct order.

Experiments 7-8. It was clear that there was fairly high sequence variability amongst the clones. They showed a number of vertex assembly patterns and chain lengths. Some could be explained by PCR mis-priming during the chain amplification step, yielding products with a v_1 sequence at both ends (These products would not cause problems in the exclusion experiments since they were not biotinylated, would not bind to the dynabeads, and would be removed during the washing step. One feature common to all the clones was the absence of sequences representing v_5 and v_6. Looking at the oligo sequences it was clear that the problem was due to identical overlapping regions between v_4 and v_5, and v_6 and v_7. making two chains possible, the shorter of which seemed to form predominantly. In this small selection of clones it looked like the vertex colourings were represented equally amongst the chains.

Experiment 9. The detection reactions showed that the PCR products from each sample were of equal intensity for each vertex tested, showing that exclusion under these conditions had failed.

Experiment 11. The PCR results showed that the detection step was specific, but that the exclusion steps had not reduced the amount of targeted sequence.

Experiment 12. Detection PCR results showed that the PCR was specific (RED control), but that the exclusion steps had not worked, and also that *Sau*3A had failed to destroy the double-stranded control. This implied that the exclusion experiments were failing due to incomplete *Sau*3A digestion of the marked sequence

Experiment 13. Comparisons of the two sets of samples showed that digestion with *Sau*3A made no difference to the detection limit of the PCR reactions. As a further control, the products of the above reactions were gel purified and split in two. One half was digested with *Sau*3A and visualised alongside the other (undigested) control on 2% agarose. The undigested DNA ran as a distinct band, whereas the digested half appeared as a high molecular weight smear.

Experiments 14-15. An overnight 37°C digestion using 20U of enzyme was found to completely destroy the template (i.e., to reduce the level of template below the limit of PCR detection).

Experiment 16. Nothing could be concluded from this set of reactions since the

positive PCR controls failed.

Experiment 17. The positive and negative detection PCR controls worked, but all other reactions failed. The problem seemed to be due to inefficient harvesting of the excluded template from the Dynabeads prior to detection PCR. To get round this problem an un-biotinylated v_8 primer was ordered. Using this primer, detection PCRs could be set up directly from template bound to the Dynabeads.

Experiment 18. The results of this experiment gave the first evidence that the exclusion method could work. The intensity of the specific PCR product band decreased with increased number of exclusion cycles, although the exclusion never reached completion. The template was still detectable after 85 cycles of primer extension.

Experiment 19. The use of Klenow produced the same effect as ~30-40 cycles of *Taq* based exclusion (i.e., exclusion was not complete), showing that Klenow offered no advantage over *Taq*.

Detection PCR showed specific exclusion of $v_2 = green$ and $v_2 = blue$ sequences, but not $v_2 = red$. The gel is depicted in figure 5.6. A summarised interpretation of this gel is presented in table 5.4.

Lanes 1-3 show the result of the removal of strands encoding $v_2 = red$. Lane 1, corresponding to $v_2 = red$ *should* be empty, but a faint band is visible. Lanes 2 and 3, corresponding to $v_2 = green$ and $v_2 = blue$ primers respectively, contain normal length product, showing that strands *not* containing the sequence $v_2 = red$ were not removed.

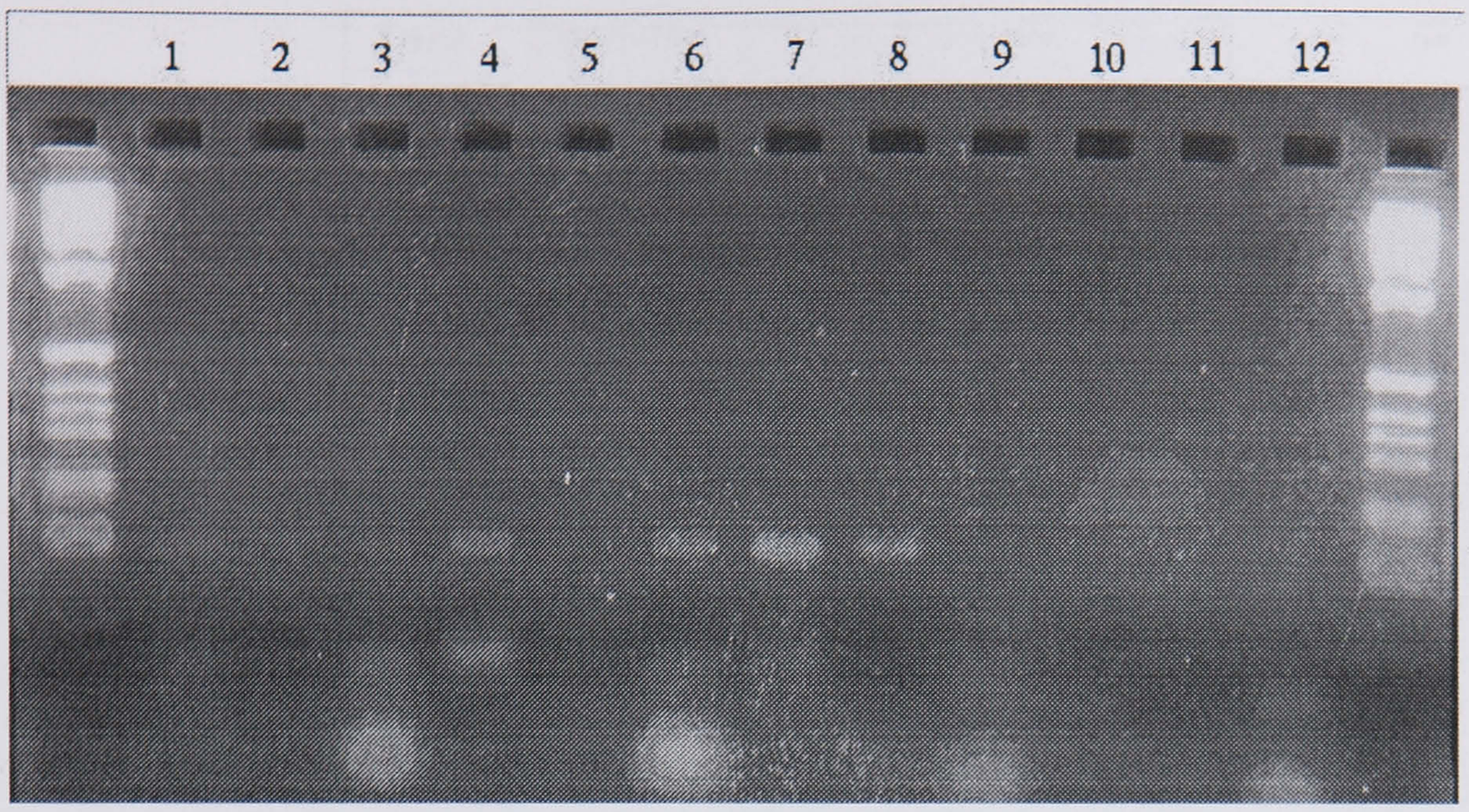

Figure 5.6: Visualisation of gel resulting from Experiment 20

We believe that the incomplete removal of $v_2 = red$ strands is due to the sequence chosen to represent red ($AAAAAA$). Because adenine only forms two hydrogen bonds with thymine, the optimum annealing temperature between strands and red primers is lower than that for green ($CCCCCC$) and blue ($GGGGGG$) primers. We believe a simple modification to the encoding sequence (described later) will solve this problem.

Lanes 4-6 show the result of the removal of strands encoding $v_2 = green$. Lane 5, corresponding to the $v_2 = green$ primer is empty, showing that no strands containing that sequence were present. Lanes 4 and 6, corresponding to $v_2 = red$ and $v_2 = blue$ primers respectively, contain normal length product, showing that strands *not* containing the sequence $v_2 = green$ were not removed.

Lanes 7-9 show the result of the removal of strands encoding $v_2 = blue$. Lane 9, corresponding to the $v_2 = blue$ primer is empty, showing that no strands containing that sequence were present. Lanes 7 and 8, corresponding to $v_2 = red$

Lane	*Excluded*	*PCR Primer*	*Result*
1		v_2=red	+
2	v_2=red	v_2=green	+
3		v_2=blue	+
4		v_2=red	+
5	v_2=green	v_2=green	-
6		v_2=blue	+
7		v_2 red	+
8	v_2=blue	v_2=green	+
9		v_2 blue	-
10	None	v_2 red	-
11	(PCR -ve	v_2 green	-
12	control)	v_2 blue	-

Table 5.4: Interpretation of figure 5.6

and $v_2 = green$ primers respectively, contain normal length product, showing that strands *not* containing the sequence $v_2 = blue$ were not removed.

