# COMPUTATIONAL METHODS FOR SELF-ASSEMBLY OF DNA NANOSTRUCTURES

ONG HUI SAN

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2016

# COMPUTATIONAL METHODS FOR SELF-ASSEMBLY OF DNA NANOSTRUCTURES

ONG HUI SAN

THESIS SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2016

**UNIVERSITY OF MALAYA**

**ORIGINAL LITERARY WORK DECLARATION**

Name of Candidate: ONG HUI SAN (I.C/Passport No: [redacted])

Registration/Matric No: WHA120040

Name of Degree: DOCTOR OF PHILOSOPHY

Title of Project Paper/Research Report/Dissertation/Thesis ("this Work"):

COMPUTATIONAL METHODS FOR SELF-ASSEMBLY OF DNA NANOSTRUCTURES

Field of Study: UNCONVENTIONAL COMPUTING (COMPUTER SCIENCE)

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This Work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date:

Subscribed and solemnly declared before,

Witness's Signature Date:

Name:

Designation:

## ABSTRACT

Deoxyribonucleic acids (DNA), Ribonucleic acids (RNA) and Proteins are the computational devices of life. Compared to conventional machine, natural hardware has data encoded as molecules and requires molecular biology tools to transform these data in order to perform computation. The feasibility of adapting these substrates into conventional silicon machines has been actively studied leading to the emergence of a new computational paradigm, known as molecular computing. One of the most notable researches under this field is DNA self-assembly computing, by which DNAs autonomously come together and formed complex nanostructures. In this study, the concept of Tetris game to facilitate DNA nanostructures fabrication was adapted; whereby different DNA Tetris shapes were used to form complex 2D DNA structures. The efforts are concerted towards self-assembly mechanism of heterogeneous DNA shapes, construction of multiple configurations that can form the identical end-structures, exploration of a less stringent sequence design and predicting connectivity map for the DNA nanostructures assembly. Several approaches have been adopted including development of an autonomous tool that incorporated evolutionary optimization algorithm in constructing these heterogeneous DNA shapes and the application of heuristic through undirected graph theory as an annotation schema to produce the connectivity maps. These approaches have lead to the successful formation of five distinct configurations based on 3 x 4 DNA rectangle, which were validated in the laboratory (using Atomic Force Microscopy (AFM) images). This study proved that the fabrication of the DNA nanostructures is no longer limited to sets of specific sequences, but liberated to the conformity of both shapes and sequence combinatorics. This proposed schema has therefore opened up the possibility for competing DNA shapes to self-organize into molecular constructs in an autonomous manner imitating their natural behaviour.

# ABSTRAK

Deoxyribonukleik asid (DNA), Ribonukleik asid (RNA) dan Protein merupakan peranti pengkomputeran bagi benda hidup. Berbanding dengan mesin konvensional, perkakasan sumber inspirasi semulajadi ini mempunyai data yang dikodkan sebagai molekul dan menggunakan perkakasan molekular biologi untuk menjana data dan melakukan aktiviti perkomputeran. Kebolehlaksanaan untuk menyesuaikan substrak-substrak ini ke dalam mesin silikon sedang giat dijalankan. Ini secara tidak langsung membawa kepada kemunculan cabang perkomputeran yang baru iaitu perkomputeran molekul. Salah satu subjek utama dalam cabang ini ialah penstrukturan sendiri DNA, dimana DNA secara automatik berkumpul dan membentuk nanostruktur kompleks. Dalam kajian ini, kajian mengadaptasi konsep permainan Tetris untuk memudahkan proses pembinaan kompleks struktur 2-dimensi DNA. Usaha ini tertumpu kepada mekanisma penstrukturan sendiri heterogen DNA, membina pelbagai konfigurasi yang boleh membentuk struktur terakhir yang identikal, mengeksplorasi kaedah reka bentuk jujukan yang longgar dan meramalkan peta perhubungan bagi tujuan pembinaan DNA nanostruktur. Antara pendekatan yang diadaptasi termasuklah pembangunan perkakasan automatik yang menggabungkan algoritma evolusi pengoptimuman dalam membina heterogen DNA. Aplikasi heuristik melalui teori graf tidak berarah digunakan sebagai anotasi skema untuk menjana peta perhubungan. Pendekatan ini telah membawa kepada kejayaan pembentukan lima konfigurasi segi empat tepat 3 x 4 yang kemudiannya ditentusahihkan dalam makmal (menggunakan imej Atomic Force Mircoscopy (AFM)). Kajian ini turut membuktikan bahawa pembinaan DNA nanostruktur kini tidak lagi terhad kepada set jujukan spesifik, sebaliknya bebas daripada konformasi bentuk dan kombinasi jujukan. Skema yang dicadangkan turut membuka peluang bagi bentuk-bentuk DNA untuk bersaing semasa proses penstrukturan sendiri dan membentuk konstruk molekul secara automatik yang mimik kepada ciri-ciri semulajadi DNA.

# ACKNOWLEDGEMENTS

First and foremost, i would like to express my gratitude and deepest appreciation to my supervisor, Dr. Effirul Ikhwan Ramlan for his invaluable guidance and constructive advice during the entire course of my research work. To Associate Professor Dr. Firdaus Mohd Raih, thanks for his guidance and valuable advice for the wet-lab experiments and Mr. Mohd Syafiq from UKM for conducting the experimental study.

I would also like to extend my utmost gratitude to Professor Hiroshi Sugiyama from the Department of Chemistry, Graduate School of Science, Kyoto University and Institute for Integrated Cell-Material Sciences (WPI-iCeMS), Kyoto University and Dr. Yuki Suzuki from the Department of Chemistry, Graduate School of Science, Kyoto University for their assistance and expertise in conducting AFM imaging.

In addition, I would also like to extend my acknowledgement to MyBrain15 scholarship awarded by the Ministry of Higher Education (MOHE), Malaysia in funding my PhD study. Not forgetting a sincere thankful to my friends and labmates from Natural Computing Laboratory, University of Malaya whom I have constantly seeking advice and suggestions. To University of Malaya, in particular staffs of Faculty of Computer Science and Information Technology, I offered my sincere recognition for providing wonderful resources and support during the course of my research work.

Next, to both my parents, I would like to extend my appreciation for their unconditional love and understanding. Last but not least, I would like to dedicate this thesis to my husband, Bernard Lee who is the true believer and supporter of my work, thank you for always being there with me.

To everyone, Thank you.

## TABLE OF CONTENTS

## LIST OF FIGURES

# LIST OF TABLES

# LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| % | Percent |
| 2D | Two-dimensional |
| AFM | Atomic force microscopy |
| APTES | (3-aminopropyl) triethoxysilane |
| DNA | Deoxyribonucleic acid |
| DRAM | Dynamic random-access memory |
| dsDNA | Double-stranded DNA |
| DSG | DNA sequence generator |
| DX | Double-crossover |
| EDTA | Ethylenediaminetetraacetic acid |
| FBS | False binding sites |
| GC | Guanine cytosine |
| $\Delta G_{i,j}$ | Free energy for node i and j |
| HCl | Hydrogen chloride |
| IC | Integrated circuit |
| kHz | Kilohertz |
| MgCl2 | Magnesium chloride |
| mM | Millimolar |
| mm | Millimeter |
| N/m | Newton/metre |
| NaCl | Sodium chloride |
| ng | Nanogram |
| nm | Nanometer |

| | |
|---|---|
| °C | Degree celcius |
| PCR | Polymerase chain reaction |
| $P_{i,j}$ | Binding affinity |
| PLA | Programmable logic array |
| RNA | Ribonucleic acid |
| RNA | Ribonucleic acid |
| ssDNA | Single-stranded DNA |
| SST | Single-stranded DNA tiles |
| TBE | Tris base, Boric acid, EDTA |
| TCL | Tool command language |
| Tm | Melting temperature |
| TSP | Travelling salesman problem |
| V | Volt/centimeter |
| µL | Microliter |
| µM | Micromolar |

## LIST OF APPENDICES

# CHAPTER 1: INTRODUCTION

## 1.1 DNA Nanostructures

Nucleic acid is one of the four major classes of biomolecules present in living organisms, aside from carbohydrates, proteins and lipids. Deoxyribonucleic acid (DNA) and ribonucleic acid (RNA) belong to the nucleic acids class by which both are chemically different (such as the five-carbon sugar). While ribose sugar is found in the RNA, deoxyribose sugar forms part of the DNA. The prominent role of DNA is to encode genetic materials (Avery, Macleod, & McCarty, 1944) needed for the biological function of living organism.

DNA nanostructures are defined as nanometer size ($1 \times 10^{-9}$ m) structures made up of DNA (Andersen et al., 2009; Han et al., 2011; He et al., 2008; Rothemund, 2006; Rothemund, Papadakis, & Winfree, 2004; Shih, Quispe, & Joyce, 2004; Winfree, Sun, & Seeman, 1998). The construction of DNA nanostructures is attainable due to its ability to interact via Watson-Crick base pairing and form double-stranded DNA molecules (Watson & Crick, 1953). Examples of the constructed DNA nanostructures are the platonic tiling based on the geometrical shapes of hexagon, square and triangle (Zhang et al., 2015) (Figure 1.1a), 2D structures with intricate curved surfaces (Han et al., 2011) (Figure 1.1b), dolphin-shaped structures with flexible tails (Andersen et al., 2008) (Figure 1.1c) and arbitrary shapes designed from a molecular canvas (Wei, Dai, & Yin, 2012) (Figure 1.1d). The successful construction of these nanostructures proved that apart from having DNA roles as hereditary materials (Avery et al., 1944), DNA can be programmed to self-assemble itself into the prescribed configurations.

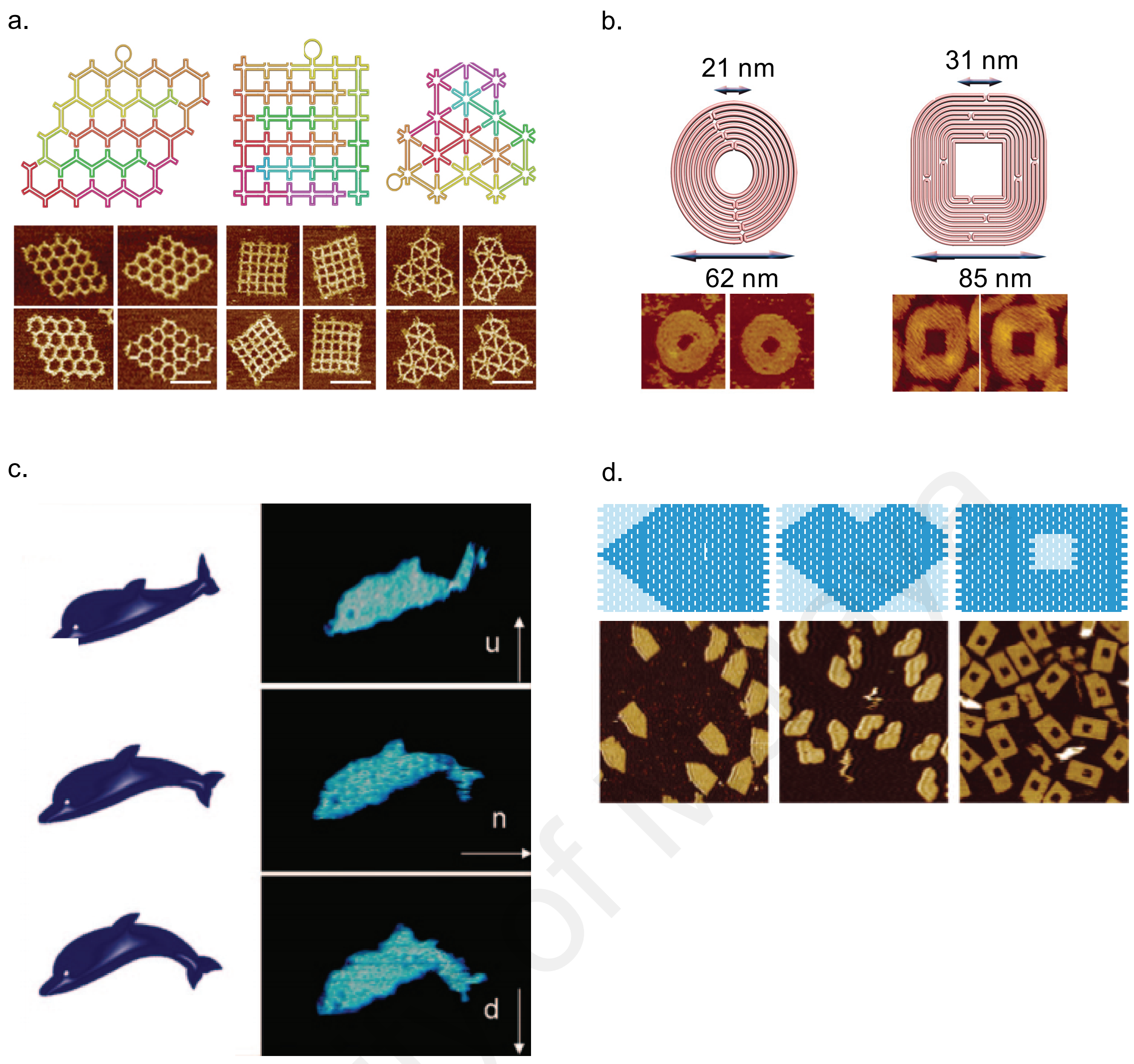


**Figure 1.1:** DNA nanostructures
The DNA nanostructures were captured using Atomic Force Microscopy (AFM). Figure 1.1a represents the platonic tiling based on the geometrical shapes of hexagon, square and triangle (Zhang et al., 2015). Figure1.1b represents the curved 2D DNA nanostructures (Han et al., 2011). Figure 1.1c is the dolphin DNA structures in various conformational states with the arrows indicate the tail orientation as up (u), down (d) and normal (n) (Andersen et. al., 2008). Figure 1.1d is the arbitrary shapes designed from a molecular canvas (Wei et al., 2012). Retrieved from (Andersen et al., 2008; Han et al., 2011; Wei et al., 2012; Zhang et al., 2015).

## 1.2 Problem Statement

The approaches used by prior studies (Ke, Ong, Shih, & Yin, 2012; Rothemund, 2006) have proven to be successful in fabricating sophisticated nanostructures. As these approaches aimed to construct DNA according to the prescribed structure, it involved the process by which each distinct structural design would required a new backbone (scaffold) routing and to have staple strand to specifically target *x* location in the backbone (Douglas et al., 2009; Han et al., 2011; Ke et al., 2009; Rothemund, 2006). A newer design technique known as single-stranded DNA tiles (SST) was subsequently developed to design modular structure by using individual components that have standardized DNA length with four domains (homogeneous block) (Wei et al., 2012; Yin et al., 2008). However, individual component that made up a structure was required to be flexible so that they could self-assemble like a puzzle and occupy the search space. This proposed flexibility feature could be achieved by having different Tetris shapes (heterogeneous blocks) to form the end-structures or in the Tetris game concept, different types of Tetris shapes that could be used to fill the Tetris board.

This Tetris concept offered tremendous dynamicity by adapting the mechanism of DNA self-organization into Tetris game. The search space in the Tetris technique is dynamic because different combinations could be used to occupy the search space ($m$ x $n$) using $y$ number of heterogeneous shapes. This advantage of constructing multiple nanostructures within the search space was able to overcome the current restriction of having one combination for one design.

The dynamicity of the design was crucial because when each component was targeted to specific coordinates in a region, any error that occurred during the targeting process would result in the structure to collapse. This was because the staple strands served to "fold" the scaffold according to the designated structures. On the contrary, each structural formation in the proposed schema was mutually exclusive allowing the remaining structures to be intact even when one of the components collapsed.

The purpose of having different types of combinations occupying the search space is important from the biological aspect of DNA nanostructures. This allowed the best fit structure to carry out its task, since DNA nanorobots have a major concern with the level of stability in cellular environment such as time limitation to resist enzymatic degradation (Mei et al., 2011; Shen et al., 2012). In order to propose the concept of multi-configurations in DNA self-assembly, Tetris concept is an ideal approach since it can be used to elucidate the use of different shapes to fill up the search space.

Similar to the mobile-agent swarm scenario, the self-organization process of Tetris shapes started when a shape occupied a coordinate in the search space (initial state), the next piece of shape will then "observe" the current state (location of the initial shape) and decide whether they could merge together. It would have two mode of actions; if it is compatible the shape would fuse with the initial shape, otherwise it will not. The entire complete DNA nanostructure was considered as the search space. The cycle will reiterate until all shapes occupied the dimension. In this way, regardless of the initial state of the search, the shape was able to self-assembly to handle dynamic situation.

## 1.3 Aims and Objectives

This research aimed to propose a new schema by integrating self-organization principle and Tetris game to create a more dynamics and flexible structure design method. The goal was to promote a complete outlook of the shapes and sequence landscape necessary in designing any DNA structures with minimal restrictions. In this work, the principles in computer science such as algorithm and prediction computing processes were used to simplify the search space, and eventually helped to find suitable solution in building the targeted structures. This was due to the fact that it was highly impossible to build a structure manually from scratch as it involved a very large search space and combinatorics issue. Hence, it is essential to develop computational methods that were able to solve these problems. The objectives of this research are as follows:

**Objective 1: To propose a new schema that simplifies the design search space**

The proposed schema implemented the concept of popular computer game, Tetris in constructing DNA nanostructures. This is because by using *y* number of heterogeneous

Tetris shapes to occupy the search space (*m* x *n*), multiple combinations could be achieved. Search space is defined as the combinatorics of potential shapes to form a given target DNA nanostructure. In this study, a 3 x 4 rectangle with multiple configurations was used as the hypothetical application to show that different combinations can be obtained using the proposed schema.

**Objective 2: To develop a computational tool for structural construction to support the proposed DNA Tetrominoes schema**

Several approaches had been adopted in the development of an autonomous tool to construct the heterogeneous DNA shapes such as evolutionary optimization algorithm. This algorithm incorporated fitness evaluation criteria in order to mutate the DNA so that it could autonomously form into the prescribed structures. A number of external tools had also been integrated into developing the computational tools based on DNA Tetrominoes concept. In order to fulfil this objective, this work will investigate the effectiveness of the undirected graph theory as an annotation schema to produce connectivity map. The map delineated all probable paths taken by the DNA sequences to form the nanostructures because during the execution of self-assembly, correctly formed DNA shapes must compete with the partially correct shapes for binding. The mapping provided insights into these correct and partially correct DNA sequence binding.

**Objective 3: To explore the plausibility of implementing self-organization principle in DNA nanostructure fabrication**

The newly proposed schema integrated self-organization property into the design method in order to construct a more flexible structure. Through this implementation, each Tetris shape was considered as one entity and a search space of a nanostructure was made up of several Tetris shapes. Each DNA Tetris shape required only complementary sticky end at the terminal site to act as glue in order to “hold” the initial shape in the search space with the incoming shape. The self-assembly process of these DNA Tetris shape would result in the formation of DNA nanostructures. Hence, in order to achieve this objective, the work will explore the formation of DNA nanostructures through wet-lab experiment, supplemented by Atomic Force Microscopy (AFM) imaging for further validation.

## 1.4 Thesis Outline

There are two leading motivations for the nucleic acids to serve as computing substrate. Firstly, the ability of nucleic acids to function inside a living system (Beisel, Bayer, Hoff, & Smolke, 2008; Rinaudo et al., 2007; Win & Smolke, 2008) since it is not feasible to program the living system by integrating silicon chips into the system. Secondly, the nucleic acids property can be predicted with the available computational tools such as *RNAstructure* (Reuter & Mathews, 2010) and *ViennaRNA* (Hofacker, 2003). For instance, the intermolecular and intramolecular bindings that occurred between nucleic acids (DNA or RNA) can be calculated using free energy. Although deoxyribonucleic acid (DNA) possed a notable challenge to work with in the laboratory, these challenges at present are still more viable to work with than attempting to design the nanostructures using RNA. This is due to relatively more complex structure in RNA such as the existence of many non Watson-Crick base pairing (non-canonical) (Das, Mukherjee, Mitra, & Bhattacharyya, 2006) that caused RNA to fold itself and formed complex tertiary motif (Chandrasekhar & Malathhi, 2003; Hermann & Westhof, 1999; Leontis & Westhof, 2003).

In chapter 1, the work began with an overview on the model structures that used the DNA as the construction material to fabricate different geometrical shapes at nanometer scale. This has enabled the understanding of the current development in structural fabrications and to strategize the protocol used to construct DNA nanostructures. Two hypotheses (search space in the context of Tetris game and self-organization principle) were introduced to drive the implementation for the newly proposed schema for the nanostructure construction. This has led to the establishment of three research objectives that centralized on the Tetris and self-organization principle.

Chapter 2 presented a general overview of the chemical structures in DNA and discussed more thoroughly about the implementation of DNA as computational substrate into solving Travelling Salesman Problem (Adleman, 1994). The structural aspects of DNA such as double helix arrangement, bending profile and flexibility in DNA were also explained in depth. The work went on to describe the applications of DNA nanostructures in the field of nanotechnology. Since the work also integrated the use of available computational tools in doing sequence and structure analysis, these tools were also discussed.

In chapter 3, the current techniques (DNA origami and Single-stranded DNA Tiles) used to construct DNA nanostructures were reviewed. This is followed by the comparison between the current methods with the proposed schema to highlight the key differences. The proposed schema, DNA Tetrominoes concept was introduced in the subsequent section, which went on to identify Tetris shapes that could be derived from DNA sequences. Each of the shape was either made up of two or four DNA sequences. An overview on the framework that displayed the graphical flow on the computational tool used for the structural construction was presented. This framework had two main modules. Module 1 was for the development of sequence design tool and module 2 for the development of DNA connectivity tool.

The development of module 1, sequence design tool was presented in chapter 4. This tool incorporated the use of optimization algorithm to construct the heterogeneous

DNA Tetris shapes. The optimization algorithm included the process of evaluating and mutating DNA Tetris shapes so that the DNA can self-assemble into the prescribed nanostructures. The developed tool began by constructing a set of DNA Tetris shapes. The respective DNA Tetris shapes were then subjected to the experimental validation. Meanwhile, extensive sequence analysis was also carried out on the generated structures such as investigating the list of mispairing bases and analysis of the free energy to conclude the successful formation of the generated structures.

Chapter 5 discussed on the development of module 2, DNA connectivity tool. This tool is a heuristic application that utilized undirected graph theory in producing the connectivity maps. In this implementation, “nodes” were used to represent DNA segments while the “edges” represented the binding affinity between the nodes. The correct graphs were represented with all the nodes visited exactly once and the edges taken by each node were correctly linked as designed, regardless of the starting points. The resulted graph aimed to provide insight into the occurrences of correct and partially correct binding between a set of DNA sequences. The developed tool will then put in practice to study the generation of multiple conformations of DNA sequences in the following chapter.

Chapter 6 demonstrated the construction of multiple configurations of Tetris shapes to occupy the skeleton of 3 x 4 rectangle. After the successful attempt on constructing the first configurations in chapter 3 (which has named as Set 1), four other combinations were subsequently generated using the sequence design tool. All of these sets were subjected to the structural validation using Atomic Force Microscopy machine. These AFM images of DNA nanostructures were then compared against the predicted configurations. In addition, the DNA connectivity tool was also used to study the interactions that occur in all five combinations (Set 1- Set 5). This has resulted in the generation of graphs and the relative binding affinity between different nodes or DNA segments in forming the end-structures.

Finally, Chapter 7 presented the summary for the implementation of the computational methods in constructing DNA nanostructures. It outlined explanation on how the proposed methodology was able to address the core research objectives. Apart from this, some of the limitations in the newly developed tool were highlighted, specifically on the occurrences of incorrect binding and the limitation of the mutation region due to existence of forbidden region within the DNA sequences. In addition, the conversion of the framework into a web service that worked as graphical user interferences (GUI) was also described. The last section in this chapter presented the conclusions of this research followed by proposition on the prospective future work.

# CHAPTER 2: LITERATURE REVIEW

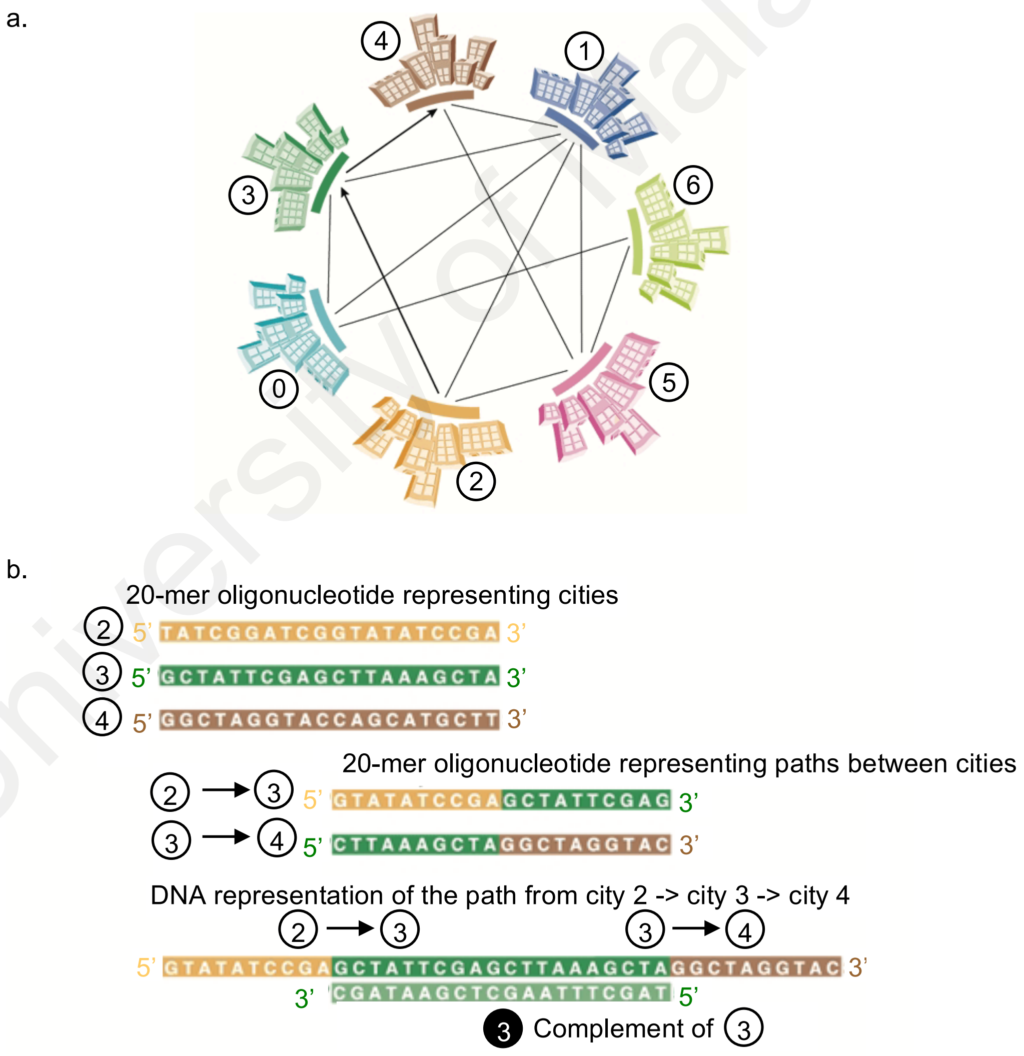
## 2.1 DNA Computer: Computing After Silicon

Computational process can occur in different environment, for instance the conventional approach uses transistor to perform computation in silicon machine (Chen, Korotkov, & Likharev, 1996). Natural computing recently emerged as an alternative computing approach (Rozenberg, Back, & Kok, 2012). It includes the development of novel problem solving methods inspired by the nature (e.g. cellular automata (Wolfram, 1983), neural computation (Von Neumann, 1958), evolutionary computation (Back, Fogel, & Michalewicz, 1997)) and the execution of computation using natural molecules such as creating molecular devices using deoxyribonucleic acid (DNA) to operate inside a living system (Amir et al., 2014). The latter belongs to the category of molecular computing. In conventional silicon computers, electron were pushed through silicon chips (Chen et al., 1996). The current trend for computers to attain high speed is by decreasing the interval between different integrated circuit (IC) or chip components and by narrowing down the size of processing chip and transistor (Moore, 1965). These ensure that the instructions take lesser time to move between one component to another during the execution protocol. As a result, more transistors are packed closely with each other and caused the chip to become more compact. Moore's law states that the number of transistors in the circuits double every one to two years (Moore, 1965). Hence, a point when it reaches the maximum limit will soon arrive. As of year 2015, the size of the chip has finally reaches 10 nm (Nenni, 2015).

Unlike silicon machine, molecular computation concentrates on harnessing the computational power of biological molecules (DNA and RNA) for information processing process (Adleman, 1994; Faulhammer, Cukras, Lipton, & Landweber, 2000; Hagiya, 1999; Heada et al., 2000). It performs parallel computation by taking the advantage in having many different DNA molecules that can search for a large number of different possibilities at one time (Adleman, 1994; Bandyopadhyay, Pati, Sahu, Peper, & Fujita, 2010; Lewin, 2002). The biological molecules are also much smaller in size (nanoscale, $1 \cdot 10^{-9}$m). On the contrary, a conventional silicon computer operates on a highly linear principle of logic (Hsu, 2002) and one computation must be completed before the following process can begin.

In 1990s, researchers started to explore the possibility of DNA computer, when Adleman made the breakthrough by showing that DNA could be used to solve mathematical problems, the Hamiltonian path problem (Adleman, 1994). It was described as a graph with seven vertices and the edge that connected between the vertices was the path. Hamiltonian path problem was similar to the travelling salesman problem (TSP) as the main purpose was to search for the shortest path to visit each vertex of a given graph exactly one time. In TSP problem, given the list of cities (vertex) and the distance between the cities, the question was to find the shortest route for the salesman to visit each city exactly one time and then back to the city of origin. Adleman experiment used seven cities to represent seven vertices (Figure 2.1a).

In order to solve this problem, Adleman represented each city as a separate single stranded DNA with 20 nucleotides length (Figure 2.1b). All possible paths (solutions) between these cities were presented as DNA molecules that were made up from a combination of the last ten nucleotides of the departure city and the first ten nucleotides of the arrival city. Entire DNAs were then mixed in a test tube. Adleman used elimination approaches to get the last remaining DNA, which were the solution to the problem. In Step 1, ligation process was used to connect DNA molecules that encoded all paths in the graph. Then, DNAs that started and ended with the correct vertices (cities) were amplified using polymerase chain reaction (PCR) (Step 2). PCR is a technology used to make many copies of DNA sequences that can be increased to over a thousand or million copies. This is to increase the number of DNA copies so that gel electrophoresis (Step 3) could be used to separate these DNA strands by length. This step filtered the DNA and only kept the paths that entered exactly *n* vertices. Then the DNAs were purified using affinity purification process (Step 4) so that only those paths that entered all the vertices of the graph at least once were kept. The final step 5 was to re-amplify the DNA from step 4 and then determined the presence of DNA sequence encoding the paths.



a.
4
1
3
6
0
5
2
b.
20-mer oligonucleotide representing cities
2 5' TATCGGATCGGTATATCCGA 3'
3 5' GCTATTCGAGCTTAAAGCTA 3'
4 5' GGCTAGGTACCAGCATGCTT 3'
20-mer oligonucleotide representing paths between cities
2 → 3 5' GTATATCCGAGCTATTCGAG 3'
3 → 4 5' CTTAAAGCTAGGCTAGGTAC 3'
DNA representation of the path from city 2 -> city 3 -> city 4
2 → 3
3 → 4
5' GTATATCCGAGCTATTCGAGCTTAAAGCTAGGCTAGGTAC 3'
3' CGATAAGCTCGAATTTCGAT 5'
3 Complement of 3

**Figure 2.1:** Principles in Adleman's DNA computer to solve travelling salesman problem

The seven vertices in the graph are represented as city labelled from 0 to 6. 20-mer oligonucleotides (or DNA sequences with 20 nucleotides length) were used to represent the cities and paths between cities. Retrieved from (Parker, 2003).

Molecular computing that uses DNA molecules to perform computations is known as DNA computing (Lewin, 2002). Different DNA computers might vary in the principles of their operation. By taking silicon computer as a reference, in DNA computer, DNA is equivalent to software whereas the biological molecule, enzyme is the hardware. One of the most notable researches under this field is DNA self-assembly (Winfree et al., 1998). DNA self-assembly or DNA self-organization is defined as a process by which DNAs autonomously came together, and merged to form DNA nanostructures (Han et al., 2011; Rothemund, 2006; Rothemund et al., 2004; Winfree, 1998; Yin et al., 2008). DNA served as an ideal substrate for structural engineering (Aldaye, Palmer, & Sleiman, 2008; Kuzuya & Komiyama, 2010; Seeman, 2010) due to its ability to self-assemble.

