DNA-based computing

Yong, Kian Yan

2013

Yong, K. Y. (2013). DNA-based computing. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/54896>

<https://doi.org/10.32657/10356/54896>

DNA-BASED COMPUTING

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

YONG KIAN YAN

DNA-BASED COMPUTING

YONG KIAN YAN

SCHOOL OF MECHANICAL AND AEROSPACE

ENGINEERING

2013

2013

DNA-BASED COMPUTING

YONG KIAN YAN

School of Mechanical and Aerospace Engineering

A thesis submitted to Nanyang Technological University in partial
fulfilment of the requirement for the degree of Doctor of
Philosophy

2013

ACKNOWLEDGEMENT

The author would like to thank Nanyang Technological University and Assoc Prof Shu Jian Jun for the opportunity to pursue a PhD research. Prof Shu has been an inspiring supervisor throughout the years of PhD study; sharing his life experiences and revealing his contagious passion towards fundamental research. The author is especially appreciative of his guidance on the ways of generating new ideas, and his vision for the potential and depth of DNA-based computing research.

In early part of this research, Assoc Prof Chan Weng Kong has provided much thoughts and ideas on how to proceed with an interdisciplinary research involving mathematics, computing and biology. This has helped lay the foundations for the DNA-based computing research. Thank you.

Appreciation is also due to Asst Prof Shao Fangwei and her dedicated team of researchers from the School of Physical and Mathematical Sciences for all the help and resources in ensuring a success in the GPS experiment.

The author would also like to thank the staff at Computer Aided Engineering laboratory for providing an environment conducive for research.

CONTENTS

Acknowledgement.....	2
List of Figures	9
List of Tables.....	11
Summary	12
Publications	12
PART I – INTRODUCTION TO DNA-BASED COMPUTING	13
1 Introduction to DNA-based computing	14
1.1 Introduction.....	14
1.1.1 History of computers	15
1.1.2 DNA-based computing.....	19
1.2 Motivation	23
1.2.1 Silicon computer versus DNA computer.....	23
1.2.2 Binary versus quaternary numeral system	26
1.3 Scope	27
2 Classification of DNA-based computing problems.....	28
2.1 DNA-based problems	28
2.1.1 Games Theory	28
2.1.2 Graph Theory	30

2.1.3	Logic gates.....	31
PART II – SYSTEMS AND LABORATORY TECHNIQUES OF DNA-BASED COMPUTING		34
3	Biocomputers and their computing systems.....	35
3.1	DNA-based computing system.....	35
3.1.1	Ligation-based system	35
3.1.2	Restriction enzymes- based system.....	36
3.1.3	Tiling system	37
3.1.4	Toe-hold and strand displacement system.....	39
3.2	RNA-based computing system	42
3.3	Protein-based computing system	44
3.4	Hybrid computing system	45
4	Laboratory techniques of DNA-based computing.....	46
4.1	DNA strands design and synthesis	46
4.2	Initial DNA pool generation.....	48
4.3	Polymerase chain reaction (PCR)	51
4.4	Affinity purification	56
4.5	Gel electrophoresis	57
4.6	DNA sequencing	58
PART III – NOVEL METHODS OF DNA-BASED COMPUTING FOR GRAPH THEORY PROBLEMS		59
5	Shortest path problem	60

5.1	Problem definition: Shortest path problem.....	60
5.2	Dijkstra Algorithm	60
5.3	Case study	61
5.4	Dijkstra Algorithm: Solution walkthrough.....	62
5.5	DNA Algorithm: DNA strands design analysis	63
5.6	Experimental procedure.....	64
5.7	Expected result.....	65
5.8	Discussion	65
6	Shortest spanning tree	67
6.1	Problem Definition: Shortest spanning tree	67
6.2	Kruskal's Greedy Algorithm.....	67
6.3	Case Study	68
6.4	Kruskal Algorithm: Solution walkthrough	69
6.5	DNA Algorithm: DNA strands design analysis	72
6.6	Experimental procedure.....	73
6.7	Expected result.....	74
6.8	Discussion	75
7	Maximum flow problem	76
7.1	Problem Definition: Maximum flow problem.....	76
7.2	Ford-Fulkerson Algorithm for Maximum Flow.....	76

7.3	Case Study	77
7.4	Ford-Fulkerson Algorithm: Solution walkthrough	78
7.5	DNA Algorithm: DNA strands design analysis	79
7.6	Experimental procedure.....	81
7.7	Expected result.....	82
7.8	Discussion	83
8	Bipartite maximum cardinality problem	84
8.1	Problem Definition: Bipartite Maximum Cardinality	84
8.2	Bipartite Maximum Cardinality Matching Algorithm.....	85
8.3	Case Study	86
8.4	Bipartite Maximum Cardinality Matching Algorithm: Solution walkthrough.....	87
8.5	DNA Algorithm: DNA strands design analysis	90
8.6	Experimental procedure.....	91
8.7	Expected result.....	92
8.8	Discussion	92
PART IV – EXPERIMENT ON GLOBAL POSITIONING SYSTEM PROBLEM		94
9	Global Positioning System problem	95
9.1	Problem definition: Global Positioning System problem.....	95
9.2	Case study	95
9.3	DNA Algorithm: DNA strands design analysis	96

9.4	Experimental procedure.....	97
9.5	Expected result.....	98
9.6	Discussion.....	98
9.7	Materials and Methods.....	99
9.7.1	Hybridization and phosphorylation of DNA strands to create DNA pool.....	99
9.7.2	Ligation of DNA strands	100
9.7.3	Purification to remove ssDNA, short DNA (less than 50 bp), enzymes and impurities.....	101
9.7.4	PCR to amplify solution strands.....	102
9.7.5	Separation and quantification of DNA strands for solution readout	103
9.8	Results and Discussion	104
9.8.1	Results.....	104
9.8.2	Discussion.....	105
10	Discussion and conclusion	107
10.1	Discussion.....	107
10.2	Limitations.....	109
10.2.1	Experimental limitations.....	110
10.2.2	Human and experimental errors	112
10.2.3	NP hard problems	112
10.2.4	Irreversible	113
10.3	Conclusion	113

11	Reference List.....	114
12	APPENDIX	124
12.1	DNA strands for Shortest Path Problem	124
12.2	DNA templates for Shortest Path Problem	126
12.3	DNA strands for Shortest Spanning Tree	128
12.4	DNA templates for Shortest Spanning Tree	130
12.5	DNA strands for Maximum Flow Problem	133
12.6	DNA templates for Maximum Flow Problem	136
12.7	DNA strands for Maximum Cardinality Problem.....	138
12.8	DNA templates for Maximum Cardinality Problem	140
12.9	DNA strands for GPS Problem	142
12.10	DNA templates for GPS Problem	143

LIST OF FIGURES

Figure 1-1. CPU transistor count versus dates of introduction (Source: Wikipedia).....	18
Figure 1-2. Double helix DNA structure and nucleotide bases A, C, G and T.	20
Figure 2-1. Boolean operations and logic gates (Source: Wikipedia).....	32
Figure 3-1. Ligation. DNA strand A has a partial complementary sequence with strand B. This results in a longer output strand consisting of both strands annealing to one another, which can be detected by gel electrophoresis.....	36
Figure 3-2. A set of 13 Wang tiles and its aperiodic assembly (Source: Wikipedia).....	38
Figure 3-3. Central Dogma of Molecular Biology.....	40
Figure 3-4. Toehold and strand displacement technique. An output strand is released into a solution. The output strand binds to the translator because it has a complementary sequence to the latter (output'). In the process, fluorophore (f) is released into the solution with increased fluorescence emission thereby signaling a positive output.....	41
Figure 3-5. Translation process involving messenger RNA (mRNA), ribosome (rRNA) and transfer RNA (tRNA) (Source: Wikipedia).	43
Figure 4-1. Polymerase chain reaction; cycles 1 and 2. DNA strands are represented by arrows running from the direction 5' to 3'. Those from previous cycle are differentiated with the newly synthesized ones by solid and dotted lines respectively. Oligonucleotide primers are characterized by rectangles.....	52
Figure 4-2. Polymerase chain reaction; cycle 3.	53
Figure 4-3. PCR machine Mastercycler ep <i>realplex</i> (Source: www.eppendorf.com).	56

Figure 4-4. An output image of gel electrophoresis. Label M stands for DNA size marker or ladder (each band is 50 bp starting from the bottom of image) and label “1” shows a high concentration band of DNA strands of 300 bp [26].	57
Figure 5-1. Shortest path problem case study	61
Figure 5-2. Shortest path problem expected result.	65
Figure 6-1. Shortest spanning tree case study	68
Figure 6-2. Kruskal algorithm - Intermediate stages of edge selection	71
Figure 6-3. Kruskal algorithm - Final stages of edge selection.	71
Figure 6-4. Shortest spanning tree expected result.	74
Figure 7-1. Maximum flow problem case study.	77
Figure 7-2. Maximum flow problem expected result.	82
Figure 8-1. Bipartite maximum cardinality between groups S and T; each having 3 elements.	84
Figure 8-2. Alternating and augmenting paths	85
Figure 8-3. Bipartite maximum cardinality problem case study.	86
Figure 8-4. Bipartite algorithm solution walkthrough – no augmenting path.	88
Figure 8-5. Bipartite algorithm solution walkthrough – augmenting path	89
Figure 8-6. Bipartite maximum cardinality problem expected result.	92
Figure 9-9-1. Global Positioning System case study.	95
Figure 9-9-2. Global Positioning System expected result.	98
Figure 9-3. Native PAGE setup.	103
Figure 9-4. Native PAGE gel result of GPS problem.	105

LIST OF TABLES

Table 1-1. Computer history (Source: Wikipedia).	16
Table 1-2. Silicon computer versus DNA-based computer [4].....	24
Table 6-1. Solution of Figure 6-1.....	70
Table 9-1. GPS distance and path for the 6 vertices.....	96

SUMMARY

DNA-based computing provides an alternative to solving optimization problems in graph theory. In this research, it is shown how DNA-based computing is used to find solutions to these problems, which involve logical thinking and are often NP hard complete. These include the shortest path, shortest spanning tree, maximum flow and maximum bipartite matching problems. DNA-based computing is a suitable tool for these problems because of its massive parallelism during computation. The success of a DNA-based experiment designed around the shortest path problem, global positioning system problem, reinforced and revealed the potential of this approach.

PUBLICATIONS

- [1] Shu, J.J., Q.W. Wang, and K.Y. Yong, *DNA-Based Computing of Strategic Assignment Problems*. Physical Review Letters, 2011. **106**(18).
- [2] Shu, J.J., K.Y. Yong, and W.K. Chan, *Lecture Notes in Computer Science: Multiple DNA Sequence Alignment Using Joint Weight Matrix*, in *Computational Science and Its Applications - Iccsa 2011, Pt Iii*, B. Murgante, et al., Editors. 2011, Springer-Verlag Berlin: Berlin. p. 668-675.
- [3] Shu, J.J., K.Y. Yong, and W.K. Chan, *An Improved Scoring Matrix for Multiple Sequence Alignment*. Mathematical Problems in Engineering, 2012.

PART I – INTRODUCTION TO DNA-BASED COMPUTING

1 INTRODUCTION TO DNA-BASED COMPUTING

1.1 Introduction

DNA-based computing has come a long way since it was first introduced by Adleman in 1994 [4]. According to the theory of computing, computing contains two parts; a method of storing information and a way of acting on the information through operations. Modern computers have flash drive and microprocessor chip such as Intel to achieve these.

In a similar sense, DNA can be used for computing. It stores information using four types of nucleotide bases. Strands of DNA can then be manipulated through operations, in the form such as chemicals and enzymes.

Why DNA-based computing? A DNA strand can store huge amount of information. The inter-strands operations are much faster than that of modern computers. It is extremely energy efficient.

Algorithms for solving mathematical problems in graph theory are derived to demonstrate the versatility of DNA-based computing. These problems include the shortest path problem, GPS problem, shortest spanning tree problem, maximum flow problem and assignment problem.

DNA-based computing can be scaled up to solve higher dimensional problems. One such problem is that of multiple sequence alignment.

Instead of competing with modern computing, DNA-based computing can be combined with it to form a new type of hybrid computation. Starting from the building blocks of a computer, biological transistors, capacitors can be built to create biological logic gates. These would form the fundamentals of a DNA computer.

Applications of DNA-based computing may include important cellular pathways identification, health monitoring and diagnosis, disease management and cure.

Instead of having computing defines what can be done, we let what can be done define computing. Danchin [5] made a philosophical study into what defines a molecular computer; one that is able to store and build on knowledge, and then duplicate this information to be passed down through generations.

Perhaps computing can be simply defined in three words; storage, operation and restore.

1.1.1 [History of computers](#)

Computers today are very powerful and can perform millions of calculation per second. They are also small and affordable to many people. It is quite astonishing if one were to look at how fast computers have developed since the first ones were built around 1940 (Table 1-1). They were once driven by mechanical and electromechanical components where instructions or programs were written using punched cards. This was the first generation computers. The second generation computers were created using vacuum tubes and capacitors between 1940 and 1950. Vacuum tubes were used as switching elements that define the various states of a computer program. Capacitors allowed computers to

have memory compartments where intermediate results could be stored and fed back into the computation system. As a result, size of computers was reduced from once taking the space of a whole room to that of a large desk.

Table 1-1. Computer history (Source: Wikipedia).

Generation	Type	Example	Remarks
First – Pre 1940	Mechanical, Electromechanical	Calculators, programmable devices	
Second – 1940 to 1950	Vacuum tubes	Calculators, programmable devices	
Third – 1950	Transistors and printed circuit board Discrete transistors and SSI, MSI, LSI Integrated circuits	Mainframes, minicomputer	Less expensive, faster, compact, lower operating temperature compared with 2 nd generation
Fourth – Post 1960s	Integrated circuit VLSI integrated circuit	Minicomputer, 4-bit to 64-bit microcomputers, embedded computer, personal computer	Microprocessor – 1971
Fifth	Theoretical, Experimental	Quantum computer, Chemical computer, DNA computer, Optical (Photonic) computer, Spintronics based computer	<ul style="list-style-type: none">• Quantum computer – Deutsch D 1970s• Photonic computer – 1989 RMRC (Photonic transistor)• DNA computer – Adleman 1994 [4]

			<ul style="list-style-type: none">• Chemical computer – Belousov 1959 [6], Adamatzky 2002 [7]
--	--	--	---

In the 1950s, vacuum tubes in computers were gradually replaced by transistors giving way to third generation computers. Transistors have many advantages over vacuum tubes for computing. They are faster, smaller, less expensive, more power efficient and reliable. The transistors were connected together along with other electronic components on a semiconductor material, known as the integrated circuit (IC). The computer system on the IC that carries out the program is known as the central processing unit (CPU). Earlier on, each CPU was capable of only one or a few functions. This meant that one had to physically switch between different IC to use different functions. This was an inefficient way to compute. The problem was solved when an IC that incorporated most or all functions was made. This is known as the microprocessor which is now the core of modern fourth generation computers. Computers are made even faster and more compact by having very small transistors using advanced nanotechnology. However, there is a limit to how small transistors can go (Figure 1-1) according to Moore's Law [8], as it approaches the size of a single atom [9].

Figure 1-1. CPU transistor count versus dates of introduction (Source: Wikipedia).

Scientists have started to explore other types of technology on which future computers can be built on. This is known as the fifth generation computers and they include the use of knowledge based on quantum technology [10-12], chemistry [13], biology [14], optical [15, 16] and spintronics [17]. These computers are either in the theoretical or experimental stage. Among these fifth generation computers, DNA-based computing demonstrates a great potential because it can be very compact as DNA strands are very small (1 bit per nm^3 versus 1 bit per 10^{12} nm^3 in modern computers). Computing is also extremely fast due to parallel processing (10^{14} operations per second versus 10^{12} operations

per second in modern computers). It is more energy efficient compared to modern computers. The energy used for one mathematical operation, represented by a reaction between two DNA strands or 10^{19} operations per joule versus 10^9 operations per joule in silicon computers.

1.1.2 DNA-based computing

At the heart of every human cell is a nucleus. Inside the nucleus are twenty three pairs of chromosomes. If we unwind those chromosomes, we will get deoxyribonucleic acids or DNA. DNA is a nucleic acid containing the code of life. Information that is used for the development and function of all living organisms is stored in the DNA [18]. It has a double helical structure that is discovered by James Watson and Francis Crick [19], and consists of four nucleotide bases; adenine (A), cytosine (C), guanine (G) and thymine (T). A and G are classified as purines; C and T are classified as pyrimidines. Purines will bind pyrimidines to form hydrogen bonds; specifically A will only pair up with T, and G will only pair up with C (Figure 1-2). Variation in the order and number of these nucleotide bases enable an infinite number of unique DNA strands to be formed. It is estimated that the human genome [20], made up of the twenty three pairs of chromosomes consists of 3 billion nucleotide base pairs. And all that information is packed inside the tiny nucleus of a cell. The vast amount of information that can be stored inside a DNA strand, the efficiently in which these information are stored, and the way in which these information can be manipulated gave rise to DNA-based computing.

DNA-based computing is invented by Adleman in 1994 [4]. Biological reactions of DNA strands coupled with enzymes are used to find solutions to problems that would otherwise be too complex to handle by silicon computer. DNA-based computing is at least a thousand times faster than the fastest super computer around. However, it is more suitable for solving problems that involve logical thinking rather than arithmetic operations. One such problem is the directed Hamiltonian path problem, which is NP complete and would have been too time consuming and complex for the silicon computer to solve. However, it is shown by Adleman that the problem can be easily solved using DNA-based computing.