The streaks visible at 74 and 18 b.p. are due to the presence of primer dimers and free primers respectively.

Experiment 21. Evidence of exclusion was seen again, but in all cases it was incomplete.

Experiment 22. There was evidence of specific exclusion (the intensity of targeted sequences reduced) but the process was incomplete. There seemed to be a basic problem with the method in that it used the enzymatic removal process to target and destroy specific sequences, followed by an incredibly sensitive technique to detect them.

Experiment 32. The PCR failed to produce any product from any of the samples, including the positive control. This was probably due to loss of the template during the washing steps, reducing its concentration below the limit of detection. There was no time to repeat this final experiment.

5.7 Implications of results

In this section we describe the lessons to be drawn from the experimental results just described. The purposes of this section are twofold: firstly, we hope that other experimentalists in the field may be made aware of various subtle aspects of the implementation of models of DNA computation. We have found that the requirements of DNA-based algorithmic experiments are often far stricter than those of "traditional" investigations in molecular biology. For example, it is rare that molecular biologists are required to sequence a heterogeneous population of DNA strands, yet, for any non-trivial problem, this task is inevitably required as the final step of the implementation of a DNA-based algorithm. We hope that these (often non-obvious) impediments to efficient and error-resistant implementation of models of DNA computation will be made apparent in the following section. The second purpose of this section is to suggest possible refinements to our original implementation, with a view towards completely redesigning the laboratory protocol.

Ensure appropriate control and optimisation experiments are performed

We quickly found that a major component of the work was comprised of finding optimal experimental conditions. Factors to be taken into account included strand concentration, salt concentration, restriction enzyme concentration, annealing tem-

perature and number of cycles. Due to the unusual nature of the experiments, we found that the system was far more sensitive to experimental conditions than is normally the case.

We also carried out extensive control experiments (Experiment 6) to ensure the specificity of the PCR detection step (i.e., to ensure that strands were not removed without this being done explicitly). Also, control experiments 14-15 proved the inefficiency of the *Sau*3A restriction enzyme, and quickly told us to use *Mbo*I (this is described in detail in a later section).

Ensure that the initial library is constructed cleanly before proceeding

A fundamental prerequisite for correct algorithmic implementation is that the initial library of strands be constructed as expected. This is especially important for algorithms within filtering models, since we must be absolutely sure that every possible solution to the given problem is represented as a strand. Whilst describing their attempt to recreate Adleman's experiment, Kaplan *et al.* [46] acknowledge the difficulty of obtaining clean generation of the initial library.

There are several potential problems inherent to the construction of an initial library by the annealing and ligation of many small strands. Incomplete or irregular ligation can result in shorter than expected strands. We check for this in Experiment 3, and observe that the majority of the product is of the expected length.

Since our 3COL algorithm operates within a filtering model, it is essential to ensure that all possible colourings are represented with roughly equal probability. Although it is not necessary to ensure a *completely* even distribution of colourings in solution, it is important that there is no significant bias towards some rather than others. In addition to checking the length of the product, we rigorously ensure that

there is sufficient variability within the initial library by cloning a sample into *E. coli.* and sequencing their DNA (Experiments 7-8). If we had simply relied upon naive length checking, we would not have spotted the fact that all initial strands were missing the sequences for v_5 and v_6. Apart from this problem, we observed the desired degree of variability.

Correct strand/primer design is vital

In [2] Adleman originally suggested using random sequences to represent vertices within the given graph. He explained this choice by stating that it was unlikely that sequences chosen to represent different vertices would share long common subsequences, and that undesirable features such as hairpin loops would be unlikely to occur. The selection of random sequences was also supported by Lipton in [52].

Since the publication of [2] and [52], the use of random sequences has been called into question [10, 58]. It is clear that for any non-trivial problem, careful thought must go into the design of sequences to represent potential solutions if we are to avoid the problems described above. As stated earlier, as the project progresses we intend to implement a software package to automate this process.

Once we had designed our sequences by hand, we checked them with the Microgenie [65] sequence analysis package for common subsequences and hairpin loops. Unfortunately, this package is now acknowledged to be rather dated. and it failed to alert us to several potential problems with the sequences. The most significant problem was that the first half of the v_4 sequences was identical to the first half of the v_6 sequences, and likewise for v_5 and v_7 sequences. This was leading to the formation of shorter than expected strands. with sequences representing v_5 and v_6 missing. The cause of this problem is unknown. though we believe it was

Colouring	*Sequence*	T_m
$v_3 = red$	AGCATGGTGATCCTATGACGGAAGGCGTAATGCTGCTAAGACGAAGAGTT	83.4
$v_3 = green$	AGCATGGTGATCCTATGACGAGGACAAGTGTGCTGCTAAGACGAAGAGTT	83.0
$v_3 = blue$	AGCATGGTGATCCTATGACGCTTCTGTAGCTGCTGCTAAGACGAAGAGTT	82.2

Table 5.5: Redesigned sequences

probably due to the fact that the sequences were designed and (more importantly) transcribed by hand. It is possible that we would not have noticed this problem had we not carried out careful checking procedures (Experiments 7-8).

The second major problem was due to the sequences chosen to represent vertex colours. We made a completely arbitrary decision to represent "red" by *AAAAAA*, "green" by *CCCCCC* and "blue" by *GGGGGG*. In retrospect, it is clear that this was a bad choice, for two main reasons. The first concerns the red and blue primers. It is clear that, in solution, these primers are complementary, and are just as likely to anneal to one another as they are to the target sequences. Obviously, this will greatly reduce the efficiency of the removal operation. The second problem concerns the melting temperatures of the primers. Because the T_m of the red primers was far lower than that of the green and blue primers, we observed incomplete removal of red sequences (Experiment 20). As a result of these problems, we redesigned the strands, the modifications being detailed in the description of Experiments 23-24.

Given the cost of redesigning and reordering every oligo, we decided to concentrate instead on the v_3 sequence. We believe that once we can show that the removal procedure works reliably for a single vertex then it will be applicable to all vertices, once the appropriate strands have been redesigned. The new sequences are given in table 5.5.

PCR can introduce problems

It is unrealistic to assume that our enzymatic method removes 100% of the targeted strands. We must therefore be prepared to accept that a small proportion of target strands will be left in solution. Normally, this residue would be undetectable, but the repeated use of PCR can quickly amplify this trace amount, causing failure of the algorithm being implemented. Experiment 22 confirmed that our removal method worked, but the use of PCR as a detection method was far too sensitive for our purposes. Kaplan *et al.* [47] confirm our belief that PCR is a major source of errors.

Biotinylated strands can introduce problems

Quite apart from the problems with biotinylation described in section 4.4, it became clear from our investigations that this can cause other significant difficulties. We found that the attached beads "settled out" in solution, dragging strands to the bottom of the PCR block and affecting the efficiency of the process. We attempted to overcome this problem in Experiment 19 by incubating the tubes in a rotating oven.

Restriction enzymes are often not as effective as they are claimed to be

Although various claims are made for the efficiency of restriction enzymes, in reality they have a non-zero error-rate associated with them. We found that *Sau*3A was completely ineffective at cleaving double-stranded DNA (Experiment 13 , but *Mbo*I worked perfectly well (Experiments 14-15). This may have been due to the fact that *Sau*3A is inefficient at cleaving *de novo* synthesised DNA.

Selection is non-trivial

A second major problem is that of final read-out: for any non-trivial combinatorial problem, we would expect the final tube to contain a heterogeneous population of strands, each encoding a final solution. Traditional sequencing techniques are useless in this situation, since they rely upon the homogeneity of the sample. Techniques such as *nested PCR* are of limited use, due to the potentially large number of different strands in the population. Unfortunately, this problem has been largely ignored in the literature, and the few papers reporting empirical results have described read-out procedures of limited use (e.g., Adleman's graduated PCR approach). Such techniques are only useful for small control experiments, since the use of PCR is inherently error-prone.