## 2.2 DNA as the Molecular Basis of Life

Deoxyribonucleic acid (DNA) is a molecule that encodes the genetic materials (Avery et al., 1944) used in the development of all living organism. DNA is made up of basic units known as nucleotides and each nucleotide consists of a phosphate group, a deoxyribose sugar and a single nitrogenous base (Figure 2.2a). There are four nitrogenous base namely adenine (A), guanine (G), cytosine (C) and thymine (T) (Figure 2.2b).

In DNA, adenine binds to thymine using two hydrogen bonds while cytosine binds to guanine using three hydrogen bonds. In the case of RNA, besides the commonly Watson-Crick base pairing or canonical base pairing (A bind to T and G bind to C), there are also a range of non-canonical base pairs that often exist in RNA secondary and tertiary structural moieties. The guanine (G) – uracil (U) (G bind to U) wobble pairs is an example of non-canonical base pairing (Masquida & Westhof, 2000; Trikha, Filman, & Hogle, 1999; Varani & McClain, 2000). This non-canonical base pairing often influences the intramolecular RNA folding into the tertiary RNA structural motif that acts as binding or recognition sites for specific protein or ligand for enzymatic activity (Chandrasekhar & Malathhi, 2003; Hermann & Westhof, 1999; Leontis & Westhof, 2003).

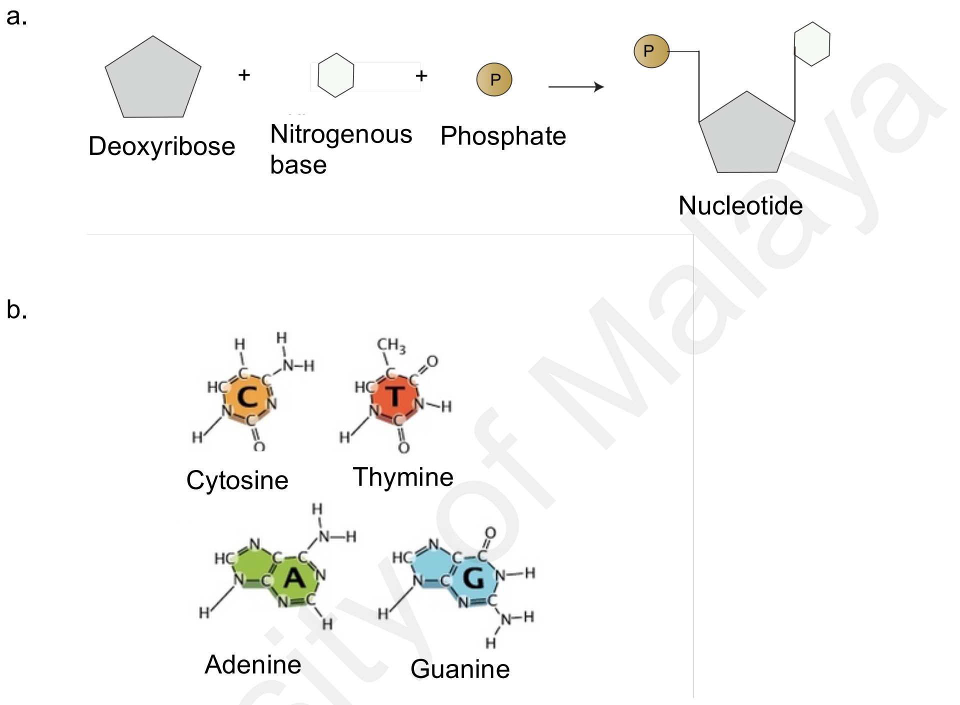


**Figure 2.2:** Nucleotide as the building block of DNA
Figure 2.2a represents a single nucleotide that is made up of a deoxyribose sugar, a single nitrogenous base and a phosphate group. Figure 2.2b represents the chemical structure for the nitrogenous base. The nitrogenous base can be cytosine, thymine, adenine or guanine in DNA. Adapted from (Pray, 2008).

Single-stranded DNA (ssDNA) occurs as random coils (Kowalczyk, Tuijtel, Donkers, & Dekker, 2010) (Figure 2.3). When two ssDNA intertwined into helix formation through hydrogen bonding, it formed double-stranded DNA (dsDNA). The structural appearance of dsDNA is known as DNA double helix. DNA double helix can adopt different conformation (right and left-handed helices), but right-handed helices are energetically favourable than left-handed helices due to less steric hindrance between their side chains and the backbone. The DNA sugar-phosphate backbones have direction, one of the DNA ends terminates with 5'-phosphate group (therefore known as 5' terminal) and another end terminates with 3'-OH group (therefore known as 3' terminal). In a dsDNA, 5'-end from one strand will bind with 3'-end from another strands and vice versa (Figure 2.4). The structure has a diameter of 2 nm and a complete turn of 10.4-10.5 base pairs per turn in B-form of DNA (Wang, 1979).

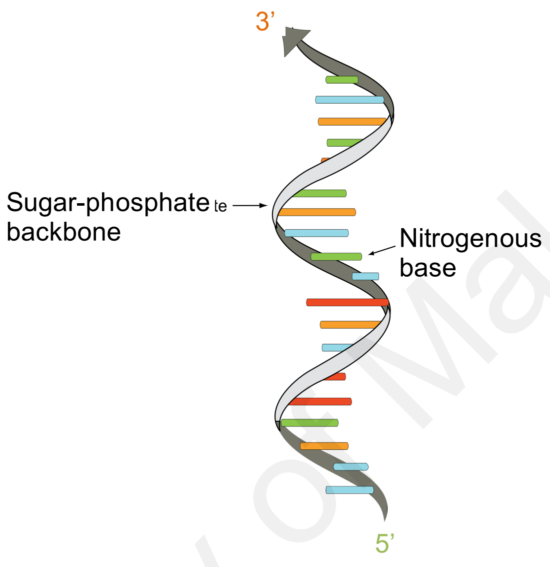


**Figure 2.3:** Single-stranded DNA in random coil conformation

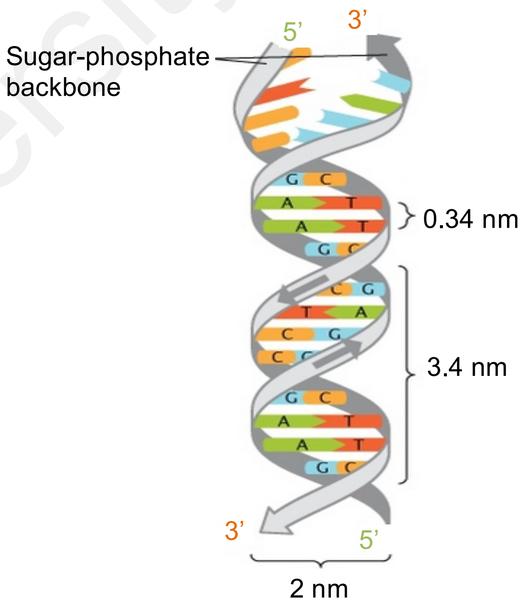


**Figure 2.4:** DNA double helix structure
The bases (adenine (A), thymine (T), guanine (G) and cytosine (C)) form Watson-Crick base pairing. Retrieved from (Pray, 2008).

### 2.2.1 DNA Double Helix Structure

The helical arrangement in DNA structure is determined by a number of helical parameters (Dickerson, 1989). These parameters use to describe the double helix conformation can be sorted into two categories; base pair parameter and base step parameter (Figure 2.5). The base pair parameter is the translational and rotation of bases within a base pair while base step parameters elucidate the translational and rotation between two neighbouring stacked base pairs (Ho & Carter, 2011). The *Rise* and *twist* parameters refer to the relative angle and the distance between two neighbouring stacked base pairs. *Slide* increases the diameter of DNA helical structure and later affects the helical twist. A-form of DNA has a large *slide* between its base pairs, but B-form DNA has smaller *slide* and thus placing the base pairs to stack on top of each other. This has caused A-DNA to have a larger overall diameter of approximately 23 Å (2.3 nm) as opposed to B-DNA 20 Å (2 nm). These different helical parameters in DNA have resulted in major forms of DNA such as A-form and B-form.

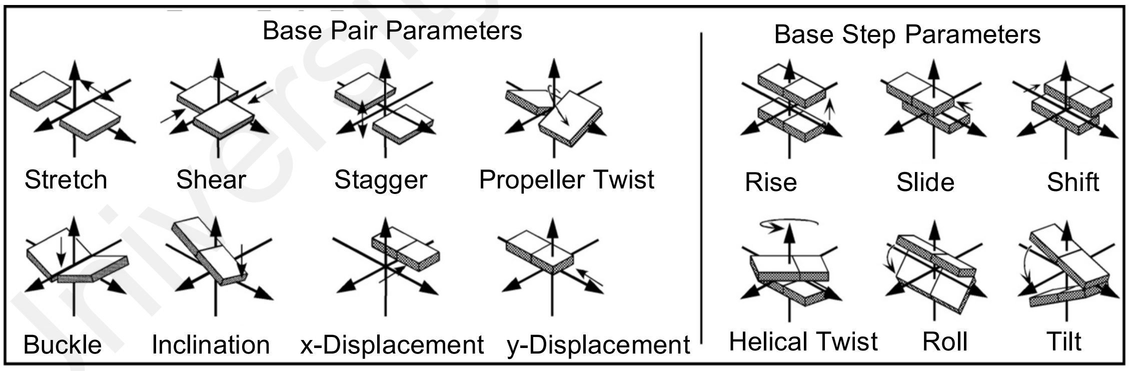


**Figure 2.5:** The base pair and base step parameters for DNA helix structure
Each DNA base (adenine, thymine, guanine, cytosine) is depicted as a block. Retrieved from (Ho & Carter, 2011).

A base pair seldom exhibits a perfect flat plane since each base has a slightly different *roll* angle in relative to one another. As a result, the measurement is conducted using the rotation per residue or *twist* angle in reference to the angle between two neighbouring base pairs. The *twist* angle can be calculated using the following case.

Considering that there are 10.5 base pairs per turn for B-DNA, and each helical turn is 360° therefore the twist angle would be 34.3° (360 / 10.5 = 34.3°) (Figure 2.6).

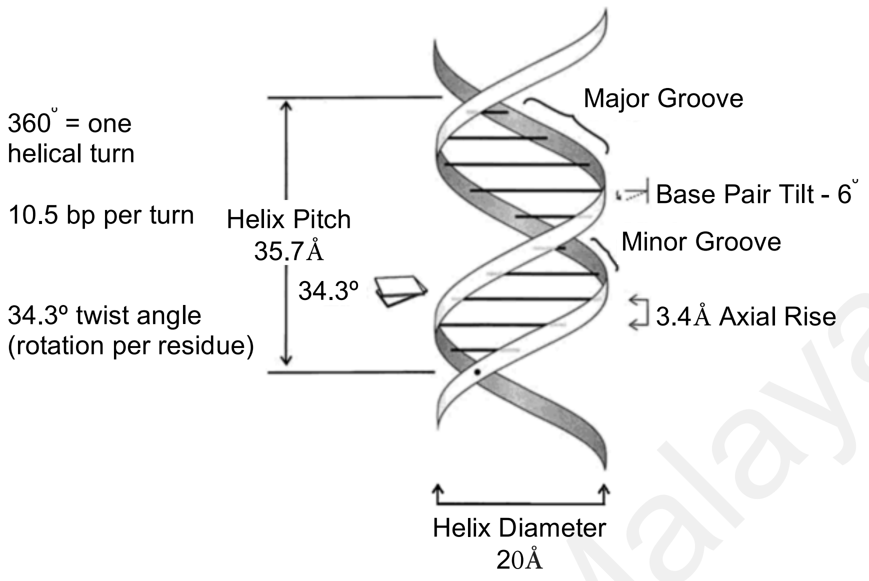


**Figure 2.6:** Parameters to calculate DNA double helix
The axial rise is the distance between two planar base pairs and tilt is the deviation from the horizontal plane. Retrieved from (Sinden, 1994).

### 2.2.2 The Stability of DNA Double Helix

As the DNA function is to encode genetic information, DNA needs to be chemically stable and avoids high mutation rate that will cause the stored genetic information to degrade over subsequent generations (Leu, Obermayer, Rajamani, Gerland, & Chen, 2011). Besides the previously mentioned hydrogen bond (A bind to T, G bind to C), the stability of the entire DNA double helix is also governed by the base-stacking interactions between adjacent bases (Yakovchuk, Protozanova, & Frank-Kamenetskii, 2006). The base stacking interaction is determined by the thermodynamics energy (Table 2.1) based on rules such as Nearest-neighbor parameters (Breslauer, Frank, Blocker, & Marky, 1986).

**Table 2.1:** Nearest-neighbor thermodynamics (Breslauer et al., 1986)

| Interaction | Thermodynamic parameter (ΔG°) |
|---|---|
| AA/TT | 1.9 |
| AT/TA | 1.5 |
| TA/AT | 0.9 |
| CA/GT | 1.9 |
| GT/CA | 1.3 |
| CT/GA | 1.6 |
| GA/CT | 1.6 |
| CG/GC | 3.6 |
| GC/CG | 3.1 |
| GG/CC | 3.1 |

All values are in reference to the disruption of the interaction in an existing duplex at 1M NaCl (1 molar of sodium chloride), temperature of 25 °C and pH 7. The unit for ΔG° is kcal/mol of interaction.

### 2.2.3 The Structural Flexibility of DNA

Since DNA is naturally contained inside the nucleus cell in a very condensed and folded form (known as chromatin) (Richmond & Davey, 2003), the flexibility of DNA is therefore very crucial. The flexibility of DNA is greatly affected by its base composition and sequence (Geggier & Vologodskii, 2010; Hormeño et al., 2011; Travers, 2004). Despite the fact that stiffness of DNA is commonly associated with guanine and cytosine content, yet the presence of base pairs repeat can also affect its stiffness (Hogan & Austin, 1987; Hogan, LeGrange, & Austin, 1983). In a case study to investigate the effect of guanine and cytosine (GC), DNA with high GC (70%) was reported to be significantly harder to stretch than the 50% GC content DNA (Hormeño et al., 2011). Despite the reduction in elasticity, double helix arrangement of the DNA remains the same (Hormeño et al., 2011).

The bending of DNA is crucial for interaction with protein in order to regulate the biological function of DNA such as gene expression (Lewis et al., 1996; Schultz, Shields, & Steitz, 1991; Schumacher, Choi, Zalkin, & Brennan, 1994). The local DNA bending is the deviation of two or three base pairs from a straight helix in a section of a helical repeat (Figure 2.7a) and it commonly takes place when the neighbouring base pairs roll on each other (Goodsell & Dickerson, 1994). Figure 2.7b showed the macroscopic curvature in DNA and the curvature is measured over several helical repeats that is approximately twenty or more base pairs (Goodsell & Dickerson, 1994). BEND (Goodsell & Dickerson, 1994) is an example of program used to calculate the DNA local bending and macroscopic curvature by using the bending model that had score for twist, roll and tilt parameters (Bolshoy, McNamara, Harrington, & Trifonov, 1991; Cacchione, Santis, Foti, Palleschi, & Savino, 1989; Calladine, Drew, & McCall, 1988; Satchwell, Drew, & Travers, 1986).

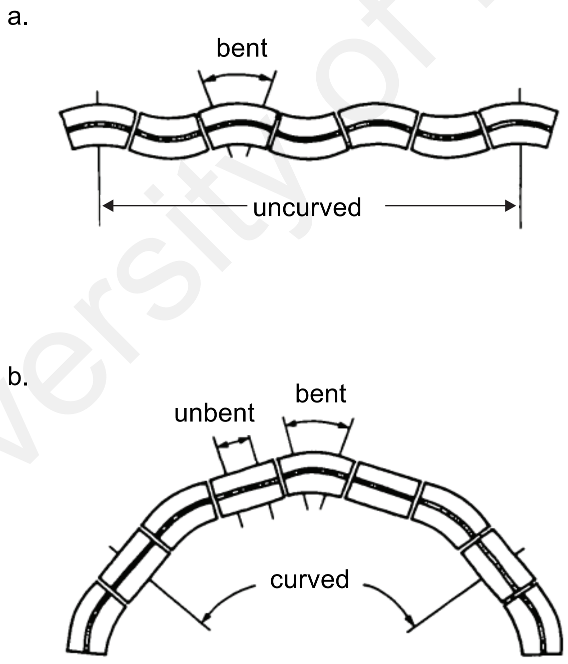


**Figure 2.7:** The DNA local bending and macroscopic curvature
Each segment represents 5 base pairs or half a turn of B-DNA helix. Figure 2.7a represents uncurved DNA that has an equivalent local bend at each base step. Figure 2.7b represents alternate fraction of bent and unbent segments in the DNA that caused macroscopic curvature. Retrieved from (Goodsell & Dickerson, 1994).

## 2.3 DNA Nanotechnology

The mechanical properties of DNA structure have great impact on the functional activities in DNA. Other biological substrates, for example protein serves as an important role for building the structural, catalytic and regulatory components of cells. Despite these, the folding and assembly of proteins remain challenging to be predicted and designed due to the complexity of their 3-dimensional structure (Poole & Ranganathan, 2006). On the contrary, DNA which serves as genetic information and has high chemical stability and highly predictable binding properties is an ideal substrate to be recruited into the construction of nanostructures via self-assembly process (Feldkamp & Niemeyer, 2006; Gothelf & LaBean, 2005; Seeman, 2005). Ribonucleic acid (RNA) has recently emerged as programmable structures (Jaeger & Leontis, 2000; Jaeger, Westhof, & Leontis, 2001). Although it has chemical structure similar to DNA, RNA is chemically more unstable than DNA (Elliott & Ladomery, 2011) and is prone to fold into complex tertiary structures similar to protein.

DNA nanotechnology is a branch of nanotechnology associated with the design, study and implementations of artificial structures based on DNA (Seeman, 1999; Seeman, 2005; Seeman, 2007). In this field, rather than using DNA as genetic information, it is used as engineering materials (Aldaye et al., 2008; Kuzuya & Komiyama, 2010; Seeman, 2010). DNA nanotechnology harnesses the information processing capabilities of nucleic acids to *program matter* (Toffoli & Margolus, 1991); by executing instruction at the molecular level. Information processing in nature is distinctly different compared to silicon machines, by which nature exploits the physics of the materials to achieve the solutions effectively with minimal energy (Conrad, 1972). The concept of informed matter (Lehn, 2004) is to design molecules to carry information that will enable them to autonomously interact with each other. It is known as the orchestration of self-organization concept (Zauner, 2005). The applications of constructing DNA nanostructures in the field of nanotechnology will be presented in the following section.

### 2.3.1 Implementation of DNA Nanotechnology

The programmability of nucleic acids to form nanodevices facilitates the advancement across various disciplines such as in nanoelectronic circuitry (Le et al., 2004; Liu, Park, Reif, & LaBean, 2004), drug and therapeutic delivery system (Lee et al., 2012), diagnostic probe that functions as pH reporter (Surana, Bhat, Koushika, & Krishnan, 2011) and as DNA cargo that transporst and delivers molecular payloads (Bhatia, Surana, Chakraborty, Koushika, & Krishnan, 2011; S.M. Douglas, Bachelet, & Church, 2012). In order to provide insights into the applications of the constructed DNA nanodevices, mechanism of these nanodevices mainly in drug delivery and in nanoelectronic circuitry will be further explained.

Small interfering RNA (siRNA) is shown as a therapeutic agent that is able to suppresses the expression of the targeted genes (Bumcrot, Manoharan, Koteliansky, & Sah, 2006; Elbashir et al., 2001). For instance, in order to deliver siRNA into the targeted tumour in mouse model, DNA nanodevice bearing siRNA and folate moieties was constructed and it was then delivered intravenously via injection into the mouse (Lee et al., 2012). The fundamental basis is that folate receptors are overexpressed in the cancerous cells (Zwicke, Mansoori, & Jeffery, 2012), giving an opportunity for DNA tetrahedral with folate moieties and siRNA to bind to the folate receptors on the cancerous cells. Details of this mechanism are illustrated in Figure 2.8.

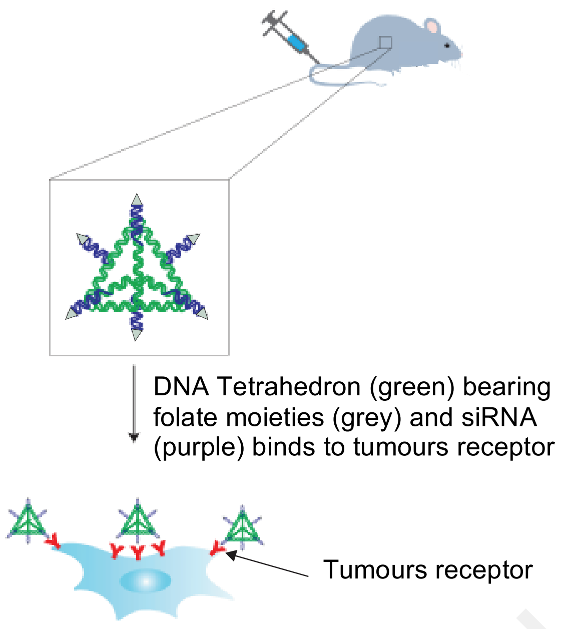

DNA Tetrahedron (green) bearing folate moieties (grey) and siRNA (purple) binds to tumours receptor
Tumours receptor

**Figure 2.8:** DNA nanodevices with tetrahedral structure
DNA tetrahedral bearing folate moieties (grey triangles) and siRNA (purple) bind to the tumour cells with overexpressing folate receptor (red). Retrieved from (Surana, Shenoy, & Krishnan, 2015).

Recent development has seen the integration of DNA nanodevices in electronic field including building a nanometer scale wire by metalizing the DNA nanotube with silver (Liu et al., 2004) and the building of 2-dimensional DNA array as nanoelectronic circuitry system (Le et al., 2004). Many electronic circuit subsystems such as dynamic random-access memory (DRAM) and programmable logic array (PLA) are organized in two-dimensional array (Rabaey, Chandrakasan, & Nikolic, 2003). Le et. al. (2004) took the same principle in their initial step towards manufacturing high-density nanoelectronic circuit, by constructing self-assembled DNA structure with gold prototype nanoelectronic components that turned into two-dimensional (2D) array of gold nanoparticles. These arrays could serve as nanoscale size memory by allowing the electronic state of these nanoparticles to be read by the scanning probes. The DNA 2D scaffold was constructed from a set of 21 synthetic oligonucleotides (DNA strands) that was used to form four different tiles (blue, red, green and yellow) (Figure 2.9a-d). These tiles further self-assembled and formed into the arrangement of 2D array (Figure 2.9e-g). The gold nanoparticles was further added onto the 2D array scaffold.

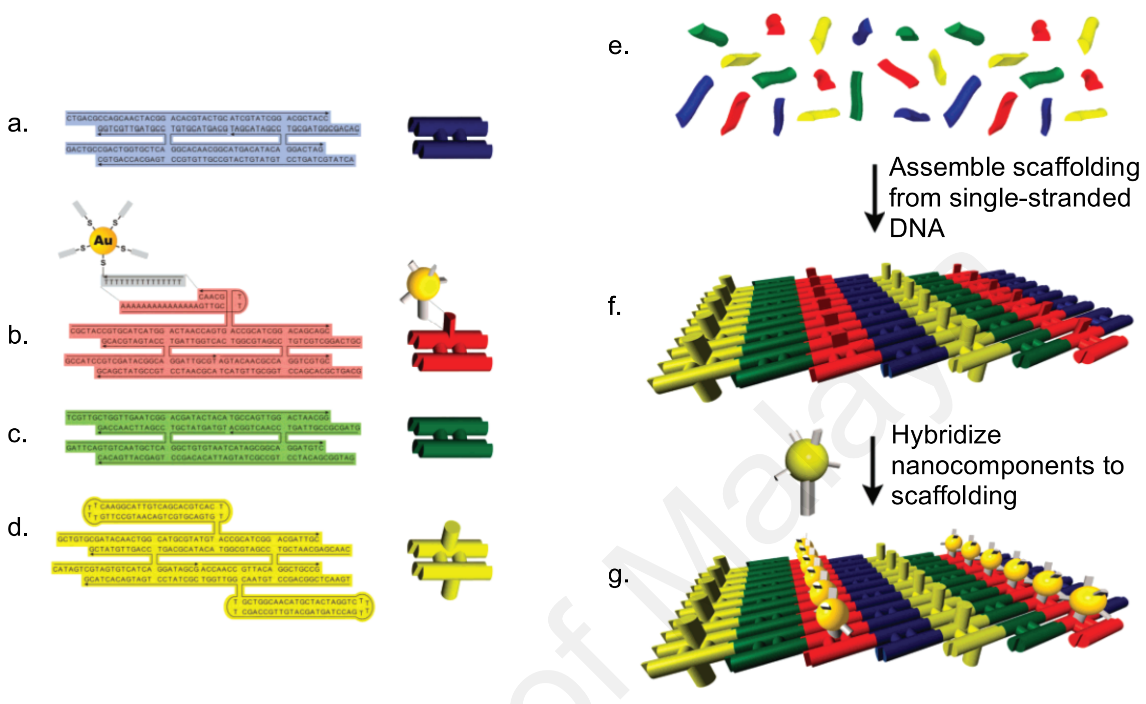

a.
b.
c.
d.
Au
e.
Assemble scaffolding from single-stranded DNA
f.
Hybridize nanocomponents to scaffolding
g.

**Figure 2.9:** Nanoelectronics device
Figure 2.9a-d represents the geometric model of each nanocomponent that is made from self-assembly DNA. Four basic components used to form the 2D scaffold are labelled using different colors: blue, red, green and yellow. Figure 2.9e-g represents the flowchart of assembly steps for constructing two-dimensional arrays. Figure 2.9e represents DNA strands in the solution. Figure 2.9f represents the deposition of DNA scaffold onto the mica surface. Figure 2.9g represents combination of the scaffold with gold particles. The gold particle was labelled as Au, bind to the red color component. Retrieved from (Le et al., 2004).

## 2.4 Computational Tools for Nucleic Acids Sequence and Structural Design

A number of computational tools have been developed to aid the nucleic acid sequence and structural analysis such as RNAstructure (Reuter & Mathews, 2010), Vienna RNA (Hofacker, 2003) and RNAsoft (Andronescu, Aguirre-Hernández, Condon, & Hoos, 2003). For instance, Vienna RNA offers the prediction of nucleic acids secondary structure and also consensus secondary structure for a set of aligned sequences (Hofacker, 2003). Strategies to assist nucleic acid sequence design are common in nanotechnology field as well as in general lab applications such as microarray probe selection and primer design for polymerase chain reaction (PCR) process (Ben-Dor, Karp, Schwikowski, & Yakhini, 2000; Gerry et al., 1999; Li & Stormo, 2001). Although the existing computational tools are universal and all-inclusive with broad applications in various fields, they are not specifically developed to construct DNA nanostructures according to the newly proposed schema. However, their concepts are crucial in sequence design process and have been integrated into this new computational tool to generate initial random sequences, prediction of free energy and calculation of melting temperature for DNA sequences.

### 2.4.1 DNA Sequences Design

In this work, *DNA Sequence Generator* (*DSG*) (Feldkamp et al., 2003; Feldkamp et al., 2001) was used to generate single-stranded deoxyribonucleic acid (ssDNA) as initial input as part of module 1 design tool (objective 2). The availability of *DSG* to work as a standalone program in Window operating system was used as part of our design tool. Different sequence design strategies (Feldkamp, Rauhe, & Banzhaf, 2003; Feldkamp, Saghafi, & Rauhe, 2001; Frutos et al., 1997; Marathe, Condon, & Corn, 2001) have been introduced to design DNA sequences that obeyed certain physical constraints. These constraints, especially in term of uniqueness are essential for DNA to act efficiently as molecular bar codes or tag identification in a chemical library (Brenner & Lerner, 1992; Shoemaker, Lashkari, Morris, Mittmann, & Davis, 1996) and to perform computation at molecular level (Adleman, 1994).

The above cases required the avoidance of non-specific binding and every DNA is expected to bind only to its targeted sequence. However, the need to have these sequences to be as dissimilar as possible creates a multitude of selection and combinatorics problem. Given that a DNA strand with length *n* and at each position in the sequence, there are four probable nucleotides; adenine (A), cytosine (C), guanine (G) and thymine (T). The number of possible combinations would be up to $4^n$. The task to design DNA words, with specified word size over the four alphabets that satisfied certain combinatorial constraints would be time consuming if it is done without the aid of computational tools.

This program employed graph-based algorithm (Niehaus, 1998) to generate a set of unique sequences that obeyed the parameters such as melting temperature and guanine-cytosine ratio. The sequence generation methods prohibit the existent of complementary DNA sequence and self-complementary base pairing within the same set of generated sequence generated. Self-complementary refers to situation whereby a sequence is able to fold with itself, and create a double-stranded like structure. In addition, DNA sequences generated by *DSG* showed a better performance in term of measurement of dissimilarity between a set of DNAs compared to other tools (Feldkamp et al., 2003).

*DSG* uses directed graph to generate sequences. Each position can accommodate either A, C, G or T bases and sequences are generated as the paths proceed throughout the graph. During each cycle of sequence generation, a base strand (or node) may only be used once, so that the paths of any two sequences will not have common node. In a similar way, if a node is being used, its complement will not be utilized for the remaining sequences and existence of self-complementary base strands are restricted. This scenario is illustrated in Figure 2.10. Algorithm 1 described the protocol used to generate the sequences (Feldkamp et al., 2001).

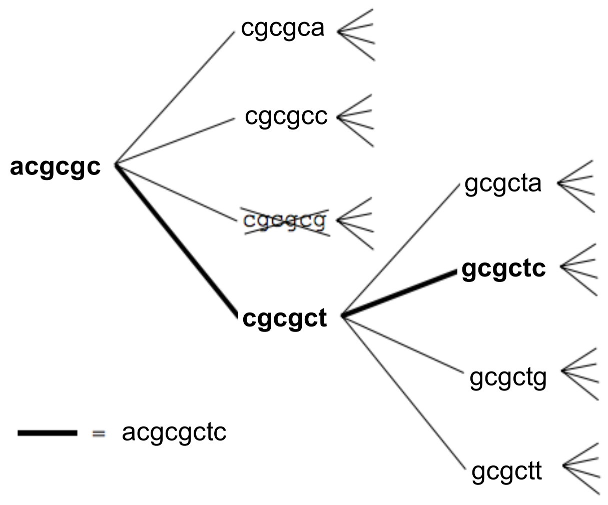


**Figure 2.10:** The graph for the sequence generation
The graph showed the paths taken to generate the sequences. Note that the node cgcgcg is self-complementary and therefore it's marked as forbidden. Retrieved from (Feldkamp et al., 2001).

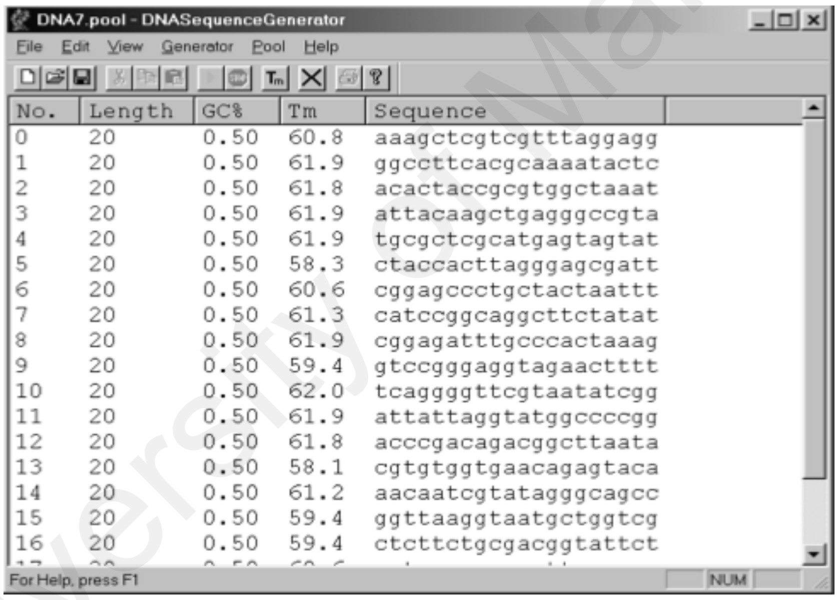
**Algorithm 1** Sequence generation algorithm (Feldkamp et al., 2001)

```
1:   create graph for base strands with x length
2:   labelled base strands that do not fulfil criteria as Forbidden
3:   for each sequence do
4:       if StartNodes exist then
5:           randomly choose one StartNode
6:           for each StartNode  do
7:               if graph path less than x length then
8:                   mark StartNode as Used
9:                   if Successor exists then
10:                      choose an Unused SucessorNode which are not Forbidden
11:                      marked the SucessorNode and its complement as Used
12:                  else
13:                      look for another StartNode
14:                      remark the existing StartNode as Unused
15:                  end if
16:              end if
17:              else if graph path equal to x length then
18:                  evaluate that new sequence
19:                  if new sequence passed all filter then
20:                      add new sequence into the pool
21:                  end if
22:                  if new sequence fail any of the filter then
23:                      find new start Nodes
24:                  end if
25:              end if
26:          end for
27:      end if
28:  end for
```

*DSG* provides an aid in searching a set of distinct DNA sequences without the manual checking for all possible combinations in $4^n$. However, the generation of the DNA as initial sequences required downstream works in order to ensure some of these sequences can be used to form different configurations. The command line to execute *DSG* is as following.