Figure 1-2. Double helix DNA structure and nucleotide bases A, C, G and T.

Since the invention of DNA-based computing by Adleman, there have been much improvements and variations to its problem solving technique. These include using ribonucleic acid (RNA) instead of DNA strands to generate the initial solution pool [21], parallel assembly methods [22, 23] and DNA hairpin formation [24]. One technique worth

mentioning is the use of restriction enzymes to replace affinity purification during the solution filtering process [25]. These developments open up more possibilities for DNA-based computing.

Different encoding methods for DNA strands are also introduced. One such method utilizes the thermodynamic properties of DNA strands for their design. It allows similar length DNA strands to be used in generating DNA pool [26] instead of having strands of varying lengths [27]. This is followed by the development of other DNA strands design software such as DNASequencesGenerator [28, 29], NACST/Seq [30] and DNA-SDT [31]. Another commonly researched encoding method is that of binary bit encoding [32]. It is inferred that if the basis of computing by binary bit encoding is possible using DNA-based computing, then it can be introduced to modern computing. One such possibility is a hybrid computer comprising of both silicon and DNA computations.

A mathematical notation for DNA-based computing is recently presented [33]. This would allow DNA-based computing to solve more general mathematical problems, without being limited to specific problems that have been solved. These problems are summarized in Chapter 2 (2.1). The following provides a more detailed overview of DNA-based computing development since its introduction in 1994.

Adleman L M, 1994 [4]

Adleman presented a novel way of solving the Hamiltonian path problem using molecular biology. A Hamiltonian path is defined as a path in an undirected graph, which visits each vertex exactly once and also returns to the starting vertex. Determining whether

such path exists is the Hamiltonian path problem, and it is NP complete. Each vertex and edge is represented by 20-mer oligonucleotides except for the starting and ending edge. About 3×10^{13} copies of them are mixed together in a single ligation reaction. The ligation reaction resulted in the formation of DNA molecules encoding random paths through the graph. Due to the large number of oligonucleotides used, it is likely that a large number of DNA molecules encoding the Hamiltonian path are created. The mixture then goes through several processes of filtering using affinity purification and gel electrophoresis to arrive at the answer.

There are many advantages of using molecular computation compared to electronic computation. Firstly, number of operations per second during the ligation step exceeds that of super computers by more than a thousand fold. Secondly, it is remarkably energy efficient. In principle, one joule is sufficient for approximately 2×10^{19} operations compared with 10^9 operations per joule in super computers. Thirdly, it is storage efficient requiring only 1 cubic nm to store 1 bit of information compared with storage media such as video tape of 1 bit per cubic nm.

Faulhammer D *et al.*, 2000 [21]

Faulhammer *et al.* expanded the field of DNA-based computing to include RNA strands for computation. A destructive algorithm is developed, which allows equal-length RNA strands that did not fit the constraints of the problem to be hydrolyzed and removed. This is done by first annealing specific DNA bit oligonucleotide to those strands. After which, ribonuclease (RNase) H digestion is used to destroy these RNA/DNA hybrids. This technique

is used to find solutions to the “Knight problem”. Using this approach, DNA algorithm is further simplified by excluding the need for DNA sequencing to get the answer. The upper bound of *in vitro* selection protocols for DNA-based or RNA-based computing experiments using exhaustive search algorithms is approximately 2^{50} or 10^{15} . This means that they can handle problems with up to a zillion possible outcomes.

Manca V *et al.*, 2008 [33]

Manca *et al.* presented a novel way of representing different mechanisms of DNA recombination using mathematical notation. This representation enables the mathematical analysis of DNA recombination, and in turn allows new technologies for DNA manipulation to be discovered. One such discovery is cross pairing PCR (XPCR).

1.2 Motivation

1.2.1 Silicon computer versus DNA computer

The following table (Table 1-2) compares DNA-based computer with silicon computer [4]. The former is faster, more energy and storage efficient. In a DNA-based computing experiment in 2003, a rate of 6.646×10^{10} operations per second per μl , with a heat dissipation of approximately 5.3×10^{-9} W/ μl and using 33.9 kT of free energy per transition for a maximum of 54 transitions, was achieved [34].

DNA-based computer can also solve non-deterministic polynomial (NP) complete problems more efficiently using parallel processing; reaction between one pair of DNA strands is taken as one operation, and up to 10^{20} DNA strands can be present in a DNA pool. One area where DNA-based computer loses out to silicon computer is that of performing mathematical calculations. The time taken to design and run laboratory experiments would be significantly larger than the seconds or even milliseconds required by that of a silicon computer. Despite this limitation, DNA-based computer can be used for other calculations and applications that are either not possible or time and resource inefficient for the silicon computer. For example, the use of DNA-based computing *in-vivo* for the diagnosis of illness in human body [35].

Table 1-2. Silicon computer versus DNA-based computer [4].

	Silicon	DNA
Speed	10^6 to 10^{12} operations per second	10^{14} to 10^{20} operations per second (ligation)
Energy	10^9 operations per joule	2×10^{19} operations per joule
Storage	1 bit per 10^{12} cubic nm	1 bit per cubic nm
Mathematical calculations	Efficient	Not practical with available protocols and enzymes
Intrinsically complex problems (directed Hamiltonian path problem)	Inefficient	Advantage of massive parallel processing

There are many advantages for a DNA-based computer, and this can be used to build on existing knowledge. Applications include a molecular sized DNA-based computer, which is able to reach within the human body and works together with it using input signals from proteins [36]. The potential and applications of a DNA-based computer provide strong motivation, and contribute to the objective of this research; to build a DNA-based computer that is capable of solving problems that is too complex, inefficient or impossible for the silicon computer. The task of building this computer is broke up into three subtasks. The first subtask is to get familiarized with DNA-based computing techniques. This is done by designing DNA algorithms and carry out laboratory experiments to solve graph theory problems. The former has been achieved and is presented in Chapters 5 to 8 of this report. The second task is to create both one-dimensional and two-dimensional DNA-based logic gates. Since silicon computers are built from logic gates, it is hypothesized that by successfully creating DNA-based ones, building a DNA-based computer is possible. This is elaborated in greater details in Chapter 2 (2.1.3). The third task is to take advantage of the unique four-nucleotide base DNA code to devise a quaternary number system, as opposed to a binary number system used in silicon computers. A computer using higher number system is conjectured to be able to compute faster. This is elaborated in the following section. The first subtask has been achieved in this research.

1.2.2 Binary versus quaternary numeral system

A binary number is a real number represented by 0 or 1 and has a base of 2. For example, number 14 is equivalent to $1110_2 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$. The binary numeral system is used by computers for processing information and calculation. This is because the binary numbers 0 and 1 can be directly translated from an on and off signal respectively. Similarly, a quaternary number is one with a base of 4. The digits 0, 1, 2, and 3 are used to represent any real number. Number 14 is equivalent to $32_4 = 3 \times 4^1 + 2 \times 4^0$ in quaternary numeral system.

Theoretically, a higher base numeral system will be able to process information faster. Each quaternary bit has a higher processing capacity as it uses four numbers (0, 1, 2 and 3) compared to two numbers (0 and 1) for the binary bit. However it is not possible to implement the quaternary numeral system for the integrated circuit boards used in computers. This is so as there are only two types of signals for the logic gates. These are measured by whether an electric current (voltage) is present in the output logic gate or not. In order to use the quaternary numeral system, there must be four types of signals.

In DNA-based computing, there are four types of bases (A, C, T and G). This could be used as the four types of signals for a quaternary numeral system. However, recent techniques used in DNA-based computing are based on a binary numeral system; a pair of DNA strands with complementary strands would then bind to each another and vice versa. A novel method that makes use of the four bases as four inputs could be introduced. Once this is done, a far more superior quaternary numeral system using DNA-based computing could be created. A quaternary numeral system can be used for analyzing problems with

hyper complex numbers, i.e. using A, C, T and G for real number, and hyper complex numbers i , j and k respectively.

1.3 Scope

DNA-based computing is a multidisciplinary field of research. It involves mathematics, computing and biology. This report is organized into four parts. An introduction to DNA-based computing and how it is used to solve some categories of problems is provided in Part I. A literature review of how computers have evolved since its first inception in 1940 has been presented in Chapter 1. This is followed by the possible structure that they may take in the future, which forms the motivation in Chapter 2. With a better understanding of DNA-based computing, its systems and laboratory techniques are then elaborated in Part II. A comprehensive set of biocomputing systems, including that of the RNA-based and protein-based ones, is presented in Chapter 3. This would allow a better appreciation of the potential of DNA-based computing. A combination of DNA-based computing with other systems enables a more complex biocomputer to be built; and hence a more complex problem to be solved. The methodology and laboratory experiments of DNA-based computing are elaborated in Chapter 4. Four novel DNA-based computing algorithms for solving graph theory problems are proposed (Chapters 5 to 8) in Part III. In last part of this report, an experiment on the shortest path problem, its design, algorithm and results are elaborated in Part IV. This is followed by an in-depth discussion and a conclusion in Chapter 10.

2 CLASSIFICATION OF DNA-BASED COMPUTING PROBLEMS

2.1 DNA-based problems

Problems that have been solved with DNA-based computing are broadly classified into three categories and summarized in this chapter; games theory, graph theory and logic gates.

2.1.1 Games Theory

Problems that involve logical thinking, strategies and payoffs are covered in games theory. Among these problems, solutions that have been proposed using DNA-based computing include the Boolean satisfiability (SAT) problem [37, 38], chess board problem [21], Chinese postman problem [27], traveling salesman problem [26, 39], maximal clique problem [25, 40-43], minimum spanning tree [44], longest common subsequence [45], poker [46] and clustering problem [47]. The development in DNA-based computing and its capabilities are best summarized in a review paper [48]. Evolutionary theories such as that of Charles Darwin, classified as evolutive games theories [49], may also be a suitable candidate for further in-depth study using DNA-based computing.

Ouyang Q *et al.*, 1997 [25]

Ouyang *et al.* applied DNA-based computing to find the solution for the maximal clique problem. Unlike Adleman's method, restriction enzymes instead of affinity purification are used to remove sites that do not form part of the solution. The DNA data pool is designed using a binary encoding method. Two DNA sections are used to represent each binary number, which correspond to its position and the bit's value (0 or 1). Each data structure is then constructed using parallel overlap assembly (POA). The solution for the maximal clique is found using gel electrophoresis, which corresponds to the lowest band. DNA cloning and sequencing are used to find vertices within the maximal clique. There are some limits pertaining to their approach. The largest maximal clique sizes that can be found are 27 vertices and 36 vertices for picomole and nanomole operations respectively. Therefore a faster and more accurate, automatic device is needed to take advantage of the massive parallelism in DNA-based computing.

Yin Z *et al.*, 2002 [27]

DNA-based computing is used to solve the Chinese postman problem. A similar approach has been used to solve this problem as proposed by Adleman. The main difference is the design of oligonucleotides. Length of each oligonucleotide representing the edges is proportional to their weights. This allows edges of varying weights to be possible compared to Adleman's method. The limitation of such sequence design is that the weights must be an integer. Also, it is difficult to solve edges with weights that are very big or small. This problem is later addressed by Lee *et al.*, 2004 [26].

Kuhn, H. W. *et al.*, 2002 [50].

Von Neumann and Morgenstern [51] introduced the theory of cooperative games that applied to two-person, non-zero-sum games and games with three or more players in their book *Theory of Games and Economic Behavior*. In 1950, Nash proposed the theory of non-cooperative games that encompassed all the cases as well as two-person zero-sum games. This was later known as Nash equilibria. Proof of Nash equilibria was first provided using Brouwer's fixed point theorem and later using Kakutani's fixed point theorem. The latter was published in *Proceedings of the National Academy of Sciences*. Von Neumann and Morgenstern's theory assumes that players have some levels of collaborations between them while playing the game. In contrast, Nash assumes the absence of such coalitions between players and introduced the notion of equilibrium point. An equilibrium point is defined as an n -tuple or set of n items such that each player's mixed strategy maximizes his payoff if strategies of the others are held fixed. Therefore at this point, each player's strategy is the best against those of the others.

2.1.2 [Graph Theory](#)

Some of the problems found in games theory can be generalized and classified under graph theory. These are problems that include structures and can be represented using a graphical method such as the traveling salesman problem where destinations and roads are represented by points and edges respectively. In this research, DNA-based computing is used to solve graph theory problems. Graph theory being more established (in the 18th

century by Leonard Euler [52]), compared with games theory in the 20th century (by John von Neumann and Oskar Morgenstern [51]), provides a wider platform of opportunities for DNA-based computing. Recently some graph theory problems have been discussed, and their respective algorithms presented [53]. In this research, DNA-based computing is used to solve four categories of problems listed in the book under the chapter of *Graphs and Combinatorial Optimization* by Kreyszig [54]. They are the shortest path, shortest spanning tree [44], maximum flow network and bipartite maximum cardinality matching problems. A literature review reveals that no attempt has been made to solve the latter two problems.

2.1.3 Logic gates

Boolean logic is a complete set of logical operations, between two variables x and y , which is created by George Boole in the 1840s. The basic Boolean operations between x and y are conjunction $x \wedge y$, disjunction $x \vee y$, and complement or negation $\neg x$ (Figure 2-1). All the other operations can be built from these three operations. In digital circuits, transistors or diodes are used to perform Boolean logic as logic gates (Figure 2-1). These are the building blocks of modern computers, where the NAND and NOR gates are the basic gates from which all the other gates can be built from.

Figure 2-1. Boolean operations and logic gates (Source: Wikipedia).

Similarly, a DNA-based computer can be built using DNA-based logic gates [55-57]. These can be built upwards starting from basic molecular switches [58] triggered by light [59], pH level [60] and metal ions [61]. Recent development in this area includes the use of toe-hold sequestering technique [35] to build simple DNA-based logic gates. The main challenges in building a DNA-based circuit with logic gates are transmitting output information from one logic gate to another, signal restoration and reusability of logic gates for later stage [62]. Researchers have proposed reversible logic gates to build more complex DNA-based circuits [58, 63, 64]. However, these designs which rely on ideal concentrations of specific DNA strands to function are time consuming and less precise. A more efficient way could be achieved using a DNA-based computer running on two-dimensional logic gates. The additional dimension could be used to provide feedback to the logic gates. This may be in the form of a quaternary logic gate, corresponding to the four nucleotide bases of DNA (A, C, T and G).

Recently a new form of biological logic gate, based on electrochemical biosensors [65], have been created [66]. Instead of using DNA strands to transmit data from one logic gate to another, current in the form of electrons are used. Mutations within DNA strands will either inhibit or allow electrons to pass through, and this property is used in the application of Boolean logic. A 2011 paper by Qian L. *et al.* [67] saw the use of DNA logic gates to build a neural network system, which is capable of playing a 'read your mind' guessing game. The logic gates are based on a modified DNA hybridization technique, known as toehold strand displacement. Also in a recent paper, DNA logic gates have been proposed for the use in drug delivery, and for the detection and killing of tumor cells [68].

PART II – SYSTEMS AND LABORATORY TECHNIQUES OF DNA-BASED COMPUTING

3 BIOCOMPUTERS AND THEIR COMPUTING SYSTEMS

3.1 DNA-based computing system

DNA-based computer is one type of biocomputers [14]. A biocomputer can be defined as a biological system that is programmable to produce an analytical answer for a given input. There are three main classes of biocomputers; DNA-based computer, RNA-based computer and protein-based computer. The three types of biocomputers and their systems of computation are explained in this chapter.

3.1.1 Ligation-based system

Several unique DNA strands are mixed together and those with complementary strands would anneal to each other either completely or partially (Figure 3-1). Rules are set so that DNA strands would anneal accordingly to the algorithm, using conditional mathematics which is similar to Boolean logic. Enzymes known as DNA ligase are then added to tie up the ends between these annealed strands, forming longer strands. The unique individual strands represent parts of a solution, while the ligated strands represent most if not all possible solutions. Selective DNA strands are then amplified through a process known as polymerase chain reaction (PCR), although annealing and ligation alone may be able to produce the solution [69].

Figure 3-1. Ligation. DNA strand A has a partial complementary sequence with strand B. This results in a longer output strand consisting of both strands annealing to one another, which can be detected by gel electrophoresis.

After PCR, the solution is usually represented by the shortest among the amplified DNA strands or a predetermined length depending on the algorithm. The Chinese postman [27] and travelling salesman [26] problems have been solved using this system, where they have been simplified to finding the shortest path linking all vectors. An expansion of this system to two dimensional matrix form has also been recently proposed [70].

3.1.2 Restriction enzymes- based system

DNA strands can be cut at specific regions using restriction enzymes. The enzymes would bind to regions of DNA with complementary bases and cut those regions. This technique has been used to create vaccines for illness, such as the one caused by flu virus. The flu virus is analyzed and regions of its DNA that code for proteins that damaged the cell is determined. These regions are then removed by restriction enzymes and the remaining regions put back together. The result is a mild form of the flu virus that is not strong enough to result in a flu but sufficient for the human body to produce antibodies to fight the virus.

This technique when used in DNA-based computing opens up more possibilities in terms of computing complexity. In addition to setting minimum conditions to be met, boundary conditions can be set. DNA strands with solutions that are beyond the boundary will be destroyed or cut. Algorithms designed around this technique have been used for problems such as the Knight problem albeit using RNA strands [21], and the assignment problem [1]. An automated and programmable biomolecular computer has been built around this technique [71], where an encoded input strand is decoded through a series of cycles. During each cycle, a portion of the strand is cleaved if it matches the restriction enzyme recognition site. The process continues until the input strand is cleaved till the end or when no restriction site is detected. The decoded output is read using gel electrophoresis. The automated biomolecular computer has sprung off several ideas including an automated gene expression mechanism [36], a potential medical diagnosis and cure for diseases [72], and a biological version of a computation model (branching program) [73].