5.8 Redesign of implementation

It is clear that there are several flaws in the experimental approach described in section 5.3. However, these problems only came to light after extensive and rigorous control experiments had been carried out, so we are confident that the major impediments to effective implementation of our model have been identified. We now describe a refinement of our laboratory implementation which we believe will yield successful results.

As stated in the previous section, the fundamental problem with our approach is that we use the enzymatic removal technique followed by an incredibly sensitive detection step (PCR). Since it is unrealistic to assume that the enzymatic removal step is 100% specific, we must desensitize the system so that single "missed" strands are undetectable, *without desensitizing to such a degree that we lose exper-*

imental precision.

As we have already stated, the final read-out stage is crucial. What is required is a *general* read-out procedure that may easily be applied within any experimental implementation, regardless of the model of computation. In this section we describe such an approach that carries the additional benefit of removing the sensitive PCR technique from the implementation.

5.8.1 The cloning implementation

In this section we describe a *cloning* method that we believe will provide an effective and error-resistant method of final read-out. The fundamental difference between this and previous approaches is that we use M13 DNA to encode solutions, rather than synthesised DNA. We may perform removal, restriction, sorting by gel electrophoresis, etc. on these strands as usual, but, unlike synthesised DNA, they may then be transfected into *E. coli* prior to the read-out stage. This allows us to pick individual clones and sequence their DNA, revealing the final result, without relying upon PCR for detection.

The new implementation comprises three main stages (depicted in figure 5.7):

1. Set-up
2. Computational
3. Read-out

Set-up stage

The initial library of strands is constructed exactly as described in Experiment 2. The strands are designed such that they can be ligated into a double stranded DNA

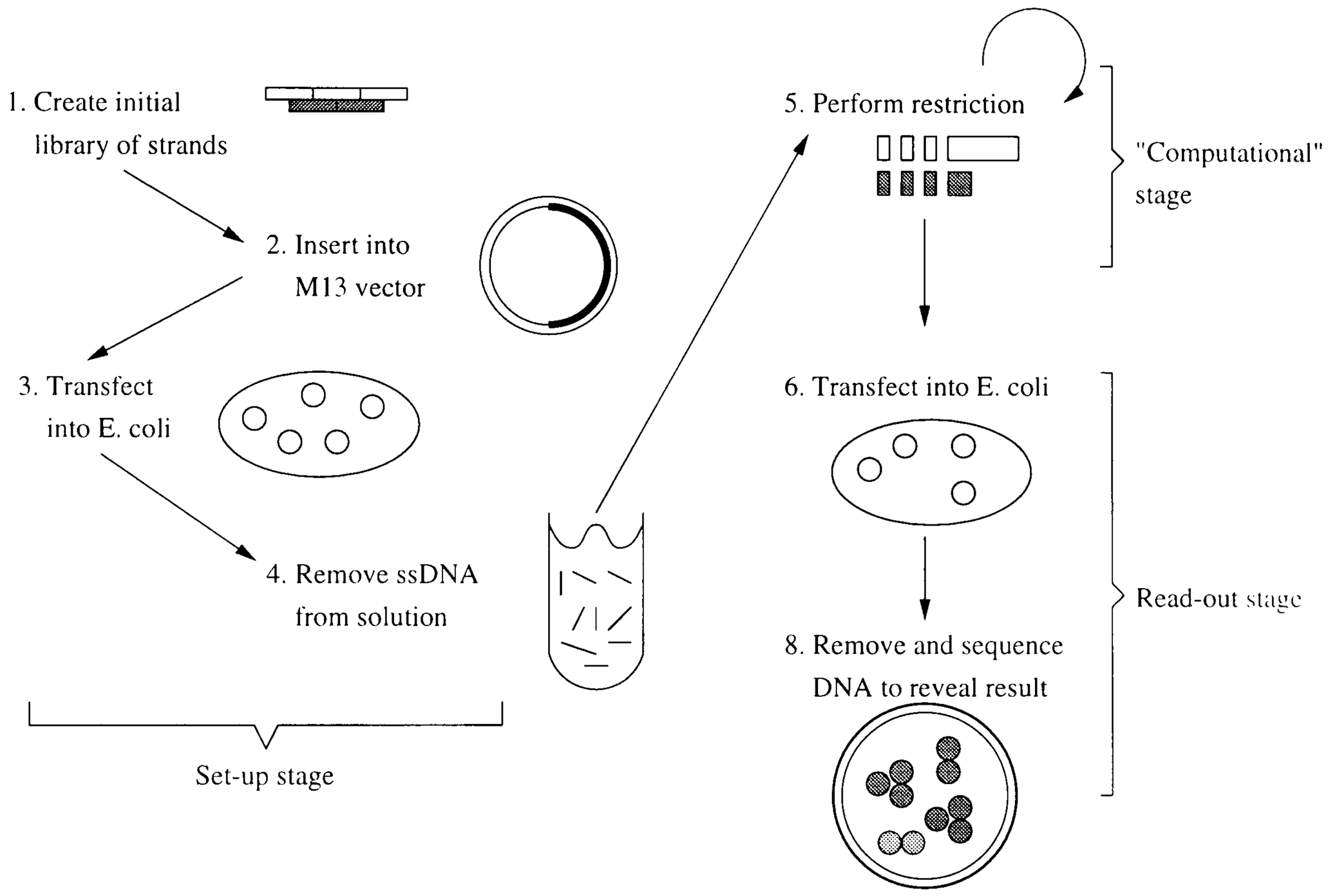

Figure 5.7: Summary of the new cloning approach

Colouring	*Sequence*	T_m
v_1	GCTCTGCTGAGCTCTCTTGATTTCACAGCATGGT	76.5
$v_3 = red$	AGCATGGTGATCCTATGACGGAAGGCGTAATGCTGCTAAGACGAAGAGTT	83.4
$v_3 = green$	AGCATGGTGATCCTATGACGAGGACAAGTGTGCTGCTAAGACGAAGAGTT	83.0
$v_3 = blue$	AGCATGGTGATCCTATGACGCTTCTGTAGCTGCTGCTAAGACGAAGAGTT	82.2
v_8	GCGGAATTCCTCTGCAGATCTTTTTTAAGCAAAGGTCTTGTCAGTG	80.3

Table 5.6: Redesigned sequences for cloning procedure

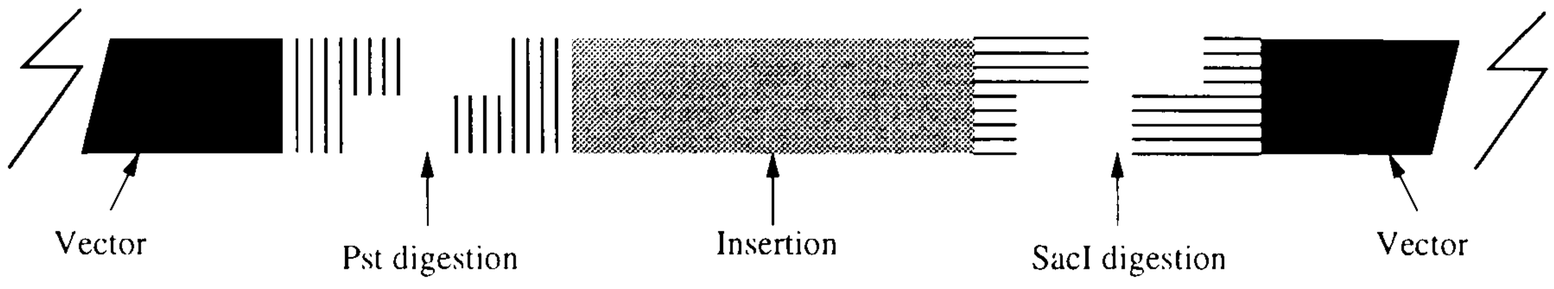

Figure 5.8: Insertion of library strands into M13 vector

vector (M13mp18) which, upon DNA transfection of a suitable bacterial host (*E. coli.*, generates single stranded DNA bacteriophage clones, each of which encodes a single solution. In order to perform the ligation, we redesign the v_1 and v_8 sequences to include a unique restriction site ($GAGCTC$ (SacI) for v_1 and $CTGCAG$ (PstI) for v_8). It is clear that the use of two different restriction enzymes prevents the formation of long chains containing multiply repeated sections. The new sequences for v_1 and v_8 are listed in table 5.6. For the purposes of testing the procedure, we do not require there to be colour sequences within the v_1 and v_8 sequences (recall that in section 5.7 we propose testing our new approach by removing only on v_3).