*dsg $file_name.dln*

The criteria for sequence generation are compiled into a file (*$filename.dln*). The main window (Figure 2.11) showed output sequences generated from *DSG* according to the user specified parameters.


DNA7.pool - DNASequenceGenerator
File Edit View Generator Pool Help
Tm
For Help, press F1
NUM

```
No.  Length  GC%   Tm    Sequence
0    20      0.50  60.8  aaagctcgtcgtttaggagg
1    20      0.50  61.9  ggccttcacgcaaaatactc
2    20      0.50  61.8  acactaccgcgtggctaaat
3    20      0.50  61.9  attacaagctgagggccgta
4    20      0.50  61.9  tgcgctcgcatgagtagtat
5    20      0.50  58.3  ctaccacttagggagcgatt
6    20      0.50  60.6  cggagccctgctactaattt
7    20      0.50  61.3  catccggcaggcttctatat
8    20      0.50  61.9  cggagatttgcccactaaag
9    20      0.50  59.4  gtccgggaggtagaactttt
10   20      0.50  62.0  tcaggggttcgtaatatcgg
11   20      0.50  61.9  attattaggtatggccccgg
12   20      0.50  61.8  acccgacagacggcttaata
13   20      0.50  58.1  cgtgtggtgaacagagtaca
14   20      0.50  61.2  aacaatcgtatagggcagcc
15   20      0.50  59.4  ggttaaggtaatgctggtcg
16   20      0.50  59.4  ctcttctgcgacggtattct
```

**Figure 2.11**: Set of DNA sequences
No. is the number of sequence, Length is the length of the sequences, GC% is the percentage of guanine-cytosine content, Tm is the melting temperature and Sequence is the DNA sequences generated by the program. Retrieved from (Feldkamp et al., 2001).

## 2.4.2 Calculation of Free Energy

The initial step to understand the mechanism of nucleic acid is to determine its structure (Mathews & Turner, 2006). In the earlier section, it is mentioned that besides hydrogen bonds that formed between complementary bases, base stacking interaction also greatly influenced the stability of DNA double helix structure (Yakovchuk et al., 2006). Therefore, the construction of DNA nanostructures or nanodevices needs to take into consideration the stability of the resulted double helix structure. The calculation of free energy is crucial as a key determinant of the stability of secondary structure formation. Free energy is predicted in the program *RNAstructure* (Reuter & Mathews, 2010) to measure the stability for the candidate structures. The structural prediction includes criteria such as the intramolecular and intermolecular binding. Intramolecular binding happens when a nucleic acid (DNA or RNA) strand forms hydrogen bond with its own strand and form pseudoknot structure in RNA. Intermolecular binding occurs when a nucleic acid forms hydrogen bond with another nucleic acids strand.

*RNAstructure* (Reuter & Mathews, 2010) has been developed to predict free energy in structural determination of nucleic acid (DNA and RNA). It is made up of several individual programs such as *AllSub* which is used to generate free energy for nucleic acid sequence (Duan, Mathews, & Turner, 2006; Wuchty, Fontana, Hofacker, & Schuster, 1999) and *bifold* program used to predict lowest free energy structure for two interacting sequences by allowing intramolecular base pairs (Mathews, Burkard, Freier, Wyatt, & Turner, 1999). *DuplexFold* program also carried out the same function as *bifold* except that it does not allow intramolecular base pairing (Piekna-Przybylska, DiChiacchio, Mathews, & Bambara, 2009).

For DNA, the parameters to calculate free energy are derived from the experimental literature (Allawi & SantaLucia, 1997; Allawi & SantaLucia, 1998a; Allawi & SantaLucia, 1998a, 1998b; Allawi & SantaLucia, 1998b; Bolewska, Zielenkiewicz, & Wierzchowski, 1984; Bommarito, Peyret, & SantaLucia, 2000; Breslauer et al., 1986; Leonard, Thomson, Watson, & Brown, 1990; Moody & Bevilacqua, 2003; Nakano, Moody, Liang, & Bevilacqua, 2002; Peyret, Seneviratne, Allawi, & SantaLucia, 1999; Plum, Grollman, Johnson, & Breslauer, 1995; Wu, Nakano, & Sugimoto, 2002). As for

RNA, the parameters to calculate free energy were extracted from Turner group (Lu, Turner, & Mathews, 2006; Mathews et al., 2004; Xia et al., 1998).

The availability of *RNAstructure* to run locally enabled the integration of this program to aid in the calculation of the free energy in our DNA sequences. Among the programs included, as part of the newly developed computational tool is the *AllSub* program that is used to generate all possible low free energy structures for a nucleic acid sequence while the *DuplexFold* program is for the prediction of lowest free energy between two interacting nucleic acid sequences. The command to run program *Allsub* is as following.

*AllSub A.fasta AllSub.ct --DNA*

The variable *A.fasta* refers to the file name of the nucleic acid, DNA or RNA. *--DNA* refers to the type of nucleic acid, DNA or RNA.

The command to run program *Duplexfold* is as following.

*DuplexFold A.fasta B.fasta DuplexFold.ct –DNA*

The variable *A.fasta* and *B.fasta* refer to the file names of the nucleic acids, DNA or RNA. *--DNA* refers to the type of nucleic acid, DNA or RNA.

### 2.4.3 Calculation of DNA Melting Temperature

The stability of the DNA structure formation needs to take into consideration the melting temperature of the DNA sequences. This is due to the fact that melting temperature is the temperature at which half of the DNAs dissociate into single-stranded state (ssDNA) and the other half are still in double-stranded form (Wetmur, 1991). The melting temperature will be used to determine the start nodes in the module 2: DNA connectivity tool (second objective). The study hypothesizes that the melting temperature of hydrogen bonds for each base pair is non-uniform. Therefore, the start nodes are important as it act as the starting points for different DNA to bind together. The prediction of melting temperature is important for PCR reaction. Upon exceeding the melting temperature, the double-stranded DNA will lose its structural formation and separate into ssDNA. The ssDNA will appear as random coil and this process is known as denaturation. The melting temperature is highly dependent on the length of the DNA and its nucleotides coupled with salt composition. Unified Nucleic Acid Folding (*UNAFold*) is an integrated program that simulates the folding, hybridization and melting pathways as well as for prediction of melting temperatures (Markham & Zuker, 2008).

One of the modules available under *UNAFold*, the *melt.pl* is used to compute melting temperature for both hetero-dimer (hybridization of two distinct sequences) and homo-dimer (hybridization of two same sequences) (Markham & Zuker, 2008). Hybridization is another term commonly used by molecular biologist to refer to nucleic acid sequences that bind or form complementary using hydrogen bonds. Therefore, we recruited *UNAFold* as an external program in order to obtain the melting temperature for DNA and then employed it into the DNA connectivity tool to generate the start nodes for the graph search in chapter 5. The command line to execute *melt.pl* is as below.

```
melt.pl –NA DNA –sodium 0.05 –temperature $T –Ct 10e-6 A.seq B.seq
```

The variable *–NA* refers to nucleic acid type, DNA or RNA. -temperature (*$T*) is the temperature for energy minimization. The default value is set at 37 $^{o}$C. *-sodium* is the concentration in molar for sodium ion while *–Ct* is the total strand concentration in molar. Variable *A.seq* and *B.seq* refer to the file name of the DNA or RNA sequences.

## 2.5 The Tetris game and Self-organization Mimicry

In this work, the first hypothesis for implementing a new schema in constructing DNA nanostructures was derived from the popular computer game, Tetris invented by mathematician Alezey Pazhitnov (Pazhitnov, Gerasimov, & Pavlovsky, 1985). A player is given pieces of Tetris shapes in sequential order (Figure 2.12) starting from the top row of a game board (Figure 2.13). While it falls down, player can rotate each piece either in the clockwise or anticlockwise direction before it stops moving permanently.

The rule is that when the entire row *r* of the board is occupied, that respective row *r* will be cleared. The objective of this game is to increase the number of cleared rows while reducing the height of the occupied space before the player lost (Breukelaar et al., 2004). Tetris was an example of a non-deterministic polynomial-time hard (NP-hard) problem because there was no systematic method to calculate the essential moves to "win", even when pieces of Tetris shapes were known in advance (Breukelaar et al., 2004).

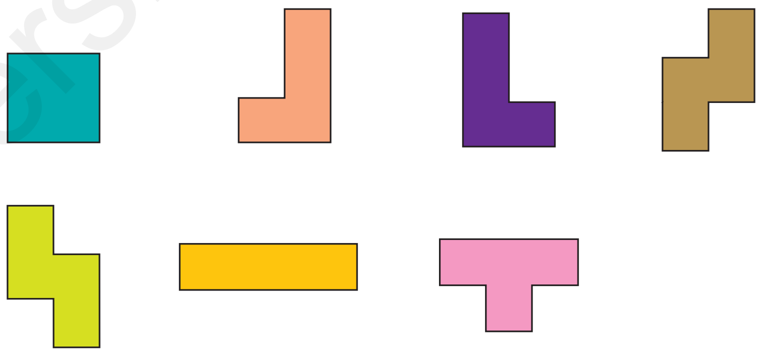


**Figure 2.12:** The Tetris shapes

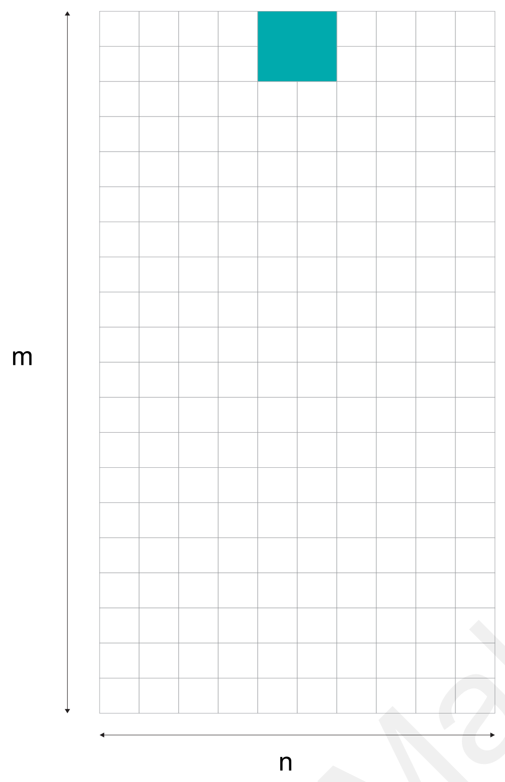


**Figure 2.13:** A Tetris board with dimension of *m* x *n*
The standard board size is set at 10 x 20 (width x height), although it may vary.

The overview of this work was to treat the entire DNA nanostructure as a Tetris board (size *m* x *n*) by filling it with pieces of Tetris shapes. In order to execute this hypothesis, DNA sequences were used to build the Tetris shapes that hereinafter were named as DNA Tetris shapes.

The second hypothesis in driving the development of a new schema for DNA nanostructures construction was derived from the principle of self-organization occurred in nature. This principle has been adapted across different disciplines ranging from physics, chemistry as well as computer science (Bray, 1921; Castets, Dulos, Boissonade, & De Kepper, 1990; Misteli, 2001; Pettinaro et al., 2002; Sahin et al., 2002; Tabony, 2006; Zhabotinsky & Zaikin, 1973). The dynamism of information technology, for instance worldwide internet computing and ubiquitous computing has made it unfeasible to predict the complicated chain of interactions and also the continuous side effect from the software applications (Mameia, Menezesb, Tolksdorfc,

& Zambonelli, 2006). In order to overcome these restrictions, each component must be able to "think" by itself and be able to handle unexpected dynamic situations. An example of the self-organizing phenomenon occurred in nature is the construction of tobacco mosaic virus coat (Camazine et al., 2001; Nicolis & Prigogine, 1977) by which the pieces of the virus's coat are able to execute like a puzzle and self-assembled themselves without interference. Hence, the resulted structure is highly stable (lowest free energy) (Camazine et al., 2001; Karsenti, 2008; Nicolis & Prigogine, 1977).

This showed that self-organizing system is essentially self-managing and not influenced by external entities (Zambonelli, 2006). Self-managing includes the capability of the software and information systems to have self-adaptation, self-configuration and self-healing properties (Zambonelli, 2006). It is fundamentally different from the present computational system, which is created by the designer using the traditional top-down technique. The main advantage of self-organization system is that it is highly robust because it can tolerate errors, deviation or partial break down (Heylighen & Gershenson, 2003). It also has the capability to adapt to any changes that occurred in the environment, repair the damage and then return to their beginning state (Heylighen & Gershenson, 2003).

Examples of robotic applications that utilize the self-organization principle into the formation of shape and pattern are mobile-robot formations and modular robot self-configurations (Nagpal, Zambonelli, Sirer, Chaouchi, & Smirnov, 2006). The study focused on the self-assembly ability of the mobile-agent swarm to program and robustly self-organize itself into coordinate system (Cheng, Cheng, & Nagpal, 2005). Figure 2.14 showed the mobile-agent swarm aggregates into a particular formation and a shape is able to recover from "death".

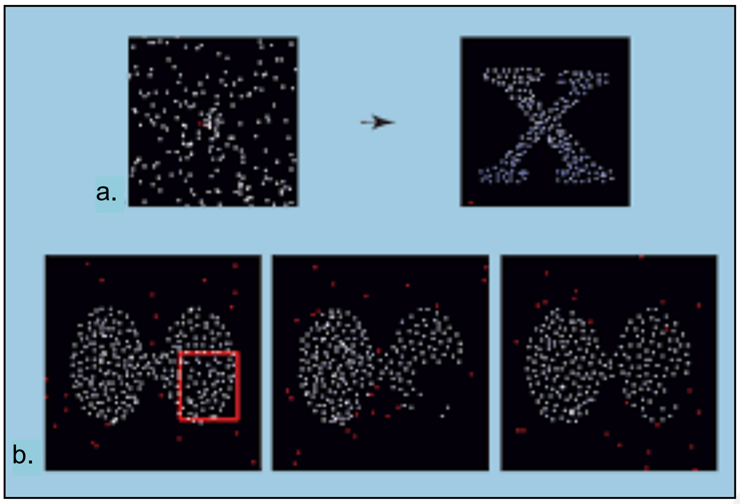


**Figure 2.14:** Self-organization in mobile-agent swarm
Figure 1.4a represents a mobile-agent swarm formed into specified shape. Figure 1.4b represents a shape that can return from death or displacement. Retrieved from (Nagpal et al., 2006).

Another example of implementing self-organization into structural formation was the Swarm-bots project (Pettinaro et al., 2002; Sahin et al., 2002). This project focused on the self-organization behaviour in structural assembly with the goal to help the robot to function; such as for the robot to form a long chain to overpass a crack in the terrain, or to climb a high step. Swarm-bot was formed via the self-assembly of individual components known as *s-bots* (Mameia et al., 2006). The hypothesis to integrate the self-organization principle into constructing DNA nanostructure was derived following the development of self-organization according to the cases mentioned above. Each piece of DNA Tetris shape was able to self-assemble itself into the specified configurations. In order to provide a simpler understanding, each DNA Tetris shape is alike to the *s-bot* mentioned in the example above.

The entire chapter conducted a background study on the core topics associated with the research subjects in various aspects in order to provide in depth review on the subject matter. Following these reviews, the work continues by presenting the concept of DNA Tetrominoes in the upcoming chapter 3.

# CHAPTER 3: THE PRESENTATION OF DNA TETROMINOES CONCEPT

## 3.1 Structural Assembly in Linear DNA

In the early 1980s, Seeman discovered that DNA is able to form other configuration such as a junction-like structure instead of only double helix structure (Seeman, 1982). From a topological point of view, DNA double helix is just a linear line and thus it is not considered as an ideal substrate for structural engineering. The idea to break DNA from its linearity is to construct synthetic branched molecules (Seeman, 1982). Naturally occurring branched DNA found in biological systems is known as Holliday junction (Holliday, 1964). The existence of branched structure proved that the DNA strands are able to associate and produce junction points or to self-assemble and form structures. If a pair of DNA strand is considered as a two-lane highway (Figure 3.1a), then the intersection point between these lanes is regarded as a junction point (Seeman & Lukeman, 2005). Therefore, in the context of four-way intersection it would allow the traffic to flow across four different lanes or directions (Figure 3.1b). A branched DNA can also be thought of as the four-way intersections.

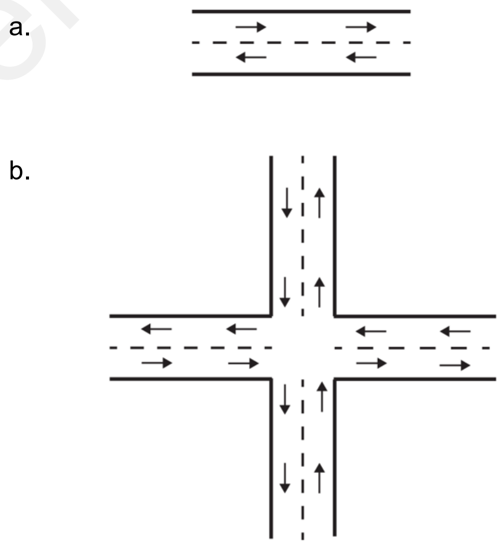


**Figure 3.1:** The illustration of linear DNA and branched structure
Figure 3.1a represents a two-lane highway with cars moving towards two different directions. Figure 3.1b represents four-way intersections with cars driving towards four different directions.

Branched DNA, for instance Holliday junction is an intermediate in genetic recombination (Holliday, 1964). Genetic recombination is a biological process to exchange DNA segments between two chromosomes in order to produce novel combinations of genetic material in the offspring. Chromosome is a DNA molecule packaged into thread-like structures. Branched DNA has the tendency of having sequence symmetry that allowed its branch point to migrate and cause the junction to be mobile (Figure 3.2) (Seeman, 1982). Sequence symmetry occurred when four DNA strands are actually made up of two pairs of DNA strands with same sequence (Seeman, 2010). However, it is more useful to design and assemble synthetic DNA sequences that lack sequence symmetry so that the resulted structure is more stable and immobile (Kallenbach, Ma, & Seeman, 1983) (Figure 3.3)

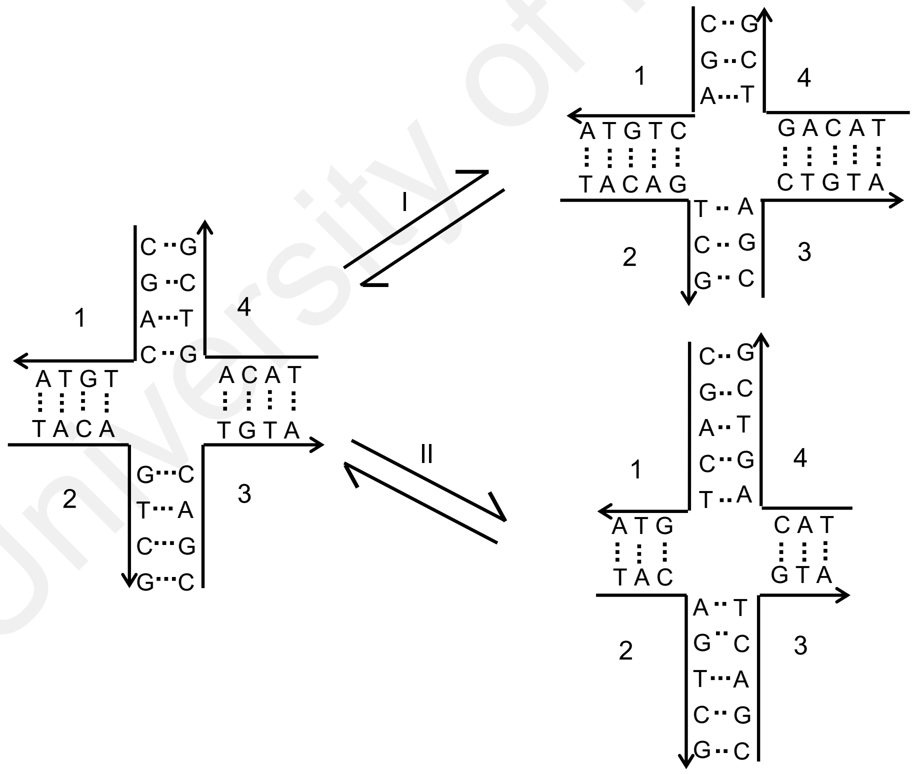


**Figure 3.2:** The illustration of mobile Holliday junction
The Holliday junctions with four DNA strands (1, 2, 3 and 4). Step I and II are the migration of the branch point towards two different directions. Sequence symmetry occurred when four DNA strands are actually made up from two pairs of same sequences. DNA strands 1 and 3 have the same sequence; DNA strand 2 and 4 are from another set of same sequence. Adapted from (Seeman, 1982).

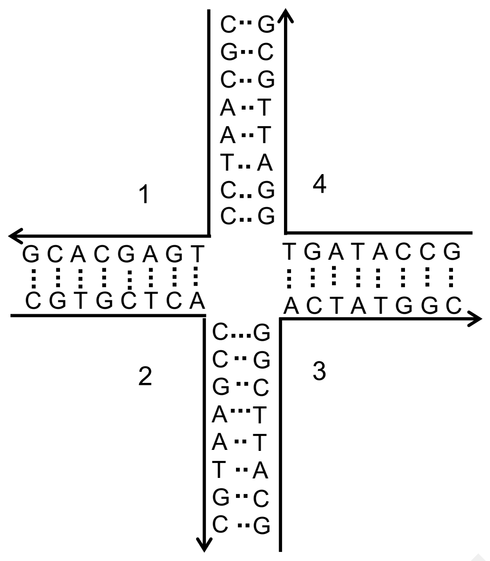
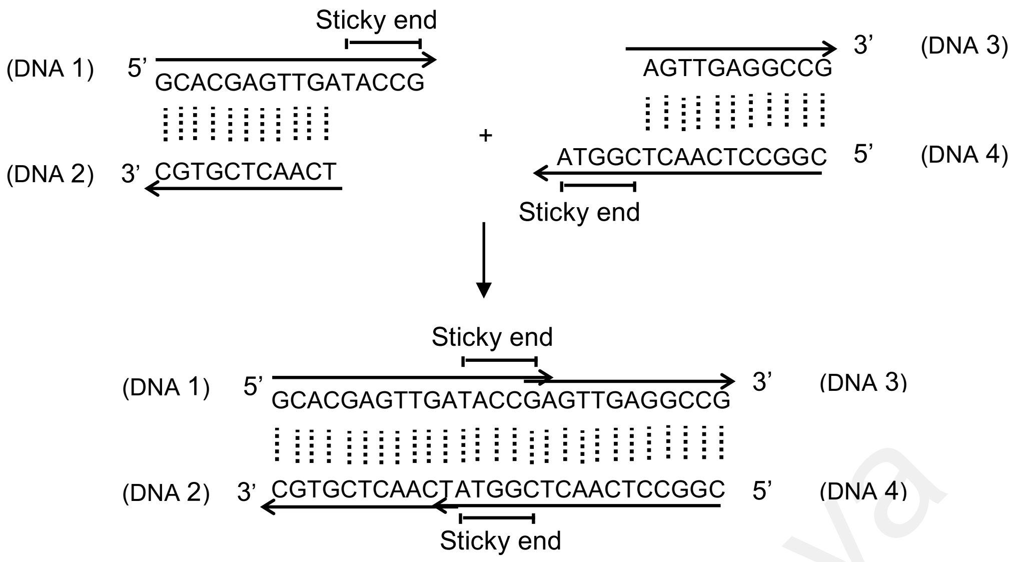


**Figure 3.3:** The illustration of an immobile Holliday junction
The Holliday junctions are made up of four DNA strands (labelled as 1, 2, 3 and 4). These sequences lack two-fold symmetry at the center region, therefore branch migration cannot occur. Adapted from (Seeman, 1982).

The ability to generate immobile-branched DNA molecule has allowed for different structural arrangements to be formed. In order to form a larger structure, many of these DNAs would need to join together using the sticky end. Sticky end is a fragment of DNA that has a stretch of unpaired nucleotides located at the terminal. This scenario is illustrated in Figure 3.4, which showed the use of sticky ends from DNA 1 and DNA 4 to form complementary base pairing and thus a larger structure. It demonstrates that DNA structural arrangement could be extended to produce structures like lattice form.



Sticky end
(DNA 1) 5' GCACGAGTTGATACCG
(DNA 2) 3' CGTGCTCAACT
+
3' (DNA 3) AGTTGAGGCCG
ATGGCTCAACTCCGGC 5' (DNA 4)
Sticky end
Sticky end
(DNA 1) 5' GCACGAGTTGATACCGAGTTGAGGCCG 3' (DNA 3)
(DNA 2) 3' CGTGCTCAACTATGGCTCAACTCCGGC 5' (DNA 4)
Sticky end

**Figure 3.4:** The sticky end
Sticky ends from DNA 1 form complementary with the sticky ends from DNA 4 through the formation of hydrogen bond. The dotted lines are the formation of hydrogen bonds. Adapted from (Seeman et al., 1999).

## 3.2 Construction of Fundamental Structures: From Linear DNA to Simple Lattice

Sticky ends have proven to be crucial in the formation of two-dimensional crystal design (Grunbaum & Shephard, 1986; Winfree, 1996). Winfree designed synthetic molecular units, by introducing two structural motifs, DAO (double crossover, antiparallel, odd spacing) and DAE (double crossover, antiparallel, even spacing) (Winfree et al., 1998) (Figure 3.5a) to form two-dimensional lattices. The DAO molecules have an odd number of half-turn (3 half-turns in ~21 base pairs) between the crossover points. On the other side, DAE molecules have an even number of half-turns (4 half-turns in ~21 base pairs). These designs depend critically upon the *twist* parameter in the DNA double helix, in which a complete turn takes place in 10.4-10.5 base pairs (Rhodes & Klug, 1980; Wang, 1979). These synthetic molecular units were designed based on Wang tiles specification; in which each tile is labelled with coloured edges. The tiles may be placed next to each other only if both of their edges are having identical color (Wang, 1961). These tiles will self-assemble into two-dimensional lattice crystals that obey colouring conditions.

In each design, the A and B tiles are arranged alternating to each other (Figure 3.5b) using sticky ends to "glue" among themselves. Therefore, sticky end sequences for each desired contact must be unique to ensure that the structures would form according to the design. The length of sticky ends is set at five nucleotides for DAO (Figure 3.5c) and six nucleotides for DAE, so that each correct contact contributes approximately 8 or 14 kcal mol$^{-1}$ to the free energy of association at 25 °C, according to nearest-neighbor model (SantaLucia, Allawi, & Seneviratne, 1996). The length of the double-crossover arms and the sticky ends, as well as the separation between crossover points, closely resemble the natural twist of the B-form DNA double helix structure.

To ensure that the component strands form the desired complexes, DNA strands must be carefully designed so that undesired base pairing and conformations are prohibited. Undesired base pairing or incorrect binding happened when a DNA strand forms complementary with DNA strands other than the one that is assigned to (Figure 3.6). Therefore, the solution to this design problem (Seeman, 1990; Sun, Brem, Chan, & Dill, 1995; Yue & Dill, 1992) is to maximize the free energy differences between the desired and all other probable conformations (Winfree et al., 1998). In each double-crossover sequence, there is no 6-bases complementarity unless required by the design and the occurrences of 5-bases complementarity is uncommon (Winfree et al., 1998). This feature is to ensure that during self-assembly process, the DNA strands would spend less time in the undesired associations and in turn could form the desired structures with higher yield (Winfree et al., 1998).

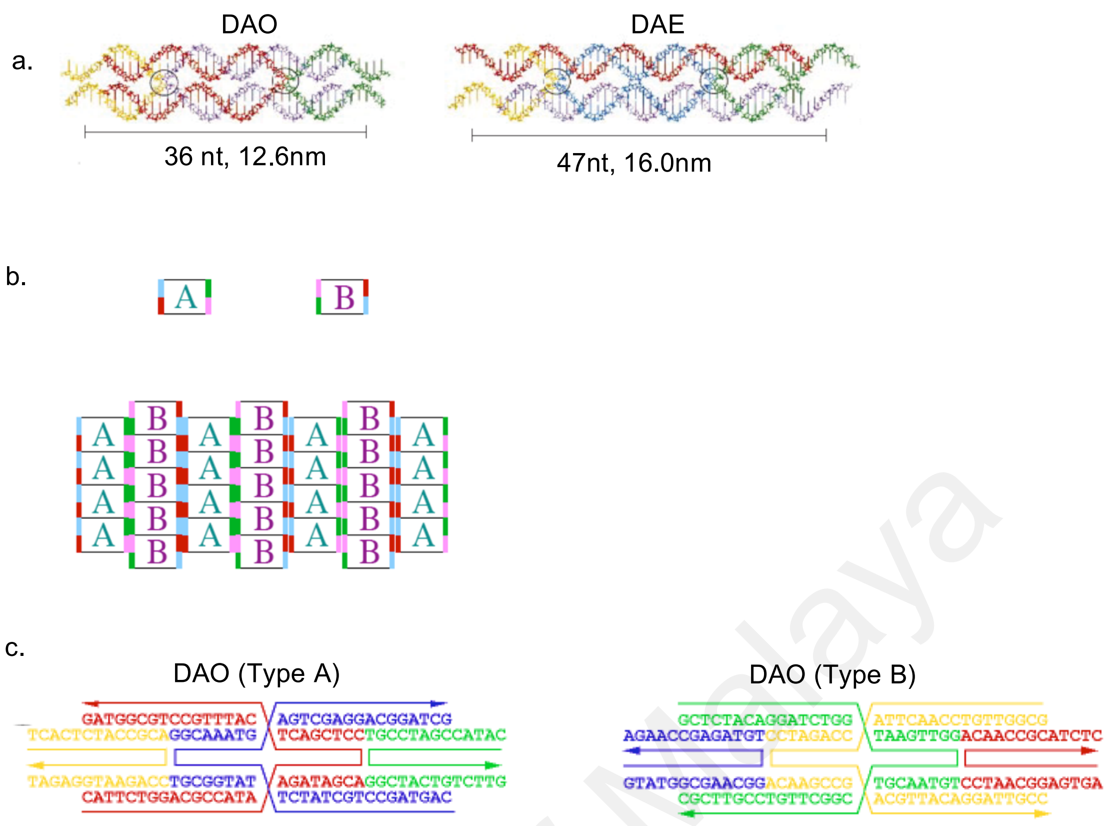

a.
DAO
36 nt, 12.6nm
DAE
47nt, 16.0nm
b.
A
B
c.
DAO (Type A)
GATGGCGTCCGTTTAC AGTCGAGGACGGATCG
TCACTCTACCGCAGGCAAATG TCAGCTCCTGCCTAGCCATAC
TAGAGGTAAGACCTGCGGTAT AGATAGCAGGCTACTGTCTTG
CATTCTGGACGCCATA TCTATCGTCCGATGAC
DAO (Type B)
GCTCTACAGGATCTGG ATTCAACCTGTTGGCG
AGAACCGAGATGTCCTAGACC TAAGTTGGACAACCGCATCTC
GTATGGCGAACGGACAAGCCG TGCAATGTCCTAACGGAGTGA
CGCTTGCCTGTTCGGC ACGTTACAGGATTGCC

**Figure 3.5:** Design of double-crossover structure and arrangement into 2D lattices
Figure 3.5a illustrates the model structure for DAO and DAE type A units. Each component strand is displayed in unique color. Crossover points are circled. Figure 3.5b is the logical structure for 2D lattices by which each design (DAO, DAE) comprises of type A and B. Type A units have four coloured edge regions, each of which matches exactly one coloured region in adjacent type B units. Figure 3.5c is the actual sequences used to form the DAO. Retrieved from (Winfree et al., 1998).