3.1.3 Tiling system

The tiling system is used to simulate earlier form of the Turing machine where programs were represented on a tape [74]. The Turing machine provides a readout using symbols based on the order of holes punched on the tape. A different set of symbols can be attained by shifting the point where the machine starts to read. The starting point is called the controller state, and together with the symbols is referred to as a configuration. A configuration can thus be changed by changing the controller state.

DNA sequences, known as tiles are used to represent symbols and controller state. A configuration is a row of tiles. In order to change a configuration, a new row of tiles is stack together on top of the initial row in a way determined by Wang tiles [75]. Wang tiles are square tiles with colored edges, arranged in a way such that edges with similar colors are placed next to each other, and forming an aperiodic pattern on a plane [76]. A set of 13 Wang tiles, with each having a unique combination of 5 choice colors and its aperiodic assembly is shown in [Figure 3-2](#). Output from the stack of tiles is obtained by means of gel electrophoresis and atomic force microscopy. The program can be continued by stacking new rows of tiles on subsequent ones.

Figure 3-2. A set of 13 Wang tiles and its aperiodic assembly (Source: Wikipedia).

The tiling system has been used for making DNA-based logic gates [77, 78] and for arithmetic computations. The latter include counting [79], addition and multiplication [80],

as well as subtraction and division [81]. Challenges of the tiling system includes deciding on the minimum types of tiles required to produce the solution, the speed of tile assembly and whether a solution can be successfully produced for nondeterministic computations [82].

An interesting experiment has been done on how these tile sets could self-heal much as in the self-healing mechanisms that is present in life (organisms) [83]. One may see the implication of this study as a possible future biological computing *in vivo*, to the far extent of self-regeneration in cells and organs within the human body.

3.1.4 Toe-hold and strand displacement system

According to the Central Dogma of Biology ([Figure 3-3](#)), DNA is the source of information from which ribonucleic acid (RNA) is produced or transcript. RNA is similar to DNA except for the following. RNA is usually single-stranded, contains ribose instead of deoxyribose (ribose that lacks an oxygen atom, hence DNA is less reactive) and has the nucleotide base uracil (U) instead of thymine (T). RNA strands are then used for producing proteins through a process known as translation. The process of transcription and translation can be considered as two-dimensional and three-dimensional operations respectively. The former process requires two factors to form RNA. The primary factor is DNA and the secondary factor includes ribosome and single nucleotide DNA (snDNA). Translation on the other hand, requires three factors. They are the RNA, ribosome and amino acids, and cofactors. Cofactors are proteins that bind to the promoter region of RNA, forming a three-dimensional shape that would fit the ribosome. Thereafter, the ribosome would attach itself to the RNA and starts translation.

Figure 3-3. Central Dogma of Molecular Biology.

Compared with transcription and translation, ligation and restriction computing systems discussed above are one-dimensional. DNA strands are either annealed at their complementary parts, or cut by restriction enzymes. A two or three-dimensional operation would be able to handle a more complex problem. However, this cannot be achieved without a more complex procedure involving transcription and translation. That is until the toehold and strand displacement system is introduced [35, 62, 84].

Figure 3-4. Toehold and strand displacement technique. An output strand is released into a solution. The output strand binds to the translator because it has a complementary sequence to the latter (output '). In the process, fluorophore (f) is released into the solution with increased fluorescence emission thereby signaling a positive output.

A double stranded DNA (dsDNA) with a toehold or extended single strand is a simple structural setup for the toehold and strand displacement technique. A fluorophore attached to the opposite side of the dsDNA is used as an output signal. The fluorophore containing strand is released when a complementary single stranded (ssDNA) sequence binds to the toehold, and slowly displaces it upon complete annealing (Figure 3-4). This is similar to transcription where dsDNA represent DNA strand, input ssDNA strand as ribosome and fluorophore as output mRNA. Hence a higher dimensional operation can be achieved at the “DNA level” without the need for transcription and translation. This is the main advantage of toe-hold and strand displacement system.

The toe-hold and strand displacement system is also be used as a catalyst for hybridization [58]. This is especially helpful when ssDNA with hairpin structures are involved; a short ssDNA would act as a catalyst by attaching to the toe-hold and ‘opening up’ the

hairpin structure for hybridization. This system has been proposed for solving medical applications, such as diagnosis of diseases [35, 85], as well as a programmable molecular controller [86]. A more complex system involving four annealed strands in the form of a triple crossover complex [87] or Holliday junction [88] have been explored, although the system may not be as robust [58].

3.2 RNA-based computing system

There are three main types of RNA; messenger RNA (mRNA), ribosomal RNA (rRNA) and transfer RNA (tRNA). Proteins are produced using information on mRNA. Other molecules involved are rRNA, tRNA and cofactors. rRNA is the machine that execute the translation process. However, in order for rRNA to attach to mRNA, cofactors such as primers must be present. Information on mRNA is read and translated by rRNA. Amino acids forming part of the protein is then brought in by tRNA. The process goes on until the stop codon is reached and the output protein completed (Figure 3-5).

Possible inputs for a RNA-based computing system are mRNA, rRNA, tRNA and its cofactors. Output is determined by presence or absence of a selected protein. This can be measured depending on the type of gene used, and thus its respective protein. For example if a fluorescence gene is used, the resulting protein will emit fluorescence light. This is then measured using a luminescence spectrometer. An example is the proposed automated RNA-based computer, where mRNA is used as an indicator or input for detection of disease-related genes, and thereafter the release of respective drugs by the computer as output [36].

Figure 3-5. Translation process involving messenger RNA (mRNA), ribosome (rRNA) and transfer RNA (tRNA) (Source: Wikipedia).

Progressively, more research has been done on RNA-based computing with other types of RNA, those that affect gene expression by interacting directly with information carrying mRNA. The notable ones are small interfering RNA (siRNA) and microRNA (miR) [89]. Such RNA-based circuits have been proposed for anticancer treatment [90].

3.3 Protein-based computing system

In addition to cofactors, there are other proteins affecting the translation of proteins from mRNA. These are known as activator and repressor proteins. As the names suggest, the former enhances the translation process resulting in more output proteins. On the other hand, the repressor protein prevents translation from taking place by binding to the cofactor or mRNA promoter region. Either way, it prevents ribosome from binding to the mRNA thus translation cannot take place.

Protein-based computing system is similar to mRNA-based system; both comprised of the translational process. However the former is more focus on whether translation has taken place using mRNA as a switch. If the switch is turned on, an output protein is detected and vice versa. On the other hand, the latter focuses on the interaction of proteins for translation. These proteins are known as transcription factors that affect translation, which in turn determine the amount of output proteins. The output proteins can then become transcription factors for another translation process. This enables the system to provide a feedback signal to adjust the output accordingly to what is required. By cascading a series of these protein networks, a complex computing system can be built. However this network is limited to no more than 3 layers. A larger network requires a longer computing time, which is more than that required for the host cell to divide, and this would result in a loss of resolution [14]. The ideas and challenges of a protein-based system has been discussed [91].

3.4 Hybrid computing system

The three systems described have their pros and cons; level of difficulty in carrying out the computation (which could be estimated [92]), and the type of problems they can solve. The next step to improving the biocomputer will be to combine these systems. A hybrid system that integrates transcription of mRNA from DNA, to translation of proteins from mRNA, and then to protein-protein interactions can perform more complex logical computations. The difficulty lies in controlling parameters that affect each level of network and how they interact with one another, as demonstrated in a hybrid experiment involving DNA, RNA and transcription [93]. In the next chapter, we will look into the techniques used in carrying out DNA-based computing.

4 LABORATORY TECHNIQUES OF DNA-BASED COMPUTING

The commonly used laboratory techniques in DNA-based computing are DNA strands design and synthesis, DNA pool generation, ligation, restriction, polymerase chain reaction (PCR), affinity purification, gel electrophoresis and DNA sequencing [37]. These are described in greater details as follows.

4.1 DNA strands design and synthesis

DNA strands are naturally produced from living cells via DNA replication. This process is expensive and time consuming. With the advancement in technology and increase in demand for artificial strands, DNA synthesis becomes an automated process by machines and is readily available at a relatively low cost [58]. Focus on the development of DNA strands can thus be shifted from DNA synthesis to DNA strands design.

Before laboratory experiment for DNA-based computation can be carried out, number and sequences of DNA strands have to be planned and designed according to the problem. Number of DNA strands is dependent on the number of vertices and edges, and how they are connected. Length and sequence of DNA strands are in terms decided by the type of sequence encoding method chosen [26], and weights assigned to the vertices and edges. Once these are decided, the challenge would be to work out the exact sequence of these

DNA strands so that they will bind correctly. In this report, a DNA sequence design system based on the concept of Pareto optimization [30] is used. If PCR would be included as part of the operators for the DNA-based algorithm, primers design would be carried out in this stage as well.

Lee J Y *et al.*, 2004 [26]

Lee *et al.* proposed a new sequence encoding method for DNA-based computing using the thermodynamic properties of DNA. This allows numeric values to be represented while at the same time not limited by length of the sequences. Cost sequences have similar length but varying melting temperatures, which are relative to their costs. A smaller cost is represented by a DNA sequence with a lower melting temperature. A more economical path therefore has a lower melting temperature. Melting temperature of a DNA strand is calculated using the GC method and the nearest-neighbor (NN) method. A novel encoding method and molecular algorithm (DTG-PCR and TGGE respectively), which are based on DNA sequence thermodynamic properties, are used to solve the traveling salesman problem (TSP). This is similar to the Chinese postman problem algorithm proposed by Yin *et al.*, 2002 [27].

Kim D *et al.*, 2003 [30]

Many objectives have been used in the design of DNA sequences for DNA-based computing. For example, the GC method is used in estimating the melting temperature of a DNA from its sequence [26]. In order to simplify the DNA sequence design process, Kim *et al.*

has created a sequence design system, which allows DNA sequences to be designed easily by selecting the required objectives. In addition, weights for each objective can be varied such that the more important objective is given a higher weight. The concept of Pareto optimization is used to design this system, NACST/Seq (Nucleic Acid Computing Simulation Toolkit). The objectives include similarity between sequences, H-measure, H-measure in 3'end, GC ratio, continuity or the measure of successive occurrence of the same base, likelihood of forming hairpin secondary structure and melting temperature. Any combination of these objectives may be used in designing DNA sequences for DNA-based computation. Multiple candidate sets are generated by the system, for specific DNA-based computing algorithm.

4.2 Initial DNA pool generation

DNA pool generation is the first experimental step in DNA-based computing. This is an important step as all possible solutions have to be generated in the pool before it undergoes filtration process to find the optimal one. A poorly generated DNA pool may result in the optimal solution not being found or worse if a wrong solution is chosen. There are two commonly used methods for initial pool generation. These are the hybridization-ligation and parallel overlap assembly (POA) methods [23]. The pros and cons of both methods have been evaluated [22], and it is concluded that POA is more suitable for initial pool generation of bigger size problems.

Ibrahim Z *et al.*, 2006 [22]

The two commonly used methods for initial pool generation are hybridization-ligation and parallel overlap assembly (POA) methods. These methods are evaluated by comparing their capability in solving the shortest path problem using direct-proportional length-based DNA computing (DPLB-DNAC). From the results, it is found that POA is better due to the following advantages. Firstly, although both methods are able to produce the correct answer, the hybridization-ligation method requires an additional input of oligos to represent weight. Secondly, initial pool size of POA is about twice that of the hybridization-ligation method when the same amount of initial oligos is used. This is because complementary strands in POA are automatically extended by polymerase. Thirdly, population size can be maintained and decided by varying the initial number of oligos in POA. Finally there is no need for ligation, and therefore phosphorylation of oligos, for POA. Because of that, POA generates the initial pool faster.

Kaplan P D *et al.*, 1997 [23]

Kaplan *et al.* proposed using the technique of parallel overlap assembly (POA) to construct computational DNA library that is more efficient than the serial assembly technique [94]. In POA, an initial pool of ordered, overlapping oligonucleotides is prepared and allowed to anneal. After annealing, the oligonucleotides are extended by DNA polymerase. A pool of molecules representing numbers from 0 to 15 are constructed and used to solve the maximal clique problem. They are represented by four-digit binary numbers. Each digit is divided into two substrings; a position string and a value string of 0 or

1. The number of stages to complete the assembly is significantly less than the serial assembly technique. The former technique requires slightly more than $\ln(L/n)$ stages, compared to L/n stages by the latter technique to complete the assembly where n is the number of digits and L is the DNA sequence length.

The limitation of POA is caused by the fact that DNA polymerase only extends the 3' end of polynucleotides. This results in the situation whereby DNA substrings are getting longer but number of strands does not change. To overcome this limitation, dilution and polymerase chain reaction (PCR) are used. Dilution removes extended DNA strands that do not start from the beginning strands and PCR is then used to duplicate the final pool of complete molecules. Another disadvantage is chain displacement, which can prevent the assembly from reaching the final stage. An overlap length that is not too short is necessary to prevent the problem. A 20 base overlap is found to be desirable. The POA technique is also prone to assembly errors but the authors suspect that the probability is low ("a few percent for $L = 1000$ and fragments of length 16 to 18 bp") and does not appear to affect the results. Kaplan *et al.* also proposed the use of assembly errors such as gene shuffling and *in vitro* evolution to generate biological combinatorial diversity for the study of molecular evolution.

4.3 Polymerase chain reaction (PCR)

Polymerase chain reaction (PCR) is an *in vitro* method to amplify the number of DNA strands. There are two main applications of PCR. Firstly, it is used to generate the initial DNA pool. Secondly, it is used to eliminate wrong solutions during the filtering process. By specifying locations where primers would attach to the DNA, only solutions that fit certain criteria would be amplified.

Primers are short strands of DNA used to initiate replication. They are used to define the starting point of a solution in DNA-based computing. Primers design is an important step of PCR. It could affect the efficiency and accuracy of amplification [95], and in terms affect the filtering process. Polymerization begins from the start point, in the 3' to 5' direction, and stop at the last nucleotide base of a DNA strand. Figure 4-1 and Figure 4-2 show how DNA strands are doubled using PCR after each cycle [96]. During gel electrophoresis, only those amplified solutions (showing a dark band) is selected. This is known as dilution. Research in this area includes improving the accuracy of PCR thermal cycling process [97].

Figure 4-1. Polymerase chain reaction; cycles 1 and 2. DNA strands are represented by arrows running from the direction 5' to 3'. Those from previous cycle are differentiated with the newly synthesized ones by solid and dotted lines respectively. Oligonucleotide primers are characterized by rectangles.

Figure 4-2. Polymerase chain reaction; cycle 3.

Loh Y J *et al.*, 2002 [97]

A significant amount of deoxyribonucleic acid (DNA) copies are needed in DNA-based computing to ensure that the data pool generated is complete. PCR is the process used to rapidly duplicate and produce multiple DNA copies from a small fragment of DNA. Hence it is an integral process of DNA-based computing. However temperature transitions during heating (denaturation and polymerization) and cooling (annealing) during a PCR process are not optimal. These will result in a longer processing time as well as increased possibility of mutations in the DNA strands. Preferably, the time taken for transition should be smooth and close to zero. Loh *et al.* proposed reducing the thermal mass and change of frame material to improve cooling rate, and hence a shorter PCR cycling time can be achieved.

After each cycle, each new DNA double strand separates to become two templates for further synthesis. Therefore after x cycles, there will be 2^x times the original number of DNA strands produced from the original sample.

Lo, Y. M. D. *et al.*, 2006 [96]

Polymerase chain reaction (PCR) is an *in vitro* method to amplify DNA using three basic steps. They are thermal denaturalization of target DNA, primer annealing of synthetic oligonucleotide primers, and extension of annealed primers by DNA polymerase. Suitable temperatures for each of these steps are 95°C, 50 to 60°C and 70 to 74°C respectively. After each cycle, the number of DNA strands is approximately doubled. Each cycle takes about 5 to 6 mins to complete. Therefore, approximately 1 billion pairs of DNA can be produced

within 2 to 3 hours in 30 cycles. This number is more than adequate for DNA-based computing applications.

The process of PCR is highly sensitive. Therefore, it is prone to false-positive results arising from contamination. Contamination can be avoided through proper setup of a PCR laboratory. A PCR laboratory can be divided into three areas: sample preparation stage, PCR setup stage, and post-PCR stage.

PCR has undergone some improvement to enable the process to be analyzed in “real-time” as opposed to “end point” analysis. The advantages include real-time quantification of DNA strands using fluorescence molecules, and monitoring the change in fluorescence during PCR. Shown in Figure 4-3 is a modern PCR machine with real time quantification of DNA strands using fluorescence dyes and light emitting diodes (LEDs). With automation and high speed technology, it is claimed that PCR amplification of high accuracy can be performed in less than 30 minutes. Edit: With recent improvement to the reagents used for PCR such as QIAGEN Fast PCR Cycling Kit, each cycle can be completed in 30 to 60 s, and 1 billion pairs of DNA would take less than an hour to be produced.

Figure 4-3. PCR machine Mastercycler ep realplex (Source: www.eppendorf.com).