We perform restrictions on the library and vector strands. We then mix the two together and allow ligation. The ligation is depicted in figure 5.8. The vectors are then introduced into *E. coli* as described in section 2.3.7.

Each clone is represented as a single bacteriophage plaque (a zone of infec-

tion) on a lawn of the bacterial host. Tens of thousands of plaques can be present on a single lawn of bacteria and a hundred such lawns can be prepared from a single ligation. Suitable dilutions are used to prevent plaques from overlapping, thus ensuring that only a single vector is introduced into each bacterium. Note that this method provides an effective mechanism for *reliably* producing many copies of the target strands, since the error-prone PCR method is not used.

The plaques are pooled and single stranded DNA isolated as described in section 2.3.7. These single strands provide the substrate for oligonucleotide primer annealing, subsequent DNA polymerase extension of double strands and restriction enzyme digestion of any such double stranded DNA. This removes any illegal solutions, as described previously, and constitutes the *computational stage.*

Read-out stage

We perform the final read-out by transfecting *E. coli* with the remaining strands. Since only intact bacteriophage DNA molecules are capable of transfecting the host bacterium, we can be confident that any clones produced contain sequences representing legal solutions to the problem. The final collection of plaques are individually picked, then the DNA is isolated and subjected to standard DNA sequencing reactions in order to obtain the solution to the problem.

5.9 Summary

In this chapter we described the results obtained from preliminary investigations of the implementation of our parallel filtering model. We decided, at this stage, to demonstrate the efficacy of our enzymatic removal technique before attempting a full algorithmic implementation. The results obtained were ambiguous, though

promising. More importantly, we have, though our own investigations, identified several potential impediments to effective implementation of all existing models of DNA computation. These problems only came to light due to the fact that we performed numerous control and optimisation experiments, an approach that seems to be unique in the literature. We hope that the community will be able to draw lessons from our investigations, and that this will aid in the development of improved laboratory protocols in the future. In particular, we highlighted problems with final read-out of solutions after the termination of a DNA-based algorithm, and described a novel *cloning* approach that is general enough to be employed within any experimental implementation.

Chapter 6

An analysis of DNA computation

This chapter addresses questions concerning the complexity and viability of DNA computations. The work described was presented at the *International Conference on Bio-Computing and Emergent Computation* [6].

6.1 Introduction

In this chapter we examine complexity issues in DNA computation. We believe that these issues are paramount in the search for so-called "killer applications", that is, applications of DNA computation that would establish the superiority of this paradigm over others in particular domains. An assured future for DNA computation can only be established through the discovery of such applications. We demonstrate that current measures of complexity fall short of reality. Consequently, we define a more realistic model, a so-called *strong* model of computation which provides better estimates of the resources required by DNA algorithms. We also

compare the complexities of published algorithms within this new model and the weaker, extant model which is commonly (often implicitly) assumed.

6.2 Motivation

Following the initial promise and enthusiastic response to Adleman's seminal work [2] in DNA computation, progress towards the realisation of worthwhile computations in the laboratory has become stalled. One reason for this is that the computational paradigm employed by Adleman, and generalised by the theoretical work of others [52, 66], relies upon filtering techniques to isolate solutions to a problem from an *exponentially sized* initial solution of DNA. This volume arises because all possible candidate solutions have to be encoded in the initial solution. As Hartmanis points out in [42], the consequence is that, although laboratory computations should work for the smallest problem sizes, the experiments do not realistically scale because vast amounts of DNA are required to initiate computations with even modest problem size. For example, Hartmanis shows that a mass of DNA greater than that of the earth would be required to solve a 200-city instance of the Hamiltonian Path Problem.

If practitioners of DNA computation insist on this mode of computation, there can be no hope of discovering so-called *killer applications*, that is, applications of DNA computation that would establish the superiority of this paradigm over others in particular domains. An assured future for DNA computation can only be established through the discovery of such applications.

It is not inherently the case that exponentially sized volumes of DNA need be used in DNA computation. Indeed, polynomially sized computations have been (at least in theory) described (e.g., in [61]). Clearly, if exponentially sized volumes

are to be avoided, then an alternative algorithmic paradigm to that employed by Adleman in [2] is required. Such a successful paradigm is always likely to emulate traditional computations which *construct* individual solutions rather than sift them out of a vast reservoir of candidates. It might still be argued that the "exponential-curse" could not, even then, be avoided for the so-called NP-complete problems [31]. If an exact solution is required for any of these, then (employing any extant algorithm) exponential sequential running time is required. A DNA computation, in seeking to reduce this to sub-exponential parallel running time, will certainly require an exponential volume of DNA. However, in general, no-one sensibly seeks exact solutions to the NP-complete problems. In traditional computation, we either employ heuristics to obtain approximate answers or use randomised methods to obtain exact solutions with high probability. These revised algorithmics lead to solutions within polynomial sequential time. Such a view should also be taken for these problems within DNA computation, that is, we should use algorithms which do not inherently require exponential resources.

It is unlikely to be enough, in the quest for killer applications, to simply have polynomial-volumed computations. We ought, at the same time, to ensure that the vast potential for parallelism is employed to obtain rapid computations. The view taken by the silicon-based parallel computing community [32] is that efficient parallel algorithms, within the so-called Parallel Random Access Machine (P-RAM) model of computation, should have polylogarithmic running time (and use a polynomial number of processors). Problems for which such solutions exist define the complexity class NC. If DNA computation is to compete within this domain, then we should clearly also look for polylogarithmic running times within polynomially-volumed computations.

At the present time, no-one has described (even theoretically) DNA computations which run in polylogarithmic time using a polynomial volume of DNA. The discovery of such solutions might well provide candidates for "killer applications". Regardless of the problem considered, it is unlikely to provide a "killer application" unless the computational resources required for a DNA computation (the product of the running time and volume of DNA required) match those needed for a conventional computation (the product of the running time and the number of processors used). For such a combination of resources, the DNA computation might well provide feasible solutions for problem sizes far greater than can be achieved by conventional computation.

It is clearly crucial, especially when judging the candidacy of a proposed DNA computation for the role of "killer application", to have a firm grasp of the computational resources that it requires. In this chapter we review claims that have been made concerning the complexity of DNA algorithms. We conclude that these claims are often unrealistic, or simply not true. It also the case that there is not an agreed model of computation in the literature within which we may agree what the required resources are for any particular computation. This chapter attempts to address these issues in a realistic way.

Traditional computational complexity theory [4, 31] is concerned with quantifying the resources (generally *time* and *space*) needed to solve computational problems. Meaningful analysis of the complexity of algorithms may only take place in the context of an agreed model of computation, or *machine model.* Many different machine models have been proposed in the past, including the Deterministic Turing Machine, Boolean circuit [25, 41] and P-RAM [28, 32]. The nascent field of DNA computing also suffers from the problem of proliferation of machine models. Several

models have been proposed, within which we may construct algorithms for the solution of computational problems. However, complexity analyses of algorithms within different models of DNA computation are meaningless, since there are no uniform definitions of the concepts of time and space. Furthermore, if we are to compare a DNA-based model with a more traditional machine model, we require a way of demonstrating equivalence between the two.

In this chapter we analyse the complexities of algorithms within a commonly employed model of DNA computation. We argue that this model, which we call the *weak model*, is actually inadequate from the point of view of obtaining *realistic* complexity results. This leads us to define a new *strong model* within which we reassess some claims that have been made concerning complexity of computations.

The chapter is organised as follows. In section 2 we recall the *weak model* of DNA computation first explicitly described in [7] although not so named there. We explain the shortcomings of this model and introduce the *strong model*, allowing us to make meaningful comparisons between DNA-based and more traditional models of DNA computation. In section 3 we discuss assumptions made about certain fundamental operations within models of DNA computation. We argue that such assumptions are false, and demonstrate the implications for the complexity analysis of various extant models. In section 4 we review the current search for the "killer application"; the one application that will establish a niche for DNA-based models of computation. We argue that the basis for such a quest is flawed, and suggest a potentially more fruitful line of enquiry in the light of the strong model.