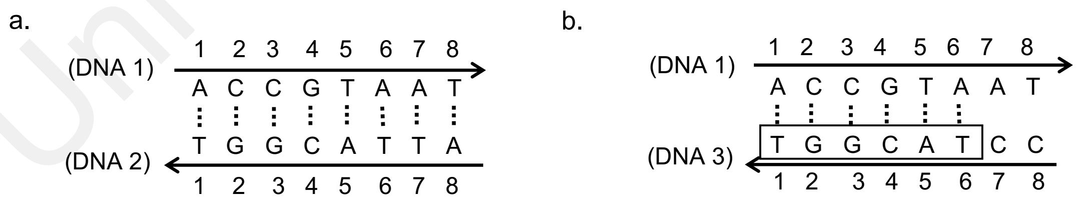


**Figure 3.6:** The correct and incorrect base pairing
Correct base pairing is defined as base pairing that is supposed to form according to the design. Incorrect base pairing refers to the DNA pair that is not supposed to form base pairing. DNA sequences are depicted as DNA 1, DNA 2, DNA 3 and DNA 4. Figure 3.6a represents correct base pairing whereby DNA 1 hybridizes with DNA 2 by forming complementary in all their 8 base pairs. Figure 3.6b demonstrates an example of incorrect base pairing whereby both DNA 1 and 3 are not supposed to bind. However, due to the existence of complementary base pairing in position 1-6 of DNA 3 with the DNA 1, both DNA strands formed partial hybridization. The arrows showed the direction of the DNA from 5’ to 3’.

## 3.3 Design Methodologies to Construct DNA Nanostructures

### 3.3.1 Structural Fabrication in DNA Origami

DNA origami is a technique of folding DNA to create large composite of double-crossover (DX) motifs. The word "Origami" refers to the traditional Japanese art of paper folding. This approach requires a long scaffold and more than hundreds of short single-stranded DNAs known as "staple strands", which are used to form complementarity with the scaffolds (Figure 3.7). The scaffolds used to build DNA origami could bend and fold when hybridized with staple strands (Rothemund, 2006). Algorithm 2 demonstrates the summary of the protocol used in the DNA origami design previously reported by Rothemund (2006).

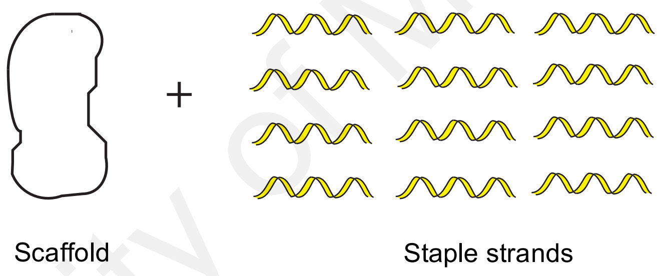


**Figure 3.7:** Biological components utilized by DNA origami to construct DNA nanostructures

**Algorithm 2** Summary of DNA origami protocol to construct nanostructures following (Rothemund, 2006)

```
1:   for each DNA shape do
2:       build a geometric model of the DNA structure (Figure 3.8a)
3:       fill the geometric model using 'cylinders'
         (cylinders = even number of parallel double helices)
4:       insert crossover (blue crosses) to hold the helices
5:       fold the scaffold back and forth in a raster fill pattern (Figure 3.8b)
6:       present the geometric model and folding path as lists of DNA lengths
7:       input these lists with DNA sequence scaffold to the computer program
8:       add staple strands to form complement with scaffold (Figure 3.8c)
9:       calculate the twist of the crossover in scaffold (Figure 3.8d)
10:      merge the adjacent staple to yield fewer, longer staples  (Figure 3.8e)
11:  end do
```

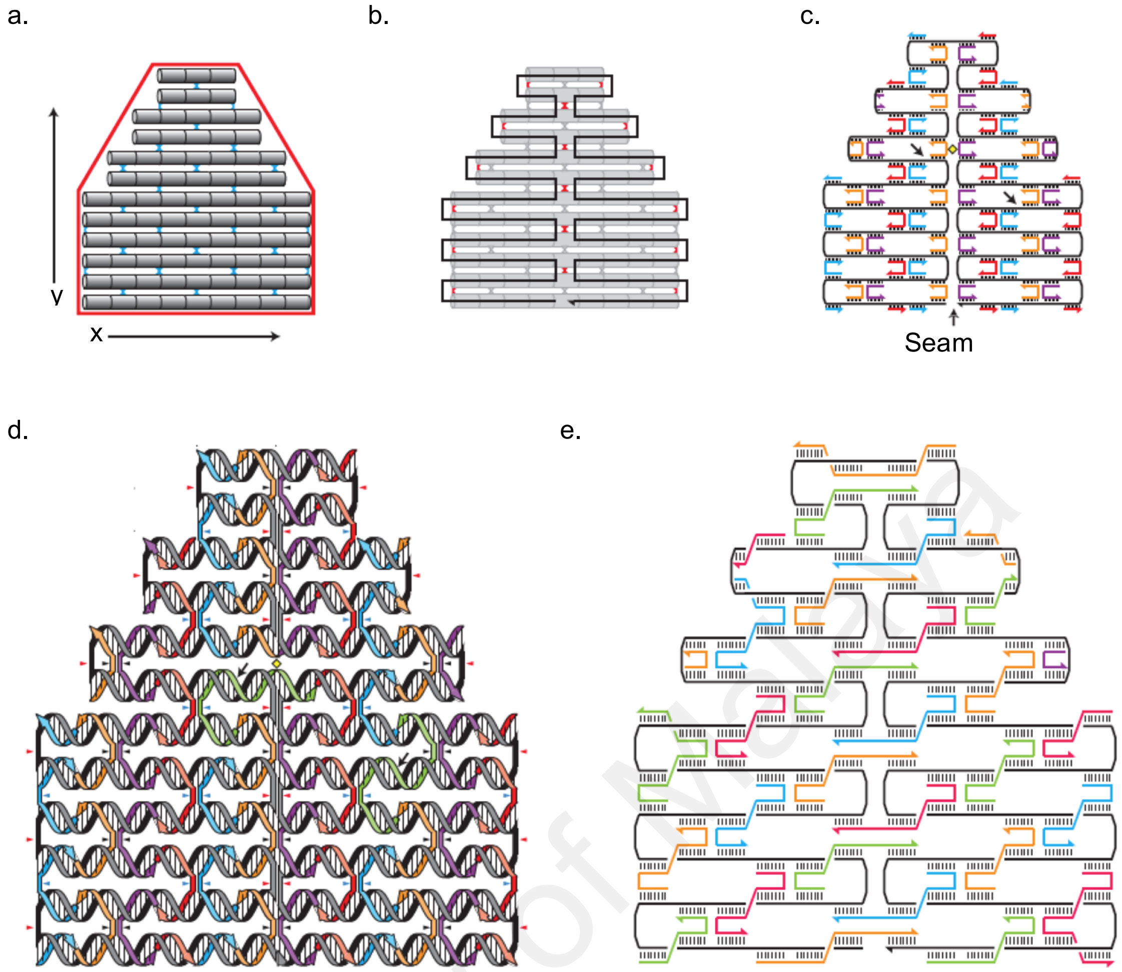


**Figure 3.8:** DNA nanostructures design using DNA origami technique
Figure 3.8a is a shape (red line) approximated by parallel double helices joined by periodic crossover (blue x). Figure 3.8b represents folding of a scaffold (black line) that runs through every helix and forms more crossovers (red x). Figure 3.8c presents the use of staples strands (colored DNA strands) to bind and fold the scaffold. Figure 3.8d represents figure similar to 3.8c but DNA strands are drawn as helices. Figure 3.8e is a finished design after merging and rearrangements along the seam. Retrieved from (Rothemund, 2006).

Every distinct DNA nanostructure would require a new scaffold routing design and the synthesis of different sets of strands (Douglas, Dietz, et al., 2009; Han et al., 2011; Ke et al., 2009; Rothemund, 2006). Different types of scaffolds are used to construct DNA origami (Table 3.1), including 7,249 nucleotides and circular DNA genome obtained from M13mp18 phage (Rothemund, 2006). The challenge of DNA origami in using long scaffold as construction material is that in order to build a gigadalton nanostructure it would require a scaffold of over 1 megabase which is mechanically fragile and difficult to synthesize (Pinheiro, Han, Shih, & Yan, 2011). The staple strands hold the adjacent portions of the scaffold together by forming crossover at every (*n+0.5*) helical turns of the DNA. DNA origami requires extensive amount of staple strands to bend the long scaffold into the desired shapes (Andersen et al., 2008; Andersen et al., 2009; Marchi, Saaem, Vogen, Brown, & LaBean, 2014; Rothemund, 2006).

As each staple strand is targeted to a specific location in the scaffold, any mispairing would result in incompatible binding and thus resulted in the non-conformity of the desired structures. On the contrary, DNA Tetrominoes concept allows each Tetris shape to be considered as an entity on its own and therefore is independent of the overall structure.

**Table 3.1:** List of scaffold and staple strands used to construct DNA nanostructures

| Nanostructures | Staple Strands | Scaffold | References |
|---|---|---|---|
| Arbitrary planar structures: Square, Rectangle, Star, Disk with holes (smiley), Rectangular domains | 200 - 250 | M13mp18 DNA | (Rothemund, 2006) |
| 3D box with a controllable lid | 220 | M13mp18 DNA | (Andersen et al., 2009) |
| 2D asymmetric origami sheets | > 1,600 | 51,466 nucleotides, single-stranded form of scaffold produced from λ/M13 hybrid virus | (Marchi, Saaem, Vogen, Brown, & LaBean, 2014) |
| Dolphin-shaped with flexible tails | > 200 | M13mp18 DNA | (Andersen et al., 2008) |

A number of sophisticated computational tools (Douglas, Marblestone, et al., 2009; Williams et al., 2008; Zhou et al., 2012) have been developed to assist in the construction of nanostructures based on DNA origami technique. For instance, *caDNAno* (Douglas, Marblestone, et al., 2009) is an open-source software package developed to construct DNA origami structures. It is equipped with graphical user interface to enable manual modification on the scaffold and staple paths that are used to design 3D DNA origami shapes (Douglas, Marblestone, et al., 2009) (Figure 3.9). The 3D DNA origami scaffolds are represented as pleated layers of DNA double helices that are based on honeycomb lattice skeleton (Douglas, Marblestone, et al., 2009).

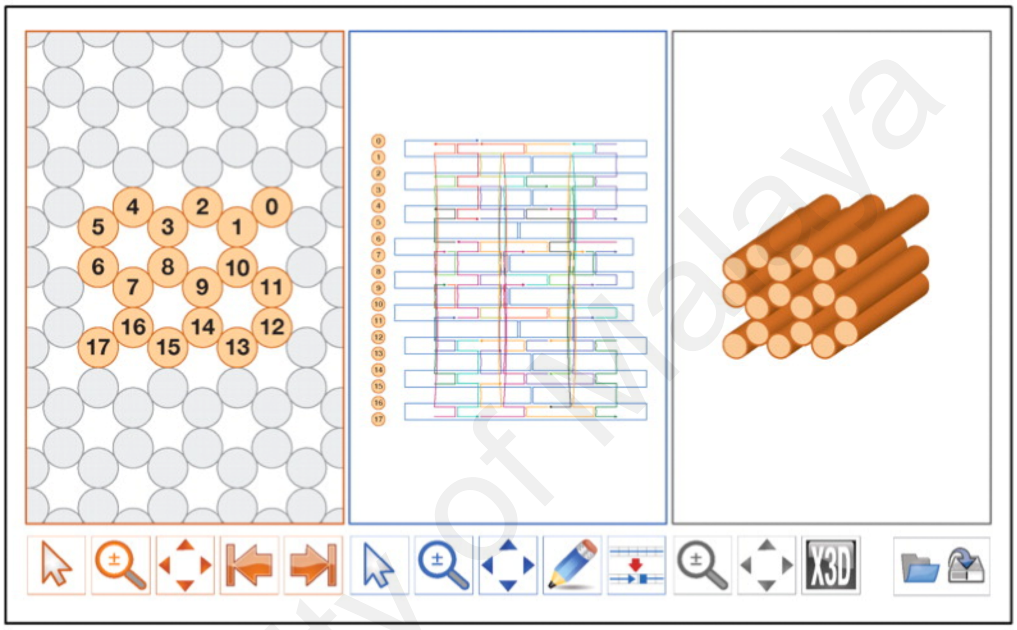


**Figure 3.9:** The screenshot of *CaDNAno* interface
The interface is separated into three sections. Left: The cross-sectional view of the honeycomb framework whereby helices are added into the design. Middle: An interface to modify the 2D schema of that particular scaffold and staple paths. Right: A real time 3D model corresponding to the design. Retrieved from (Douglas, Marblestone, et al., 2009).

The automated processes of *caDNAno* involved steps such as scaffold selection, assignment of staple paths and setting of complementary sequences (Douglas, Marblestone, et al., 2009). Despite this, subsequent studies are needed to optimize the design parameters that may influence the folding yield, for example scaffold routing and densities of scaffold versus staple crossover (Jungmann, Liedl, Sobey, Shih, & Simmel, 2008). There is another tool known as *FOLDNA* web server that is developed to autonomously design 2D nanostructures upon user input image (Zhou et al., 2012). This web server, executed by first generating the scaffold pathway based on the custom picture, followed by generation of the DNA strands that bind to the scaffold and then filling of the staple DNA based on DNA helix torque angle (Zhou et al., 2012).

### 3.3.2 Structural Fabrication in Single-stranded DNA Tiles (SST)

Following the DNA origami technique, single-stranded DNA tiles (SST) technique is introduced to address the modularity characteristic surrounding the construction of DNA nanostructure. SST used modular components to build the structures by which each single component can be included, excluded or replaced without changing the remaining structure (Ke et al., 2012). The challenge in this method is to develop a universal strategy that will allow every component (small monomer of DNA) to be mediated strictly by local interactions and then formed into the specific global shape (Figure 3.10) (Wei et al., 2012).

Each component exhibits homogeneous property by which all components used within a structure are always uniform in term of nucleotides length. For example, a molecular cube was constructed using single-stranded DNA with 42 nucleotides (Wei et al., 2012) while Ke et al. (2012) built a cuboid shape using single-stranded DNA with 32 nucleotides (Ke et al., 2012). The SST technique regards each of these single-stranded DNAs as one component and multiple components are used to assemble into the end-structures. Figure 3.11a illustrates each SST component that is comprised of four concatenated domains; the domain 1 (orange color), domain 2 (blue color), domain 3 (green color) and domain 4 (pink color) (Wei et al., 2012). By pairing up the complementary domains (Figure 3.11b), the motifs can be arranged to form DNA lattices composed of parallel DNA helices connected by single-stranded linkages.

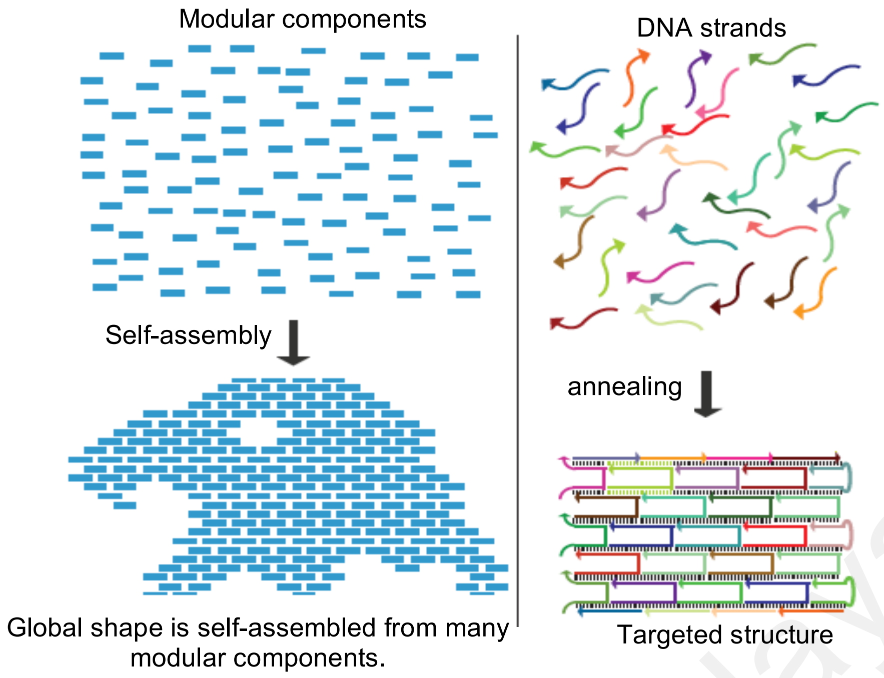


**Figure 3.10:** Self-assembly of modular components in SST
Figures on the left and right are the relative comparison of using DNA strands as components to assemble into the prescribed targeted structure. Retrieved from (Wei et al., 2012).

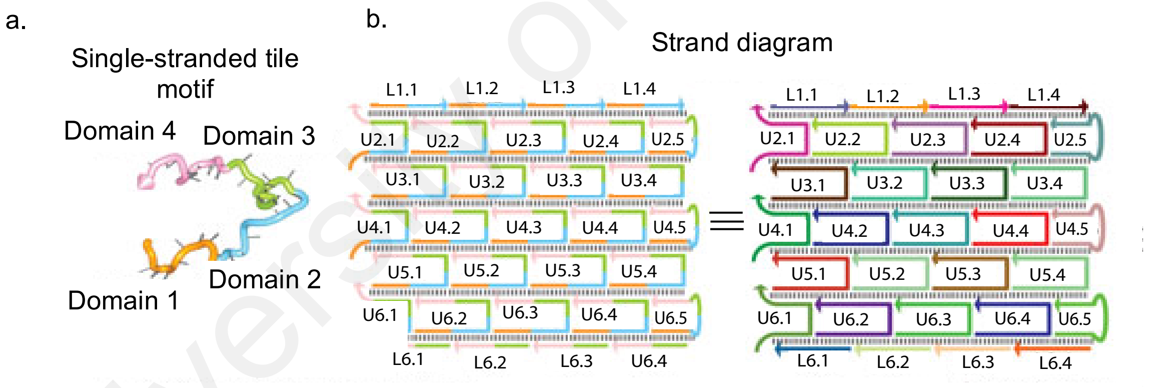


**Figure 3.11:** The Single-stranded DNA tiles units
Figure 3.11a represents a SST motif with four domains (orange domain 1, blue domain 2, green domain 3 and pink domain 4). Figure 3.11b represents the assembly of multiples SST motif into a structure. Each standard (full) tile has 42 bases (labelled as U), and each top and bottom boundary (half) tile has 21 bases (labelled as L). Retrieved from (Wei et al., 2012).

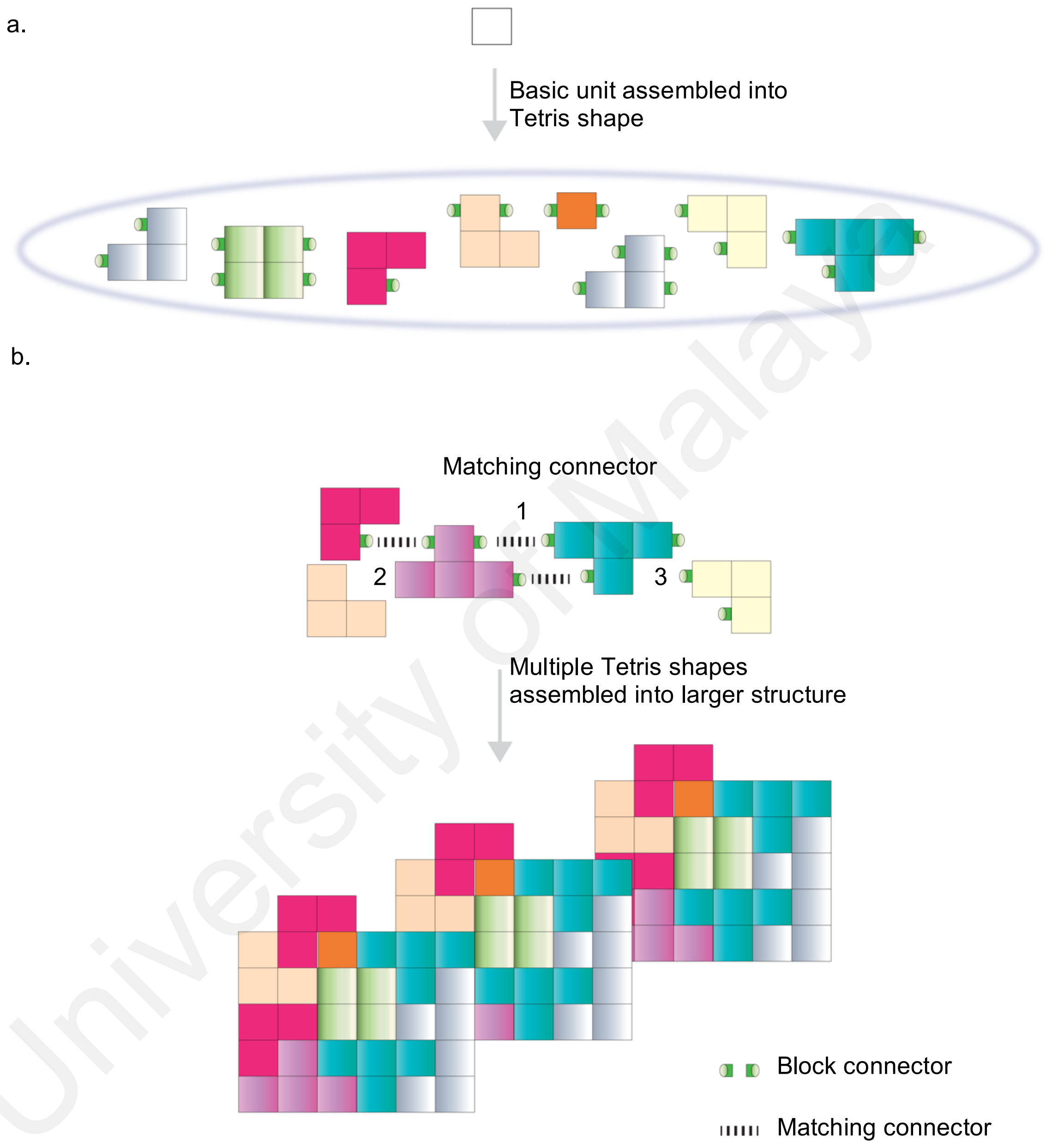
Although the sequence design for SST is straightforward, individualized sequence design is still required in order to enable intermolecular binding to form between every domain. Meanwhile in the newly proposed DNA Tetrominoes concept, each Tetris shape is considered as an entity, and each of this entity is independent of the overall structural design. The goal is to have these Tetris shapes to merge together with different type of combinations to generate a larger structure. It only requires the users to program the sticky ends of each DNA Tetris shape in order for the shapes to be able to form complementary and join with other shapes. Moreover, the proposed schema allows the construction of heterogeneous components, which means the components that are used to make a larger structure need not always be the same or uniform. This is because different Tetris shapes (heterogeneous blocks) or from the aspect of Tetris game, different types of Tetris shapes can be used to fill the Tetris board (search space). The DNA sequences used to construct the DNA nanostructures were designed using *UNIQUIMER* software (Zhu, Wei, Yuan, & Mi, 2009) by minimizing the sequence symmetry (Seeman, 2010) or by utilizing an entirely random sequences of SST motif (Wei et al., 2012). The sequence design criteria for the SST based on sequence symmetry minimization (Wei et al., 2012) is presented as Algorithm 3.

**Algorithm 3** SST sequence design criteria (Wei et al., 2012)

```
1:  for each DNA sequence do
2:      randomly generate every nucleotide from (A, T, G, C)
3:      generate complementary nucleotide for the sequence generated in (2)
4:      check for repeating segments beyond length 8 or 9 nucleotides
5:      if repeating segments more than 8 or 9 then
6:          mutate the most recent nucleotide till the condition is satisfied
7:      end if
8:      avoid four consecutive A, C, G, T bases
9:  end for
```

## 3.4 Fundamental Concepts in DNA Tetrominoes

This section begins by presenting the fundamental concept in the proposed schema, DNA Tetrominoes which is a new nano-fabrication strategy based on the paradigm of self-organization. Hence, Tetris shapes were used as the representative in demonstrating the feasibility of using multiple elementary blocks in structural assembly. The hierarchical schema in DNA Tetrominoes started with an elementary block, followed by shapes and then larger structural formation. As a basis, each block used two single-stranded DNAs to form a block. Then, multiple units of these blocks assembled into Tetris shape. Different Tetris shape would then assemble into a larger structure (Figure 3.12).



a.
Basic unit assembled into Tetris shape
b.
Matching connector
1
2
3
Multiple Tetris shapes assembled into larger structure
Block connector
Matching connector

**Figure 3.12:** Hierarchical schematic of using Tetris shape to form structures
Figure 3.12a represents the Tetris shapes made up from one or multiple basic blocks. Each block may or may not have its block connector used to join different Tetris shapes. Figure 3.12b represents the example of joining Tetris shapes through the connector. The shapes can only merge when both shapes have matching connector. In the case (1), both Tetris shapes had matching connector and therefore these shapes could join together. The Tetris shape in case (2) does not have connector, thus it cannot join with other shape. While in case (3) only one of the shapes has the connector and thus also cannot merge together. Multiple Tetris shapes further assembled to form larger structures.

Each of these Tetris shapes may comprise of one or more connector on the horizontal sides of the shape. Its function is to enable the shape to bind to another shape that had matching connector and thus forming larger structures. In the context of DNA sequences, the matching connector was defined as DNA with complementary sticky ends (Figure 3.13).

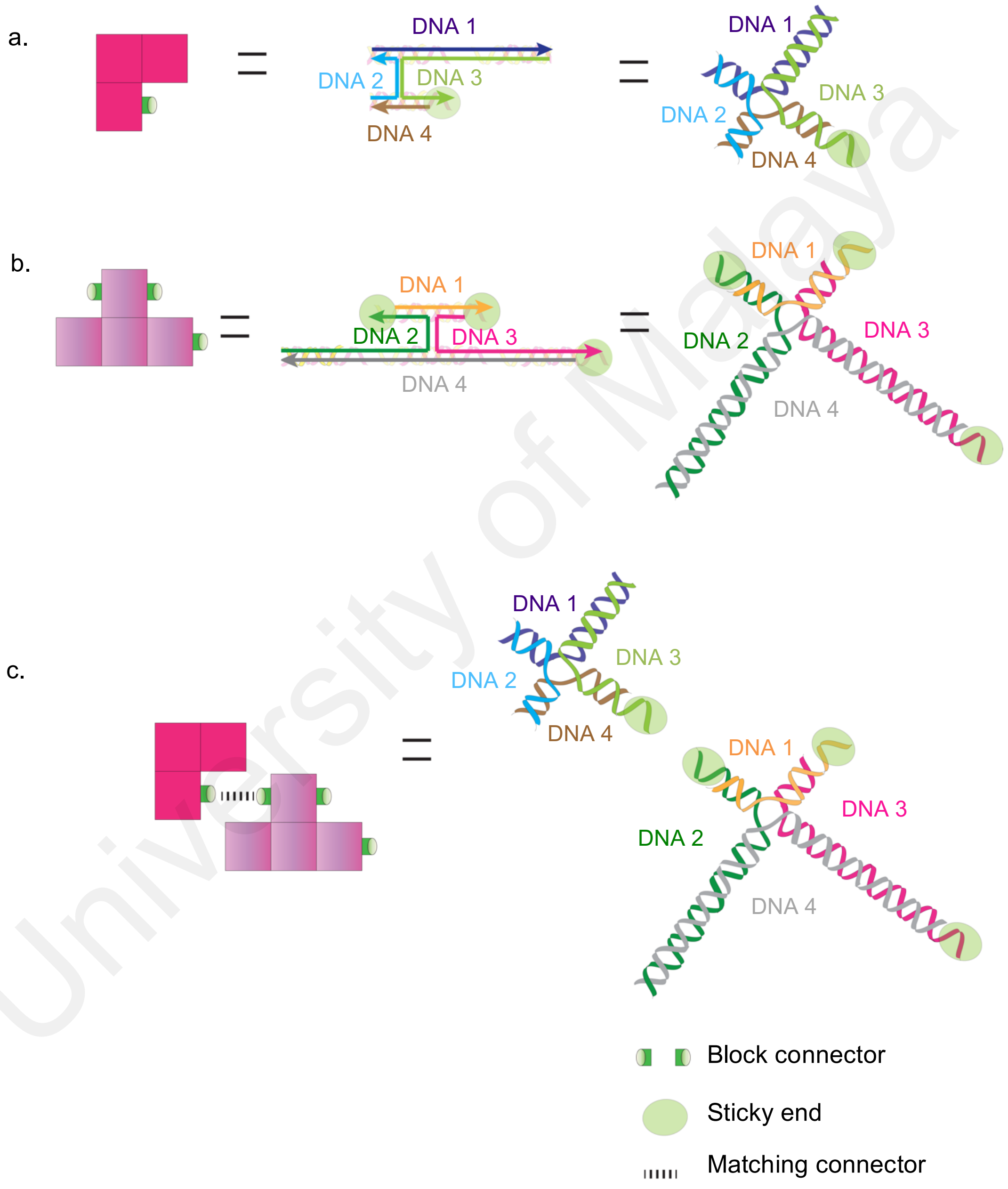


**Figure 3.13:** Conceptual illustration on the assembly of DNA Tetris shapes Different DNA strands were labelled as DNA 1, DNA 2, DNA 3 and DNA 4. Figure 3.13a represents the Tetris shape that has 1 block connector. Figure 3.13b represents Tetris shape with 3 block connectors. Figure 3.13c represents two Tetris shapes with matching connector and then joined together to form a larger structure. Whenever there is a presence of block connecter, its corresponding region in DNA sequence will have sticky end. Sticky end is needed to enable two Tetris shapes to bind together.

## 3.5 Construction of DNA Tetris Shape

Before beginning to construct DNA nanostructures, a list of Tetris shapes that could be derived from DNA sequences were identified (Figure 3.14). Tetris shapes were made from DNA sequences and were identified as DNA Tetris shape. In order to do so, every two ssDNA was treated as one Tetris block (Figure 3.15). The initial random DNA sequences used to form the Tetris block were generated using *DSG* and a Perl script.

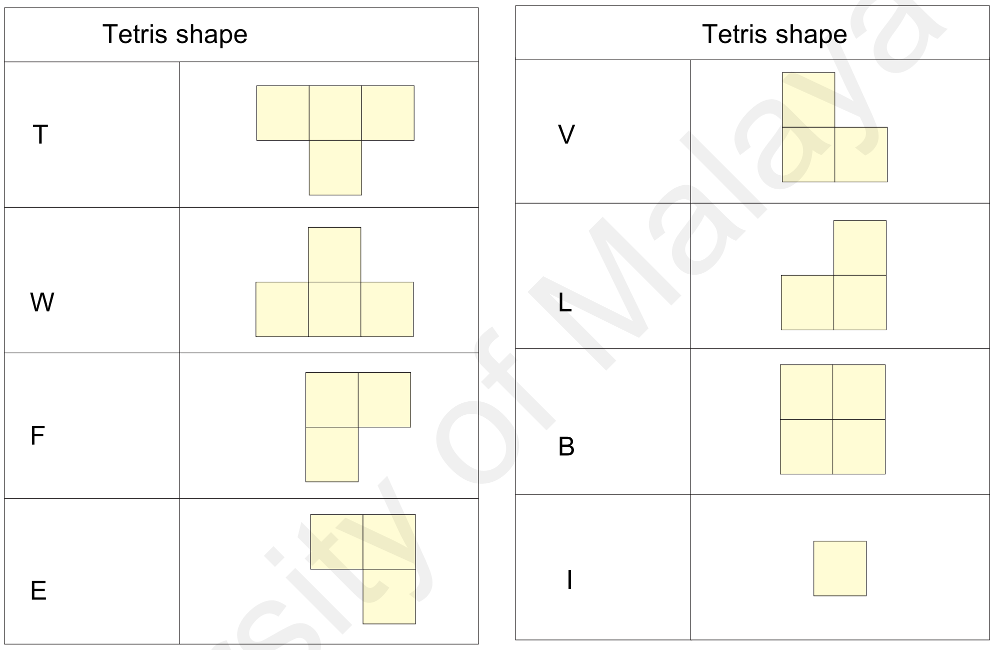


**Figure 3.14:** List of Tetris shapes
T, W, B-shapes were formed from 4 basic blocks while F, E, V and L-shapes were formed from 3 basic blocks. I-shape was formed from 1 basic block.

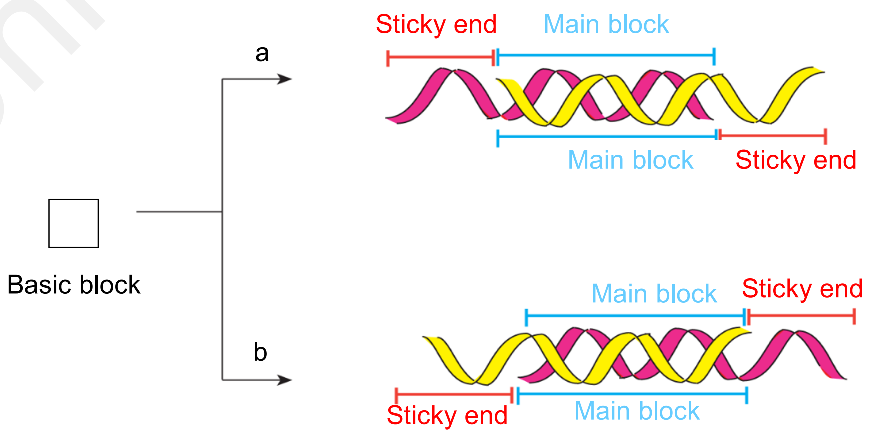


**Figure 3.15:** Basic blocks used to design DNA Tetris shapes
There were two types of blocks used to design the structures. Figure 3.15a represents Type-1 block. Figure 3.15b represents Type-2. Each strand was compartmentalised into a main block and sticky end.