4.4 **Affinity purification**

Affinity purification is used to pick out DNA strands containing specific sequences or markers from the DNA pool. Complementary sequences of the markers or tags are first synthesized and attached on the surface of a tube or plate. The DNA pool mixture is then poured into the tube, and those strands with marker sequences would be retained while the rest are washed away. The remaining strands are then detached from their tags and ready for the next step. This process can be repeated for other markers. A similar technique is used in DNA microarray; where up to thousands of markers or gene sequences can be analyzed in one step. Fluorophores are released when complementary samples hybridized to the tags, and the microarray can then be analyzed in the form of a two-dimensional spectral map [98].

4.5 Gel electrophoresis

The remaining DNA strands have different sequences, order and length. The solution is generally designed as the shortest DNA strand. This is singled out using gel electrophoresis. A solid gel is prepared with loading compartments known as wells. The DNA mixture is loaded into one or more of the wells. In one of the well, DNA ladder is loaded. DNA ladder is a mixture of DNA strands of known lengths, which is used as a measure of the sample DNA's length(s). The gel is placed either into a horizontal or vertical container. An electrically conducting buffer is poured into the container and an electric current is passed through the container from one end of container to the other. The setup is such that current flows from the further side of the container towards the wells. The negatively charged DNA strands would then travel along the container, with the shorter and lighter strands making a longer distance due to less resistance.

Figure 4-4. An output image of gel electrophoresis. Label M stands for DNA size marker or ladder (each band is 50 bp starting from the bottom of image) and label “1” shows a high concentration band of DNA strands of 300 bp [26].

Output from gel electrophoresis is a column of short lines or bands, each representing DNA strands of different lengths (Figure 4-4). The shortest DNA strand that travelled the furthest distance is isolated from the group. DNA sequencing is then used to provide a read out to the solution.

4.6 DNA sequencing

DNA sequencing is a method used to provide a read out of DNA sequences in the form of four nucleotides A, C, T and G. Several DNA sequencing methods have been developed in the 20th century but Sanger method [99] is most widely used due to its higher efficiency and lesser chemicals requirement. It is also known as the chain-terminator method. This is because DNA chain terminators, dideoxynucleotide triphosphates (ddNTPs) are used to break up the solution strands into fragments of varying lengths. A mixture of DNA solution strands is first separated into four tubes. In each tube, one of the four types of dideoxynucleotide triphosphates (ddATP, ddCTP, ddGtG, ddTTG) is added. In the tube containing ddATP, fragments of DNA strands terminating at nucleotide A are formed. The same goes for the other respective tubes. The fragments then undergo gel electrophoresis where respective nucleotides can be deciphered based on its length. Recently, new DNA sequencing technologies which are more cost efficient than Sanger's method have been introduced [100]. This would allow the study of genomics to be more accessible to researchers and scientists.

PART III – NOVEL METHODS OF DNA- BASED COMPUTING FOR GRAPH THEORY PROBLEMS

5 SHORTEST PATH PROBLEM

5.1 Problem definition: Shortest path problem

Find the shortest path of travel from one vertex to another. Vertices can represent post office and letter box for a postman. Assuming the postman get paid by the number of deliveries he makes in a day, he would want to take the shortest path to maximize his pay. Other applicable fields include airline industry whereby the route that each airplane takes is planned, such that the total distance of travel required to cover all destinations is minimized. This in turn maximizes the profit. The problem becomes exponentially complex as number of vertices increases. An extension of the shortest path problem is the travelling salesman problem.

5.2 Dijkstra Algorithm

One of the mathematical algorithms used to solve the shortest path problem is Dijkstra algorithm. At the start of the algorithm, all vertices are put into a temporary label group. During the calculation process, vertices that match a certain criteria will be moved into a permanent label group. The algorithm stops when all vertices have been moved into the permanent label group.

The following are steps of Dijkstra algorithm.

1. Select a starting vertex. This vertex is moved to the permanent label group.
2. Label the total distance from selected vertex to each vertex in the temporary label group according to their number.
3. Find the distance from selected vertex to all other connecting vertices within the temporary label group. Add this distance to the respective labels.
4. The vertex belonging to the shortest total distance label is moved to the permanent label group. This vertex is then selected.
5. Repeat steps 2 and 4 until no vertex is left in the permanent label group.
6. The shortest path is the order in which the vertices are put into the permanent group.

5.3 Case study

Figure 5-1. Shortest path problem case study.

The example in Figure 5-1 is used as a case study to compare the efficiency of mathematical algorithm (Dijkstra) to DNA algorithm for solving the shortest path problem. It contains 6 vertices and 13 edges. The objective is to find the shortest path between any pair of vertices.

5.4 Dijkstra Algorithm: Solution walkthrough

Any vertex can be selected as the starting point, in this case vertex 1 is chosen. The Dijkstra algorithm is applied to Figure 5-1 to find the shortest path from vertex 1 to the other five vertices.

Solution: Steps and computations.

1. $L_1 = 0, \tilde{L}_2 = 8, \tilde{L}_3 = 10, \tilde{L}_4 = \infty, \tilde{L}_5 = 5, \tilde{L}_6 = \infty, \quad PL = \{1\}, \quad TL = \{2, 3, 4, 5, 6\}$
2. $L_5 = \min\{\tilde{L}_2, \tilde{L}_3, \tilde{L}_4, \tilde{L}_5, \tilde{L}_6\} = 5, k = 5, \quad PL = \{1, 5\}, \quad TL = \{2, 3, 4, 6\}$
3. $\tilde{L}_2 = \min\{\tilde{L}_2, \tilde{L}_5 + l_{52}\} = \min\{8, 5 + 2\} = 7$
 $\tilde{L}_3 = \min\{\tilde{L}_3, \tilde{L}_5 + l_{53}\} = \min\{10, 5 + 6\} = 10$
 $\tilde{L}_4 = \min\{\tilde{L}_4, \tilde{L}_5 + l_{54}\} = \min\{\infty, 5 + 5\} = 10$
 $\tilde{L}_6 = \min\{\tilde{L}_6, \tilde{L}_5 + l_{56}\} = \min\{\infty, 5 + 2\} = 7$
4. $L_2 = \min\{\tilde{L}_2, \tilde{L}_3, \tilde{L}_4, \tilde{L}_6\} = 7, k = 2, \quad PL = \{1, 2, 5\}, TL = \{3, 4, 6\}$
5. $\tilde{L}_3 = \min\{\tilde{L}_3, \tilde{L}_2 + l_{23}\} = \min\{10, 7 + 2\} = 9$
 $\tilde{L}_4 = \min\{\tilde{L}_4, \tilde{L}_2 + l_{24}\} = \min\{10, 7 + 8\} = 10$
 $\tilde{L}_6 = \min\{\tilde{L}_6, \tilde{L}_2 + l_{26}\} = \min\{7, 7 + 6\} = 13$

6. $L_3 = \min\{\tilde{L}_3, \tilde{L}_4, \tilde{L}_6\} = 9, k = 3, \quad PL = \{1, 2, 3, 5\}, \quad TL = \{4, 6\}$
7. $\tilde{L}_4 = \min\{\tilde{L}_4, \tilde{L}_3 + l_{34}\} = \min\{10, 9 + \infty\} = 10$
 $\tilde{L}_6 = \min\{\tilde{L}_6, \tilde{L}_3 + l_{36}\} = \min\{13, 9 + \infty\} = 13$
8. $L_4 = \min\{\tilde{L}_4, \tilde{L}_6\} = 10, k = 4, \quad PL = \{1, 2, 3, 4, 5\}, \quad TL = \{6\}$
9. $\tilde{L}_6 = \min\{\tilde{L}_6, \tilde{L}_4 + l_{46}\} = \min\{13, 10 + 1\} = 11$
10. $L_6 = \min\{\tilde{L}_6\} = 11, k = 6, \quad PL = \{1, 2, 3, 4, 5, 6\}, \quad TL = \{\}$

5.5 DNA Algorithm: DNA strands design analysis

Two DNA strands are used for representing each edge, one for each direction. There are 11 edges in Figure 5-1, giving a total of $2 \times 11 = 22$ directional DNA strands. In addition, DNA strands are needed to include the possibility that any vertex can be a starting point in the solution. For each vertex, number of DNA strands required is equal to the number of edges from each vertex. For vertex 1, three edges are extending from it; hence three DNA strands are required. This adds up to $3 + 5 + 3 + 3 + 5 + 3 = 22$ (or $2 \times$ number of edges) start DNA strands. An addition edge from each vertex to stop is created to signify end of path. This is necessary so that DNA strands in the mixture will not extend nonstop. This brings the total number of DNA strands to $22 + 22 + 6 = 50$. These strands and templates are presented under appendix (12.112.9 and 12.2).

Number of DNA strands required

$$N = 4e + v$$

where e = number of edges and v = number of vertices (excluding stop).

5.6 Experimental procedure

The following lists the steps for DNA-based computing.

1. Mix a portion of directional, start and stop strands into a container. Allow time for them to react. This forms the DNA pool consisting of all possible solutions. For example, if the start and stop vertices are 1 and 6 respectively, then add the following start and stop strands besides all the directional strands.

2. Use PCR to amplify DNA strands that begin from the start vertex (one primer) and end at the stop vertex (one primer). Only those solutions that contain these start and stop vertices are retained for the next step.
3. Gel electrophoresis to identify shortest strand (solution strand), which represents the shortest path.
4. DNA sequencing to reveal the order of travel (answer).

5.7 Expected result

Figure 5-2. Shortest path problem expected result.

The shortest path between vertex 1 and 6 is found to be {1, 5, 6}. In DNA-based computing, this is represented by the shortest DNA strand (Figure 5-2) with a length of 75bp.

5.8 Discussion

Comparing results of the mathematical algorithm represented by Dijkstra and DNA-based computing algorithm, the following are observed. Number of steps to find the shortest path between vertex 1 and 6 for the former are ten steps or around sixteen arithmetic operations. This number varies for different pairs of start and stop vertices. The number of steps using DNA-based computing to find the solution is four and this is fixed regardless of the pair of vertices chosen.

Time needed to design DNA strands and run the experiments is many times more than that of the mathematical model for this case study. However for large problems, DNA-based

computing could prove to be more time efficient than the mathematical model. The critical number of vertices at which this happens can be further investigated.

DNA-based computing is suitable for solving problems whereby number of vertices is large and positions of these vertices are fixed. The latter means a lesser amount of time is required to design DNA strands for new vertices.

6 SHORTEST SPANNING TREE

6.1 Problem Definition: Shortest spanning tree

A tree is defined as a connected graph that has no cycle or closed path. A spanning tree is one that has connections to all vertices within the graph. A shortest spanning tree is connected by exactly $(v - 1)$ edges. The problem can thus be simplified as finding the shortest total length of $(v - 1)$ edges connecting all vertices.

Applications of shortest spanning trees include the planning and construction of transportation facilities such as railway lines. A proper plan would have the shortest possible railway lines connecting every city. This saves construction cost as well as time of travel between cities after they are built. Other applications include communication networks, such as the laying of television and broadband lines. One algorithm used to find the shortest spanning tree is Kruskal's greedy algorithm.

6.2 Kruskal's Greedy Algorithm

Kruskal algorithm finds the shortest spanning tree by selecting and joining edges starting with the shortest one. The process continues until all vertices are connected. Along the way if a selected edge results in a cycle being formed, that edge is excluded from the

solution. The following are detailed steps of Kruskal algorithm for a connected graph $G = (V, E)$ with edges (i, j) .

1. Arrange edges of G in ascending order of length.
2. Choose edges of T in this order, eliminating an edge if it forms a cycle with those already chosen.
3. Stop when $v - 1$ edges have been selected. Output shortest spanning tree T in G .

6.3 Case Study

Figure 6-1. Shortest spanning tree case study.

The example in Figure 6-1 is used as a case study to compare mathematical algorithm (Kruskal) to DNA algorithm in solving the shortest spanning tree problem. It contains 6 vertices and 9 edges. The objective is to find the shortest spanning tree.

6.4 Kruskal Algorithm: Solution walkthrough

The Kruskal algorithm is used to find the shortest spanning tree for the graph in Figure 6-1.

Solution: Steps and computations.

1. Arrange the edges according to ascending order of length (Table 6-1).
2. The first two edges are always selected. This is because a cycle can only be formed with a minimum of three edges. Edges (3, 4) and (2, 3) are selected (Figure 6-2).
3. Next shortest edge (2, 4) is rejected because it will form a closed path of (2, 3, and 4).
4. Edges (3, 5), (1, 5) and (2, 6) are subsequently selected as they are next in the order of shortest length and do not form a cycle with previously selected edges.
5. The algorithm stops as $n - 1 = 6 - 1 = 5$ edges have been selected.
6. Output shortest spanning tree (Figure 6-3).

Table 6-1. Solution of Figure 6-1.

Edge	Length	Choice
(3, 4)	2	1 st
(2, 3)	4	2 nd
(2, 4)	6	Reject
(3, 5)	6	3 rd
(1, 5)	8	4 th
(2, 6)	10	5 th
(4, 5)	12	
(1, 2)	20	
(1, 6)	30	

Figure 6-2 and Figure 6-3 show the intermediate stages of edge selection. The final solution is shown in Figure 6-1.

Figure 6-2. Kruskal algorithm - Intermediate stages of edge selection.

Figure 6-3. Kruskal algorithm - Final stages of edge selection.

6.5 DNA Algorithm: DNA strands design analysis

Two DNA strands are used for representing each edge, one for each direction. Next select vertex 1 as the start vertex. Therefore there will be no directional strands going towards it. There are 9 edges in Figure 6-1, giving a total of $2 \times 9 - 3 = 15$ directional DNA strands. Rank the edges according to their length; the shortest one is ranked highest. There are a total of eight ranks ($R = 8$). Length of directional DNA strands is designed as

$$l_d = 20 + 10/r$$

where r is number of rank and j is the maximum number of “jumps” expected in the solution. This is elaborated further in the next paragraph.

Vertices of the shortest spanning tree are connected by edges that do not form a cycle (loop). Therefore the solution should contain $v - 1$ number of edges where v is the number of vertices. In DNA-based computing, this is represented by a DNA strand that connects all vertices in a pathway. However this is not possible if the solution has secondary branches. In order to overcome this problem, the path should allow “jumping” from one vertex to another not connected by an edge. This would increase length of the solution DNA strand. If the solution path can go from any vertices to another and back, number of directional DNA strands would be $2 \times \sum_{i=1}^n n_i = 2 \times 21 = 42$. This number is relatively large. In order to reduce this number, a different set of DNA strands design is used rather than the one used for Dijkstra algorithm. Directional DNA strands are joined by an intermediate connector DNA strand, instead of a direct connection between. This allows a path reaching one vertex to

continue its journey from another vertex through “jumping”. The number of connector DNA strands is $2v = 2 \times 6 = 12$.

An addition edge from each vertex to stop is created to signify end of path. This is necessary so that DNA strands in the mixture will not extend nonstop. This brings the total number of DNA strands to $15 + 12 + 6 = 33$. These strands and templates are presented under appendix (12.3 and 12.4).

Number of DNA strands required

$$N = 2e + 3v - e_1$$

where e = number of edges, e_1 = number of edges connected to vertex 1 and v = number of vertices (excluding stop).

6.6 Experimental procedure

The following lists the steps for DNA-based computing.

1. Mix a portion of directional, connector, vertex 1 start and stop strands into a container.
Allow time for them to react.
2. Affinity purification to identify strands containing all seven vertices (including stop).
3. Gel electrophoresis to identify shortest strand (solution strand), which is the solution or one of the possible solution for shortest spanning tree.
4. DNA sequencing to reveal edges of the shortest spanning tree selected (answer).

6.7 Expected result

Figure 6-4. Shortest spanning tree expected result.

The shortest spanning tree is connected by five edges {1-5, 3-5, 3-4, 2-3 and 2-6}. In DNA-based computing, this is represented by the shortest DNA strand as shown in Figure 6-4 with a length of 390bp.

6.8 Discussion

Comparing results of the mathematical and DNA-based computing algorithms, the following are observed.

In the mathematical algorithm, edges are first arranged according to increasing length. After which they are added one by one to form the solution. While the solution is being formed, it has to check if the addition of an edge would result in a cycle. Arranging of edges and checking of cycle formation takes up significant computation time and power compared with basic addition and subtraction operations. This becomes more significant for larger problems. For DNA-based computing, edges are arranged during the design stage. After which, there is no step by step verification of whether a cycle has formed. All possible solutions are created in the DNA pool at about the same time. The solution is obtained in two steps of filtering techniques; affinity purification and gel electrophoresis.

The possible problems DNA algorithm may face are the unpredictability of DNA ligation. For example, connector DNA strands may end up forming loops of “endless” extended strands among themselves. This will result in insufficient directional DNA strands coming into contact with connector DNA strands for a “jump” to occur. Hence the solution DNA strand may not be formed. Experiments are required to calculate the best proportion of each DNA strands and way in which they are mixed together in the DNA pool.

Simplicity of DNA-based computing method in solving the shortest path problem provides an attractive approach to the problem, once the limitations are ironed out.

7 MAXIMUM FLOW PROBLEM

7.1 Problem Definition: Maximum flow problem

A network is a digraph $G = (V, E)$ in which each edge (i, j) has a capacity $c_{ij} > 0$ assigned to it. It has an unique vertex s , called the source, where a flow is produced that run along the edges to another unique vertex t , called the target or sink, where the flow disappears. A digraph is a graph with directional edges (Figure 7-1). The objective is to maximize flow through the network from s to t . Applications include maximizing the flow of water in pipes. This will minimize time required to transfer water from the reservoir or water tank to a designated place thus saving time and cost.