6.3 Weak and strong models

Attempts have been made to characterise DNA computations using traditional measures of complexity, such as time and space. Such attempts, however, are misleading due to the nature of the laboratory implementation of the computation. We first examine these algorithms from a time complexity standpoint. Most extant models quantify the time complexity of DNA-based algorithms by counting the number of "biological steps" required to solve the given problem. Such steps include the creation of an initial library of strands, separation of subsets of strands, sorting strands on length, chopping and ligating strands.

Within these models, operations such as those described above are considered to be atomic actions performed in constant time. This assumption is patently false. In this section we rigorously define the time complexity of various laboratory operations, so that an accurate assessment of various DNA-based algorithms may be made.

6.3.1 The weak model

Here we recall [7] the basic legal operations on sets within what we now refer to as the *weak* model. The operation set described here is constrained by biological feasibility, but all operations are currently realisable with current technology.

- $remove(U, \{S_i\})$. This operation removes from the tube U, in parallel, any string which contains at least one occurrence of any of the substrings S_i.

- $union(\{U_i\}, U)$. This operation, in parallel, creates the tube U which is the set union of the tubes U_i.

- *copy*$(U, \{U_i\})$. In parallel, this operation produces a number of copies, U_i, of the tube U.

- *select*(U). This operation selects an element of U uniformly at random, if U is the empty set then *empty* is returned.

From the point of view of establishing the parallel time complexities of algorithms within the model, these basic operations are assumed to take constant time. This assumption has been commonly made by many authors in the literature [7, 52, 61]. However, these operations are frequently implemented in such a way that it is difficult to sustain this claim. For example, the *union* operation consists of pouring a number of tubes into a single tube, and this number is usually, in some way, problem size dependent. Assuming that in general we have a single laboratory assistant, this implies that such operations run in time proportional to the problem size.

Obviously, in the general case, a single laboratory assistant may not pour n tubes into one tube in parallel, nor may s/he split the contents of one tube into n tubes in parallel. This observation, if we are to be realistic in measuring the complexity of DNA computations, requires us to introduce the following constant time atomic operation:

- *pour*(U, U'). This operation creates a new tube, U, which is the set union of the tubes U and U'.

As we have observed, the *pour* operation is a fundamental component of all compound operations. It therefore follows that more realistic analyses of the time complexities of algorithms may be obtained by taking this operation into consideration.

6.3.2 The strong model

In what follows we refine the weak model just described. We assume that the initial tube (which takes at most linear time to construct) is already set up.

The *pour* operation is fundamental to all compound operations within our weak model. We must therefore reassess the time complexity of these operations. The *remove* operation requires the addition to U of

1. i tubes containing primers, and

2. A tube containing restriction enzymes

This operation is inherently sequential, since there must be a pause between steps 1 and 2 in order to allow the primers to anneal correctly. Therefore, the *remove* operation takes $O(i)$ time. Creating the *union* of i tubes is an inherently sequential operation, since the technician must first pour U_1 into U, then U_2, and so on, up to U_i. Rather than taking constant time, the *union* operation actually takes $O(i)$ time. It is clear that the *copy* operation may be thought of as a reverse-*union* operation, since the contents of a single tube U are split into many tubes, $\{U_i\}$. Therefore, *copy* takes $O(i)$ time.

6.4 Complexity comparisons in the weak and strong models

In this section we compare time complexities for algorithms previously described [7] within both the weak and strong models. In particular, we examine in detail the problem of generating a set of permutations. This will characterise the general form of comparisons that can be made, so that we merely tabulate comparisons for other

algorithms (table 6.1).

- **Problem: Permutations**

  Generate the set P_n of all permutations of the integers $\{1, 2, \ldots . n\}$.

- **Solution**

  - *Input*: The input set U consists of all strings of the form $p_1 i_1 p_2 i_2 \ldots p_n i_n$ where, for all j, p_j uniquely encodes "position j" and each i_j is in $\{1, 2, \ldots, n\}$. Thus each string consists of n integers with (possibly) many occurrences of the same integer.

  - *Algorithm*

    for $j = 1$ to $n - 1$ **do**

    begin

    copy$(U, \{U_1, U_2, \ldots, U_n\})$

    for i=1, 2, ... , n and all $k > j$

    remove$(U_i, \{p_j \neq i, p_k i\})$

    union$(\{U_1, U_2, \ldots, U_n\}, U)$

    end

    $P_n \leftarrow U$

  - *Complexity*: $O(n^2)$ parallel-time.

In [7] the authors claimed a time complexity for this algorithm of $O(n)$. We justify the new time complexity of $O(n^2)$ as follows: at each iteration of the **for** loop we perform one *copy* operation. n *remove* operations and one *union* operation. The *remove* operation is itself a compound operation, consisting of $2n$ *pour* operations. The *copy* and *union* operations consist of n *pour* operations.

Algorithm	*Weak*	*Strong*
Three colouring	$O(n)$	$O(n^2)$
Hamiltonian path	$O(1)$	$O(n)$
Subgraph isomorphism	$O(n)$	$O(n^2)$
Maximum clique	$O(n)$	$O(n^2)$
Maximum independent set	$O(n)$	$O(n^2)$

Table 6.1: Time comparisons of algorithms within the Weak and Strong models

Similar considerations cause us to reassess the complexities of the algorithms described in [7], according to table 6.1.

Although we have concentrated here on adjusting time complexities of algorithms described in [7], similar adjustments can be made to other work. An example is given in the following section.

6.5 Analysis of the Boolean circuit model

The authors of [61] claim real-time simulation of the class NC^1 [63] in time proportional to the depth of the circuit. Recall that NC^1 defines the class of problems of size n solved by bounded fan-in circuits of $O(\log n)$ depth and polynomial size. We point out that, with a single laboratory assistant, this estimate of the time complexity should be proportional to the *size* of the circuit. Thus, the claim of polylogarithmic running time (with, incidentally, a polynomial volume of DNA) in the weak model translates to polynomial running time in the strong (realistic) model. Essentially, the simulation does not harness the massive potential for parallelism that DNA offers. We now justify this claim.

Ogihara and Ray claim that set-up "requires only one step". Given the inherent sequentiality of the *pour* operation, this statement is clearly false. An

n-input Boolean circuit with $O(1)$ set-up time requires n technicians. We now consider the simulation of gates at level $l > 0$. We omit detailed discussion of the implementation (described in section 4.7), and concentrate purely on the number of *pour* operations required at each level. Let $i_1, i_2, \ldots, i_a$ be the indices of gates at level $l-1$, and $j_1, j_2, \ldots j_b$ those of the gates at level l. After amplifying the contents of the working tube, for each s, $1 \leq s \leq b$ the operation $pour(U, \sigma[j_s])$ is executed. This takes s time. Now U contains many copies of the strand representing each gate at level l. In order to simulate the operation of some gate G_s, two "linker" strands, representing the inputs to G_s are poured. This takes $2s$ time. Ogihara and Ray claim that they only require $O(d)$ ligation steps during the course of the simulation. However, we believe that *in the general case* it is more meaningful to talk in terms of $O(m)$ *pour* operations, even if we discount set-up time.

Despite these remarks, Ogihara and Ray's work is important because it establishes the Turing-completeness of DNA computation. This follows from the work of Fischer and Pippenger [27] and Schnorr [70], who described simulations of Turing Machines by combinational networks. Although a Turing Machine simulation using DNA has previously been described by Reif [66], Ogihara and Ray's method is simpler, if less direct.

6.6 Summary

In this chapter we have emphasised the rôle that complexity considerations are likely to play in the identification of "killer applications" for DNA computation. We have examined how time complexities have been estimated currently within the literature. We have shown that these are often likely to be inadequate from a realistic point of view. In particular, many authors implicitly assume that arbitrarily large numbers

of laboratory assistants are available for the mechanical handling of tubes of DNA. This has often led to serious under-estimates of the resources required to complete a computation.

We have proposed a so-called *strong* model of DNA computation, which we believe allows *realistic* assessment of the time complexities of algorithms within it. This model, if the *splice* operation is trivially included, not only provides realistic estimates of time complexities, but is also Turing-complete.

We believe that success in the search for "killer applications" is the only means by which there will be sustained interest in DNA computation. Success is only a likely outcome if DNA computations can be described which will require computational resources of similar magnitude to those required by conventional solutions. At present, we believe that no realistic estimates of time complexities of DNA computations have been made, despite the claims of some authors. However, if, for example, we were to establish polylogarithmic time computations using only a polynomial volume of DNA, then this would be one scenario in which "killer applications" might well ensue. In this case, we might imagine that the vast potential for parallelisation would yield feasible solutions to very much larger problem sizes than could be achieved using existing, silicon-based parallel machines.