Multiples basic blocks were used to form a Tetris shape. These blocks were stacked and merged together to form the shapes. Whenever one DNA block stacked upon another block, a crossover between the blocks would be implemented into the design. Each basic Tetris shape (with an exception of the I-Shape) was built from either six or eight single-stranded DNA (ssDNA), which was then merged to form four long continuous ssDNAs. The I-Shape, on the other hand was formed using just two ssDNAs.

Figure 3.16 illustrates the formation of eight distinct Tetris shapes using previously mentioned Type-2 basic block. Each of the DNA Tetris shape was made up of 4 DNA strands except for I-shape that was formed from 2 DNA strands. Six of these shapes were mirror image of each other (i.e., T-shape with W-shape; F-shape with E-shape and V-shape with L-shape).

In addition to the above, the DNA strands were also subjected for sequence modification processes such as the position of block stacking, nucleotide shifting, sequence insertion and deletion to ensure a greater versatility in nucleotide combinations for the resulting structures. It also described the process of the DNA sequences to merge and form into four long continuous DNA (Figure 3.17). In this case, Type-1 block was used to build the L-Shape, while the Type-2 was used to build the T-Shape, B-Shape and I-Shape. Two neighbouring blocks were linked using the existing sticky ends while a single crossover was utilised to ensure the linkage between two blocks formed when the two blocks were stacked on top of each other. The merging of short sequences from 3 blocks (L-Shape) and 4 blocks (T-Shape, B-Shape) resulted in four long stretches of DNA sequences.

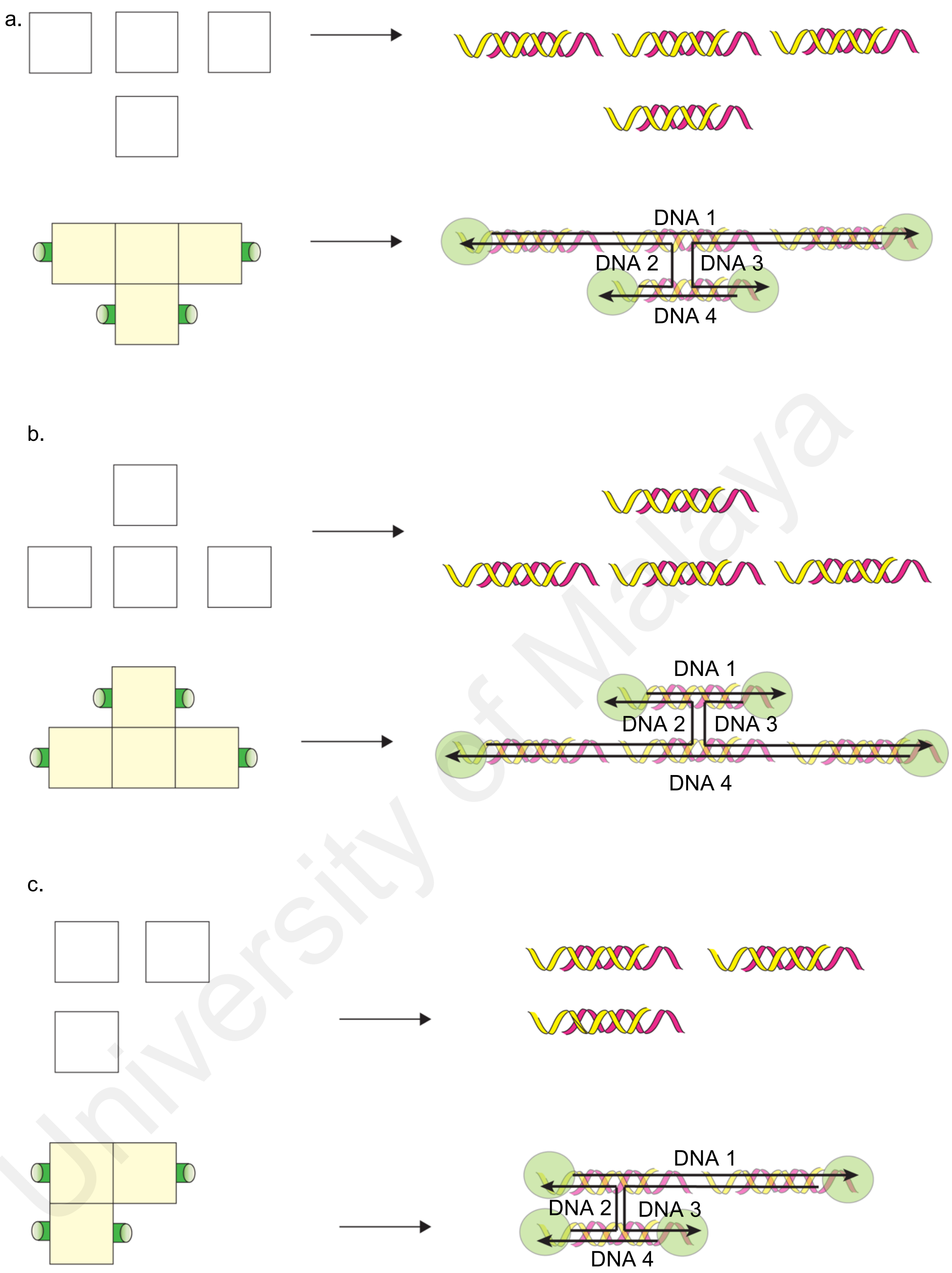


**Figure 3.16:** Conceptual representation of the formation of DNA Tetris shapes Figure 3.16a-c represents the formation of T-shape, W-shape and F-shape. Basic blocks (Type-2) were used to form four long continuous single-stranded DNAs (ssDNAs). DNA strands were represented as DNA 1, DNA 2, DNA 3 and DNA 4. The arrows in the DNA strands indicated the 5' to 3' direction.

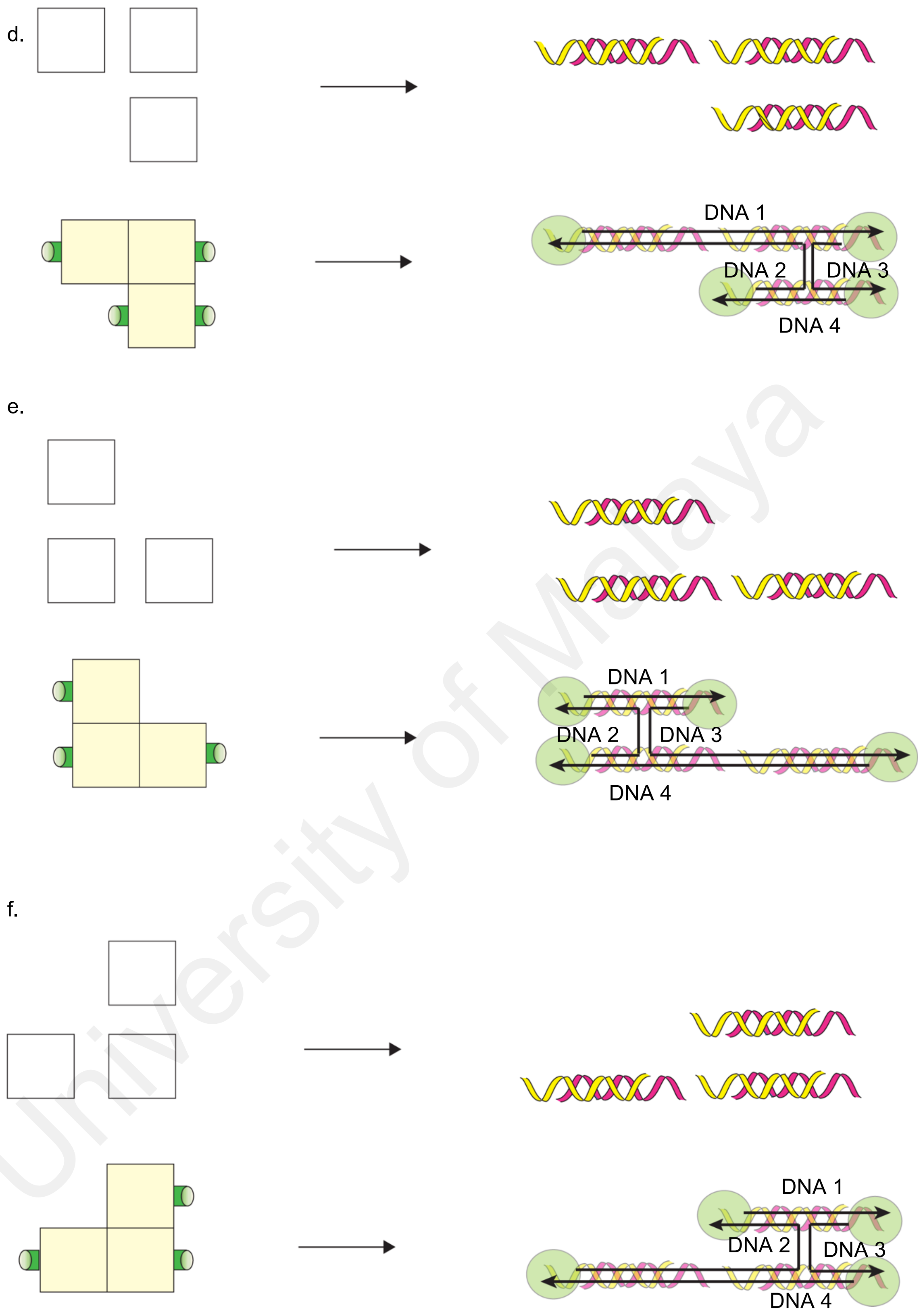


**Figure 3.16 (continued):** Conceptual representation for the formation of DNA Tetris shapes

Figure 3.16d-f represents the formation of E-shape, V-shape and L-shape. Basic blocks (Type-2) were used to form four long continuous single-stranded DNAs (ssDNAs). DNA strands were represented as DNA 1, DNA 2, DNA 3 and DNA 4. The arrows in the DNA strands indicated the 5’ to 3’ direction.

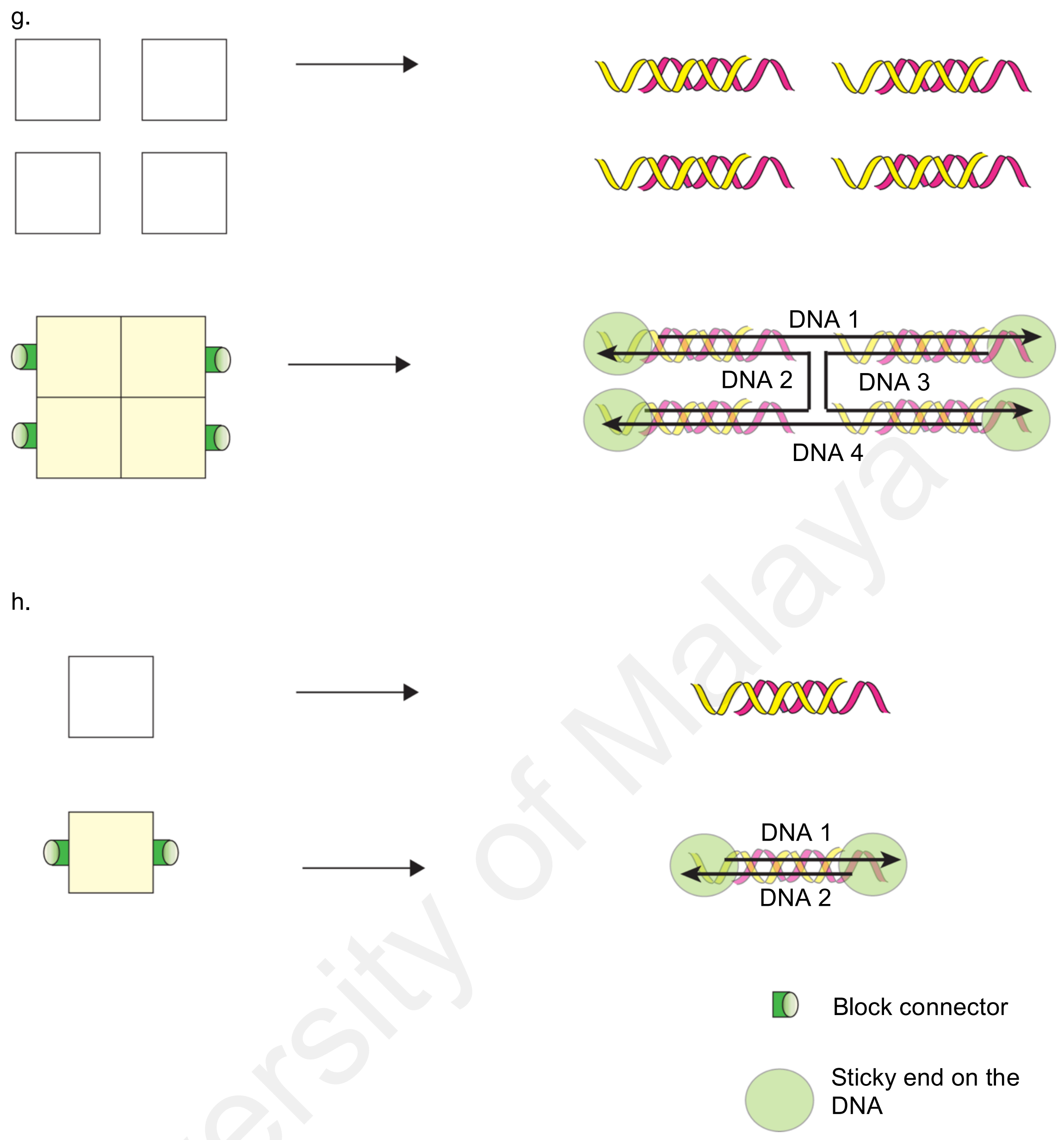


**Figure 3.16 (continued):** Conceptual representation for the formation of DNA Tetris shapes

Figure 3.16e represents the formation of B-shape and I-shape. Basic blocks (Type-2) were used to form four long continuous single-stranded DNAs (ssDNAs). DNA strands were represented as DNA 1, DNA 2, DNA 3 and DNA 4. The arrows in the DNA strands indicated the 5’ to 3’ direction.

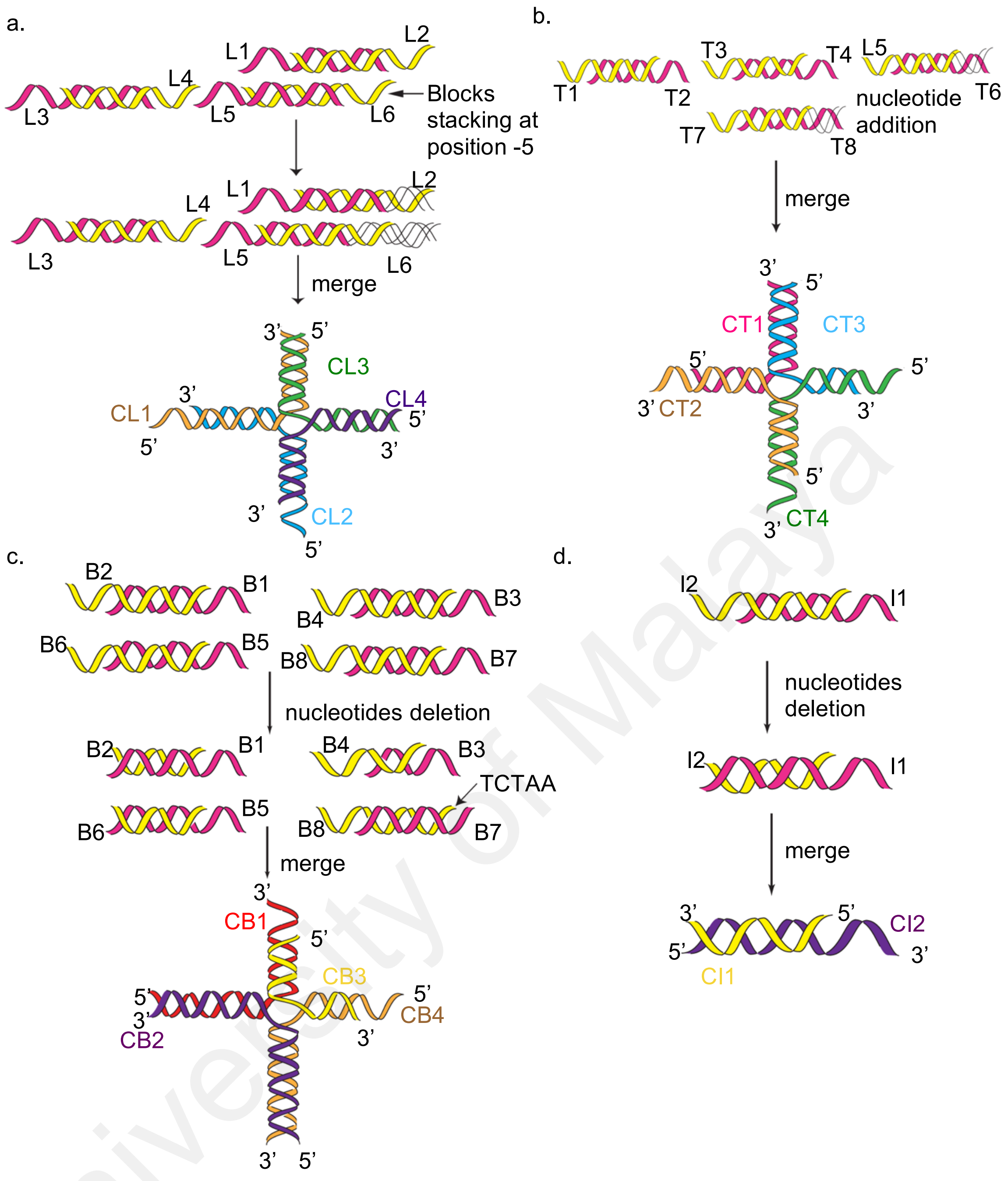


**Figure 3.17:** Schematic illustration of DNA sequence modifications
Figure 3.17a represents the formation of L-shape using 3 blocks or 6 DNA strands (L1-L6). These strands were then subjected for modifications (insertion of 10 and 15 nucleotides to L1 and L5, insertion of 5 nucleotides to L6) and block stacking (Block L5-L6 was stacked on position -5 (to the left) relative to Block L1-L2). After modification, these 3 blocks then merged to form 4 long strands (CL1-CL4). Figure 3.17b represents formation of T-shape using 4 blocks or 8 DNA strands (T1-T8). These strands were modified (insertion of 10 and 20 nucleotides to form blunt end on T6 and sticky ends on T8. After modification, it was merged to form 4 newly combined strands (CT1-CT4). Figure 3.17c represents B-shape formation using 4 blocks or 8 DNA strands (B1-B8). These strands were subjected for modifications (Deletion of 10 nucleotides on strand B2-B4, B6 and B7) and fragment shifting (fragment "TCTAA" shifted from strand B7 to B8). Thereafter, blocks were merged to form 4 long strands (CB1-CB4). Figure 3.17d represents I-shape using a single block or 2 DNA strands (I1, I2). These strands were subjected for modification whereby deletion of 10 nucleotides occurred on I2 sticky ends. Following modification, CI1 and CI2 were the new strands.

## 3.6 Framework To Implement DNA Tetrominoes

The use of computational tools to support the proposed schema was crucial since it was improbable to build a structure manually from scratches as it involved a very large search space and combinatorics issues. As a consequence, there was a need to develop tools that were able to solve these problems. This schema would also allow different types of combinations to be formed whereby 3 x 4 rectangle was used as a hypothetical application to proof the feasibility of building multiple combinations of DNA nanostructures using the newly proposed schema.

In order to demonstrate the execution of DNA Tetrominoes concept, the entire framework was presented by displaying the graphical flow on the processes that occurred during structural construction. It began with the input parameters such as minimum and maximum guanine-cytosine ratio *($minGC*, *$maxGC*) and the type of combinations (*$Rec*). *$Rec* was used to call out its own pre-generated sequence dependency file *($Rec*)DefineSeq.txt. The derivation of sequence dependency file would be discussed in details in chapter 6.

The entire framework diagram was segregated into two main modules (Figure 3.18). Module 1 was the development of sequence design tool to construct heterogeneous DNA Tetris shapes that could conform to the designed structures while module 2 was the development of DNA connectivity tool used to compute the binding affinities between the DNA Tetris shapes. Both module 1 and 2 would be discussed thoroughly in Chapter 4 and Chapter 5. All scripts were written in Tool Command Language (TCL) and Perl version 5.12.4 and the completed scripts were then tested in Unix environment in Mac OS X, version 10.7.5.

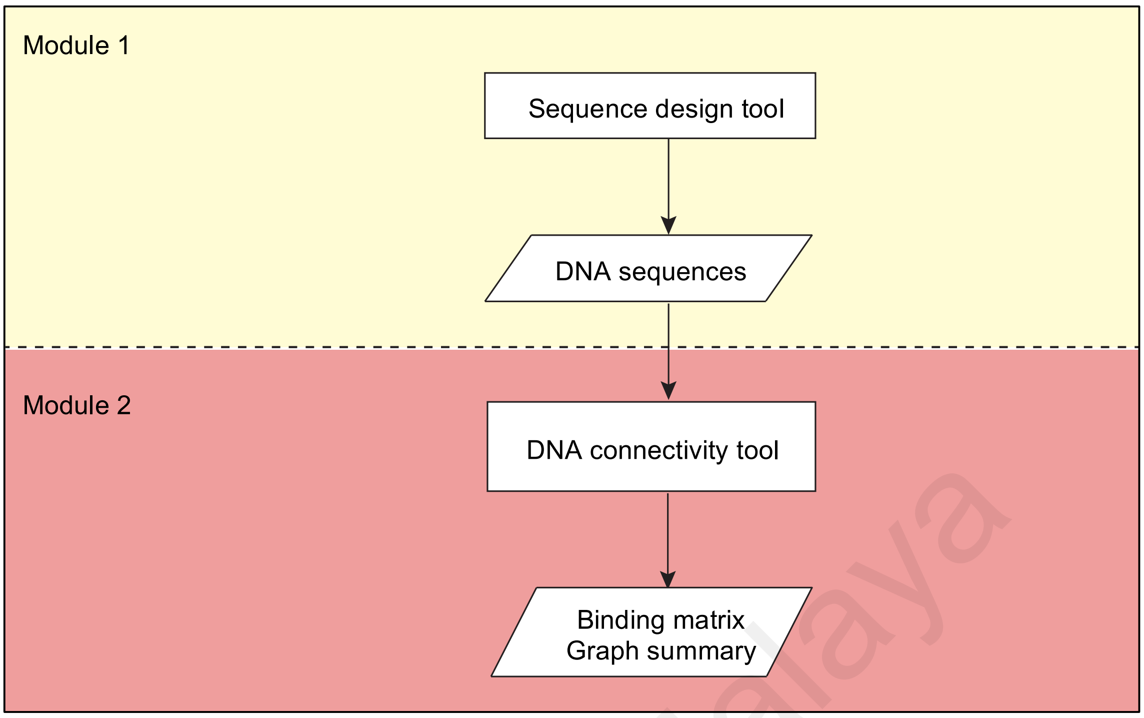


**Figure 3.18:** Framework of the implementation of DNA Tetrominoes concept
It was segregated into module 1 and module 2. Module 1 was to generate DNA sequences that could conform into a particular structure and module 2 comprised of DNA connectivity tool to compute binding affinities between the DNA. The rectangular shapes were the script/program used to execute the task and the parallelogram shapes were the output/input generated by the programs.

## 3.7 Comparison between DNA Origami, SST and DNA Tetrominoes

This section summarized the differences that were previously discussed between DNA origami and SST techniques as well as DNA Tetrominoes schema (Table 3.2)

**Table 3.2:** Comparison between DNA origami, SST and DNA Tetrominoes

| Aspects | DNA Origami | Single-stranded DNA Tiles (SST) | DNA Tetrominoes |
|---|---|---|---|
| Schema | Each distinct structure (S. M. Douglas et al., 2009; Han et al., 2011; Ke et al., 2009; Rothemund, 2006) requires a new scaffold routing design and the synthesis of different sets of staple strands. | Modular assembly: Every single component in the structure can be included, excluded or replaced without changing the remaining structure (Ke et al., 2012).<br><br>Homogeneous: Components that make up the nanostructures are always uniform (standardized components). | Modular assembly: Every single component in the structure can be included, excluded or replaced without changing the remaining structure.<br><br>Heterogeneous: Components used to make up a larger structure need not always be the same (Different Tetris shapes) |

**Table 3.2 (continued):** Comparison between DNA origami, SST and DNA Tetrominoes

| Aspects | DNA Origami | Single-stranded DNA Tiles (SST) | DNA Tetrominoes |
|---|---|---|---|
| Sequence Design | Each staple strand is targeted to a specific location in the scaffold. | Individualized sequences design is required to enable intermolecular binding between domains in every SST motif (Ke et al., 2012; Wei et al., 2012; Yin et al., 2008). | Each shape is independent of the overall structure design (and only conforms to its individual shape) |
| Programma-bility | Required a long scaffold as construction material. To build a gigadalton nanostructure, it would require a scaffold (> 1 megabase) (Pinheiro et al., 2011). | One standardized length with four domains is used throughout a structure. (e.g. 42 nucleotides ssDNA (Wei et al., 2012), 32 nucleotides (Ke et al., 2012)). | Any Tetris shape can combine together with different combinations to generate larger structures. The users only need to program the sticky ends of each Tetris shape to be compatible. |

## 3.8 Summary

This chapter began by discussing different approaches adopted by the DNA origami and SST techniques in constructing DNA nanostructures. Comparisons between various techniques against the DNA Tetrominoes revealed the differences in strategy taken for nanostructures fabrication. After successfully identified the core features presence in the schema, it was crucial to identify Tetris shapes that could be derived to form DNA Tetris shapes. These shapes were identified and then the procedures to build individual DNA Tetris shapes through the process of block stacking and merging were elucidated. A framework of computational tool to oversee the execution and development of the entire computational process to support the proposed schema was presented. This framework was segregated into two modules; by which both the development of module 1 (sequence design tool) and module 2 (DNA connectivity tool) would be described in details in Chapter 4 and Chapter 5.

# CHAPTER 4: MODULE 1 SEQUENCE DESIGN TOOL

## 4.1 Introduction

The presentation of DNA Tetrominoes concept for structural construction had been introduced in the prior chapter. In order to support the newly proposed schema, the development of new computational tool was essential. Therefore, a sequence design tool (Module 1) that used to generate a set of DNA sequences that could conform to the prescribed structures were developed. This tool incorporated the use of optimization algorithm to evaluate the candidate DNA sequences based on the fitness criteria and further exerted mutation on the DNA sequences. Penalty scores had been derived to enable the calculation of the total score for each of the fitness criterion. Less stringent criteria were adopted and allowed the structures to form, subjected to the occurrences of some unwanted aggregates. This was necessary to handle the formation of structures under undesirable and uncontrollable physicochemical conditions. It would be applicable for a specific scenario such as when the DNA nanostructures were built inside the living cells, as compared to the conventional method of building the structures externally (thus requiring a complicated delivery mechanism afterwards) (Amir et al., 2014). The newly developed tool would autonomously generate a set of DNA sequences based on the DNA Tetrominoes concept.

## 4.2 Framework for the Implementation of Sequence Design Tool

The module 1 sequence design tool was utilized to generate DNA sequences based on the designed structures (Figure 4.1). The core script in this module is *Main_RunDNATetris.tcl* and it comprised of the following six scripts namely *Main2GenerateSeq.tcl, GenerateRandomSeq.pl, Adjust_RandomSeq.pl, FindStartPosition.pl, CleanEmptyPosition.pl* and *GetLongestComplement.pl* (Appendix A-G). It began by utilizing *GenerateRandomSeq.pl* or *DNA sequence generator (DSG)* to generate a set of random DNA sequences (RandomSeq.txt). These random DNA sequences were then modified according to the sequence dependency file (*$Rec*)DefineSeq.txt to produce a list of unoptimized DNA sequences (UnOptimized_Seq.txt).

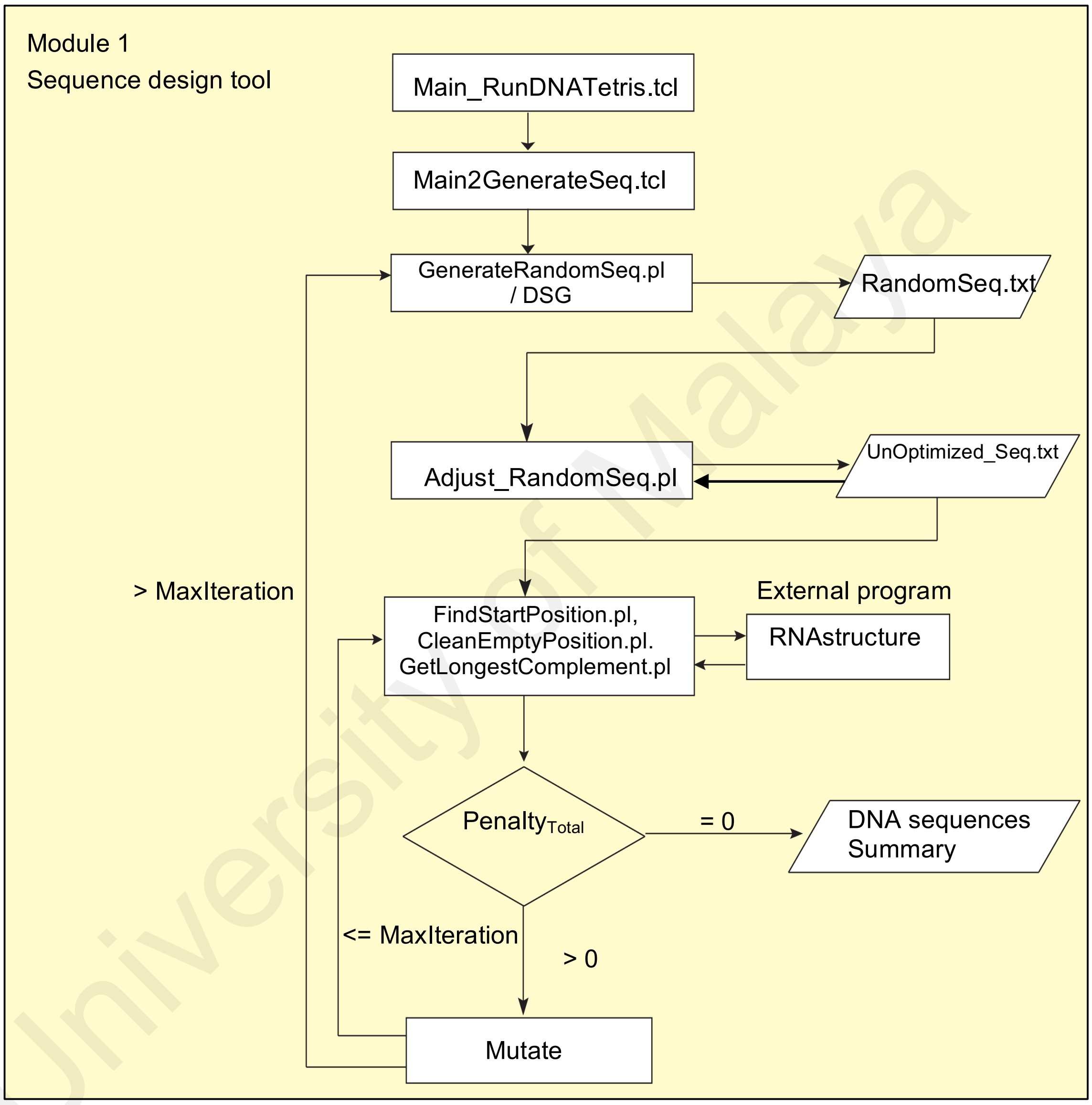


**Figure 4.1:** Flowchart for sequence design tool (Module 1)
The tool generated random DNA sequences to optimize the DNA sequences so that they formed respective structure as of designed. The rectangular shapes were the script/program used to execute the task and the parallelogram shapes were the output/input generated by the programs. The final outputs were DNA sequences and summary file.