7.2 Ford-Fulkerson Algorithm for Maximum Flow

One of the mathematical algorithms used to solve the shortest path problem is Ford-Fulkerson algorithm. The following are steps of the algorithm.

1. Assign an initial flow from source f_{ij} .
2. Mark source vertex as ϕ , and the rest as “unlabeled”.
3. Start from the source index. Scan each labeled vertex i by labeling each unlabeled vertex j that is adjacent to it. A label consists of the difference between given flow and its

capacity, Δ_{ij} and whether it is a forward or backward flow. If no such vertex j exists then output maximum flow f . Else go to Step 4.

4. Repeat Step 3 until the target t is reached. A flow augmenting path $P: s \rightarrow t$ is specified.

If it is impossible to reach t , then output f . Else go to Step 5.

5. Backtrack path P by means of the labels.
6. Using P , augment the existing flow by Δ_P . Set $f = f + \Delta_P$.
7. Remove all labels from vertices $2, \dots, n$. Go to step 3.

7.3 Case Study

Figure 7-1. Maximum flow problem case study.

The example in Figure 7-1 is used as a case study to compare the mathematical algorithm (Ford-Fulkerson) with DNA algorithm efficiency in finding the maximum flow from source (vertex 1) to sink (vertex 6). It contains 6 vertices and 9 edges.

7.4 Ford-Fulkerson Algorithm: Solution walkthrough

Ford-Fulkerson algorithm is applied to Figure 7-1 to find the maximum flow from vertex 1 through the other four vertices, and exiting from vertex 6.

Solution: Steps and computations.

1. An initial flow $f = 2 + 2 = 4$ is given.
2. Label $s (= 1)$ by $\emptyset$. Mark 2, 3, 4, 5, 6 as “unlabeled”.
3. Scan 1.

Compute $\Delta_{12} = 4 - 2 = 2 = \Delta_2$. Label 2 by $(1^+, 2)$.

Compute $\Delta_{13} = 3 - 2 = 1 = \Delta_3$. Label 3 by $(1^+, 1)$.

4. Scan 2.

Compute $\Delta_{24} = 5 - 3 = 2$, $\Delta_4 = \min(\Delta_2, 2) = 2$. Label 4 by $(2^+, 2)$.

Compute $\Delta_{23} = 2 - 1 = 1$, $\Delta_3 = \min(\Delta_2, 1) = 1$. Label 3 by $(2^+, 1)$.

Compute $\Delta_5 = \min(\Delta_2, 2) = 2$. Label 5 by $(2^-, 2)$.

Scan 3.

Compute $\Delta_{46} = 10 - 4 = 6$, $\Delta_6 = \Delta_t = \min(\Delta_4, 6) = 2$. Label 6 by $(3^+, 2)$.

5. $P: 1 - 2 - 4 - 6 (= t)$ is a flow augmenting path.
6. $\Delta_t = 2$. Augmentation gives $f_{12} = 4$, $f_{24} = 5$, $f_{46} = 6$, other f_{ij} unchanged. Augmented flow $f = 4 + 2 = 6$.
7. Remove labels on vertices 2, ..., 6. Go to Step 3.

3. Scan 1.

Compute $\Delta_{13} = 3 - 2 = 1 = \Delta_3$. Label 3 by $(1^+, 1)$.

4. Scan 2.

[No vertex left for labeling].

Scan 3.

Compute $\Delta_2 = \min(\Delta_3, 1) = 1$. Label 2 by $(2^-, 1)$.

Compute $\Delta_{35} = 6 - 3 = 3$, $\Delta_5 = \min(\Delta_3, 3) = 1$. Label 5 by $(2^+, 1)$.

Scan 5.

Compute $\Delta_{54} = 3 - 1 = 2$, $\Delta_4 = \min(\Delta_5, 2) = 1$. Label 4 by $(3^+, 1)$.

Compute $\Delta_{56} = 1 - 0 = 1$, $\Delta_6 = \min(\Delta_5, 1) = 1$. Label 6 by $(3^+, 1)$.

5. $P: 1 - 3 - 5 - 6 (= t)$ is a flow augmenting path.

6. $\Delta_t = 1$. Augmentation gives $f_{13} = 3$, $f_{35} = 4$, $f_{56} = 1$, other f_{ij} unchanged. Augmented flow $f = 4 + 3 = 7$.

7. Remove labels on vertices 2, ..., 6. Go to Step 3.

No Δ_j exists, output f .

7.5 DNA Algorithm: DNA strands design analysis

There are 9 edges in Figure 7-1. A pair of numbers is used to represent each edge. The first and second numbers stand for capacity and given flow rate through the pipe respectively. The arrow shows the flow direction. For pipe 2-5, flow capacity is 3 units and flow rate is 2 units from vertex 2 to 5. There is room for increasing the flow rate by 1 unit (3 – 2) in this direction. For the opposite direction, i.e. vertex 5 to 2, the maximum room for increment is 2 units.

DNA strands are used for each level of increment and for each direction. For pipe 2-5, 3 DNA strands are used; one for direction 2-5 and two for direction 5-2. An inspection of all pipes reveals that the maximum increment is 6 units ($10 - 4$) for pipe 4-6 in the direction of 4-6. This is followed by 3 units in pipe 3-5 for both directions. Therefore it will be adequate to fix the maximum increment at 3 units, i.e. assume a value of (7, 4) for pipe 4-6. There are a total of 26 increment DNA strands for Figure 4. Length of increment DNA strands is $20 + 10r$, where r is the level of increment. The highest increment is equivalent to the highest rank (1).

Not all pipes will end up having a higher flow rate in the final solution. Some may remain the same while the rest may have a lower flow rate with respect to the flow direction. For those pipes without a change in flow rate, an additional DNA strand type is necessary to indicate this. One indicator DNA strand is needed for each edge. The number of indicator DNA strands is thus equal to 9. Length of each indicator DNA is designed such that a significantly high cost is assigned, if an edge is not used as part of the solution. This is to encourage the algorithm to explore all possible edges for maximum flow. The indicator length nc is designed as follows.

$$nc = e \times [(max\ increment\ length + 10)]$$

$$nc = 9 \times [(30 + 10)] = 360$$

The solution may comprise of a number of sub solutions that extends from source (vertex 1) to sink (vertex 6). In order to include this possibility, a link DNA strand is used. This DNA strand connects the sink to source vertex.

An addition edge from the sink to stop vertex is created to signify end of path. This is necessary so that DNA strands in the mixture will not extend nonstop. This brings the total number of DNA strands to $26 + 9 + 1 + 1 = 37$. These strands and templates are presented under appendix (12.5 and 12.6).

7.6 Experimental procedure

The following lists the steps for DNA-based computing.

1. Mix a portion of directional (includes source and sink), indicator, link and stop strands into a container. Allow time for them to react.
2. Affinity purification to identify strands containing all nine edges and one stop.
3. Gel electrophoresis to identify shortest strand (solution).
4. DNA sequencing to reveal flow at each edge to achieve maximum flow.

7.7 Expected result

Figure 7-2. Maximum flow problem expected result.

Maximum flow is achieved by increasing two flow units to edges **a**, **d**, **h** and one flow unit to edges **b**, **f**, **g**. Edges **c**, **e** and **g** remain unchanged. In DNA-based computing, this is represented by the shortest DNA strand shown in Figure 7-2 with a length of 1350bp.

7.8 Discussion

Comparing results of the mathematical algorithm and DNA-based computing algorithm, the following are observed. It takes the mathematical algorithm approximately two rounds of seven steps to find the maximum flow solution. For DNA-based computing, it takes four number of steps. Moreover, if the solution requires many rounds of augmentation, number of steps will be significantly greater than that for DNA-based computing.

Despite small number of steps needed to get the solution, time taken to run the wet lab experiment makes DNA-based computing impractical for problems with small number of vertices and edges. For larger sized problems with more vertices and intersecting edges, DNA-based computing may be more efficient.

Possible problems that DNA-based computing may face include unpredictability of DNA ligation. For example, non thorough mixing of DNA strands such as the indicator strands may result in them forming loops of “endless” extended strands among themselves. Hence the correct solution may not be found. Experiments are required to get the best proportion of each DNA strands, and the way they are mixed together in the DNA pool.

8 BIPARTITE MAXIMUM CARDINALITY PROBLEM

8.1 Problem Definition: Bipartite Maximum Cardinality

In a work environment, workers are assigned to jobs in a way that would ensure optimal work productivity is achieved. Similarly, time period between examinations are set such that students will have enough rest and preparation time before their next paper. And this is done while making sure that the whole examination can be carried out with a given time frame. These are examples of assignment problems and they can be further classified as bipartite assignment problem. Bipartite can be defined as an agreement between two parties or groups. To find the maximum possible number of matches between elements of two independent groups, which have the same number of elements, is known as the bipartite maximum cardinality problem (Figure 8-1).

Figure 8-1. Bipartite maximum cardinality between groups S and T; each having 3 elements.

8.2 Bipartite Maximum Cardinality Matching Algorithm

The bipartite maximum cardinality matching algorithm finds the maximum number of matches between two groups, by looking for augmenting paths (Figure 8-2 (b)). An augmenting path can be loosely defined as an extension of an alternating path (Figure 8-2 (a)). In this context, an augmenting path is one that starts from an exposed/unexposed vertex and ends at another similar one. An exposed vertex is one that is not matched to another vertex, shown as dashed lines in Figure 8-2. If an augmenting path starting from an exposed vertex is found, then the connecting edges are reversed. This way, number of unexposed vertex will increase by one. Figure 8-2 (c) shows the resulting augmenting path with four matches, which is one higher than the previous one (Figure 8-2 (b)).

(a) Alternating path

(b) Augmenting path

(c) Augmenting path (reverse of (b))

Figure 8-2. Alternating and augmenting paths.

The following are steps of the bipartite maximum cardinality matching algorithm for a bipartite graph $G = (S, T; E)$, where S and T are bipartite groups and E is the number of edges.

1. Label exposed vertices in S as empty set. Output maximum cardinality matching M if there is no exposed vertex in S .
2. For each element i in S and edge (i, j) not in M , label j with i , unless already labeled.
3. For each nonexposed j in T , label i with j , where i is the other end of the unique edge (i, j) in M .
4. Backtrack alternating paths P ending on an exposed vertex in T by using labels on the vertices.
5. If no P in Step 4 is augmenting then output M . Else augment M by using an augmenting path P . Remove all labels and go to step 1.

8.3 Case Study

Figure 8-3. Bipartite maximum cardinality problem case study.

The example in Figure 8-3 is used as a case study to compare bipartite maximum cardinality matching algorithm to DNA algorithm. It contains 6 vertices and 6 edges. Edges represent possible matches between vertices, while the darker edges are those matches in M . The objective is to find the highest number of matches between vertices in S and T . Each vertex in S can only be matched to another one in T at most once, and vice versa.

8.4 Bipartite Maximum Cardinality Matching Algorithm: Solution walkthrough

The bipartite maximum cardinality matching algorithm is applied to Figure 8-3 to find the highest number of matches between vertices in S and T . The original matching $M_1 = \{(1,4), (3,6)\}$.

Solution (a): Steps and computations.

1. Label 2 with $\emptyset$.
2. Label 6 with 1. Label 4 and 5 with 2.
3. Label 1 with 4, and 3 with 6.

[All vertices are now labeled as shown in Figure 8-4.]

4. $P_1: 2 - 4 \rightarrow 1 - 6 \rightarrow 3$. [By backtracking, P_1 is not augmenting.]
5. There is no augmenting path.

Figure 8-4. Bipartite algorithm solution walkthrough – no augmenting path.

There is in fact an augmenting path in Figure 8-3. However, it is not found by the algorithm due to labeling of vertices in Step 2. Vertices in T are labeled starting from the smaller ones in S, i.e. 1, 2 then 3. If labeling is done on the reverse, i.e. starting from vertices 3 to 1, then a maximum cardinal match can be found. This is shown in the following solution.

Solution (b): Steps and computations.

1. Label 2 with 0.
2. Label 5 with 3. Label 4 with 2. Label 6 with 1.
3. Label 1 with 4, and 3 with 6.

[All vertices are now labeled as shown in (a).]

4. $P_1: 2 - 4 \rightarrow 1 - 6 \rightarrow 3 - 5$. [By backtracking, P_1 is augmenting.]
5. Augment M_1 by using P_1 , dropping (1, 4) and (3, 6) from M_1 and including (2, 4), (1, 6) and (3, 5). Remove all labels. Go to Step 1.

[Updates shown in Figure 8-5 (b), $M_2 = \{(1,6), (2,4), (3,5)\}$]

6. There is no exposed vertex in S. Output M_2 .

Figure 8-5. Bipartite algorithm solution walkthrough – augmenting path.

Note that an alternative and simpler solution would be to add edge (2, 3) to M_1 , i.e. $M_2 = \{(1,4), (2,5), (3,6)\}$. However, the bipartite maximum cardinality matching algorithm is not able to find this solution as there is no edge from vertex 5 that will lead to an augment path. The solution is thus dependent on the initial matching of M_1 .

8.5 DNA Algorithm: DNA strands design analysis

Two DNA strands are used for representing each edge, one for each direction. There are 6 edges in Figure 8-3, giving a total of $2 \times 6 = 12$ directional DNA strands. In addition, DNA strands are needed to include the possibility that any vertex in S group can be a starting point in the solution. For each vertex, number of DNA strands required is equal to the number of edges extending it. For vertex 1, there are two edges. Hence two DNA strands are required. This adds up to $2 + 2 + 2 + 2 + 2 + 2 = 12$ (or $2 \times$ number of edges) start DNA strands. Another type of DNA strands is needed to connect individual pairs of answers together. For S group vertices, the connectors are 1-2, 1-3, 2-1, 2-3, 3-1 and 3-2 = 6 top row connectors (or 3!). Similarly, there are 6 DNA strands for T connectors. An addition edge from each vertex to stop is created to identify end of the path. This is necessary so that DNA strands in the mixture will not extend nonstop. This brings the total number of DNA strands to $12 + 12 + 12 + 6 = 42$.

Number of DNA strands required

$$N = 4e + 2e + v$$

where e = number of edges and v = number of vertices (excluding stop). These strands and templates are presented under appendix (12.7 and 12.8).

8.6 Experimental procedure

The following lists the steps for DNA-based computing.

1. Mix a portion of directional, connector, start and stop strands into a container. Allow time for them to react. This will form the DNA pool consisting of all possible solutions.
2. Use PCR to amplify DNA strands that begin from the start vertices (six primers) and end at the stop vertex (one primer). Only solutions that contain these start and stop vertices are retained for the next step.
3. Affinity purification to identify strands containing all seven vertices (including stop).
4. Gel electrophoresis to identify shortest strand (solution strand), which is the solution or one of the possible solutions (if there are more than one).
5. DNA sequencing to reveal the maximum pair of matches.
6. Answer: Pair of odd to even number vertices, till stop vertex or till vertex pair contains repeat vertex.

8.7 Expected result

Figure 8-6. Bipartite maximum cardinality problem expected result.

Figure 8-6 shows the bipartite maximum cardinality solution of three matches $\{(1, 4), (2, 5), (3, 6)\}$ with a length of 170bp.

8.8 Discussion

Comparing results of the mathematical algorithm and DNA algorithm, the following are observed. Solution obtained by the mathematical algorithm is dependent on the order in which vertices are labeled. On the other hand, the DNA algorithm does not require this step to get the solution. Therefore in order to find if a solution or better one exist in the former algorithm, it has to be run several times with a different order by which the vertices are labeled.

The DNA algorithm can be easily expanded to find tripartite maximum cardinality matches, a maximum number of cardinal matches between three groups, or above. This is done by including all possible interactions among vertices. Time used to find the solution will be similar to that for bipartite maximum cardinality except during the initial DNA strands design step. This is so as more DNA strands are required to represent an increased number of vertices and their interactions. However for the mathematical algorithm, processing time and power will increase exponentially to solve higher dimensional problems, which have a larger number of vertices and interactions.

PART IV – EXPERIMENT ON GLOBAL POSITIONING SYSTEM PROBLEM

9 GLOBAL POSITIONING SYSTEM PROBLEM

9.1 Problem definition: Global Positioning System problem

The Global Positioning System (GPS) problem is one application using algorithm of the shortest path problem. It is similar to the traveling salesman problem, except that there is no requirement for all the vertices to be visited. Hence the solution can consists of any number of vertices in between, that is available. An experiment was designed and run to prove the concept of DNA-based computing using the GPS problem. The following describes the case study and design of experiment.

9.2 Case study

Figure 9-9-1. Global Positioning System case study.

The example (Figure 9-9-1) above consists of 6 vertices and 8 edges. The objective is to find the GPS path between any two given vertices. The following table shows the GPS distance and path for each pair of start and stop vertices.

Table 9-1. GPS distance and path for the 6 vertices.