Chapter 7

Conclusions

This doctoral thesis is the first ever to be written solely on the subject of DNA-based computation. As such, it is appropriate to address a broad spectrum of issues and problems. Although several of these issues remain unresolved, we hope that the work presented provides clarification and suggests potentially fruitful lines of enquiry in the future.

In chapter 2 we presented a comprehensive review of extant models of DNA computation, in the course of which we described the novel *parallel filtering* model. This model provides a generalisation of the motivating work of Adleman and Lipton, and facilitates the elegant expression of algorithms for a variety of NP-complete problems. We described several such algorithms within the parallel filtering model. We also described the work of Reif and others, who show how the addition of a *splicing* operation to existing models of DNA computation provides full Turing-computability. We then described a model for the DNA-based simulation of Boolean Circuits, due to Ogihara and Ray.

In chapter 4 we considered the practical implementation of models of DNA

computation. In particular, we showed how the models described in chapter 2 may be implemented in the laboratory, concentrating in particular on the implementation of the parallel filtering model. We point out practical difficulties in implementing extant models and argue that our proposed implementation of the parallel filtering model provides a greater degree of error-resistance than those previously described.

In chapter 3 we explained the structure of the DNA molecule, and described most of the common laboratory techniques for its manipulation. This review provides a foundation for subsequent chapters. In particular, we concentrated on the *cloning procedure*, since it is fundamental to the work presented in the second half of chapter 5.

At this time, the empirical basis for being optimistic about the future of DNA computation is rather slim. Much more work needs to be done to develop error-resistant and scalable laboratory computations. This implies improved empirical techniques and algorithmic designs. The pressure to develop more efficient and accurate laboratory techniques will not only benefit researchers in DNA computation, but will also yield positive "spin-off" benefits for molecular biologists. As many authors have demonstrated, it is not difficult to design algorithms within a variety of models which, on a naive basis, appear to be realisable. The fact is that it is a difficult matter to design experiments that are likely to be successful in the laboratory. In chapter 5 we described the results of a preliminary implementation of the parallel filtering model. Although early results are promising, the main contribution of this chapter is to highlight several potential difficulties in implementing models of DNA computation. We describe a *cloning* approach that removes reliance upon error-resistant techniques such as PCR and hybridisation extraction, and allows easy final read-out of solutions to the given problem. This technique is

sufficiently general to allow it to be integrated into any future implementation.

In chapter 6 we considered the complexity and viability of DNA computations. Such issues have, to date, been largely underestimated in the literature. We argue that existing analyses of models of DNA computation are flawed and unrealistic. In order to obtain more realistic measures of the time and space complexity of DNA computations we describe a new *strong* model, and reassess previously described algorithms within it. An important area of enquiry that is outstanding is the quest for algorithms which proceed through polynomial-sized volumes of DNA. In principle, this ought to be possible for problems which can be solved by conventional computers in a time which is polynomial in the problem size. Success in this area would give real hope for solving problems with very large problem size through the exploitation of the potentially massive parallelism offered by the biochemistry of DNA. Such applications would therefore constitute "killer applications": applications of DNA computing that will establish the superiority of this paradigm within a certain domain.

We have therefore identified the following open problems in the field of DNA computation:

- Does there exist a class of problems for which DNA computers will out-perform conventional, silicon-based machines?

- Can methods be developed to reduce the error-rate of commonly-used biological operations?

- Can the techniques and models developed be integrated into a general- purpose computer?

- Can naturally-occurring biological processes suggest novel algorithmic tech-

niques and approaches?

Although we believe that there is great potential for DNA computing, the field is still at a very early stage. It is unlikely that DNA computers will be used for tasks like word processing, but they may ultimately find a niche for solving large-scale intractable combinatorial problems. Although the basic operations required for DNA computers already exist, the problem of automating, miniaturising and integrating them into a general-purpose desktop DNA computer is unlikely to be solved in the near future. We hope that this thesis contributes to this long-term goal by stimulating realistic investigations in the future.

Appendix A

Laboratory Protocol

A.1 Initial library construction

Resuspend oligos in distilled water to a final stock concentration of 100pmol/μl.

Prepare a stock mixture containing all 24 oligos at 2.5pmol/μl each. Prepare hybridisation mixture containing:

- 2μl oligo mix (5pmol, $\sim 3 \times 10^{12}$ copies)
- 10μl 5 $\times$ Polig buffer
- 34μl distilled water

5 $\times$ Polig buffer:

- 250mM Tris.Cl pH 7.8
- 450mM KCl
- 25mM $MgCl_2$
- 50mM $(NH_4)2SO_4$

- 20mM DTT

In a PCR block (with heated lid) denature at 95°C 5 minutes, then allow to cool at 2 °C/min to 52°C and continue to incubate at 52°C for 1 hour. Chill on ice then add:

- 1μl E.coli DNA polymerase I (10U)
- 1μl E.coli DNA ligase (10U)
- 1μl 10mM dNTPs
- 1μl 20mM B NAD

Incubate for 2 hours at 16°C. Store at -20°C.

A.2 Amplification of hybridisation product

Dilute hybridisation product $\frac{1}{10}$ in water. Use this as the PCR template. Set up PCR reaction:

- 10μl 10 $\times$ PCR buffer
- 4μl 50mM $MgCl_2$
- 30pmol $v_1 = red$
- 30pmol $v_1 = green$
- 30pmol $v_1 = blue$
- 30pmol $\overline{v_8}$
- 2μl 10mM dNTPs

- 1μl template

- 5U *Taq* polymerase

Cycle at: 94°C 2'(95°C 15s, 48°C 15s, 72°C 30s) × 35, 72*4'. Purify major band of ~200bp on 2% agarose and extract using QIA kit. Elute in 50*l water.

A.3 Capture of biotinylated product

2 × Binding and washing (2 × B&W) buffer:

- 10mM Tris.Cl pH 7.5

- 1mM EDTA

- 1M NaCl

Wash 20μl Dynabeads with 20μl of 2 × B&W buffer. Resuspend in 20μl B&W buffer. Add 1μl from a $\frac{1}{10}$ dilution of PCR product (ie. a maximum of 0.06 pmol) plus 19μl water to the washed beads. Incubate for 15 minutes at room temperature, keeping the beads in suspension by gently tipping the tube.

A.4 Melting the DNA duplex

Using a magnet, capture the beads and remove the supernatant. Wash the beads once in 200*l 1 × B&W. Resuspend the beads in 50*l freshly prepared 0.1M NaOH and incubate at room temperature for 5 minutes. Collect the beads and remove the supernatant. Wash the beads once with 200μl 0.1M NaOH, then three times with 200*l 1 × *Taq* PCR buffer.

A.5 Removal of illegal sequences

Resuspend the beads in 15*l 10 × *Taq* PCR buffer and 45μl water then split the sample into three. Add 500pmol of the appropriate primers to each tube. (a template:primer ratio of roughly 1:8000). Add $MgCl_2$ to 2mM, dNTPs to 200uM, 5U *Taq* polymerase and make each tube up to 50μl with water. In a heated lid PCR block cycle at 95°C 1' (45°C 2', 72°C 1') × 30, 72°C 10'. Resuspend the beads about every 5 cycles. Wash the beads three times in 200*l 1 × REact 2 buffer, then resuspend in 50μl 1 × REact 2 buffer plus 50U MboI. Incubate O/N at 37°C in a rotating oven to keep the beads in suspension. Wash the beads three times in 200μl 1 × *Taq* buffer then resuspend in 50μl 1 × *Taq*. For subsequent rounds of exclusion repeat, using appropriate primers.

A.6 Identification of result

Using an aliquot of the beads (1-5μl) as template amplify between one of the three coloured primers for v_1 and the $\overline{v_8}$ primer.

Basic PCR mix:

- 5μl 10 × PCR buffer
- 2mM $MgCl_2$
- 30pmol specific colour primer
- 30pmol $\overline{v_8}$ primer
- 200uM dNTPs
- 1-5μl template

- 5U *Taq*
- water to 50μl

Cycle at 94°C 2',(94°C 15s, 48°C 15s, 72°C 30s) × 30, 72°C 5'. Run 20μl of the PCR product on 2.5% agarose. A positive result indicates that the sequence was present in the mix.