The unoptimized sequences were later subjected to optimization algorithms to allow mutation to be exerted on the candidate DNA sequences until all the candidates complied with all fitness criteria. The fitness criteria were used to calculate the penalty scores (*$Penalty_{Total}$*) for the candidate sequences. During each mutational cycle, whenever the score for *$Penalty_{Total}$* was more than zero, the sequence needed to undergo mutation. However, if the sequence still did not pass the fitness evaluations after the maximum iteration (*MaxIteration*), which was set to default 500 mutation cycles, the program would proceed to regenerate a new set of initial unoptimized sequence for the optimization algorithm. This was to reduce the time frame required to perform the computation process. In addition, the optimization algorithm was required the call out an external program named *RNAstructure* (Reuter & Mathews, 2010). It then output a list of DNA sequences that conformed to the desired structure and the keywords to identify the correspond Tetris Shape.

The optimization algorithm had two main sections that were the sequence evaluation based on four fitness parameters (Table 4.1) and the sequence mutation. The four fitness criteria were the false binding sites (FBS), free energy, G4 pattern and percentage of guanine-cytosine. They were used to calculate the penalty scores for all the generated candidates sequences. The protocol for the optimization process (*OptimizedSeq.tcl*) was presented as Algorithm 4.

**Table 4.1:** Fitness evaluation criteria for sequence optimisation algorithm

| Fitness criterion | Description |
|---|---|
| Base pairing at false binding sites | The detection program exhaustively looked for maximum consecutive base pairing at unwanted positions (also known as false binding sites or FBS), $FBS_{max} = 6$. |
| Free energy | The calculation of free energy incorporated the use of program *AllSub* and *DuplexFold* (Reuter & Mathews, 2010). The free energy of intra-molecular pairing must be higher compared to inter-molecular pairing. |
| Percentage of guanine-cytosine (GC) content | Percentage of GC must be in the range of 40% to 70% inclusive. |
| No existence of G4 pattern | No existence of GGGG pattern was allowed in the sequence. |

**Algorithm 4** Sequence optimization algorithm

```
for each sequence ($S_n$) in population do
initialize PenaltyScore (P)=1, initialize LoopValue (L)=1
    while L =1 do
        if n=1 then
            exec program AllSub
            initialize $\Delta$DuplexFold = 0
        else if n>1
            exec program AllSub
            for each Target (MSeq) complement with Query do
                exec program DuplexFold
            end for
            extract minimum energy of DuplexFold, $\Delta_{LowestDuplexFold}$
            for each $S_1..S_{n-1}$ do
                if $S_n$ had region hybridized with $S_1..S_{n-1}$ then
                    substitute the reverse complement into $S_n$
                    remove hybridized match in $S_n$ and record as $T_n$
                    record start position, QStart in $T_n$ if FBS > $FBS_{Max}$
                    find LongestComplementarity from QStart
                end if
            end for
        end if
        grab "GGGG" and "CCCC" from $S_n$
        calculate PercentageCG = $(T_C + T_G)/ T_L)$
        if $\Delta$AllSub < $\Delta$DuplexFold or $\Delta$LowestDuplexFold then
            incr P
        else if LongestComplementarity > $FBS_{Max}$ then
            incr P
        else if "GGGG" or "CCCC" exist then
            incr P
        else if PercentageCG < 40% or > 70% then
            incr P
        end if
        if P >0 then
            if forbidden region exist then
                MutateRegion = AllPosition–ForbidPosition
            else if forbidden region does not exist
                MutateRegion = AllPosition
            end if
            Randomly select a nucleotide, $N_{old}$ from MutateRegion
            Randomly select $N_{New}$ from {A, T, G, C}, $N_{New}$ != $N_{old}$
            Replace $N_{old}$ with $N_{New}$, initialize L=1
        end if
        if P = 0 then
            output $S_n$, initialize L = 0, move to next sequence $S_{n+1}$
            incr n
        end if
    end while
end for
```

### 4.2.1 Base Pairing at False Binding Sites (FBS)

As a general rule during DNA assembly, it was crucial for DNA sequences to form base pairings exactly at the pre-defined positions and at the same time avoided pairings at unwanted positions (mispairing). Unfortunately, such false-binding sites (FBS) were inevitable; otherwise the sequence diversity would be extremely low. As a consequence, base pairings at false-binding sites were limited to shorter lengths (Dirks, Lin, Winfree, & Pierce, 2004; J. SantaLucia et al., 1996; J. J. SantaLucia & Hicks, 2004) so that the false-binding sites were predicted to have higher free energy and accordingly, low probability of hybridization. This criterion was included as a crucial filter and was used to detect the longest complementary region that existed between two sequences. In this work, base pairing at a false-binding site was defined as the occurrence of two sequences that formed base pairings at unwanted positions.

The detection of base pairing at FBS was processed using the following three scripts that are *FindStartPosition.pl*, *CleanEmptyPosition.pl* and *GetLongestComplement.pl* (Figure 4.2). The calculations were conducted by aligning a query sequence against the remaining corresponding target sequences. Each query sequence was shifted a nucleotide at a time towards the 3' terminal to search for any complementary nucleotide in the target sequence. During each shift, if a nucleotide from a targeted strand was complementary with the nucleotide from the query strand, the FBS score increased by one, (if and only if the longest complementarity at unwanted position was more than six, otherwise the FBS-score remained unchanged). The final FBS-score represented the longest consecutive stretch of complementary bases that were detected between the two strands.

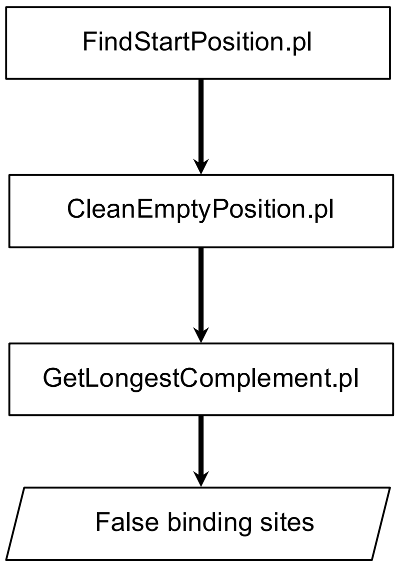


**Figure 4.2:** The search for false binding sites

The first script (*FindStartPosition.pl*) was employed to find all positions that had a minimum of seven consecutive complementary (*$Qmin*) nucleotides between the query and targeted sequence. It listed out every start position that matched to the minimum complementary bases. The output of *FindStartPosition.pl* listed every start position in the query *($QStart*) and the target (*$TStart*). The function of *CleanEmptyPosition.pl* was to remove the query, which did not meet the threshold value of at least seven consecutive matching nucleotides. The *GetLongestComplement.pl* script was then executed to obtain the longest matching complementary sequence using the start position output from the first script. Parameters used in the sequence design process were flexible to any design specification (e.g. minimum consecutive complementary might be different depending on the DNA shapes design).

### 4.2.2 Free Energy of Inter-molecular and Intra-molecular DNA

The free energy for a DNA sequence to form self-folding (intra-molecular) and double-stranded folding (intermolecular) were calculated using the *AllSub* and *DuplexFold* programs available in the *RNAstructure* package (Reuter & Mathews, 2010). The program *AllSub* was selected to generate all possible low free energy structures of a given DNA sequence. The program *DuplexFold* was used to predict the lowest free energy structure for two interacting sequences with a constraint of not allowing any intra-molecular base pairing to occur. Default parameters were selected with the exception of the RNA/DNA option, which was set to only DNA. This fitness evaluation required the free energy of *AllSub* to be higher (less negative) than the energy of *DuplexFold* (more negative). This was to ensure a relatively more stable structure when bindings occurred between the two ssDNAs as compared to the stability of ssDNA self-folding. It was also to make sure that correct base-pairing formation for inter-molecular assembly would occur.

### 4.2.3 G4 Pattern

The sequence design was prevented from having a G4 sub-sequence pattern because such sequences is favourable to form an unintended four-stranded G4 DNA structure (Sen & Gilbert, 1988).

### 4.2.4 Percentage of GC Content

The number of guanine-cytosine in oligonucleotides was set between 40% and 70% inclusive. The GC content was calculated by obtaining the number of GC versus the total nucleotide content. The maximum percentage of guanine and cytosine (GC) was set to 70% since DNA with high GC (70%) content was reported to be significantly harder to stretch than the 50% GC content DNA (Hormeño et al., 2011). The minimum percentage of GC content was set to 40% because a low GC content will cause the resulted DNA structure to be less stable (i.e. wobbly and flexible).

### 4.2.5 Mutation

The penalty score increased (i.e., increment by a point) whenever the sequence did not pass any of the fitness evaluation criteria (otherwise, the penalty score would be nil). If the total penalty score of the four fitness criteria exceeded 0, the sequence would undergo a mutation process. The algorithm would randomly select a new nucleotide to replace the existing nucleotide (at any random position) in the permissible region. Only a single nucleotide would be mutated at a time; the penalty score would be recalculated and mutations would be conducted repeatedly until the penalty score became nil.

The regions for the mutations to be exercised were based on 2 conditions depending whether a forbidden region existed. Variable *$MutateRegion* was a list of nucleotide positions that allowed mutations to occur, while variable *$ForbidPosition* was a list of nucleotide positions that did not allow mutations to occur mainly because these nucleotides were hybridized with the previous strands.

The formula to determine the mutation regions was *$MutateRegion* = *$AllPosition* - *$ForbidPosition*. An example of the calculation for *$MutateRegion* during the existence of forbidden region was depicted in Table 4.2. For this instance, sequence CB2 had 30 nucleotides, and the nucleotides numbered 16-30 from CB2 were complementary with nucleotides numbered 1-15 from strand CB1.

**Table 4.2:** Condition 1 (the existence of the forbidden region in CB2)

| | |
|---|---|
| *$AllPosition* | 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 |
| *$ForbidPosition* | 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 |
| *$MutateRegion* | 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 |

Table 4.3 depicted an example of mutating region (*$MutateRegion*) where the forbidden region was non-existence. In this instance, sequences in CL1 did not have complementary binding with any sequences. The length of the molecule was 35 nucleotides.

**Table 4.3:** Condition 2 (the non-existence of the forbidden region in CL1)

| | |
|---|---|
| *$AllPosition* | 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 |
| *$MutateRegion* | 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 |

Therefore, in order for a mutation to occur, a position would be randomly selected, identified as X in *$MutateRegion* and X would be replaced with a randomly selected nucleotide, $N_{New}$.

## 4.3 Laboratory Validation Protocol

### 4.3.1 DNA Annealing

Oligonucleotides (DNA sequences) were purchased from Integrated DNA Technologies Pte Ltd. To form individual shapes, oligonucleotides (4 oligonucleotides for T, L and B-shapes; 2 oligonucleotides for I-shape) were mixed stoichiometrically in a buffer containing 40 mM Tris base, 2.5 mM EDTA, and 13 mM $MgCl_2$. The final concentration of oligonucleotides was set to 0.5 μM. Then, the complexes were formed by annealing the reaction mixture for three hours from 90 $^{o}$C to 4 $^{o}$C in an Eppendorf Mastercycler Pro S thermocycler (Eppendorf, Hamburg, Germany). DNA annealing is a process for single-stranded DNA/RNA to hybridize and form double-stranded. The solution containing DNA sequences was not treated with any DNA polymerases, to ensure that they were held together only by non-covalent interactions (e.g. hydrogen bonds and base stacking).

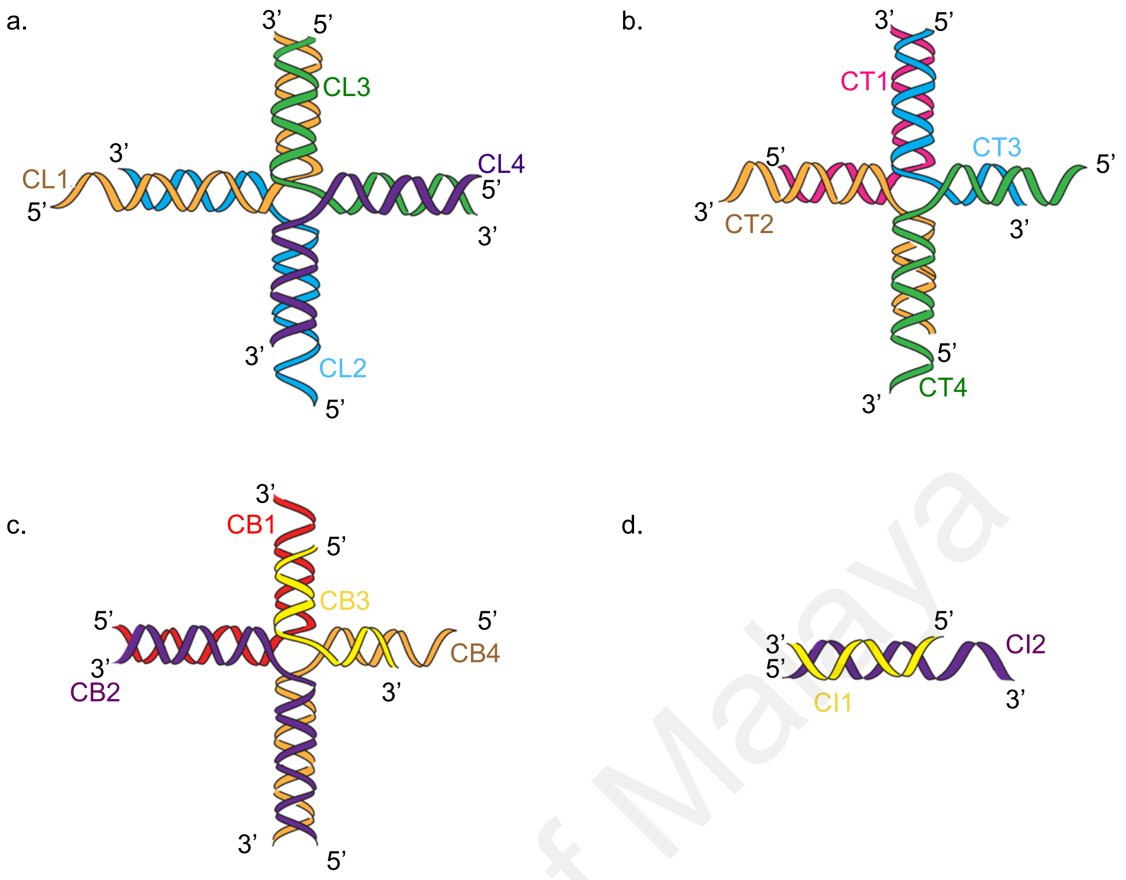
### 4.3.2 Gel Electrophoresis

The results of annealing reactions were analysed by electrophoresis using 12% non-denaturing 0.75 mm thick polyacrylamide gel (29:1 acrylamide: bisacrylamide). The running buffers contained 1· TBE (89 mM Tris base, 89 mM Boric acid and 2 mM

EDTA pH8.3) and 10 mM $MgCl_2$. The loading buffers contained 30% glycerol and 0.25% Bromophenol blue tracking dye. The gels were run at approximately 12 $V/cm^{-1}$ for 4 hours (for L-, B-, T-, I-Shapes) at 4 $^{o}C$ and then stained with the GelRed Nucleic Acid gel stain (Biotium, US). Gel electrophoresis is a method to separate biological molecules (DNA, RNA and proteins) based on their size and charge. It uses electrical current to separate DNA fragments by size as they migrate through a gel matrix (e.g. polyacrylamide gel). Smaller size DNA moves further away through the gel pores faster than the larger size DNA.

## 4.4 Results and Discussion

The sequence design tool began by generating individual DNA Tetris shapes. In this work, the autonomous tool generated 500 populations for each individual shape (L-Shape, T-Shape, B-Shape and I-Shape). A random sample from each shape was taken for further investigation to detect the assembly of the ssDNA components into the Tetris structures. This set of random sequences was being named as Set 1 (Figure 4.3). The generated sequence was the result of having sequence modifications employed during the design process. The automated sequence design tool generated these shapes based on the sequence dependency file *($Rec)*DefineSeq.txt in Table 4.4. This file listed all positions of nucleotides that formed complementary binding between different DNA strands.



| DNA strands | DNA sequences (In the direction of 5’ to 3’) |
|---|---|
| CL1 | CGAGTTGCATGTTAGGACGTACTCACTACCACGTA |
| CL2 | GCATGAGATTCCCATTTGATGATTCGCGTTAGTGGTTCCTACGACAAGATGCAGATG AGTACGTCCTAAC |
| CL3 | TACGTGGTAGATCCGTCGAT |
| CL4 | ATCGACGGATTCTGCATCTTGTCGTAGGAACCACTAACGCGAATCATCAAATGGG |
| CT1 | CTGCCGACATCAGGTCAGGCTCCGAACAGTAGATGGTGGAAGAGTATTCCGCTCGA TGCTTATCGGTATCCTGGA |
| CT2 | GAATTCCTGCACTATACTGTTCGGAGCCTGACCTGATGTCGGCAGCTAGAAGTTA |
| CT3 | TCCAGGATACCGATAAGCATCGAGCGGAATACTCTTCCACCATCTCTCCACAAGG |
| CT4 | ATGCAACTCGCCTTGTGGAGATAGTGCAGGAATTCCAAGACTCGA |
| CB1 | CCTCTGACACTAAGATCGTTGCTATGACGTTCGAGTCTTG |
| CB2 | GCGACCATGAGTGATTCTTAGTGTCAGAGG |
| CB3 | ACGTCATAGCAACGAATGGACACGTCAAGCTCAAG |
| CB4 | AATCTCATGCCTTGAGCTTGACGTGTCCATATCACTCATGGTCGC |
| CI1 | GTAAGACGATCACCTTAACTTCTAG |
| CI2 | AGGTGATCGTCTTAC |

**Figure 4.3:** DNA Tetris shapes generated by the sequence design tool (Set 1) Figure 4.3a-d represents L-shape, T-shape, B-shape and I-shape respectively. L-Shape, T-Shape and B-Shape are made up of 4 single-stranded DNA oligomers (CL1-CL4, CT1-CT4, CB1-CB4). I-Shape is made up of 2 single stranded DNA oligomers (CI1 and CI2). The table shows the DNA sequences used to form the shapes.

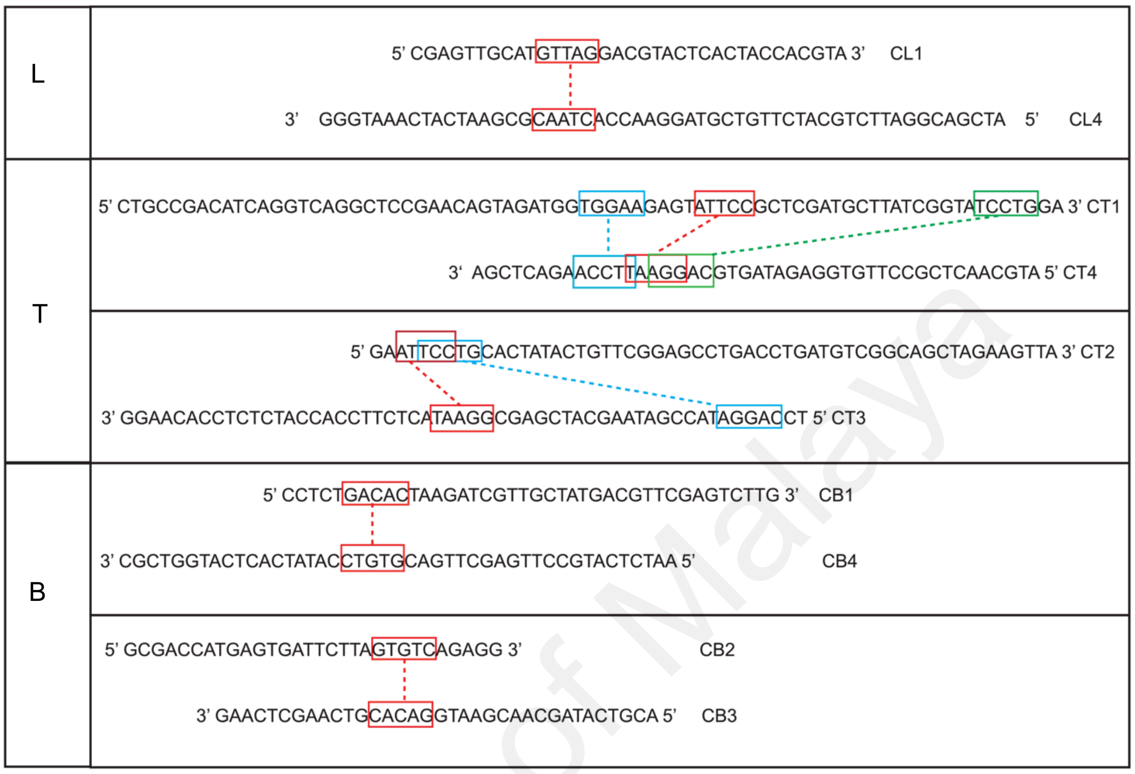
**Table 4.4:** Sequence dependency file for Set 1

| DNA strands, Curr | CurrLength | CurrStart | CurrEnd | MSeq | MStart | MEnd |
|---|---|---|---|---|---|---|
| CI1 | 25 | NIL | NIL | NIL | NIL | NIL |
| CI2 | 15 | 1 | 15 | CI1 | 1 | 15 |
| CT1 | 75 | NIL | NIL | NIL | NIL | NIL |
| CT2 | 55 | 46 | 55 | CI1 | 16 | 25 |
| CT2 | 55 | 16 | 45 | CT1 | 1 | 30 |
| CT3 | 55 | 1 | 45 | CT1 | 31 | 75 |
| CT4 | 45 | 21 | 35 | CT2 | 1 | 15 |
| CT4 | 45 | 11 | 20 | CT3 | 46 | 55 |
| CB1 | 40 | 31 | 40 | CT4 | 36 | 45 |
| CB2 | 30 | 16 | 30 | CB1 | 1 | 15 |
| CB3 | 35 | 1 | 15 | CB1 | 16 | 30 |
| CB4 | 45 | 31 | 45 | CB2 | 1 | 15 |
| CB4 | 45 | 11 | 30 | CB3 | 16 | 35 |
| CL1 | 35 | 1 | 10 | CT4 | 1 | 10 |
| CL2 | 70 | 56 | 70 | CL1 | 11 | 25 |
| CL2 | 70 | 1 | 10 | CB4 | 1 | 10 |
| CL3 | 20 | 1 | 10 | CL1 | 26 | 35 |
| CL4 | 55 | 11 | 55 | CL2 | 11 | 55 |
| CL4 | 55 | 1 | 10 | CL3 | 11 | 20 |

MSeq is the DNA strand that formed complementarity with Curr.
Region between MStart and MEnd in MSeq formed complementarity with region between CurrStart and CurrEnd in Curr.
NIL indicates an empty value.

### 4.4.1 Analysis of False Binding Sites for the Generated Sequences

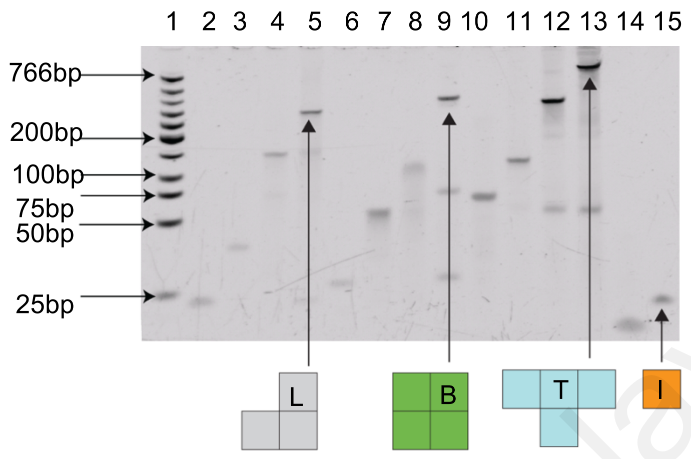
Previous study reported that five nucleotides (Winfree et al., 1998) are sufficient to create the possibility of binding, although six (Seiffert & Huhle, 2008) or more are more commonly used; and anything less than five is regarded as insufficient to form stable binding. The generated sequences were then subjected for analysis in order to detect the occurrences of the mispairings bases that might influence the result of the laboratory validation (Figure 4.4). Mispairing bases was defined as binding between bases at incorrect base positions.



L
5' CGAGTTGCATGTTAGGACGTACTCACTACCACGTA 3' CL1
3' GGGTAAACTACTAAGCGCAATCACCAAGGATGCTGTTCTACGTCTTAGGCAGCTA 5' CL4
T
5' CTGCCGACATCAGGTCAGGCTCCGAACAGTAGATGGTGGAAGAGTATTCCGCTCGATGCTTATCGGTATCCTGGA 3' CT1
3' AGCTCAGAACCTTAAGGACGTGATAGAGGTGTTCCGCTCAACGTA 5' CT4
5' GAATTCCTGCACTATACTGTTCGGAGCCTGACCTGATGTCGGCAGCTAGAAGTTA 3' CT2
3' GGAACACCTCTCTACCACCTTCTCATAAGGCGAGCTACGAATAGCCATAGGACCT 5' CT3
B
5' CCTCTGACACTAAGATCGTTGCTATGACGTTCGAGTCTTG 3' CB1
3' CGCTGGTACTCACTATACCTGTGCAGTTCGAGTTCCGTACTCTAA 5' CB4
5' GCGACCATGAGTGATTCTTAGTGTCAGAGG 3' CB2
3' GAACTCGAACTGCACAGGTAAGCAACGATACTGCA 5' CB3

**Figure 4.4:** List of mispairing bases

To fully implement the proposed hierarchical schematic, a less stringent approach was adopted during the sequence design. Mispairing of bases was allowed to occur in the designed sequences (with subtle limitations). By referring to the list of mispairing bases at Figure 4.4 and the resulting gel electrophoresis experiment in Figure 4.5, there were two extra bands appeared below the major bands observed in Lane 9 (B-shape) which were the unwanted aggregates proceeding from the mispairing between CB1-CB4 and CB2-CB3. As for the T-shape, there was an extra band with the same band size observed in both Lane 12 and Lane 13. Similarly, these were the unwanted aggregates derived from the mispairing of CT2-CT3. The complementary binding at the correct position was set at least 10 nucleotides to provide sufficient strength in the structure formation. Supported by the gel electrophoresis results, the formation of the designed DNA Tetris shape was satisfactory except for some minor unwanted aggregates (which was expected due to the allowance of the tool).



Legend

| | | | |
|---|---|---|---|
| 1 | Marker | 10 | CT1 |
| 2 | CL4 | 11 | CT1+CT2 |
| 3 | CL2+CL4 | 12 | CT1+CT2+CT3 |
| 4 | CL2+CL3+CL4 | 13 | CT1+CT2+CT3+CT4 |
| 5 | CL1+CL2+CL3+CL4 | 14 | CI1 |
| 6 | CB1 | 15 | CI1+CI2 |
| 7 | CB1+CB2 | | |
| 8 | CB1+CB2+CB3 | | |
| 9 | CB1+CB2+CB3+CB4 | | |

**Figure 4.5:** Gel electrophoresis for the formation of individual DNA Tetris shape The gel result showed the band increment for the sequence used to form the Tetris shape. Gel electrophoresis was conducted on 12% non-denaturing PAGE gel.

### 4.4.2 Free Energy Distribution of the Populations

The free energy for the interaction pairs, $\Delta G_{DuplexFold}$ was plotted in Figure 4.6. The distribution of the median (thick horizontal black line) showed a relatively uniform distribution between the first and third quartile. This implied that the majority of the populations have relatively similar free energy approximations. However for CI1-CI2 some of the generated sequences from the 500 populations were outliers and had relatively higher free energy (less negative). The asterisks (*) show the free energy for sequences in Set 1.

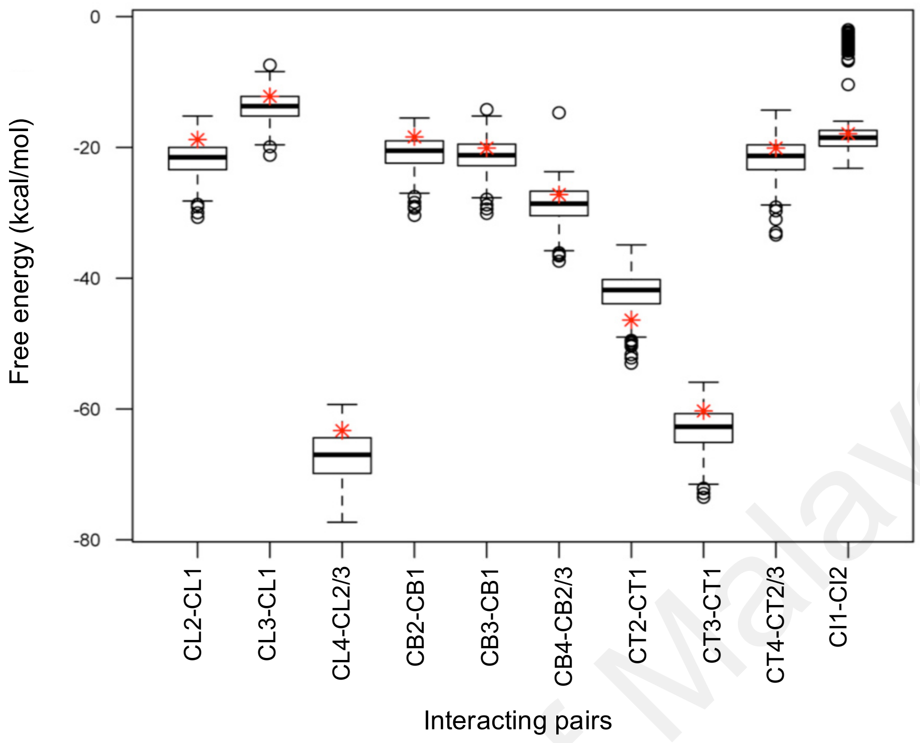


**Figure 4.6:** Boxplot of free energy for 500 populations in each shape CL4-CL2/3 implied that CL4 hybridized with CL2 and CL3. Free energy between CL4-CL2 and CL4-CL3 were generated and the lowest energy was used to plot the graph. The asterisk (*) represents the free energy of the strands used for gel electrophoresis study (CL2-CL1: -18.8kcal/mol, CL3-CL1: -12.2kcal/mol, CL4-CL2/3: -63.3kcal/mol, CB2-CB1: -18.4kcal/mol, CB3-CB1: -20.1kcal/mol, CB4-CB2/3: -27.2kcal/mol, CT2-CT1: -46.4kcal.mol, CT3-CT1: -60.3kcal/mol, CT4-CT2/3: -20.1kcal.mol and CI1-CI2: -17.9kcal/mol). Free energy were obtained using program *DuplexFold* and graphs were generated using R software version 2.15.1 (Team, 2005).

### 4.4.3 Number of Sequence Iterations

The average number of iterations for B-Shape is 9.9±0.46 cycle, L-Shape 8.5±0.53 cycle, T-Shape 22.4±1.13 cycle and I-Shape 3.1±0.15 cycle. The number of iterations increased linearly as the number of nucleotides in mutated regions increased. Furthermore, the number of iterations was also dependent on the complexity of the fitness criteria. However, the approach was still effective and did not require complicated heuristics in order to generate candidate sequences for each DNA Tetris shape. The number of iterations required for each shape was relatively small and the computational process was relatively fast.

Each sequence was defined to be dependent or partially dependent on the nucleotide pattern from the previous sequence using a top-down method (e.g. L1-L2-L3-L4). The optimization process would only proceed when sequence L1 had satisfied all the four criteria, and then continued with the following sequence (L2) until the design for all sequences were completed. The lack of positions for sequence mutations such as the I-shape (made up of two strands) caused the resulting structure to be less susceptible to changes. This was because the sequence arrangement in CI2 depended entirely on CI1 (CI2 does not have sticky ends that could be mutated) (Figure 4.7).

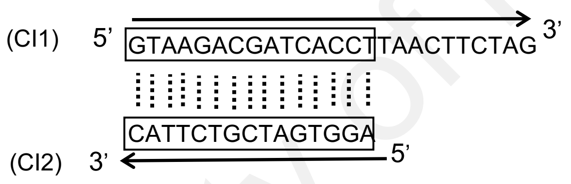


**Figure 4.7:** An example of lack of positions for sequence mutation Sequence arrangement in CI2 depended entirely on CI1 nucleotides position 1-15.

## 4.5 Summary

In this chapter, the development of module 1 sequence design tool for the construction of DNA Tetris shapes to support the newly proposed DNA Tetrominoes concept were presented. This autonomous tool revealed successful formations of the respective DNA Tetris shapes through integration of optimization algorithm. The successful formations of these shapes have been concluded following the experimental validation of gel electrophoresis and the extensive analyses on all possible mispairing bases. The development of module 2 DNA connectivity tool would be presented in the following Chapter 5.