Vertices	1	2	3	4	5	6
1	0	46 (1, 2)	90 (1, 5, 3)	62 (1, 5, 4)	32 (1, 5)	84 (1, 2, 6)
2	46 (2, 1)	0	42 (2, 3)	32 (2, 4)	62 (2, 4, 5)	38 (2, 6)
3	90 (3, 5, 1)	42 (3, 2)	0	36 (3, 4)	38 (3, 5)	80 (3, 2, 6)
4	62 (4, 5, 1)	32 (4, 2)	36 (4, 3)	0	30 (4, 5)	40 (4, 2, 6)
5	32 (5, 1)	62 (5, 4, 2)	38 (5, 3)	30 (5, 4)	0	116 (5, 1, 2, 6)
6	84 (6, 2, 1)	38 (6, 2)	80 (6, 2, 3)	40 (6, 2, 4)	116 (6, 2, 1, 5)	0

9.3 DNA Algorithm: DNA strands design analysis

Two DNA strands are used to represent each edge, one for each direction. There are 8 edges in Figure 9-1, hence a total of 16 directional strands. The length of each directional strand is used to represent the distance between two vertices, i.e., the length of directional strand 1 to 2 is 46 bp. These are double stranded DNA with sticky ends of 10 bp on each side,

so that vertex strands can bind to them. For the GPS problem, any vertex can be a start point, as well as a stop point. 6 vertex strands are used to represent the 6 vertices. This brings the total number of strands to $16 + 6 = 22$ strands. These strands and templates are presented under appendix (12.9 and 12.10).

9.4 Experimental procedure

The following lists the steps for DNA-based computing.

5. Mix a portion of directional and vertex strands into a tube. Allow time for them to hybridize.
6. Add kinase and ligase enzymes to seal up the connection among hybridized strands.
7. PCR to pick out the possible path(s) between the selected vertices.
8. Gel electrophoresis to identify shortest strand (solution strand), which is the solution for the GPS problem.
9. DNA sequence to reveal the edges involved, and hence the shortest path of travel for the GPS problem (answer).

9.5 Expected result

Figure 9-9-2. Global Positioning System expected result.

The GPS selected path between vertex 1 and 4 is found to be {1, 5, 4}. In DNA-based computing, this is represented by the shortest DNA strand (Figure 9-9-2) with a length of 82 bp. This length is 20 units longer than the GPS distance (62 units) due to the additional 10 bp on each side of the strand.

9.6 Discussion

A modified design of DNA strands was used compared to the one on Shortest Path problem (Chapter 5). There were no unique start and stop strands. This means that DNA pool formed during the hybridization step can be much longer than the solution strand, since they can connect to the nearest neighbouring strand with complimentary ends. However, these long strands could, and most probably would contain the solution strands of selected start and end points. And these solution strands and be filtered out using PCR. This

design is more elegant and simple to create as only the directional and vertex strands need to be considered.

9.7 Materials and Methods

A comprehensive list of working steps includes hybridization, phosphorylation, ligation, purification, polymerase chain reaction (PCR) and DNA strands separation using gel electrophoresis for viewing output results.

9.7.1 Hybridization and phosphorylation of DNA strands to create DNA pool

DNA pool was first created by mixing an amount of unique DNA strands from each tube. These contained single stranded DNA (ssDNA) of approximately 50 μM , and duplex DNA strands of approximately 100 μM , representing vertices and edges respectively. For duplexes, amount of DNA added were inversely proportional to their lengths as follows. This ensures that shorter paths are preferred.

DNA pool (Total 22 μl)

- Tubes 1 to 6 (ssDNA, 20 bp): 1.00 μl
- Tubes 7, 8 (duplexes, 46 bp): 0.78 μl
- Tubes 9, 10 (duplexes, 32 bp): 1.13 μl
- Tubes 11, 12 (duplexes, 42 bp): 0.86 μl
- Tubes 13, 14 (duplexes, 32 bp): 1.13 μl

- Tubes 15, 16 (duplexes, 38 bp): 0.95 μ l
- Tubes 17, 18 (duplexes, 36 bp): 1.00 μ l
- Tubes 19, 20 (duplexes, 38 bp): 0.95 μ l
- Tubes 21, 22 (duplexes, 30 bp): 1.20 μ l

The resultant 22 μ l mixture was subjected to a process known as hybridization. This is where complementary ssDNA, DNA duplexes with sticky ends would readily anneal via non-covalent hydrogen bonds to form extended duplexes. 2.4 μ l of 10X ligation buffer was added to the mixture and left at room temperature for 15 min.

The 24.4 μ l mixture then underwent a process of phosphorylation, where phosphates are added to the 5' end of DNA strands for ligation later. 2 μ l of NEB kinase was added to the mixture and incubated for 1 hr at 37 °C.

9.7.2 Ligation of DNA strands

After phosphorylation, DNA strands are held together by weak hydrogen bonds within the DNA pool. In order to ensure that the strands stay together, covalent bonds are created by adding 2 μ l of NEB T4 DNA ligase. These are enzymes that help form strong covalent bonds among annealed DNA strands. The mixture was incubated at 25 °C for 20 min to complete the ligation process.

9.7.3 Purification to remove ssDNA, short DNA (less than 50 bp), enzymes and impurities

After taking 2 µl of the ligated mixture for concentration analysis, the remaining 26.4 µl was then pipetted into a 0.2 ml tube. The QIAquick® PCR Purification Kit was used for the following step. 132 µl of Buffer PB, or 5 times the mixture volume, was added to the tube and mixed. It takes three steps to purify the mixture; bind, wash and elute. The mixture was poured into a provided QIAquick column and attached to a 2 ml collection tube. A centrifuge was used to centrifuge the sample for 60 s @ 13000 rpm. This would bind large sized DNA of 50 bp and above to the filter. The flow-through was discarded and the QIAquick column was placed back in the same tube.

The next step is to wash unwanted part of the mixture away. 0.75 ml Buffer PE was added to the QIAquick column and centrifuged for 60 s. The flow-through was discarded and the QIAquick column placed back in the same tube. The QIAquick column was centrifuged once more in the 2 ml collection tube for 1 min to remove residual wash buffer. The QIAquick column was then placed in a clean 1.5 ml microcentrifuge tube.

50 µl Buffer EB was added to the membrane at center of the QIAquick column, left at room temperature for 1 min, and centrifuged to elute DNA. The final elute contained 40 to 50 µl of purified DNA strands.

9.7.4 PCR to amplify solution strands

Elute from purification contains all possible solutions of the problem. PCR is used to amplify the solution needed. TaKaRa PCR Thermal Cycler Dice and QIAGEN Fast Cycling PCR Kit were used for this step. 14 μl of reagents for PCR including 10 μl of PCR Master Mix and 4 μl of Q solution were prepared. 2 μl of each primer of approximately 100 μM representing the start and end of path were added to the mixture. 2 μl from the purified DNA pool was then added to make a mixture of 20 μl for PCR. The following PCR protocol was used.

PCR Protocol

- Step 1: 95 °C for 5mins
- Step 2: 96 °C for 5 s (denaturalization)
- Step 3: 48 °C for 5 s (primers annealing)
- Step 4: 68 °C for 5 s (polymerase extension)
- Step 5: Repeat step 2 to 4 for 29 (30 – 1) more time
- Step 6: 72 °C for 1min
- Step 7: 25 °C forever

Strands representing path(s) of start and end points from their respective forward and reverse primers were amplified.

9.7.5 Separation and quantification of DNA strands for solution readout

Product from PCR was viewed using native polyacrylamide gel electrophoresis or PAGE. A 12 % gel was prepared according to the following recipe (Figure 9-3).

Gel recipe

- H ₂ O:	15.7 ml
- 30 % acrylamide:	16.0 ml
- 5 X TBE buffer:	8.0 ml
- 10 % ammonium persulfate:	0.28 ml
- TEMED:	0.026 ml

Figure 9-3. Native PAGE setup.

The gel was run for 1h 45 min at a fixed power of 18 W, and with variable voltage and current at approximately 560 V and 32 mA. The gel was then stained in 3X Gel Red for 30

min, and viewed using 2D multi-fluorescence scanner Typhoon 9410. The *Green-excited mode*, where dyes are excited at 532 nm, was used.

9.8 Results and Discussion

9.8.1 Results

Result of the Native PAGE gel is shown in [Figure 9-4](#). An 82 bp strand with the exact sequence representing the shortest path from points 1 to 4 (Path 1-5-4) was loaded in Lane 2. This was used as a comparison for the shortest strand produced in Lane 4.

In Lane 3, DNA pool after purification is shown. All possible solutions and their distributions are represented here.

In Lane 4, primers that represent the start and end points of 1 and 4 were added. A number of bands with the shortest one in the region of 80 to 90 bp are observed. When compared to Lane 2, this is the shortest path (Path 1-5-4) of 82 bp.

In Lane 5, primers that represent the start and end points of 6 and 4 were added. One significant band is seen in the region of 90 bp, and it coincides with the shortest path (Path 6-2-4) of 90 bp.

In Lane 6, primers that represent start and end points of 6 and 5 were added. Two visible bands are seen. The shorter band is in the region of 120 bp, and it coincides with the shortest path (Path 6-2-4-5) of 120 bp.

Figure 9-4. Native PAGE gel result of GPS problem.

9.8.2 Discussion

The shortest path was successfully produced for each of the chosen start and end points. This is despite the solution for points 6 to 5 on Lane 6 showing a rather weak band.

This could be attributed to the non-homogenous mixing of each unique DNA strands before hybridization took place.

The result in Lane 3 shows a successful process of hybridization, phosphorylation and ligation to produce a DNA pool, which consists of a wide range of possible paths of travel, among the vertices. Majority of these paths are within 60 to 200 bp.

Besides the shortest path, the next best choice of travel could be seen in lanes 4 and 6. They were represented by DNA strands of 98 bp for Path 1-2-4 in Lane 4, and 136 bp for Path 6-2-1-5 in Lane 6.

10DISCUSSION AND CONCLUSION

10.1 Discussion

From simulated results, DNA-based computing is able to solve all four categories of graph theory problems. In all four problems, DNA-based computing would have taken a longer time to solve due to time consuming wet-lab experiments. These include PCR, gel electrophoresis and DNA sequencing that usually takes up to a few hours. Recently, researchers had tried to work around this problem with proposed algorithms for simulating these wet lab experiments using computer architecture [101]. However, as technology for DNA-based computing improves, more competitive experimental timing for these processes can be achieved. Until then, machine and technology improvements are not the focus of this research. Instead the focus is on comparing the efficiency and simplicity of the mathematical and DNA algorithms in solving these problems.

In the shortest path problem, DNA algorithm outperforms Dijkstra algorithm. The latter uses arithmetic operations such as addition, comparison, storing and updating of data for solving the problem. There are at least ten steps taken to arrive at the solution. For DNA-based computing, there are four steps regardless of the number of vertices and edges; generating DNA pool, PCR, gel electrophoresis and DNA sequencing.

In the shortest spanning tree problem, there are three main steps using Kruskal's Greedy algorithm; arranging edges according to length, add edge as part of solution and rejecting the edge if it forms a cycle with previously selected ones. For DNA-based computing, there are four steps regardless of the number of vertices and edges; generating

DNA pool, affinity purification, gel electrophoresis and DNA sequencing. Kruskal algorithm seems to be less complex than DNA algorithm in this problem. However, if the problem size is large, the latter would be a better choice. The time taken to check for cycles as each edge is added increases exponentially with the number of vertices added.

In the maximum flow problem, Ford-Fulkerson algorithm uses addition, subtraction, comparison, storing and updating of data, and backtracking to get the solution. This amounts to more than ten mathematical operations. For DNA-based computing, the steps are similar to that for shortest spanning tree; generating DNA pool, affinity purification, gel electrophoresis and DNA sequencing. However, design of DNA strands for maximum flow problem is relatively more tedious as compared to the other problems. In Ford-Fulkerson algorithm, whenever an augmented path is found, network edges are updated with new flow rates. After that, the calculation is repeated to find another augmented path. For DNA algorithm, all calculations are done in one step during DNA pool generation. In order to ensure that all possible solutions are generated in the DNA pool, more DNA strands are used to represent each edge. This is especially so for edges with high capacities but low given flow rate. In this problem, Ford-Fulkerson algorithm has an edge over DNA algorithm.

In the bipartite maximum cardinality problem, the mathematical algorithm uses arithmetic operations including comparison, storing and updating of data, and backtracking. For DNA-based computing, these are translated into five steps; generating DNA pool, PCR, affinity purification, gel electrophoresis and DNA sequencing. There is an additional step compared with previous DNA algorithms. Comparatively, there is also a limitation with the mathematical algorithm due to the augmenting path technique. Because of this, some solutions cannot be derived using the mathematical algorithm. There is however no such

limitation in DNA algorithm. And if there is more than one solution, they would be in the final solution pool and reflected during DNA sequencing.

Number of DNA strands required for DNA pool generation varies among the four problems with same number of vertices. DNA design strands required are 50, 36, 37 and 42 for shortest path, shortest spanning tree, maximum flow and bipartite maximum cardinality matching problems respectively. Shortest path problem requires greatest number of DNA design strands to include the possibility that any vertex can be a starting vertex. Design of DNA strand is most challenging for maximum flow problem. This is because the maximum flow increment for all pipes has to be found and thereafter their increments have to be ranked. The number of DNA strands needed is also directly affected by pipe capacities and given flow rates.

For the GPS experiment, results agree with simulated results of the shortest path problem. GPS paths of all three selected pairs of vertices have been successfully found using DNA-based computing. Experimental protocols can be optimized further to improve the quality of results.

10.2 Limitations

The advantage of using DNA-based computing is massive parallel processing whereby many DNA strands react together at the same time. However, there are limitations with DNA-based computing to be addressed. On top of this, as with many wet-lab experiments,

there exist the possibilities of both human and experimental errors. Some of the limitations are listed as follows.

10.2.1 Experimental limitations

The following example demonstrates the experimental limitation, in terms of number of vertices possible, with algorithm of this nature.

Take the flight paths of Singapore Airlines to 62 major cities (Source: Wikipedia) as an example of a travelling salesman problem. There is a maximum of $\sum_{m=1}^{n=62} i = \frac{(n+1-m)(n+m)}{2} = \frac{(62+1-1)(62+1)}{2} = 1953$ connecting paths, although the actual number is much less (< 100) as majority of the flight paths are to and from Singapore only.

The objective is to find the shortest path of travel to all major cities once. With 62 cities and 1953 connecting paths, the number of possible paths is $62! \approx 3.15 \times 10^{85}$. Even with the latest supercomputer, it will take a very long time to calculate the distances of all possible paths, and then find the shortest one.

On the other hand, using DNA-based computing, the problem could be simplified to designing $62 + 1953 = 2015$ unique DNA strands that would only bind with their respective counterparts. However, there is the challenge of using large numbers of long DNA strands, where mismatch binding could occur. Another more important issue would render this method impossible, i.e. the amount of DNA required [102]. Mathematically, to have 2015 unique DNA strands would require each DNA strands to have a minimum length of

$$l_{\min} = \frac{85 \log 10}{\log 4} \approx 141.2 \text{ or } 142 \text{bp. Assuming DNA of length } 150 \text{bp is used, a complete path}$$

consisting of traveling to all cities once would be as follows.

$$l_{\text{complete}} = (62 + 61) \times 150 = 18450 \text{bp}$$

Ensuring that the DNA pool would contain about 100 strands of each complete path, the total amount of DNA needed in terms of number of base pairs is then

$$\text{DNA}_{\text{needed}} = 18450 \times 100 \times 3.15 \times 10^{85}$$

$$\text{DNA}_{\text{needed}} \approx 5.81 \times 10^{91} \text{bp}$$

Since each DNA base pairs is approximately 660 Daltons or atomic mass unit, the weight of DNA needed is

$$\text{DNA}_{\text{needed}} \approx 5.81 \times 10^{91} \times 660 \times 1.66 \times 10^{-27}$$

$$\text{DNA}_{\text{needed}} \approx 6.37 \times 10^{67} \text{kg}$$

Hence there is a limitation as to how far DNA-based computing using this algorithm can go. A backward calculation reveals that a traveling salesman problem of up to 19 cities is solvable with 209 unique DNA strands of 100bp each, using 10kg of DNA.

With improvement in bio-chemical technologies and new techniques, new algorithms may evolve to solve this type of problems in the near future.

10.2.2 Human and experimental errors

Mismatch binding among non-complementary DNA strands may result. This happens when a high proportion of their sequences are similar. In order to prevent or minimize the possibility of this from happening, DNA sequence encodings and lengths are carefully designed to fall within a certain range of criteria [31, 103]. This limits the number of DNA templates possible and thus size of the problem. Yang C. N. *et al.* [104] have proposed the use of a feedback system to overcome this problem. Another way of preventing such error is to do away with DNA binding altogether. This can be done by designing algorithms using restriction enzymes to filter out sequences as opposed to hybridization separation [105].

Human error may result in the form of an incomplete DNA pool due to poor mixing of DNA strands and/or an incorrect amount of concentrations added.

10.2.3 NP hard problems

For NP hard problems, number of DNA templates needed increase exponentially as size of problem increases. Nevertheless, once the set of DNA strands required for a particular size problem has been designed, variable solutions to that problem can be easily found using the same or similar templates, but with different parameters.

10.2.4 Irreversible

Chemicals used in DNA-based computing cannot be reused. Although DNA strands can be recovered after the solution is found, the additional chemical processes needed made it not feasible. However once the DNA strands required has been designed and produced using PCR, cost of running the experiment can be kept at a relatively low cost. Alternatively, an enzymatic methodology as opposed to molecular biological techniques for DNA-based computing may be explored. This would eliminate the need for PCR and gel electrophoresis, thus reducing the chemicals required and computing time [106].

10.3 Conclusion

Organisms perform biological computation at the molecular level, much as in how humans compute in their daily lives. DNA-based computing can be seen as a bridge between them. A better understanding of biological computation would thus lead to a better understand of the mechanisms of life. The potentials of DNA-based computing have been discussed and demonstrated in this research. The next step would be to expand the research on these ideas so that a more powerful computer, an automatic medical diagnostic device, a cure for cancer or even a self-regeneration human gene could be realized.