Appendix B

Glossary

Affinity purification See **hybridisation extraction**.

Aliquot Small sample taken from a larger solution.

Annealing The joining of two **complementary** strands of **DNA**, forming a double strand. The opposite of **denaturing**.

Bacteriophage (abbrev. *phage*) A virus that infects bacteria.

Base Organic compound found in nucleic acids. The four bases found in **DNA** are adenine (A), guanine (G), cytosine (C) and thymine (T).

Biotinylation The attachment of magnetic beads (Dynabeads) to the end of a single **strand** of **DNA**.

Cloning The insertion of a **DNA** sequence into a **vector**. After transfer of this recombinant molecule into a suitable host (e.g., *E. coli*). the resulting colony consists of cells that all contain the recombinant **DNA** molecule.

Complement, complementary See **Watson-Crick complement**.

DNA (deoxyribose nucleic acid). The genetic material of all cellular organisms. Two **complementary** single **strands** of DNA form the famous double-helix structure.

Denaturing The heating of a solution of double-stranded **DNA** to break the hydrogen bonds joining them, yielding single **strands**. The opposite of **annealing**.

Digestion The cleaving of double-stranded **DNA** by a **restriction enzyme**.

E. coli Bacterium commonly used in **cloning** experiments.

Extension The addition of **nucleotides** to a **primer**. Directed by **DNA polymerase**.

Gel electrophoresis A process by which a solution of **strands** of **DNA** is sorted according to length.

Hybridisation extraction Technique for removal from solution of **strands** containing a specific sequence. **Biotinylated primers complementary** to the target sequence are added to the solution, which then **anneal** to **strands** containing the sequence. A magnet is then used to draw out the beads with the targeted **strands** attached to them.

Killer application An application of a particular mode of computation that establishes its superiority over others in a particular domain.

Lawn A flat surface (usually a petri dish) covered with a bacterial culture.

Ligase An enzyme that catalyzes **ligation**.

Ligation The joining of two linear molecules of **DNA**.

M13 A particular **bacteriophage** commonly used in **cloning** experiments.

NC Nic's Class. Named after the computer scientist Nic Pippenger. Efficient parallel algorithms, within the so-called Parallel Random Access Machine (P-RAM) model of computation, should have polylogarithmic running time (and use a polynomial number of processors). Problems for which such solutions exist define the complexity class *NC*.

NP The class of problems that can be solved in polynomial time by a nondeterministic computer. See also **NP-complete**.

NP-complete A problem is **NP-complete** if it can be reduced in polynomial time to any other problem known to be **NP-complete**. **NP-complete** problems are characterised by an exponential-sized search space, and are therefore referred to as intractable.

Nucleotide The base-sugar-phosphate group found in nucleic acids. See also **base**.

Oligonucleotide A chain of **nucleotides**.

PCR Polymerase Chain Reaction. A technique for massively amplifying an initial sample of **DNA**.

Plaque A region of infection on a **lawn** of bacteria.

Polymerase An enzyme that synthesises a new **strand** of **DNA complementary** to an existing **DNA** template. See also **PCR**.

Primer A short **complementary oligonucleotide** used to initiate synthesis of a new **DNA strand**. See also **PCR, polymerase**.

Restriction enzyme An enzyme that recognises and cleaves double-stranded **DNA** at a specific **restriction site**.

Restriction site A short, specific sequence in double-stranded **DNA** that is recognised and cleaved by a **restriction enzyme**.

Strand See **oligonucleotide**.

Transfection The **transformation** of a bacterium with DNA isolated from a virus (e.g., a **bacteriophage**).

Transformation The introduction of exogenous **DNA** into cells.

Turing machine Named after Alan Turing. A finite state automaton with an unbounded memory. An abstract machine that is used to define the concept of computability. A function is computable if it can be proved that there exists a Turing machine to evaluate it at any given point.

Vector A **DNA** molecule (e.g., a **bacteriophage**) into which foreign **DNA** sequences may be inserted. The vector may then be **cloned** into a suitable host.

Watson-Crick complement Named after the co-discoverers of the structure of **DNA**, James Watson and Francis Crick. Two strands of **DNA** are complementary if one is able to form a perfect hydrogen-bonded duplex with the other, according to the rules of **base** pairing (i.e., A $\leftrightarrow$ T, and G $\leftrightarrow$ C). For example, the sequences AGGCT and TCCGA are complementary.

Bibliography

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

[2] Leonard Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.

[3] Leonard M. Adleman. On constructing a molecular computer. In Baum and Lipton [12].

[4] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley, 1974.

[5] American Mathematical Society. *Proceedings of the Second Annual Meeting on DNA Based Computers*, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science., June 1996. To appear.

[6] Martyn Amos, Alan Gibbons, and Paul E. Dunne. The complexity and viability of DNA computations. In Lundh, Olsson, and Narayanan, editors, *Proceedings of the First International Conference on Bio-Computing and Emergent Computation*, pages 165–173, University of Skövde, Sweden, 1997. World Scientific.

[7] Martyn Amos, Alan Gibbons, and David Hodgson. Error-resistant implementation of DNA computations. In *Proceedings of the Second Annual Meeting on DNA Based Computers* [5]. To appear.

[8] Anon. *Popular Mechanics*, page 258, March 1949.

[9] Eric Bach, Anne Condon, Elton Glaser, and Celena Tanguay. *DNA Models and Algorithms for NP-complete Problems*, pages 290–299. IEEE Computer Society Press, 1996.

[10] Eric B. Baum. DNA sequences useful for computation. In *Proceedings of the Second Annual Meeting on DNA Based Computers* [5]. To appear.

[11] Eric B. Baum and Dan Boneh. Running dynamic programming algorithms on a DNA computer. In *Proceedings of the Second Annual Meeting on DNA Based Computers* [5]. To appear.

[12] Eric B. Baum and Richard J. Lipton, editors. *DNA Based Computers*, volume 27 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science. ISSN: 1052-1798.* American Mathematical Society, 1996.

[13] Donald Beaver. A universal molecular computer. In Baum and Lipton [12].

[14] C. H. Bennett. The thermodynamics of computation – a review. *International Journal of Theoretical Physics*, 21:905–940, 1982.

[15] New England Biolabs. Catalog, 1996.

[16] Dan Boneh, Christopher Dunworth, Richard J. Lipton, and Jiří Sgall. Making DNA computers error resistant. In *Proceedings of the Second Annual Meeting on DNA Based Computers* [5]. To appear.

[17] K. J. Breslauer, R. Frank, H. Blocker, and L. A. Marky. Predicting DNA duplex stability from the base sequence. *Proc. Natl. Acad. Sci.*, pages 3746–3750, 1986.

[18] T. A. Brown. *Genetics: A molecular approach.* Chapman and Hall, 1993.

[19] T.A. Brown. *Gene cloning: an introduction.* Chapman and Hall, second edition, 1990.

[20] B. Bunow. On the potential of molecular computing. *Science*, 268:482–483, April 1995.

[21] Erzsébet Csuhaj-Varjú, R. Freund, Lila Kari, and Gheorghe Păun. DNA computation based on splicing: universality results. In Lawrence Hunter and Teri Klein, editors, *Biocomputing: Proceedings of the 1996 Pacific Symposium.* World Scientific Publishing Co., Singapore, January 1996.

[22] J. H. M. Dassen. Molecular computation and splicing systems. M.Sc. thesis, Leiden University, August 1996.

[23] Dónall A. Mac Dónaill. On the scalability of molecular computational solutions to NP problems. *The Journal of Universal Computer Science*, 2(2):87–95, February 1996.

[24] S. M. Du and N. C. Seeman. The construction of a trefoil knot from a DNA branched junction motif. *Biopolymers*, 34:31–37, 1994.

[25] Paul E. Dunne. *The Complexity of Boolean Networks.* Academic Press, 1988.

[26] Richard P. Feynman. There's plenty of room at the bottom. In D. Gilbert, editor, *Miniaturization*, pages 282–296. Reinhold, 1961.

[27] M. Fischer and N.J. Pippenger. Relations among complexity measures. *Journal of the ACM*, 26:361–381. 1979.