# CHAPTER 5: MODULE 2 DNA CONNECTIVITY TOOL

## 5.1 Introduction

DNA sequences generated from the preceding chapter would require further study on the binding interactions that occurred between these DNA sequences. As a result, DNA connectivity tool (module 2) was developed to act as an annotation schema by generating connectivity maps between a set of interacting DNAs forming into the prescribed structures. It provides mapping of all probable paths taken by the DNA to form the structures using graph theory (Biggs, Lloyd, & Wilson, 1986). This is essential since molecular self-assembly is asynchronous and may have multitude of errors (Rothemund, Papadakis, & Winfree, 2004). The probable shapes (i.e. the "best" unit) must compete with partially probable shapes (i.e., the "next best" unit) during the assembly process at all time.

## 5.2 Framework for the Implementation of DNA Connectivity Tool

A computational tool was developed to map the binding interaction between DNA nucleotides based on the principle of binding dependencies (Ramlan & Zauner, 2013). An undirected graph representation was implemented, in which DNA segments were stored as nodes that could be connected using edges. This allowed the automated tool to compute all probable paths taken by each DNA segment (node) in forming the structures. It incorporated the use of two external program that is *UNAFold-3.8* (Markham & Zuker, 2008) and *DuplexFold* (Reuter & Mathews, 2010). Figure 5.1 outlined the processes, which began by utilizing the segmented DNA sequences to calculate the free energy and to build the binding affinity matrix. The scripts for the connectivity tool were attached in Appendix H-Q.

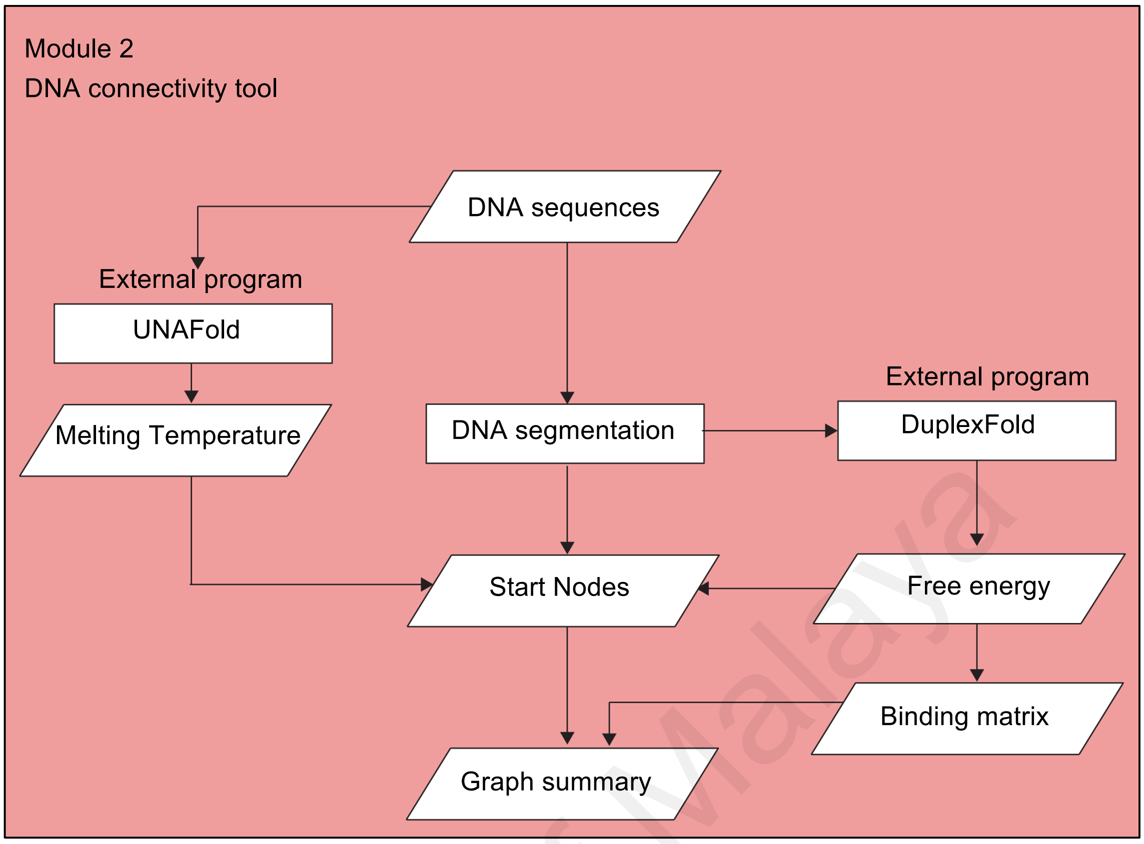


**Figure 5.1:** Framework for DNA connectivity tool (module 2)

## 5.3 DNA Segmentation

Every DNA strand was separated into different segments based on perfect complementarity (i.e. where all the intended bases hybridize as specified in the design) between its pairs. The lists of segments in each set of generated sequences can be obtained from their respective sequence dependency files. A TCL script *FindSegment_TmMatchPair.tcl* was written to extract these lists of DNA segments from the sequence dependency files and then utilized the lists to create the binding matrix. Example of how each perfect complementarity that took place in a double-stranded DNA and 4-way junction were illustrated in Figure 5.2a and Figure 5.2b. This had resulted in the double-stranded DNA to have 3 segments and the 4-way junction to have 10 segments. If both of these double-stranded DNA and 4-way junction were present in the same set of sequences, this set would have a total of 13 segments (3 segments + 10 segments). So the binding matrix would comprise of 13 rows and 13 columns.

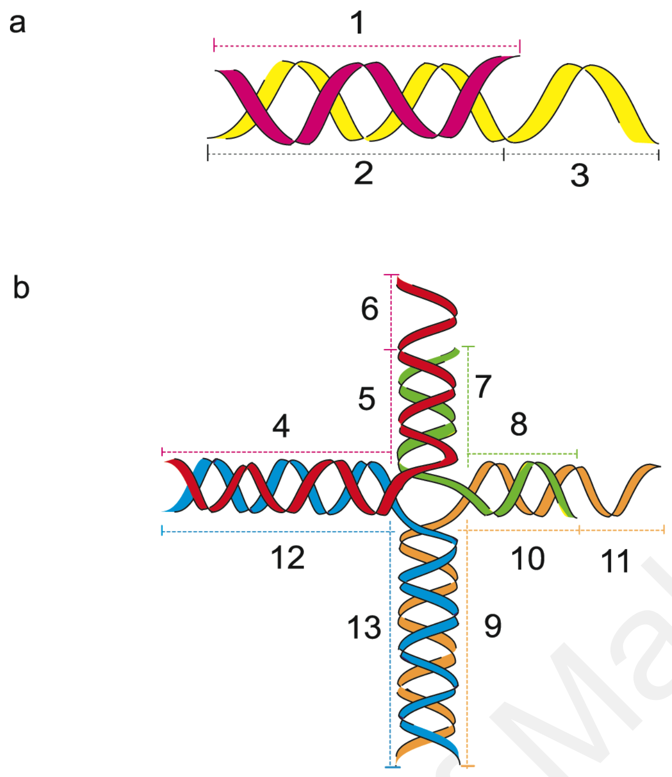


**Figure 5.2:** Example of segmentation in DNA sequence
Figure 5.2a represents the double-stranded DNA with three segments (1, 2, 3). Figure 5.2b represents the 4-way junction with ten segments (3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13).

## 5.4 Construction of Binding Matrix

After obtaining a list of DNA segment from prior procedure, the binding affinity between these DNA segments were measured through the calculation of free energy. The calculation of free energy on a matrix with 13 row and 13 columns was conducted on segment 1-1, segment 1-2…segment 1-13, segment 2-1, segment 2-2 … segment 2-13, until segment 13-1, segment 13-2 … segment 13-13. Each segment was then represented as node. The general formula for creating the free energy matrix with *n* number of nodes was presented in Eq. 5.1, by which $\Delta G_{i,j}$ is the free energy for node *i* and *j*.

$$\begin{bmatrix} X_{1,1} & X_{1,2} & \dots & X_{1,n} \\ X_{2,1} & X_{2,2} & \dots & X_{2,n} \\ \vdots & \vdots & \dots & \vdots \\ X_{n,1} & X_{n,2} & \dots & X_{n,n} \end{bmatrix} \tag{5.1}$$

$X_{i,j} = \Delta G_{i,j}; i,j = 1,2..n; n =$ Total number of nodes

The free energy between each node was calculated using the program *DuplexFold* (Reuter & Mathews, 2010). Default parameters for the program were used with the "DNA" setting. This free energy matrix was later converted into a binding affinity matrix. The conversion (Eq. 5.2) was done by dividing the free energy in every position of the matrix, $\Delta G_{i,j}$ with the lowest free energy obtained in each row $\left(\min\left(\Delta G_{i,j_{1..n}}\right)\right)$. This would generate the binding affinity between every node. The value of 1.0 indicated the lowest free energy (strongest binding) between all available binding nodes. For instance, when the binding affinity between node 1 and node 2 is 1.0, it indicates that the binding strength of node 1 with node 2 is the strongest compared to the remaining nodes (e.g. 3, 4, 5…etc). The formula for binding affinity was as follows:

$$\text{Binding affinity for } P_{i,j} = \frac{\Delta G_{i,j}}{\min\left(\Delta G_{i,j_{1...n}}\right)} \tag{5.2}$$

The edges connecting nodes with 1.0 binding affinity values represented the most favourable binding among *n* number of nodes. However, in circumstances where no edges carried the most favourable binding affinity values (1.0), the highest value would take precedence (in this implementation binding affinity must be above the threshold of 0.7).

## 5.5 Computation of All Probable Paths

Determining the start point: In order to determine the start point, the melting temperature (Tm) for every DNA pair $X_{i,j}$ was calculated using *UNAFold-3.8* (Markham & Zuker, 2008). The DNA pairs with Tm value equal or higher than third quartile were selected. From this pair, the segments (nodes) with lowest free energy were used as the starting point and the remainder of the nodes would act as the sticky ends. These sticky ends would then operate as precursors in determining which node to be selected next (Figure 5.3).

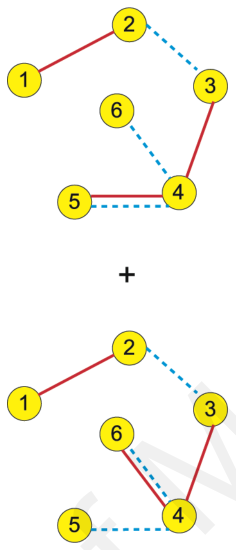


**Figure 5.3:** The graph search algorithm
The start point is from node 1 to node 2, leaving node 3 as sticky ends. During the next iteration, node 3 is bound with node 4 and hence resulting in the emergence of new potential sticky ends between node 5 and node 6. Because there are now two new sticky ends, the search is duplicated so that node 4 will bind to both node 5 and node 6. This will generate two paths: graph 1 (1-2-3-4-5) and graph 2 (1-2-3-4-6).

Greedy search phase: The graph would only proceed to the node with the binding affinity value $\left(P_{i,j} > 0.7\right)$. The default value was fixed at 0.7 to ensure that the graph was restricted to only displaying strong estimation values (i.e. representative of preferable binding interactions). Lower assignment of threshold would generate convoluted paths full of weak interactions, which would then complicate the search process. For every new node, two conditions were considered: the emergence of one or more new sticky end(s) or the non-availability of sticky ends. The initial value of every node started at 1.0. The DNA uptake rate was set at 0.001. Whenever a node was selected the value of that node would be deducted by the DNA uptake rate. The formula (Eq. 5.3) for node calculation was as follows:

$$[\mathrm{Node}_{\mathrm{NewCurrent}}] = [\mathrm{Node}_{\mathrm{Current}}] - [\mathrm{Node}_{\mathrm{UptakeRate}}] \tag{5.3}$$

The value of every node was evaluated during each cycle. The search would continue until the values of any node became nil (Algorithm 5).

**Algorithm 5** Computation of all paths in a set of interacting DNA sequences

```
1:  split DNA into different segments (Node), N
2:  define bound node (form base pairing)=$N_b$
3:  define unbound node (free sticky ends)=$N_u$,
4:  initialize all initial node concentration, [N] = 1.0
5:      **for** each $N_u$ **do**
6:          check probability matrix
7:          **if** $P_e$ > ThresholdValue **then**
8:              record new node, $N_{TempoNew}$ bind to $N_b$
9:              **for** each $N_{TempoNew}$ **do**
10:                 check all nodes concentration, [N] in the solution
11:                 **if** $[N_{all}] > 0\%$ **then**
12:                     $N_{TempoNew}$ binds to $N_u$
13:                     compute new sticky ends, $N_u$
14:                     record $N_u$
15:                     Update latest total solution concentration
16:                     $[N_{Latest}] = [N_{Current}] - [N_{UptakeRate}]$
17:                 **else**
18:                     No binding, $[N_{NewCurren}] = [N_{NewCurrent}]$
19:             **end for**
20:         **end if**
21:     **end for**
```

## 5.6 Graph Search

To address the complexity of determining "many sequences to many shapes configuration" allowance introduced in this approach, the concept of undirected graph was implemented (i.e. each node/vertex could be visited more than once; with no emphasis on the order of the path taken) as an annotation schema. In our implementation, “nodes” represented the DNA segments while the “edges” represented binding affinity. The decision of visiting any of these nodes was dependent on the free sticky ends resulted from prior binding (edge). As long as the new sticky ends had a probability value of more than the defined threshold value (0.7), it would be predicted to be able to bind to the existing parent DNA (node). This process would be repeated iteratively for each node (similar to a greedy search method where all paths were traversed). The generated graph served as an annotation schema to provide insights into the interaction of multiple DNA sequences presence in every combination of 3x4 rectangle. The following Chapter 6 will present the generation of multi-combinations of DNA Tetris shape in 3 x 4 rectangle whereby this DNA connectivity tool would be used to annotate the interactions that happened in the combinations.

## 5.7 Summary

This chapter described the development of module 2 DNA connectivity tool. This tool was employed to compute the binding affinities between a set of interacting DNA sequences. The binding affinity matrices were then converted into graph mapping so that it could provide insights into the level of competition in forming the end structures. This connectivity tool would be put into practice in the following chapter 6 to describe the formation of DNA sequence for five sets of different structural combinations.

# CHAPTER 6: MULTIPLE COMBINATIONS FOR DNA SELF-ASSEMBLY

## 6.1 Introduction

In this chapter, the plausibility in supporting self-assembly DNA through the implementation of many shapes to many combinations relationship was explored. This approach allowed the heterogeneous property of DNA Tetris shapes to form different combinations in a search space. As mentioned earlier, 3 x 4 rectangle was used as a hypothetical application to show the feasibility of this multiple combinations concept.

The multiple conformations of DNA Tetris shapes increased the flexibility in constructing DNA nanostructures given that the formation of the structures was achieved through the self-organization of competing DNA shapes. The core principle was to allow the most preferred shape and sequence combinations to take precedence (i.e., survival of the fittest). For instance, if *n* sets (where *n* is more than 1) of DNAs were initially designed to assemble into the desired conformations, in cases where a single set of the structure collapsed, the remaining *n-1* sets would still be able to form the targeted structure. In fact, individual units inside the *n-1* sets could replace the default unit of the original set. This interchangeability was the key in this approach.

Every component in each set is modular, whereby the failure of any particular unit would not affect the completeness of the set. The mechanism allowed a specific substitution (i.e. to replace any incompatible shapes) or replacement of the entire shape configurations to be executed. Total programmability was not promoted in this approach and the formation of the structures was entirely dependent on the self-organized characteristics of the molecule. This was necessary to handle the formation of structures under undesirable and uncontrollable physicochemical conditions. The multiple conformation concept would be highly significant in cases where DNA nanorobots inside a cellular environment that have time limitation to resist the enzymatic degradation (Mei et al., 2011; Shen et al., 2012).

Based on the concept of having the most preferred shape and sequence combinations to take precedence, the hypothesis would be to let the most stable DNA nanostructures against degradation to take precedence and carry out their tasks inside a living cell (cellular environment). Given the biological importance of having multiple sets of configurations the plausibility of using self-organization principle was explored to propose the concept of multi-configurations in DNA self-assembly.

## 6.2 Self-assembly into Multiple Combinations

After successful attempt of designing the first DNA Tetris configuration, which had since been named as Set 1 (as discussed in Chapter 4), another additional four sets of DNA sequences were constructed. These combinations were named as Set 2, Set 3, Set 4 and Set 5. These additional designs were an attempt to proof that in a specific dimension, (in this case 3 x 4 rectangle) different shape arrangements could be obtained that led to different combinations.

Figure 6.1 illustrates five different combinations that were generated to conform to the layout in 3 x 4 rectangle. The intermolecular bindings between various DNA shapes were loosely programmed using complementary sticky ends. Sticky ends were positioned at the intersection point (marked with *), where different shapes were adjacently located next to each other. The default lengths of sticky ends (for all DNA shapes) were set to 10 nucleotides. In this study, the complementary sticky ends were predefined to ensure different configurations were attainable.

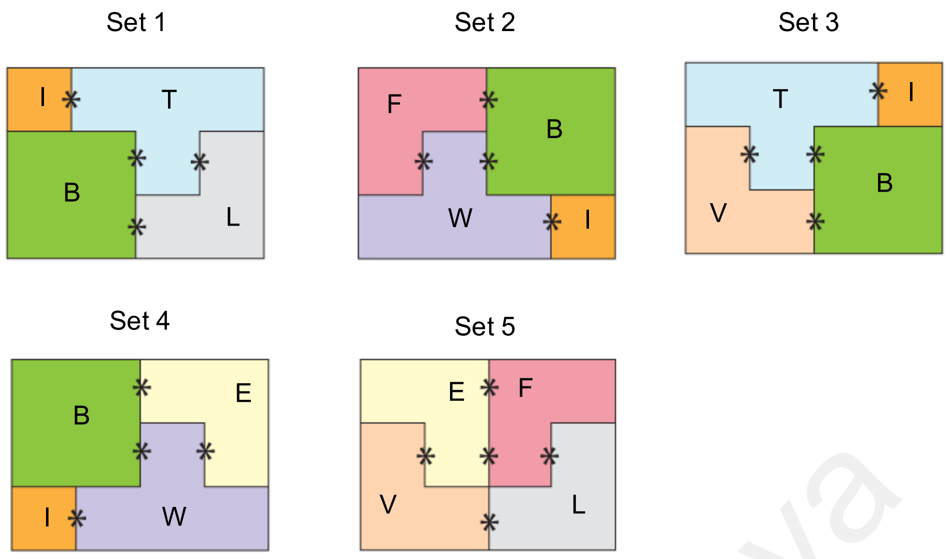
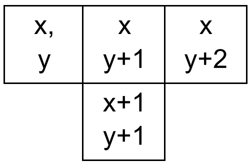
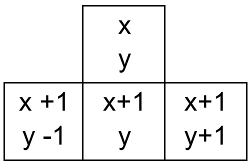
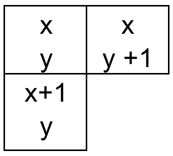
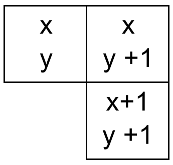


**Figure 6.1:** Five different combinations in 3 x 4 rectangles
Five different combinations (Set 1, Set 2, Set 3, Set 4 and Set 5) based on 3 x 4 rectangles skeleton. The symbol (*) represents the sticky ends, which are required to bind different DNA Tetris shapes through intermolecular binding.

In order to demonstrate the ability of DNA Tetris shapes to form multiple combinations for the same search space (many shapes to many combinations), a program to search for different combinations in 3 x 4 rectangles was developed. The 3 x 4 rectangle was translated into an array with 3 horizontal rows and 4 vertical columns. The process adapted Brute Force algorithm to evaluate whether each position could be used to occupy the Tetris shape candidates.

Each position in the rectangle was labelled in *x* and *y* format based on the coordinate system (Figure 6.2). For every incoming shape, the program would check whether the candidate Tetris shape was able to fit into the respective dimension. If the candidate passed the evaluation criteria, the array would automatically update to include that particular shape.

| Shape | Coordinates formula |
|---|---|
| T |  |
| W |  |
| F |  |
| E |  |

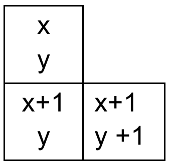
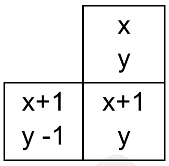
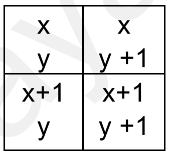
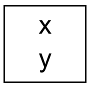
| Shape | Coordinate formula |
|---|---|
| V |  |
| L |  |
| B |  |
| I |  |

**Figure 6.2:** Coordinate system for Tetris shapes

The array of 3 x 4 rectangle was initialized with the value of zero, which indicated that the specific position was not occupied, by any of the Tetris shape (Figure 6.3a). Any value other than zero, for instance T, as illustrated in Figure 6.3b indicates that the respective position had already been occupied. Figure 6.3c demonstrates the circumstances whereby F-Shapes cannot occupy the dimension because position 0,4 was located outside the grid. Therefore, F-Shapes could not fit into the array and the program would then proceed with other candidate Tetris shapes to fill up the entire array. Figure 6.3d presents another circumstances whereby E-Shape cannot occupy a position because of its collision with the existing T-Shape. As a result, E-Shape could not fit into the dimension and the program would proceed to the remaining shapes.

a.

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 |

b.

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | T | T | T | 0 |
| 1 | 0 | T | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 |

c.

| | 0 | 1 | 2 | 3 | |
|---|---|---|---|---|---|
| 0 | T | T | T | F | F |
| 1 | 0 | T | 0 | F | |
| 2 | 0 | 0 | 0 | 0 | |

d.

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | T | T | TE | E |
| 1 | 0 | T | 0 | E |
| 2 | 0 | 0 | 0 | 0 |

**Figure 6.3:** Array of 3 x 4 rectangles

The 3 x 4 rectangles were treated as an array of 3 rows and 4 columns. Figure 6.3a displays an array that was initialized with zero. Figure 6.3b shows that T-Shape occupied the array at position (0,0), (0,1), (0,2) and (1,1). Figure 6.3c illustrates F-Shapes (designated as F) that could not occupy the dimension due to position (0,4) that was located outside the grid. Figure 6.3d illustrates E-Shapes (designated as E) that could not occupy the dimension due to its position (0,2) occupied by T-Shape.

In this work, five different combinations were used to comply the 3 x 4 rectangles framework (Figure 6.4).

| | | | |
|---|---|---|---|
| I | T | T | T |
| B | B | T | L |
| B | B | L | L |

| | | | |
|---|---|---|---|
| F | F | B | B |
| F | W | B | B |
| W | W | W | I |

| | | | |
|---|---|---|---|
| T | T | T | I |
| V | T | B | B |
| V | V | B | B |

| | | | |
|---|---|---|---|
| B | B | E | E |
| B | B | W | E |
| I | W | W | W |

| | | | |
|---|---|---|---|
| E | E | F | F |
| V | E | F | L |
| V | V | L | L |

**Figure 6.4:** Multiple combinations of Tetris shapes

DNA sequences representing respective DNA Tetris shapes were generated using the autonomous tool developed in Chapter 4. The program focused on the stacking and merging of blocks to form DNA Tetris shapes. It relied on the dependency information (sequence dependency file) of all nucleotide positions in different DNA strands. The sequence dependency files generated by Generate_Defineseq.pl for the following four combinations (Set 2, Set 3, Set 4 and Set 5) were presented in Table 6.1 - 6.4.

The parameters that were used during the sequence design: Length of main block, *$m*=30 nucleotides and Length of sticky ends, *$s*=10 nucleotides. Column 1 is the current sequence numbers; column 2, 3 and 4 are the sequence length, start and end positions of the current sequence. Column 5, 6 and 7 are the sequence name, start and end positions of the sequence that formed complementary regions with the respective sequence in the first column. NIL indicated an empty value.

**Table 6.1:** Sequence dependency file for Set 2

| DNA strands, Curr | CurrLength | CurrStart | CurrEnd | MSeq | MStart | MEnd |
|---|---|---|---|---|---|---|
| F1 | 80 | NIL | NIL | NIL | NIL | NIL |
| F2 | 30 | 16 | 30 | F1 | 1 | 15 |
| F3 | 80 | 1 | 55 | F1 | 16 | 70 |
| F4 | 30 | 16 | 30 | F2 | 1 | 15 |
| F4 | 30 | 1 | 15 | F3 | 56 | 70 |
| B1 | 70 | NIL | NIL | NIL | NIL | NIL |
| B2 | 80 | 71 | 80 | F1 | 71 | 80 |
| B2 | 80 | 36 | 70 | B1 | 1 | 35 |
| B3 | 70 | 1 | 35 | B1 | 36 | 70 |
| B4 | 80 | 36 | 70 | B2 | 1 | 35 |
| B4 | 80 | 1 | 35 | B3 | 36 | 70 |
| W1 | 40 | 31 | 40 | B4 | 71 | 80 |
| W2 | 80 | 71 | 80 | F3 | 71 | 80 |
| W2 | 80 | 56 | 70 | W1 | 1 | 15 |
| W3 | 80 | 1 | 15 | W1 | 16 | 30 |
| W4 | 110 | 56 | 110 | W2 | 1 | 55 |
| W4 | 110 | 1 | 55 | W3 | 16 | 70 |
| I1 | 30 | NIL | NIL | NIL | NIL | NIL |
| I2 | 40 | 31 | 40 | W3 | 71 | 80 |
| I2 | 40 | 1 | 30 | I1 | 1 | 30 |

MSeq is the DNA strand that formed complementarity with Curr.

Region between MStart and MEnd in MSeq formed complementarity with region between CurrStart and CurrEnd in Curr.

NIL indicates an empty value.

**Table 6.2:** Sequence dependency file for Set 3

| DNA strands, Curr | CurrLength | CurrStart | CurrEnd | MSeq | MStart | MEnd |
|---|---|---|---|---|---|---|
| T1 | 120 | NIL | NIL | NIL | NIL | NIL |
| T2 | 70 | 16 | 70 | T1 | 1 | 55 |
| T3 | 80 | 1 | 55 | T1 | 56 | 110 |
| T4 | 40 | 16 | 30 | T2 | 1 | 15 |
| T4 | 40 | 1 | 15 | T3 | 56 | 70 |
| I1 | 30 | NIL | NIL | NIL | NIL | NIL |
| I2 | 40 | 31 | 40 | T1 | 111 | 120 |
| I2 | 40 | 1 | 30 | I1 | 1 | 30 |
| V1 | 40 | 31 | 40 | T4 | 31 | 40 |
| V2 | 30 | 16 | 30 | V1 | 1 | 15 |
| V3 | 80 | 1 | 15 | V1 | 16 | 30 |
| V4 | 70 | 56 | 70 | V2 | 1 | 15 |
| V4 | 70 | 1 | 55 | V3 | 16 | 70 |
| B1 | 70 | NIL | NIL | NIL | NIL | NIL |
| B2 | 80 | 71 | 80 | T3 | 71 | 80 |
| B2 | 80 | 36 | 70 | B1 | 1 | 35 |
| B3 | 70 | 1 | 35 | B1 | 36 | 70 |
| B4 | 80 | 71 | 80 | V3 | 71 | 80 |
| B4 | 80 | 36 | 70 | B2 | 1 | 35 |
| B4 | 80 | 1 | 35 | B3 | 36 | 70 |

MSeq is the DNA strand that formed complementarity with Curr.

Region between MStart and MEnd in MSeq formed complementarity with region between CurrStart and CurrEnd in Curr.

NIL indicates an empty value.

**Table 6.3:** Sequence dependency file for Set 4

| DNA strands, Curr | CurrLength | CurrStart | CurrEnd | MSeq | MStart | MEnd |
|---|---|---|---|---|---|---|
| B1 | 80 | NIL | NIL | NIL | NIL | NIL |
| B2 | 70 | 36 | 70 | B1 | 1 | 35 |
| B3 | 80 | 1 | 35 | B1 | 36 | 70 |
| B4 | 70 | 36 | 70 | B2 | 1 | 35 |
| B4 | 70 | 1 | 35 | B3 | 36 | 70 |
| E1 | 70 | NIL | NIL | NIL | NIL | NIL |
| E2 | 80 | 71 | 80 | B1 | 71 | 80 |
| E2 | 80 | 16 | 70 | E1 | 1 | 55 |
| E3 | 30 | 1 | 15 | E1 | 56 | 70 |
| E4 | 40 | 16 | 30 | E2 | 1 | 15 |
| E4 | 40 | 1 | 15 | E3 | 16 | 30 |
| W1 | 40 | 31 | 40 | E4 | 31 | 40 |
| W2 | 80 | 71 | 80 | B3 | 71 | 80 |
| W2 | 80 | 56 | 70 | W1 | 1 | 15 |
| W3 | 70 | 1 | 15 | W1 | 16 | 30 |
| W4 | 120 | 56 | 110 | W2 | 1 | 55 |
| W4 | 120 | 1 | 55 | W3 | 16 | 70 |
| I1 | 40 | 31 | 40 | W4 | 111 | 120 |
| I2 | 30 | 1 | 30 | I1 | 1 | 30 |

MSeq is the DNA strand that formed complementarity with Curr.

Region between MStart and MEnd in MSeq formed complementarity with region between CurrStart and CurrEnd in Curr.

NIL indicates an empty value.

**Table 6.4:** Sequence dependency file for Set 5

| DNA strands, Curr | CurrLength | CurrStart | CurrEnd | MSeq | MStart | MEnd |
|---|---|---|---|---|---|---|
| E1 | 80 | NIL | NIL | NIL | NIL | NIL |
| E2 | 70 | 16 | 70 | E1 | 1 | 55 |
| E3 | 40 | 1 | 15 | E1 | 56 | 70 |
| E4 | 40 | 16 | 30 | E2 | 1 | 15 |
| E4 | 40 | 1 | 15 | E3 | 16 | 30 |
| F1 | 70 | NIL | NIL | NIL | NIL | NIL |
| F2 | 40 | 31 | 40 | E1 | 71 | 80 |
| F2 | 40 | 16 | 30 | F1 | 1 | 15 |
| F3 | 80 | 1 | 55 | F1 | 16 | 70 |
| F4 | 40 | 31 | 40 | E3 | 31 | 40 |
| F4 | 40 | 16 | 30 | F2 | 1 | 15 |
| F4 | 40 | 1 | 15 | F3 | 56 | 70 |
| V1 | 40 | 31 | 40 | E4 | 31 | 40 |
| V2 | 30 | 16 | 30 | V1 | 1 | 15 |
| V3 | 80 | 1 | 15 | V1 | 16 | 30 |
| V4 | 70 | 56 | 70 | V2 | 1 | 15 |
| V4 | 70 | 1 | 55 | V3 | 16 | 70 |
| L1 | 30 | NIL | NIL | NIL | NIL | NIL |
| L2 | 80 | 71 | 80 | F3 | 71 | 80 |
| L2 | 80 | 56 | 70 | L1 | 1 | 15 |
| L3 | 30 | 1 | 15 | L1 | 16 | 30 |
| L4 | 80 | 71 | 80 | V3 | 71 | 80 |
| L4 | 80 | 16 | 70 | L2 | 1 | 55 |
| L4 | 80 | 1 | 15 | L3 | 16 | 30 |

MSeq is the DNA strand that formed complementarity with Curr.

Region between MStart and MEnd in MSeq formed complementarity with region between CurrStart and CurrEnd in Curr.

NIL indicates an empty value.

### 6.2.1 Derivation of Sequence Dependency Files

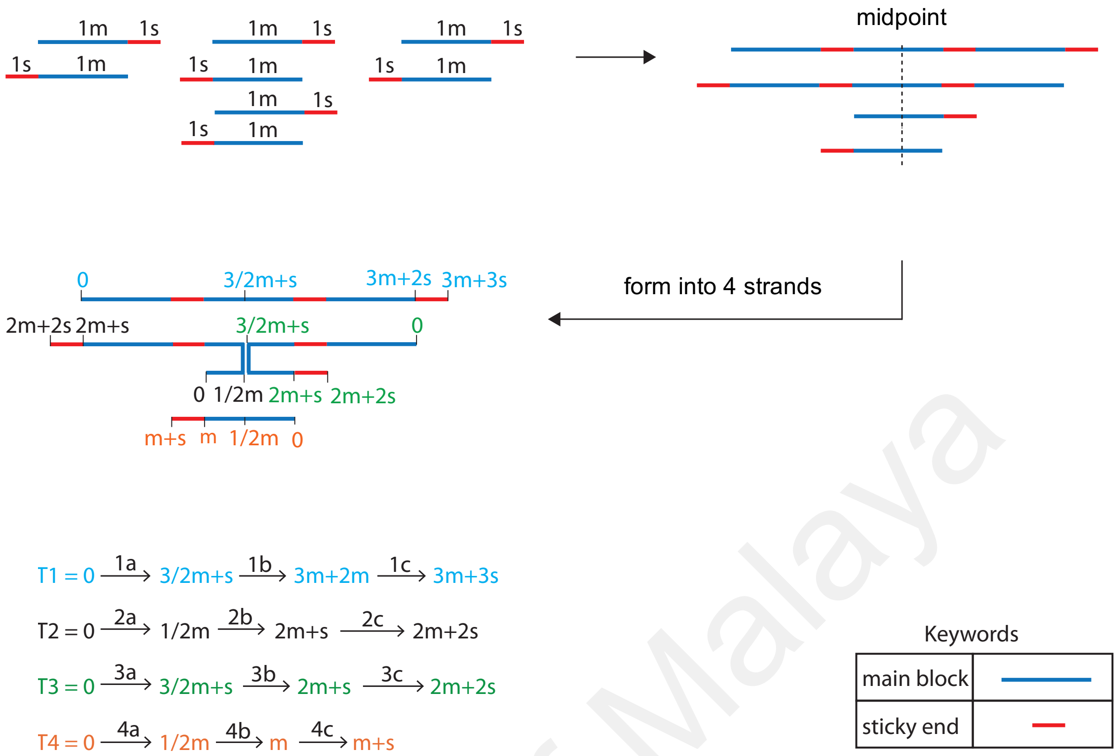
The sequence dependency files, *($Rec)*DefineSeq.txt was a list of all nucleotide positions that formed complementary binding between DNA strands. The algorithm for Generate_DefineSeq.pl to generate sequence dependency files was presented in Algorithm 6.