11 REFERENCE LIST

1. Shu, J.J., Q.W. Wang, and K.Y. Yong, *DNA-Based Computing of Strategic Assignment Problems*. Physical Review Letters, 2011. **106**(18).
2. Shu, J.J., K.Y. Yong, and W.K. Chan, *Lecture Notes in Computer Science: Multiple DNA Sequence Alignment Using Joint Weight Matrix*, in *Computational Science and Its Applications - Iccsa 2011, Pt Iii*, B. Murgante, et al., Editors. 2011, Springer-Verlag Berlin: Berlin. p. 668-675.
3. Shu, J.J., K.Y. Yong, and W.K. Chan, *An Improved Scoring Matrix for Multiple Sequence Alignment*. Mathematical Problems in Engineering, 2012.
4. Adleman, L.M., *Molecular computation of solutions to combinatorial problems*. Science, 1994. **266**(5187): p. 1021-1024.
5. Danchin, A., *Bacteria as computers making computers*. Fems Microbiology Reviews, 2009. **33**(1): p. 3-26.
6. Belousov, B.P., *A periodic reaction and its mechanism*. Compilation of Abstracts on Radiation Medicine, 1959: p. 147:145.
7. Adamatzky, A. and B. De Lacy Costello, *Experimental logical gates in a reaction-diffusion medium: The XOR gate and beyond*. Physical Review E, 2002. **66**(4): p. 046112.
8. Moore, G.E., *Cramming more components onto integrated circuits (Reprinted from Electronics, pg 114-117, April 19, 1965)*. Proceedings of the Ieee, 1998. **86**(1): p. 82-85.
9. Fuechsle, M., et al., *A single-atom transistor*. Nature Nanotechnology, 2012. **7**(4): p. 242-246.

10. Ma, X., et al. *Reversible gates and testability of one dimensional arrays of molecular QCA*. in *21st IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*. 2006. Arlington, VA: Springer.
11. Tarasov, V.E., *Quantum computer with mixed states and four-valued logic*. Journal of Physics a-Mathematical and General, 2002. **35**(25): p. 5207-5235.
12. You, J.Q., J.S. Tsai, and F. Nori, *Scalable quantum computing with Josephson charge qubits*. Physical Review Letters, 2002. **89**(19): p. 4.
13. Sinha, S., et al., *Exploiting the effect of noise on a chemical system to obtain logic gates*. Epl, 2009. **86**(6): p. 5.
14. Benenson, Y., *Biocomputers: from test tubes to live cells*. Molecular Biosystems, 2009. **5**(7): p. 675-685.
15. Mehra, R., S. Jaiswal, and H.K. Dixit, *Optical computing with semiconductor optical amplifiers*. Optical Engineering, 2012. **51**(8).
16. Smit, M., J. van der Tol, and M. Hill, *Moore's law in photonics*. Laser & Photonics Reviews, 2012. **6**(1): p. 1-+.
17. Seneor, P., et al., *Spintronics with graphene*. Mrs Bulletin, 2012. **37**(12): p. 1245-1254.
18. Watson, J.D., et al., *Molecular biology of the gene*. 2004. **Fifth Edition**.
19. Macgregor, R.B. and G.M.K. Poon, *The DNA double helix fifty years on*. Computational Biology and Chemistry, 2003. **27**(4-5): p. 461-467.
20. Falcón de Vargas, A., *The Human Genome Project and its importance in clinical medicine*. International Congress Series, 2002. **1237**: p. 3-13.
21. Faulhammer, D., et al., *Molecular computation: RNA solutions to chess problems*. Proceedings of the National Academy of Sciences of the United States of America, 2000. **97**(4): p. 1385-1389.

22. Ibrahim, Z., Y. Tsuboi, and O. Ono, *Hybridization-ligation versus parallel overlap assembly: An experimental comparison of initial pool generation for direct-proportional length-based DNA computing*. Ieee Transactions on Nanobioscience, 2006. **5**(2): p. 103-109.
23. Kaplan, P.D., et al., *Parallel overlap assembly for the construction of computational DNA libraries*. Journal of Theoretical Biology, 1997. **188**(3): p. 333-341.
24. Sakamoto, K., et al., *Molecular computation by DNA hairpin formation*. Science, 2000. **288**(5469): p. 1223-1226.
25. Ouyang, Q., et al., *DNA solution of the maximal clique problem*. Science, 1997. **278**(5337): p. 446-449.
26. Lee, J.Y., et al., *Solving traveling salesman problems with DNA molecules encoding numerical values*. Biosystems, 2004. **78**(1-3): p. 39-47.
27. Yin, Z.X., F.Y. Zhang, and J. Xu, *A chinese postman problem based on DNA computing*. Journal of Chemical Information and Computer Sciences, 2002. **42**(2): p. 222-224.
28. Feldkamp, U., R. Hilmar, and B. Wolfgang, *Software Tools for DNA Sequence Design*. Genetic Programming and Evolvable Machines, 2003. **4**(2): p. 153-171.
29. Feldkamp, U., et al., *DNASquenceGenerator: A Program for the Construction of DNA Sequences*, in *DNA Computing2002*. p. 23-32.
30. Kim, D., et al., *NACST/Seq: A sequence design system with multiobjective optimization*. DNA Computing, 2003. **2568**: p. 242-251.
31. Tanaka, F., et al., *Design of nucleic acid sequences for DNA computing based on a thermodynamic approach*. Nucleic Acids Research, 2005. **33**(3): p. 903-911.
32. ZhiXiang, Y., Z. Fengyue, and X. Jin, *The general form of 0-1 programming problem based on DNA computing*. Biosystems, 2003. **70**(1): p. 73-78.

33. Manca, V. and G. Franco, *Computing by polymerase chain reaction*. Mathematical Biosciences, 2008. **211**(2): p. 282-298.
34. Benenson, Y., et al., *DNA molecule provides a computing machine with both data and fuel*. Proceedings of the National Academy of Sciences of the United States of America, 2003. **100**(5): p. 2191-2196.
35. Picuri, J.M., B.M. Frezza, and M.R. Ghadiri, *Universal Translators for Nucleic Acid Diagnosis*. Journal of the American Chemical Society, 2009. **131**(26): p. 9368-9377.
36. Benenson, Y., et al., *An autonomous molecular computer for logical control of gene expression*. Nature, 2004. **429**(6990): p. 423-429.
37. Boneh, D., et al., *On the computational power of DNA*. Discrete Applied Mathematics, 1996. **71**(1-3): p. 79-94.
38. Lin, C.H., et al., *Solving satisfiability problems using a novel microarray-based DNA computer*. Biosystems, 2007. **90**(1): p. 242-252.
39. Wang, Z.C., et al., *Solving traveling salesman problem in the Adleman-Lipton model*. Applied Mathematics and Computation, 2012. **219**(4): p. 2267-2270.
40. Chen, R.C.S. and S.J.H. Yang, *Applying DNA computation to intractable problems in social network analysis*. Biosystems, 2010. **101**(3): p. 222-232.
41. Darehmiraqi, M., *A New Solution for Maximal Clique Problem based Sticker Model*. Biosystems, 2009. **95**(2): p. 145-149.
42. García-Arnau, M., et al., *A P system and a constructive membrane-inspired DNA algorithm for solving the Maximum Clique Problem*. Biosystems, 2007. **90**(3): p. 687-697.
43. Kim, I., J. Watada, and W. Pedrycz, *A DNA-based algorithm for arranging weighted cliques*. Simulation Modelling Practice and Theory, 2008. **16**(10): p. 1561-1570.

44. Liu, X.K., L. Yan, and X. Jin. *Solving minimum spanning tree problem with DNA computing*. in *International Conference on Management Science and Engineering*. 2002. Moscow, Russia: Harbin Institute Technology Publishers.
45. Iliopoulos, C. and M. Rahman, *A New Efficient Algorithm for Computing the Longest Common Subsequence*. *Theory of Computing Systems*, 2009. **45**(2): p. 355-371.
46. Wood, D., et al., *DNA Starts to Learn Poker*, in *DNA Computing* 2002. p. 92-103.
47. Bakar, R.B.A., J. Watada, and W. Pedrycz, *DNA approach to solve clustering problem based on a mutual order*. *Biosystems*, 2008. **91**: p. 1-12.
48. Wood, D.H., *DNA computing capabilities for game theory*. *Natural Computing*, 2003. **2**(1): p. 85-108.
49. Nowak, M.A. and K. Sigmund, *Evolutionary dynamics of biological games*. *Science*, 2004. **303**(5659): p. 793-799.
50. Kuhn, H.W. and S. Nasar, *The Essential John Nash*. Princeton University Press, 2007.
51. Morgenstern, O. and J. Von Neumann, *Theory of Games and Economic Behavior*. Princeton University Press, 1944.
52. Biggs, N.L., E.K. Lloyd, and R.J. Wilson, *Graph Theory 1736-1936*. Clarendon Press, Oxford, 1976.
53. Martínez-Pérez, I.M. and K.-H. Zimmermann, *Parallel bioinspired algorithms for NP complete graph problems*. *Journal of Parallel and Distributed Computing*, 2009. **69**(3): p. 221-229.
54. Kreyszig, E., *Advanced Engineering Mathematics*. 1998. **8th Edition**.
55. Carell, T., *MOLECULAR COMPUTING DNA as a logic operator*. *Nature*, 2011. **469**(7328): p. 45-46.

56. Lederman, H., et al., *Deoxyribozyme-based three-input logic gates and construction of a molecular full adder*. Biochemistry, 2006. **45**(4): p. 1194-1199.
57. Wasiewicz, P., et al., *DNA computing: implementation of data flow logical operations*. Future Generation Computer Systems, 2001. **17**(4): p. 361-378.
58. Krishnan, Y. and F.C. Simmel, *Nucleic Acid Based Molecular Devices*. Angewandte Chemie-International Edition, 2011. **50**(14): p. 3124-3156.
59. Liang, X.G., et al., *A DNA nanomachine powered by light irradiation*. Chembiochem, 2008. **9**(5): p. 702-705.
60. Chakraborty, S., et al., *The poly dA helix: a new structural motif for high performance DNA-based molecular switches*. Nucleic Acids Research, 2009. **37**(9): p. 2810-2817.
61. Ma, D.L., et al., *Oligonucleotide-Based Luminescent Detection of Metal Ions*. Chemistry-an Asian Journal, 2011. **6**(4): p. 986-1003.
62. Seelig, G., et al., *Enzyme-free nucleic acid logic circuits*. Science, 2006. **314**(5805): p. 1585-1588.
63. Komiya, K., et al., *Successive state transitions with I/O interface by molecules*, in *DNA Computing* 2001. p. 17-26.
64. Qian, L. and E. Winfree, *Scaling Up Digital Circuit Computation with DNA Strand Displacement Cascades*. Science, 2011. **332**(6034): p. 1196-1201.
65. Liu, G., et al., *Deoxyribonucleic Acid Molecular Design for Electrochemical Biosensors*. Chinese Journal of Analytical Chemistry, 2011. **39**(7): p. 953-962.
66. Zhu, X., et al., *Design of a DNA electronic logic gate (INHIBIT gate) with an assaying application for Ag⁺ and cysteine*. Chemical Communications, 2011. **47**(32): p. 9080-9082.

67. Qian, L., E. Winfree, and J. Bruck, *Neural network computation with DNA strand displacement cascades*. Nature, 2011. **475**(7356): p. 368-372.
68. Zhu, J.B., et al., *Enzyme-Free Unlabeled DNA Logic Circuits Based on Toehold-Mediated Strand Displacement and Split G-Quadruplex Enhanced Fluorescence*. Advanced Materials, 2013. **25**(17): p. 2440-2444.
69. Winfree, E., *On the computational power of DNA annealing and ligation*. 1996.
70. Zhang, H.Y. and X.Y. Liu, *A CLIQUE algorithm using DNA computing techniques based on closed-circle DNA sequences*. Biosystems, 2011. **105**(1): p. 73-82.
71. Benenson, Y., et al., *Programmable and autonomous computing machine made of biomolecules*. Nature, 2001. **414**(6862): p. 430-434.
72. Adar, R., et al., *Stochastic computing with biomolecular automata*. Proceedings of the National Academy of Sciences of the United States of America, 2004. **101**(27): p. 9960-9965.
73. Soloveichik, D. and E. Winfree, *The computational power of Benenson automata*. Theoretical Computer Science, 2005. **344**(2-3): p. 279-297.
74. Fujibayashi, K., et al., *Toward reliable algorithmic self-assembly of DNA tiles: A fixed-width cellular automaton pattern*. Nano Letters, 2008. **8**(7): p. 1791-1797.
75. Winfree, E., et al., *Design and self-assembly of two-dimensional DNA crystals*. Nature, 1998. **394**(6693): p. 539-544.
76. Wang, H., *Proving theorems by pattern recognition, II*. Bell System Technical Journal, 1961. **40**(1): p. 1-41.
77. Carbone, A. and N.C. Seeman, *Circuits and programmable self-assembling DNA structures*. Proceedings of the National Academy of Sciences of the United States of America, 2002. **99**(20): p. 12577-12582.

78. Mao, C.D., et al., *Logical computation using algorithmic self-assembly of DNA triple-crossover molecules*. Nature, 2000. **407**(6803): p. 493-496.
79. Barish, R.D., P.W.K. Rothmund, and E. Winfree, *Two computational primitives for algorithmic self-assembly: Copying and counting*. Nano Letters, 2005. **5**(12): p. 2586-2592.
80. Brun, Y., *Arithmetic computation in the tile assembly model: Addition and multiplication*. Theoretical Computer Science, 2007. **378**(1): p. 17-31.
81. Zhang, X., et al., *Arithmetic computation using self-assembly of DNA tiles: subtraction and division*. Progress in Natural Science, 2009. **19**(3): p. 377-388.
82. Brun, Y., *Solving NP-complete problems in the tile assembly model*. Theoretical Computer Science, 2008. **395**(1): p. 31-46.
83. Winfree, E., *Self-healing Tile Sets*, in *Nanotechnology: Science and Computation*, J. Chen, N. Jonoska, and G. Rozenberg, Editors. 2006, Springer Berlin Heidelberg. p. 55-78.
84. Zhang, D.Y. and E. Winfree, *Control of DNA Strand Displacement Kinetics Using Toehold Exchange*. Journal of the American Chemical Society, 2009. **131**(47): p. 17303-17314.
85. de Murieta, I.S. and A. Rodriguez-Paton, *DNA biosensors that reason*. Biosystems, 2012. **109**(2): p. 91-104.
86. Muscat, R.A., J. Bath, and A.J. Turberfield, *A Programmable Molecular Robot*. Nano Letters, 2011. **11**(3): p. 982-987.
87. LaBean, T.H., et al., *Construction, analysis, ligation, and self-assembly of DNA triple crossover complexes*. Journal of the American Chemical Society, 2000. **122**(9): p. 1848-1860.

88. Murphy, N., D. Woods, and T.J. Naughton, *Bio-Computation using Holliday junctions*, in *International Conference on Information and 4th Irish Conference on the Mathematical Foundations of Computer Science and Information Technology (MFCSIT)2006*. p. 317-320.
89. Benenson, Y., *Synthetic biology with RNA: progress report*. Current Opinion in Chemical Biology, 2012. **16**(3-4): p. 278-284.
90. Xie, Z., et al., *Multi-Input RNAi-Based Logic Circuit for Identification of Specific Cancer Cells*. Science, 2011. **333**(6047): p. 1307-1311.
91. Ramakrishnan, N., U.S. Bhalla, and J.J. Tyson, *COMPUTING WITH PROTEINS*. Computer, 2009. **42**(1): p. 47-56.
92. Garzon, M.H., N. Jonoska, and S.A. Karl, *The bounded complexity of DNA computing*. Biosystems, 1999. **52**(1-3): p. 63-72.
93. Kim, J., K.S. White, and E. Winfree, *Construction of an in vitro bistable circuit from synthetic transcriptional switches*. Molecular Systems Biology, 2006. **2**.
94. Guarnieri, F., M. Fliss, and C. Bancroft, *Making DNA add*. Science, 1996. **273**(5272): p. 220-223.
95. Pearson, W.R., et al., *On the primer selection problem in polymerase chain reaction experiments*. Discrete Applied Mathematics, 1996. **71**(1-3): p. 231-246.
96. Lo, Y.M., R.W. Chiu, and K.C. Chan, *Clinical Applications of PCR*. Methods in Molecular Biology, 2006. **336**.
97. Loh, Y.J. and H.-Q. Gong, *DNA amplification experimental study*. 8th National Undergraduate Research Opportunities Programme (NUROP), 2002. **68**.

98. Shalon, D., S.J. Smith, and P.O. Brown, *A DNA microarray system for analyzing complex DNA samples using two-color fluorescent probe hybridization*. Genome Research, 1996. **6**(7): p. 639-645.
99. Sanger, F., *Determination of nucleotide sequences in DNA*. Bioscience Reports, 1981. **1**(1): p. 3-18.
100. Chan, E.Y., *Advances in sequencing technology*. Mutation Research/Fundamental and Molecular Mechanisms of Mutagenesis, 2005. **573**(1-2): p. 13-40.
101. Eghdami, H. and M. Darehmiraki, *Application of DNA computing in graph theory*. Artificial Intelligence Review, 2012. **38**(3): p. 223-235.
102. Hartmanis, J., *On the Weight of Computations*. Bulletin of the EATCS, 1995. **55**.
103. Deaton, R., et al., *Reliability and efficiency of a DNA-based computation*. Physical Review Letters, 1998. **80**(2): p. 417-420.
104. Yang, C.N., et al., *Error-Tolerant Minimum Finding With DNA Computing*. International Journal of Innovative Computing Information and Control, 2009. **5**(10A): p. 3045-3057.
105. Amos, M., A. Gibbons, and D. Hodgson, *Error-resistant implementation of DNA computations*. 1996.
106. Murphy, R.C., et al. *A new algorithm for DNA based computation*. in *1997 IEEE International Conference on Evolutionary Computation (ICEC 97)*. 1997. Indianapolis, In: I E E E.