[28] Steven Fortune and James Wyllie. Parallelism in random access machines. In *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing*, pages 114–118, San Diego, California, 1978.

[29] T.-J. Fu, Tse-Dinh Y.-C., and N. C. Seeman. Holliday junction crossover topology. *J. Molec. Biol.*, 236:91–105, 1994.

[30] Tsu-Ju Fu and Nadrian C. Seeman. DNA double-crossover molecules. *Biochemistry*, 32:3211–3220, 1993.

[31] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[32] A. Gibbons and W. Rytter. *Efficient Parallel Algorithms.* Cambridge University Press, 1988.

[33] A. M. Gibbons. *Algorithmic Graph Theory.* Cambridge University Press, 1985.

[34] Alan Gibbons, Martyn Amos, and David Hodgson. Models of DNA computation. In Penczek and Szalas, editors, *Mathematical Foundations of Computer Science (MFCS)*, volume 1113 of *Lecture Notes in Computer Science*, pages 18–36. Springer-Verlag, 1996.

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

[36] David K. Gifford. On the path to computation with DNA. *Science*, 266:993–994, 1994.

[37] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, Reading, MA, 1989.

[38] Larry Gonick and Mark Wheelis. *The cartoon guide to genetics.* HarperPerennial, 1983.

[39] Frank Guarnieri, Makiko Fliss, and Carter Bancroft. Making DNA add. *Science*, 1996. In press.

[40] Z. Guo, R. A Guilfoyle, A. J Thiel, R. Wang, and L. M Smith. Direct fluorescence analysis of genetic polymorphisms by hybridization with oligonucleotide arrays on glass supports. *Nucl. Acids Res.*, 22:5456–5465, 1994.

[41] M.A. Harrison. *Introduction to switching and automata theory.* McGraw-Hill, 1965.

[42] Juris Hartmanis. On the weight of computations. *Bulletin of the European Association For Theoretical Computer Science*, 55:136–138, 1995.

[43] Thomas Head. Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, 49(6):737–759, 1987.

[44] Tom Head. Splicing schemes and DNA. In G. Rozenberg and A. Salomaa, editors, *Lindenmayer systems: Impacts on theoretical computer science, co mputer graphics and developmental biology*, pages 371–384. Springer-Verlag, 1992.

[45] D.A. Jackson, R.H. Symons, and P. Berg. Biochemical method for inserting new genetic information into DNA of simian virus 40: circular SV40 DNA molecules containing lambda phage genes and the galactose operon of *escherichia coli*. *Proc. Natl. Acad. Sci. USA*, 69:2904–2909, 1972.

[46] Peter Kaplan, Guillermo Cecchi, and Albert Libchaber. Molecular computation: Adleman's experiment repeated. Technical report, NEC Research

Institute, 1995.

[47] Peter D. Kaplan, Guillermo Cecchi, and Albert Libchaber. DNA based molecular computation: template-template interactions in PCR. In *Proceedings of the Second Annual Meeting on DNA Based Computers* [5].

[48] Richard M. Karp, Claire Kenyon, and Orli Waarts. Error-resilient DNA computation. In *7th ACM-SIAM Symposium on Discrete Algorithms*, pages 458–467. SIAM, 1996.

[49] Stuart A. Kurtz, Stephen R. Mahaney, James S. Royer, and Janos Simon. Active transport in biological computing. In *Proceedings of the Second Annual Meeting on DNA Based Computers* [5]. To appear.

[50] G.E. Liepins and M.D. Vose. Characterizing crossover in genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 5(1):27–34, April 1992.

[51] M. Linial and N. Linial. On the potential of molecular computing. *Science*, 268:481, April 1995.

[52] Richard J. Lipton. DNA solution of hard computational problems. *Science*, 268:542–545, 1995.

[53] Quinghua Liu, Zhen Guo, Anne E. Condon, Robert M. Corn, Max G. Lagally, and Lloyd M. Smith. A surface-based approach to DNA computation. In *Proceedings of the Second Annual Meeting on DNA Based Computers* [5]. To appear.

[54] Y-M. D. Lo, K.F.C. Yiu, and S.L. Wong. On the potential of molecular computing. *Science*, 268:481–482, April 1995.

[55] P.E. Lobban and C.A. Sutton. Enzymatic end-to-end joining of DNA molecules. *J. Mol. Biol.*, pages 453–471, 1973.

[56] Rong-Ine Ma, Neville R. Kallenbach, Richard D. Sheardy, Mary L. Petrillo, and Nadrian C. Seeman. Three-arm nucleic acid junctions are flexible. *Nucl. Acids Res.*, 14:9745–9753, 1986.

[57] J. Marmur. A procedure for the isolation of deoxyribonucleic acid from micro-organisms. *Journal of Molecular Biology*, 3:208–218, 1961.

[58] Kalim U. Mir. A restricted genetic alphabet for DNA computing. In *Proceedings of the Second Annual Meeting on DNA Based Computers* [5].

[59] Kary B. Mullis. The unusual origin of the polymerase chain reaction. *Scientific American*, 262:36–43, 1990.

[60] Kary B. Mullis, François Ferré, and Richard A. Gibbs, editors. *The polymerase chain reaction.* Birkhauser, 1994.

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

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

[63] N. Pippenger. On simultaneous resource bounds. In *20th Annual Symposium on Foundations of Computer Science*, pages 307–311, Long Beach, Ca., USA, October 1979. IEEE Computer Society Press.

[64] Robert Pool. A boom in plans for DNA computing. *Science*, 268:498–499, 1995.

[65] C. Queen and L.J. Korn. A comprehensive sequence analysis programme for the IBM personal computer. *Nucleic Acids Research*, 12(1):581–599, 1984.

[66] John H. Reif. Parallel molecular computation: Models and simulations. In *Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms*

and Architectures (SPAA95), Santa Barbara, June 1995, pages 213–223. Association for Computing Machinery, June 1995.

[67] L. Roberts and C. Murrell, editors. *Understanding genetic engineering.* Ellis Horwood, Ltd., 1989.

[68] Paul W. K. Rothemund. A DNA and restriction enzyme implementation of Turing Machines. Unpublished manuscript, 1995.

[69] 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* [5]. To appear.

[70] C.P. Schnorr. The network complexity and Turing machine complexity of finite functions. *Acta Informatica*, 7:95–107, 1976.

[71] Nadrian C. Seeman. Nucleic acid junctions and lattices. *Journal of theoretical biology*, 99(2):237–247, 1982.

[72] Nadrian C. Seeman. Denovo design of sequences for nucleic-acid-structural-engineering. *Journal of Biomolecular structure and dynamics*, 8(3):573–581, 1990.

[73] Nadrian C. Seeman, Hui Wang, Bing Liu, Jing Qi, Xiaojun Li, Xiaoping Yang, Furong Liu, Weiqiong Sun, Zhiyong Shen, Ruojie Sha, Chengde Mao, Yinli Wang, Siwei Zhang, Tsu-Ju Fu, Shouming Du, John E. Mueller, Yuwen Zhang, and Junghuei Chen. The perils of polynucleotides: The experimental gap between the design and assembly of unusual DNA structures. In *Proceedings of the Second Annual Meeting on DNA Based Computers* [5]. To appear.

[74] Robert Shapiro. *The human blueprint.* Cassell, 1992.

[75] H. Stubbe. *History of genetics - from pre-historic times to the rediscovery of Mendel's laws.* MIT Press, 1972.

[76] J. D. Watson, M. Gilman, J. Witkowski, and M. Zoller. *Recombinant DNA.* Scientific American Books, second edition, 1992.

[77] J.D. Watson and F.H.C. Crick. Genetical implications of the structure of deoxyribose nucleic acid. *Nature*, 171:964, 1953.

[78] J.D. Watson and F.H.C. Crick. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, 171:737–738, 1953.

[79] J. Williams, A. Ceccarelli, and N. Spurr. *Genetic Engineering.* βios Scientific Publishers, 1993.

[80] 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* [5]. To appear.

[81] C. Yanisch-Perron, J. Vieira, and J. Messing. Improved m13 phage cloning vectors and host strains: nucleotide sequences of the m13mp18 and puc19 vectors. *Gene*, 33(1):103–119. 1985.

[82] Y. Zhang and N. C. Seeman. The construction of a DNA truncated octahedron. *J. Am. Chem.*, 116:1661-1669, 1994.