12 APPENDIX

12.1 DNA strands for Shortest Path Problem

	Start	Forward	Backward
Vertex 1	1 L12 2f	1b L12 2f	2b L12 1f
	1 L13 3f	1b L13 3f	3b L13 1f
	1 L15 5f	1b L15 5f	5b L15 1f
Vertex 2	2 L12 1f		
	2 L23 3f	2b L23 3f	3b L23 2f
	2 L24 4f	2b L24 4f	4b L24 2f
	2 L25 5f	2b L25 5f	5b L25 2f
	2 L26 6f	2b L26 6f	6b L26 2f
Vertex 3	3 L13 1f		
	3 L23 2f		
	3 L35 5f	3b L35 5f	5b L35 3f
Vertex 4	4 L24 2f		
	4 L45 5f	4b L45 5f	5b L45 4f
	4 L46 6f	4b L46 6f	6b L46 4f

	Start	Forward	Backward
	<div><div></div><div>5</div><div>L15</div><div>1f</div></div>		
	<div><div></div><div>5</div><div>L25</div><div>2f</div></div>		
Vertex 5	<div><div></div><div>5</div><div>L35</div><div>3f</div></div>		
	<div><div></div><div>5</div><div>L45</div><div>4f</div></div>		
	<div><div></div><div>5</div><div>L56</div><div>6f</div></div>	<div><div></div><div>5b</div><div>L56</div><div>6f</div></div>	<div><div></div><div>6b</div><div>L56</div><div>5f</div></div>
	<div><div></div><div>6</div><div>L26</div><div>2f</div></div>		
Vertex 6	<div><div></div><div>6</div><div>L46</div><div>4f</div></div>		
	<div><div></div><div>6</div><div>L56</div><div>5f</div></div>		
	Stop		
Vertex 1	<div><div></div><div>1b</div><div>S</div></div>		
Vertex 2	<div><div></div><div>2b</div><div>S</div></div>		
Vertex 3	<div><div></div><div>3b</div><div>S</div></div>		
Vertex 4	<div><div></div><div>4b</div><div>S</div></div>		
Vertex 5	<div><div></div><div>5b</div><div>S</div></div>		
Vertex 6	<div><div></div><div>6b</div><div>S</div></div>

12.2 DNA templates for Shortest Path Problem

Name	Sequence	T _m (1μm)	Remarks
1	CCGATGACTT	43.9	
2	CACTGCACTG	44.5	
3	GCCTTGAAGT	44.8	
4	CAGTCGGTAT	42.2	
5	AGACTGACGA	44.0	
6	AGTGCAACAC	43.6	
S	CCACCAAGAT	43.6	
1f	CTGCGAATGT	44.4	
1b	GACGCTTACA		1f rev comp
2f	CCCAAAGTTC	43.6	
2b	GGGTTTCAAG		2f rev comp
3f	GCTGTCTACC	44.6	
3b	CGACAGATGG		3f rev comp
4f	GACTTGGTCA	42.9	
4b	CTGAACCACT		4f rev comp
5f	TGACGCTACT	44.3	
5b	ACTGCGATGA		5f rev comp
6f	CCCTCATTTG	42.4	
6b	GGGAGTAAAC		6f rev comp
L12	CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA	84.2	
L13	CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAAC	86.7

	CGATGACTT		
L15	TAGGCGAAAAGGCCAAGATAGACCA	77.9	
L23	AGGTCAACTC	42.6	
L24	CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA	84.2	
L25	AGGTCAACTC	42.6	
L26	TCCCTCTTCCCAGTCGGTATCTATGACTCG	53	
L35	TCCCTCTTCCCAGTCGGTATCTATGACTCG	53	
L45	TAGGCGAAAAGGCCAAGATAGACCA	77.9	
L46	GTCAG		
L56	AGGTCAACTC	42.6	

Note: “rev comp” stands for reverse complementary.

12.3 DNA strands for Shortest Spanning Tree

	Start	Forward	Backward
Vertex 1	1b L12 2a		
	1b L15 5a		
	1b L16 6a		
Vertex 2		2b L23 3a	3b L23 2a
		2b L24 4a	4b L24 2a
		2b L26 6a	6b L26 2a
Vertex 3		3b L34 4a	4b L34 3a
		3b L35 5a	5b L35 3a

Note: Strands with a joins b (and c joins d), with the exception for connector DNA strands where Cf joins Cb.

12.4 DNA templates for Shortest Spanning Tree

Name	Sequence	T _m (1μm)	Remarks
1d	CCGATGACTT	43.9	
1a	GGCTACTGAA		1d rev comp
2d	CACTGCACTG	44.5	
2a	GTGACGTGAC		2d rev comp
3d	GCCTTGAAGT	44.8	
3a	CGGAACTTCA		3d rev comp
4d	CAGTCGGTAT	42.2	
4a	GTCAGCCATA		4d rev comp
5d	AGACTGACGA	44.0	
5a	TCTGACTGCT		5d rev comp
6d	AGTGCAACAC	43.6	
6a	TCACGTTGTG		6d rev comp
1b	CCACCAAGAT	43.6	
1c	GGTGGTTCTA		1b rev comp
2b	CTGCGAATGT	44.4	
2c	GACGCTTACA		2b rev comp
3b	CCCAAAGTTC	43.6	
3c	GGGTTTCAAG		3b rev comp
4b	GCTGTCTACC	44.6	
4c	CGACAGATGG		4b rev comp
5b	GACTTGGTCA	42.9

5c	CTGAACCAGT		5b rev comp
6b	TGACGCTACT	44.3	
6c	ACTGCGATGA		6b rev comp
Cf	CCCTCATTTG	42.4	
Cb	GGGAGTAAAC		Cf rev comp
S	AGGTCAACTC	42.6	
L34	CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA	47	
L24	CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA CCGATGACTTAACTGGCGTC		L34 +20
L26, L35	CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA		L34 x 2
L26	CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA CCGATGACTTAACTGGCGTC		L34 x 2 + 20
L15	CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA		L34 x 3
L45	CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA CCGATGACTTAACTGGCGTC		L34 x 3 + 20
L12	CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA		L34 x 4

	CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA		
L16	CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA CAAGGAGCCACCCTATGTGCCCAAACACTTCGTTTTGTAA CCGATGACTTAACTGGCGTC		L34 x 4 + 20

12.5 DNA strands for Maximum Flow Problem

Forward

Backward

Vertex 1

1b	a30	2a1
1b	a20	2a2
1b	b30	3a1

Vertex 2

2b1	c30	3a1
2b1	d30	4a1
2b2	d20	4a2
2b1	e30	5a1
2b2	e20	5a2

	Forward	Backward
Vertex 3	3b1 f30 5a1	3b1 c30 2a1
	3b2 f20 5a2	
	3b3 f10 5a3	
Vertex 4	4b1 g30 5a1	4b1 d30 2a1
	4b1 h30 6a	4b2 d20 2a2
	4b2 h20 6a	4b3 d10 2a3
	4b3 h10 6a	
Vertex 5	5b1 g30 6a	5b1 d30 2a1
		5b1 d20 3a1
		5b2 d10 3a2
		5b3 h30 3a3
		5b1 h20 4a1
		5b2 h10 4a2

	Indicator	Link					
Edge a	<table><tr><td>6b</td><td>anc</td><td>6a</td></tr></table>	6b	anc	6a	<table><tr><td>6b</td><td>1a</td></tr></table>	6b	1a
6b	anc	6a					
6b	1a						
Edge b	<table><tr><td>6b</td><td>bnc</td><td>6a</td></tr></table>	6b	bnc	6a			
6b	bnc	6a					
Edge c	<table><tr><td>6b</td><td>cnc</td><td>6a</td></tr></table>	6b	cnc	6a	<table><tr><td>Stop</td></tr></table>	Stop	
6b	cnc	6a					
Stop							
Edge d	<table><tr><td>6b</td><td>dnc</td><td>6a</td></tr></table>	6b	dnc	6a	<table><tr><td>6b</td><td>s</td></tr></table>	6b	s
6b	dnc	6a					
6b	s						
Edge e	<table><tr><td>6b</td><td>enc</td><td>6a</td></tr></table>	6b	enc	6a			
6b	enc	6a					
Edge f	<table><tr><td>6b</td><td>fnc</td><td>6a</td></tr></table>	6b	fnc	6a			
6b	fnc	6a					
Edge g	<table><tr><td>6b</td><td>gnc</td><td>6a</td></tr></table>	6b	gnc	6a			
6b	gnc	6a					
Edge h	<table><tr><td>6b</td><td>hnc</td><td>6a</td></tr></table>	6b	hnc	6a			
6b	hnc	6a					
Edge i	<table><tr><td>6b</td><td>inc</td><td>6a</td></tr></table>	6b	inc	6a			
6b	inc	6a					

Note: Endpoint of strands with a joins b, a1 joins b1, a2 joins a2, etc. 'a30' stands for edge a with length of 30 base pairs (bp) and 'anc' stands for no change in pipe a's flow rate. In DNA strands design, 'a30' is equivalent to template 'a' combining with template '20'. This results in a length of 30 bp. Similarly, 'a20' is equivalent to template 'a' combining with template '10'. 'anc' is equivalent to template 'a' combining with template 'nc'.

12.6 DNA templates for Maximum Flow Problem

Name	Sequence	T _m (1μm)	Remarks
1a	CCGATGACTT	43.9	
1b	GGCTACTGAA		1a rev comp
2a1	CACTGCACTG	44.5	
2b1	GTGACGTGAC		2a1 rev comp
2a2	GCCTTGAAGT	44.8	
2b2	CGGAACTTCA		2a2 rev comp
3a1	CAGTCGGTAT	42.2	
3b2	GTCAGCCATA		3a1 rev comp
3a2	AGACTGACGA	44.0	
3b2	TCTGACTGCT		3a2 rev comp
3a3	AGTGCAACAC	43.6	
3b3	TCACGTTGTG		3a3 rev comp
4a1	CCACCAAGAT	43.6	
4b1	GGTGGTTCTA		4a1 rev comp
4a2	CTGCGAATGT	44.4	
4b2	GACGCTTACA		4a2 rev comp
4a3	CCCAAAGTTC	43.6	
4b3	GGGTTTCAAG		4a3 rev comp
5a1	GCTGTCTACC	44.6	
5b1	CGACAGATGG		5a1 rev comp
5a2	GACTTGGTCA	42.9

5b2	CTGAACCAGT		5a2 rev comp
5a3	TGACGCTACT	44.3	
5b3	ACTGCGATGA		5a3 rev comp
6a	CCCTCATTTG	42.4	
6b	GGGAGTAAAC		6a rev comp
S	AGGTCAACTC	42.6	
a	ATGGGCAATT	43.5	
b	TCGCTATTGG	44.2	
c	GGACAATCGA	44.3	
d	TTCGATGTCG	44.6	
e	CCGTTTCTGT	45.0	
f	ACCCTACCAT	44.7	
g	CCTCAGTAGC	44.1	
h	CGACTTCGTA	43.5	
i	GATGCGTACT	42.8	
nc			Design Pending
10	GGTGAAATCG	43.3	
20	CCGATGACTTAACTGGCGTC	71.6

12.7 DNA strands for Maximum Cardinality Problem

	Start	Forward	Backward
Vertex 1	1 L14 4f	1b L14 4f	4b L14 1f
	1 L16 6f	1b L16 6f	6b L16 1f
Vertex 2	2 L24 4f	2b L24 4f	4b L24 2f
	2 L25 5f	2b L25 5f	5b L25 2f
Vertex 3	3 L35 5f	3b L35 5f	5b L35 3f
	3 L35 6f	3b L36 6f	6b L36 3f
Vertex 4	4 L14 1f		
	4 L24 2f		
Vertex 5	5 L25 2f		
	5 L35 3f		
Vertex 6	6 L16 1f		
	6 L36 3f

	Forward Connectors	Backward Connectors
Vertex 1	<div><div>1b</div><div>L12</div><div>2f</div></div>	<div><div>2b</div><div>L12</div><div>1f</div></div>
	<div><div>1b</div><div>L13</div><div>3f</div></div>	<div><div>3b</div><div>L13</div><div>1f</div></div>
Vertex 2	<div><div>2b</div><div>L23</div><div>3f</div></div>	<div><div>3b</div><div>L23</div><div>2f</div></div>
Vertex 4	<div><div>4b</div><div>L45</div><div>5f</div></div>	<div><div>5b</div><div>L45</div><div>4f</div></div>
	<div><div>4b</div><div>L46</div><div>6f</div></div>	<div><div>6b</div><div>L46</div><div>4f</div></div>
Vertex 5	<div><div>5b</div><div>L56</div><div>6f</div></div>	<div><div>6b</div><div>L56</div><div>5f</div></div>
	Stop	
Vertex 1	<div><div>1b</div><div>S</div></div>	
Vertex 2	<div><div>2b</div><div>S</div></div>	
Vertex 3	<div><div>3b</div><div>S</div></div>	
Vertex 4	<div><div>4b</div><div>S</div></div>	
Vertex 5	<div><div>5b</div><div>S</div></div>	
Vertex 6	<div><div>6b</div><div>S</div></div>

12.8 DNA templates for Maximum Cardinality Problem

Name	Sequence	T _m (1μm)	Remarks
1	CCGATGACTT	43.9	
2	CACTGCACTG	44.5	
3	GCCTTGAAGT	44.8	
4	CAGTCGGTAT	42.2	
5	AGACTGACGA	44.0	
6	AGTGCAACAC	43.6	
S	CCACCAAGAT	43.6	
1f	CTGCGAATGT	44.4	
1b	GACGCTTACA		1f rev comp
2f	CCCAAAGTTC	43.6	
2b	GGGTTTCAAG		2f rev comp
3f	GCTGTCTACC	44.6	
3b	CGACAGATGG		3f rev comp
4f	GACTTGGTCA	42.9	
4b	CTGAACCAGT		4f rev comp
5f	TGACGCTACT	44.3	
5b	ACTGCGATGA		5f rev comp
6f	CCCTCATTTG	42.4	
6b	GGGAGTAAAC		6f rev comp

L12	CCGATGACTTAACTGGCGTC	71.6	Same sequence for all templates
L13			
L15			
L23			
L24			
L25			
L26			
L35			
L45			
L46			
L56

12.9 DNA strands for GPS Problem

	Vertex	Directional Forward	Directional Backward
Vertex 1	$\overline{1f \quad 1b}$	$\overline{1b' \quad L12 \quad 2f'}$ $\overline{1b' \quad L15 \quad 5f'}$	$\overline{2b' \quad L12 \quad 1f'}$ $\overline{5b' \quad L15 \quad 1f'}$
Vertex 2	$\overline{2f \quad 2b}$	$\overline{2b' \quad L23 \quad 3f'}$ $\overline{2b' \quad L24 \quad 4f'}$ $\overline{2b' \quad L26 \quad 6f'}$	$\overline{3b' \quad L23 \quad 2f'}$ $\overline{4b' \quad L24 \quad 2f'}$ $\overline{6b' \quad L26 \quad 2f'}$
Vertex 3	$\overline{3f \quad 3b}$	$\overline{3b' \quad L34 \quad 4f'}$ $\overline{3b' \quad L35 \quad 5f'}$	$\overline{4b' \quad L34 \quad 3f'}$ $\overline{5b' \quad L35 \quad 3f'}$
Vertex 4	$\overline{4f \quad 4b}$	$\overline{4b' \quad L45 \quad 5f'}$	$\overline{5b' \quad L45 \quad 4f'}$
Vertex 5	$\overline{5f \quad 5b}$		
Vertex 6	$\overline{6f \quad 6b}$

12.10 DNA templates for GPS Problem

Name	Sequence (From 5' to 3')	T _m (1μm)	Remarks
1f	CCGTGTCTAC		
1f'	GGCACAGATG		
1b	AACAGAAGGA		
1b'	TTGTCTTCCT		
2f	CCCTCATTTG		
2f'	GGGAGTAAAC		
2b	ACGAGGAATG		
2b'	TGCTCCTTAC		
3f	TGGTTAGACG		
3f'	ACCAATCTGC		
3b	CAGAGAGTTC		
3b'	GTCTCTCAAG		
4f	AGGCACACGA		
4f'	TCCGTGTGCT		
4b	TTATGGACAG		
4b'	AATACCTGTC		
5f	CGAATTTAGC		
5f'	GCTTAAATCG		
5b	ACCGCATGTG		
5b'	TGGCGTACAC		
6f	GGGGTCTTCA

6f'	CCCCAGAAAGT		
6b	ACTATTGTCC		
6b'	TGATAACAGG		
L12	GGGTCAATTACGTTGGAACATCGCCA		
L15	ATATTACGGGC		
L23	ATACGACTGGTAGGATGGAAGG		
L24	TTTCGAGATCCC		
L26	TGTACTGTAGAGGCGACT		
L34	TAGACCATAAGCAGGG		
L35	GGGGATCAGTAGCAATGA		
L45	CGTTCAAACC		

Note: (') in the template name stands for reverse complementary.