**Algorithm 6** Generation of sequence dependency file

```
1:   initialize $mainblockLength, $StickyEndLength
2:   for each rectangle do
3:       for each shape do
4:           compute every segmented position in the sequences
5:       end for
6:     identify shapes at leftmost and rightmost position in the dimension
7:     identify the sticky ends positions in each shape
8:     compute total number of strands
9:     substitute missing positions with NIL
10:    output as ($Rec)DefineSeq.txt
11:  end for
```

The generation of sequence dependency files required the identification of each segmented position in the DNA sequences (Line 4, Algorithm 7). This segmented sequence would then be used to identify the complementary nucleotides in their corresponding sequence within the same shape. In addition, the generation of sequence dependency files also required the identification of sticky end position whereby the sticky end from one shape was needed to bind to other DNA Tetris shapes (Line 7, Algorithm 7). The diagram to illustrate each segmented position in DNA sequences and identification of intersection points for all eight DNA Tetris shapes were presented in Figure 6.5 - 6.12 and Algorithm 7 - 14.

- **T-shape**



1m 1s
1s 1m
midpoint
form into 4 strands
0
3/2m+s
3m+2s 3m+3s
2m+2s 2m+s
3/2m+s
0
0 1/2m 2m+s 2m+2s
m+s m 1/2m 0
T1 = 0 1a → 3/2m+s 1b → 3m+2m 1c → 3m+3s
T2 = 0 2a → 1/2m 2b → 2m+s 2c → 2m+2s
T3 = 0 3a → 3/2m+s 3b → 2m+s 3c → 2m+2s
T4 = 0 4a → 1/2m 4b → m 4c → m+s
Keywords
main block
sticky end

**Figure 6.5:** The segmented positions in T-shape

The T-shape consists of four strands, T1 (blue), T2 (black), T3 (green) and T4 (orange). Each of the strands has three segments, with T1 (1a, 1b and 1c), T2 (2a, 2b, 2c), T3 (3a, 3b, 3c) and T4 (4a, 4b, 4c). For example in T1, segment 1a starts from position 0+1 to 3/2m+s; segment 1b starts from position 3/2m+s+1 to 3m+2s and segment 1c starts from position 3m+2s+1 to 3m+3s. Overall, the 4 strands have segment 1a complement with 2b, segment 1b complement with 3a, segment 2a complement with 4b and segment 3b complement with 4a. Variable m is the length of the main block and s is the length of the sticky end. Midpoint is the middle position of the design and is the position where T2 and T3 are formed. Position 0 indicates 5' terminal.

**Algorithm 7** Identification of sticky end in T-shape that bind with adjacent shape

```
1:  for each block linked to adjacent shape do
2:      if intersection point = row 1 and Position T = left then
3:          segment 1c bind with adjacent shape
4:      else if intersection point = row 2 and Position T = left then
5:          segment 3c bind with adjacent shape
6:      else if intersection point = row 1 and Position T = right then
7:          segment 2c bind with adjacent shape
8:      else if intersection point = row 2 and Position T = right then
9:          segment 4c bind with adjacent shape
10:     end if
11: end for
```

- **W-shape**

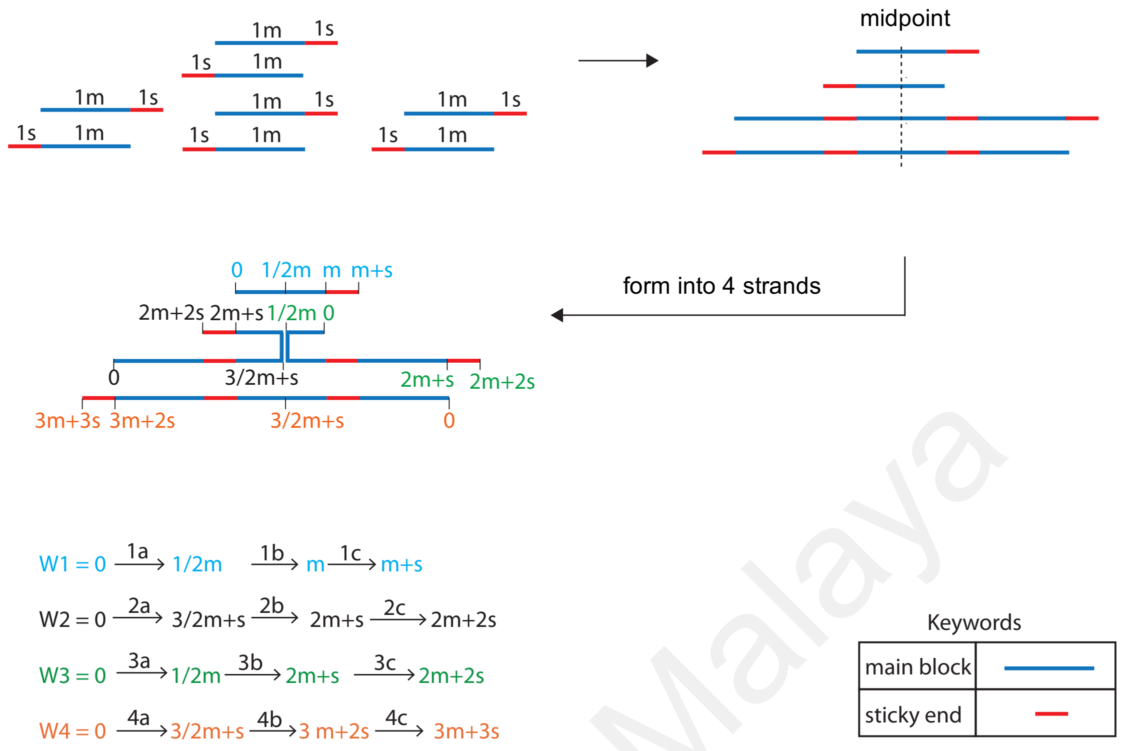


**Figure 6.6:** The segmented positions in W-shape

The W-shape consists of four strands, W1 (blue), W2 (black), W3 (green) and W4 (orange). Each of the strands has three segments, with W1 (1a, 1b and 1c), W2 (2a, 2b, 2c), W3 (3a, 3b, 3c) and W4 (4a, 4b, 4c). For example in W1, segment 1a starts from position 0+1 to 1/2m; segment 1b starts from position 1/2m+1 to m and segment 1c starts from position m+1 to m+s. Overall, the 4 strands have segment 1a complement with 2b, segment 1b complement with 3a, segment 2a complement with 4b and segment 3b complement with 4a. Variable m is the length of the main block and s is the length of the sticky end. Midpoint is the middle position of the design and is the position where W2 and W3 are formed. Position 0 indicates 5' terminal.

**Algorithm 8** Identification of sticky end in W-shape that bind with adjacent shape

```
  1:    for each block linked to adjacent shape do
  2:        if intersection point = row 1 and Position W = left then
  3:            segment 1c bind with adjacent shape
  4:        else if intersection point = row 2 and Position W = left then
  5:            segment 3c bind with adjacent shape
  6:        else if intersection point = row 1 and Position W = right then
  7:            segment 2c bind with adjacent shape
  8:        else if intersection point = row 2 and Position W = right then
  9:            segment 4c bind with adjacent shape
 10:        end if
111:    end for
```

- **F-shape**

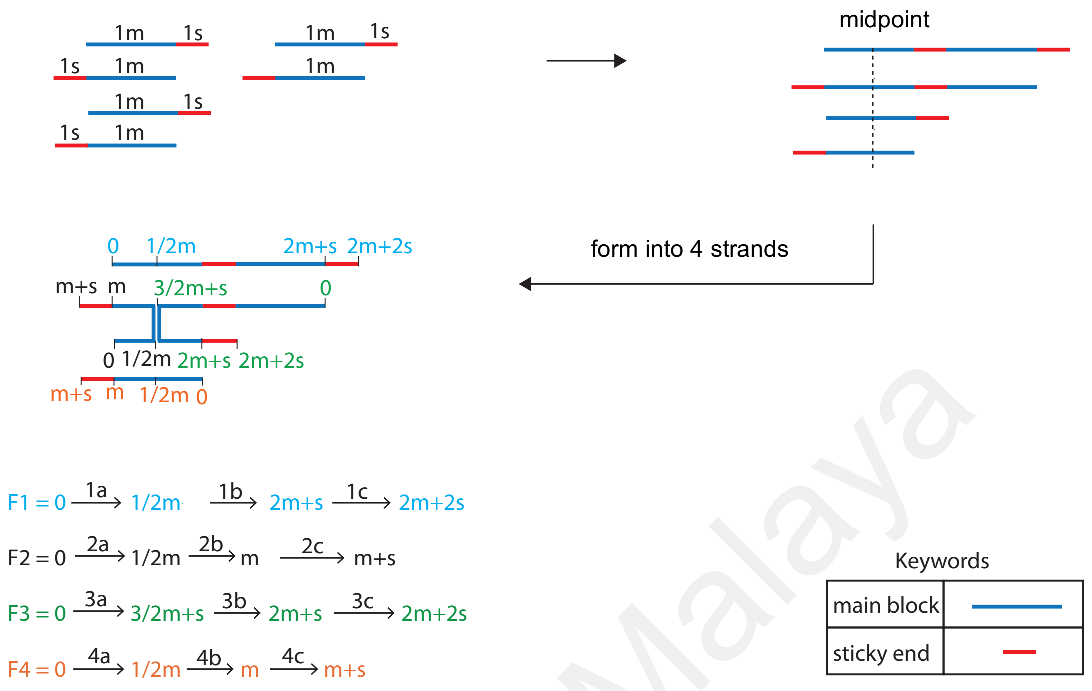


**Figure 6.7:** The segmented positions in F-shape

The F-shape consists of four strands, F1 (blue), F2 (black), F3 (green) and F4 (orange). Each of the strands has three segments, with F1 (1a, 1b and 1c), F2 (2a, 2b, 2c), F3 (3a, 3b, 3c) and F4 (4a, 4b, 4c). For example in F1, segment 1a starts from position 0+1 to 1/2m; segment 1b starts from position 1/2m+1 to 2m+s and segment 1c starts from position 2m+s+1 to 2m+2s. Overall, the 4 strands have segment 1a complement with 2b, segment 1b complement with 3a, segment 2a complement with 4b and segment 3b complement with 4a. Variable m is the length of the main block and s is the length of the sticky end. Midpoint is the middle position of the design and is the position where F2 and F3 are formed. Position 0 indicates 5' terminal.

**Algorithm 9** Identification of sticky end in F-shape that bind with adjacent shape

```
1:  for each block linked to adjacent shape do
2:      if intersection point = row 1 and position F = left then
3:          segment 1c bind with adjacent shapes
4:      else if intersection point = row 2 and position F = left then
5:          segment 3c bind with adjacent shapes
6:      else if intersection point = row 1 and position F = right then
7:          segment 2c bind with adjacent shapes
8:      else if intersection point = row 2 and position F = right then
9:          segment 4c bind with adjacent shapes
10:     end if
11: end for
```

- **E-shape**

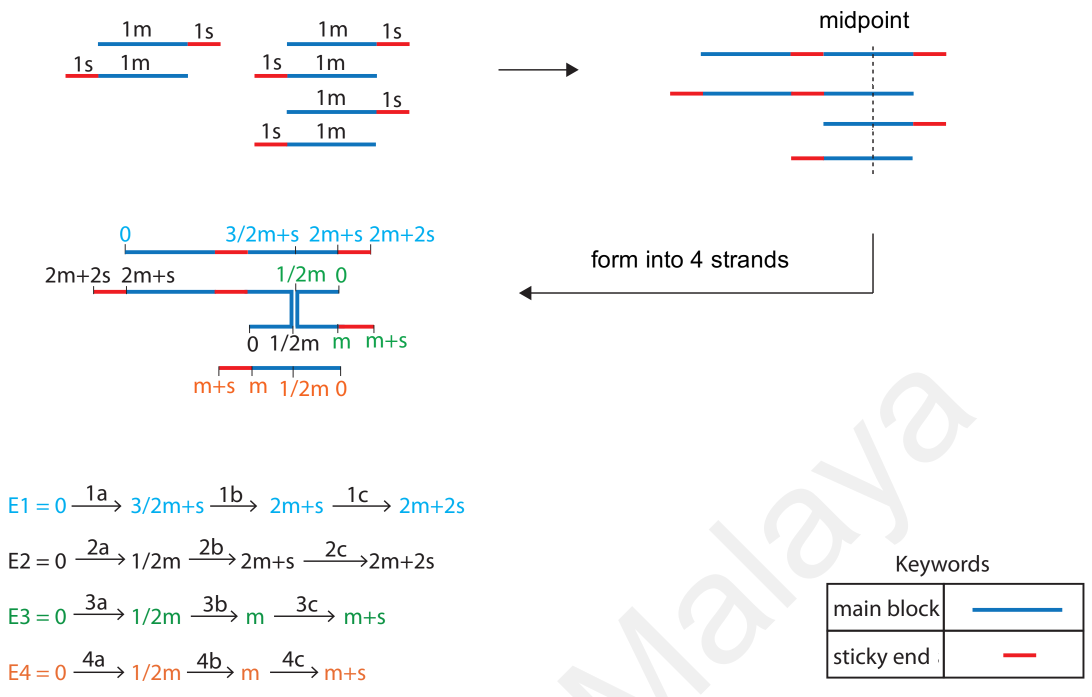
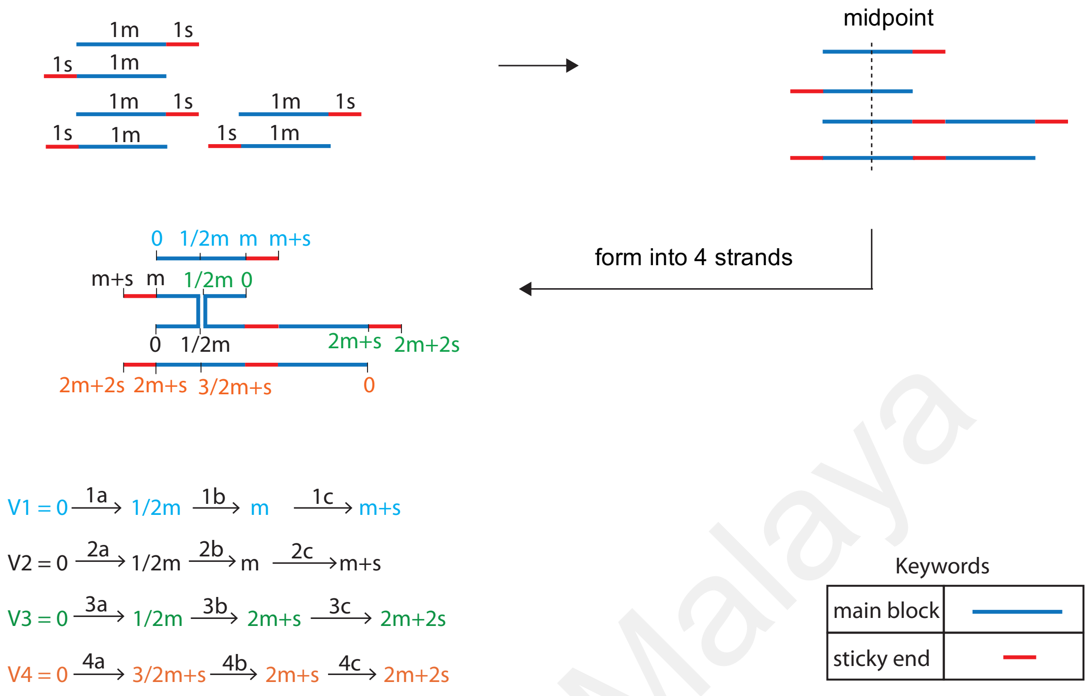


**Figure 6.8:** The segmented positions in E-shape
The E-shape consists of four strands E1 (blue), E2 (black), E3 (green) and E4 (orange). Each of the strands has three segments, with E1 (1a, 1b and 1c), E2 (2a, 2b, 2c), E3 (3a, 3b, 3c) and E4 (4a, 4b, 4c). For example in E1, segment 1a starts from position 0+1 to 3/2m+s; segment 1b starts from position 3/2m+s+1 to 2m+s and segment 1c starts from position 2m+s+1 to 2m+2s. Overall, the 4 strands have segment 1a complement with 2b, segment 1b complement with 3a, segment 2a complement with 4b and segment 3b complement with 4a. Variable m is the length of the main block and s is the length of the sticky end. Midpoint is the middle position of the design and is the position where E2 and E3 are formed. Position 0 indicates 5' terminal.

**Algorithm 10** Identification of sticky end in E-shape that bind with adjacent shape

```
1:   for each block linked to adjacent shape do
2:       if intersection point = row 1 and position E = left then
3:           segment 1c bind with adjacent shapes
4:       else if intersection point = row 2 and position E = left then
5:           segment 3c bind with adjacent shapes
6:       else if intersection point = row 1 and position E = right then
7:           segment 2c bind with adjacent shapes
8:       else if intersection point = row 2 and position E = right then
9:           segment 4c bind with adjacent shapes
10:      end if
11:  end for
```

- **V-shape**



1m 1s
1s 1m
midpoint
form into 4 strands
0 1/2m m m+s
m+s m 1/2m 0
0 1/2m 2m+s 2m+2s
2m+2s 2m+s 3/2m+s 0
V1 = 0 $\xrightarrow{1a}$ 1/2m $\xrightarrow{1b}$ m $\xrightarrow{1c}$ m+s
V2 = 0 $\xrightarrow{2a}$ 1/2m $\xrightarrow{2b}$ m $\xrightarrow{2c}$ m+s
V3 = 0 $\xrightarrow{3a}$ 1/2m $\xrightarrow{3b}$ 2m+s $\xrightarrow{3c}$ 2m+2s
V4 = 0 $\xrightarrow{4a}$ 3/2m+s $\xrightarrow{4b}$ 2m+s $\xrightarrow{4c}$ 2m+2s
Keywords
main block
sticky end

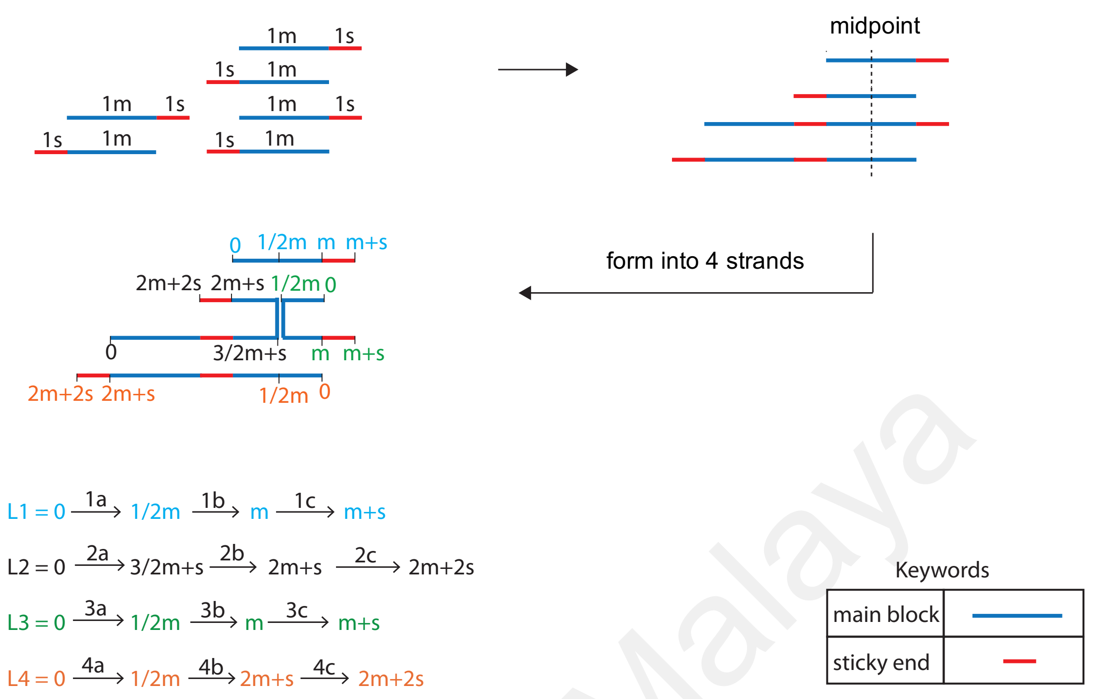
**Figure 6.9:** The segmented positions in V-shape

The V-shape consists of four strands V1 (blue), V2 (black), V3 (green) and V4 (orange). Each of the strands has three segments, with V1 (1a, 1b and 1c), V2 (2a, 2b, 2c), V3 (3a, 3b, 3c) and V4 (4a, 4b, 4c). For example in V1, segment 1a starts from position 0+1 to 1/2m; segment 1b starts from position 1/2m+1 to m and segment 1c starts from position m+1 to m+s. Overall, the 4 strands have segment 1a complement with 2b, segment 1b complement with 3a, segment 2a complement with 4b and segment 3b complement with 4a. Variable m is the length of the main block and s is the length of the sticky end. Midpoint is the middle position of the design and is the position where V2 and V3 are formed. Position 0 indicates 5' terminal.

**Algorithm 11** Identification of sticky end in V-shape that bind with adjacent shape

```
1:   for each block linked to adjacent shape do
2:       if intersection point = row 1 and position V = left then
3:           segment 1c bind with adjacent shapes
4:       else if intersection point = row 2 and position V = left then
5:           segment 3c bind with adjacent shapes
6:       else if intersection point = row 1 and position V = right then
7:           segment 2c bind with adjacent shapes
8:       else if intersection point = row 2 and position V = right then
9:           segment 4c bind with adjacent shapes
10:      end if
11:  end for
```

- **L-shape**



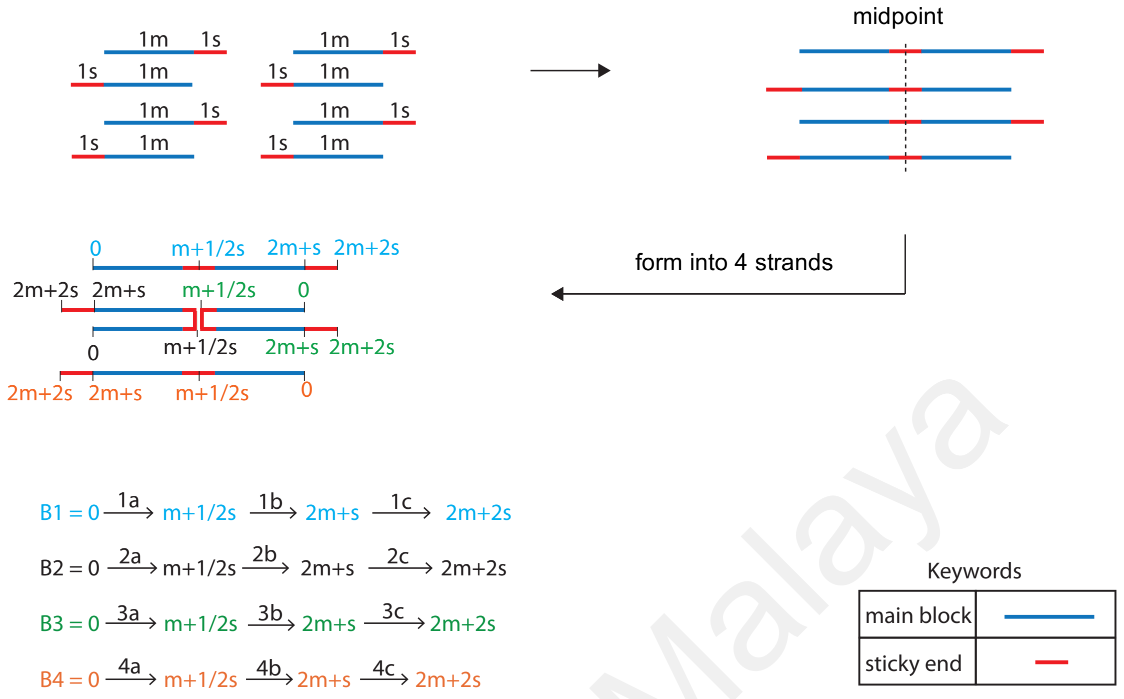
1m 1s
1s 1m
1m 1s
1s 1m
midpoint
form into 4 strands
0 1/2m m m+s
2m+2s 2m+s 1/2m 0
0 3/2m+s m m+s
2m+2s 2m+s 1/2m 0
L1 = 0 1a → 1/2m 1b → m 1c → m+s
L2 = 0 2a → 3/2m+s 2b → 2m+s 2c → 2m+2s
L3 = 0 3a → 1/2m 3b → m 3c → m+s
L4 = 0 4a → 1/2m 4b → 2m+s 4c → 2m+2s
Keywords
main block
sticky end

**Figure 6.10:** The segmented positions in L-shape
The L-shape consists of four strands L1 (blue), L2 (black), L3 (green) and L4 (orange). Each of the strands has three segments, with L1 (1a, 1b and 1c), L2 (2a, 2b, 2c), L3 (3a, 3b, 3c) and L4 (4a, 4b, 4c). For example in L1, segment 1a starts from position 0+1 to 1/2m; segment 1b starts from position 1/2m+1 to m and segment 1c starts from position m+1 to m+s. Overall, the 4 strands have segment 1a complement with 2b, segment 1b complement with 3a, segment 2a complement with 4b and segment 3b complement with 4a. Variable m is the length of the main block and s is the length of the sticky end. Midpoint is the middle position of the design and is the position where the L2 and L3 are formed. Position 0 indicates 5' terminal.

**Algorithm 12** Identification of sticky end in L-shape that bind with adjacent shape

```
1:   for each block linked to adjacent shape do
2:       if intersection point = row 1 and position L = left then
3:           segment 1c bind with adjacent shapes
4:       else if intersection point = row 2 and position L = left then
5:           segment 3c bind with adjacent shapes
6:       else if intersection point = row 1 and position L = right then
7:           segment 2c bind with adjacent shapes
8:       else if intersection point = row 2 and position L = right then
9:           segment 4c bind with adjacent shapes
10:      end if
11:  end for
```

- **B-shape**



1m
1s
midpoint
form into 4 strands
0
m+1/2s
2m+s
2m+2s
$B1 = 0 \xrightarrow{1a} m+1/2s \xrightarrow{1b} 2m+s \xrightarrow{1c} 2m+2s$
$B2 = 0 \xrightarrow{2a} m+1/2s \xrightarrow{2b} 2m+s \xrightarrow{2c} 2m+2s$
$B3 = 0 \xrightarrow{3a} m+1/2s \xrightarrow{3b} 2m+s \xrightarrow{3c} 2m+2s$
$B4 = 0 \xrightarrow{4a} m+1/2s \xrightarrow{4b} 2m+s \xrightarrow{4c} 2m+2s$
Keywords
main block
sticky end

**Figure 6.11:** The segmented positions in B-shape

The B-shape consists of four strands B1 (blue), B2 (black), B3 (green) and B4 (orange). Each of the strands has three segments, B1 (1a, 1b and 1c), B2 (2a, 2b, 2c), B3 (3a, 3b, 3c) and B4 (4a, 4b, 4c). For example in B1, segment 1a starts from position 0+1 to m+1/2s; segment 1b starts from position m+1/2s+1 to 2m+s and segment 1c starts from position 2m+s+1 to 2m+2s. Overall, the 4 strands have segment 1a complement with 2b, segment 1b complement with 3a, segment 2a complement with 4b and segment 3b complement with 4a. Variable m is the length of the main block and s is the length of the sticky end. Midpoint is the middle position of the design and is the position where the B2 and B3 are formed. Position 0 indicates 5' terminal.

**Algorithm 13** Identification of sticky end in B-shape that bind with adjacent shape

```
1:   for each block linked to adjacent shape do
2:       if intersection point = row 1 and position B = left then
3:           segment 1c bind with adjacent shapes
4:       else if intersection point = row 2 and position B = left then
5:           segment 3c bind with adjacent shapes
6:       else if intersection point = row 1 and position B = right then
7:           segment 2c bind with adjacent shapes
8:       else if intersection point = row 2 and position B = right then
9:           segment 4c bind with adjacent shapes
10:      end if
11   end for
```

- **I-shape**

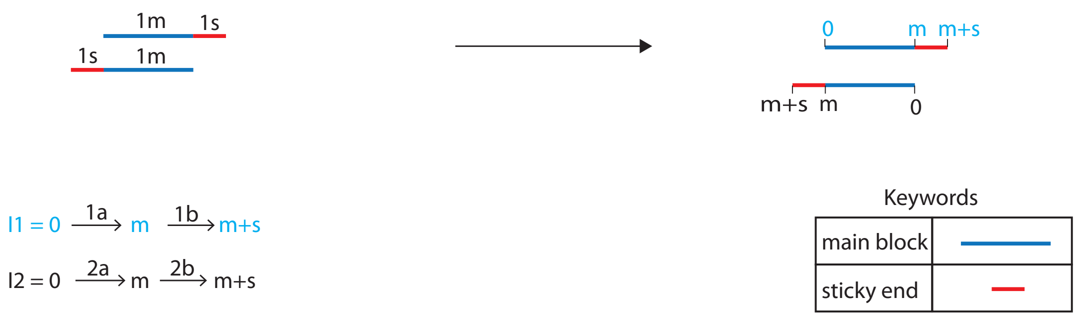

1m
1s
1s
1m
0
m
m+s
m+s
m
0
I1 = 0 $\xrightarrow{1a}$ m $\xrightarrow{1b}$ m+s
I2 = 0 $\xrightarrow{2a}$ m $\xrightarrow{2b}$ m+s
Keywords
main block
sticky end

**Figure 6.12:** The segmented positions in I-shape

The I-shape consists of two strands I1 (blue) and I2 (black). Each of the strands has two segments, with I1 (1a and 1b) and I2 (2a and 2b). For example in I1, segment 1a starts from position 0+1 to m; segment 1b starts from position m+1 to m+s. The 2 strands have segment 1a complement with 2a. Variable m is the length of the main block and s is the length of the sticky end. Position 0 indicates 5' terminal.

**Algorithm 14** Identification of sticky end in I-shape that bind with adjacent shape

```
1:  for block linked to adjacent shape do
2:      if intersection point = row 1 and position I = left then
3:          segment 1b bind with adjacent shapes
4:      else if intersection point = row 1 and position I = right then
5:          segment 2b bind with adjacent shapes
6:      end if
7:  end for
```

## 6.3 Architecture for the Predicted DNA Nanostructures

Figure 6.13 illustrates the sequence arrangements for DNA Tetris shapes in Set 1-Set 5 that conformed based on the 3 x 4 rectangle skeleton. However, the final DNA structures would not remain static as if they were being "glued" to a planar surface. Since every DNA Tetris shape was entirely made up of 4-way junctions except for the double-stranded DNA structure of I-Shape, the resulted DNA architectures were predicted to appear as in Figure 6.14. In this figure, the labelling such as CI1, T1, T2, W1 and W2 represent the names of the DNA strands that were used to form each DNA Tetris shape. For example, CT1-CT4 represent four DNA strands that were used to form

T-shape in Set 1. The predicted structures showed that Set 1, 2, 3 and 4 had similar configurations since each set was made up of 4-way junctions and 1 double-stranded DNA. Despite this, the size of the Set 1 was smaller compared to Set 2, 3, and 4. This was because Set 1 was made up of 25 nucleotides in each basic unit; while the remaining sets was made up of 40 nucleotides for their basic units.

On the other side, Set 5 had a different structural configuration compared to the remaining set. This was because all DNA Tetris shapes in Set 5 were entirely made up of 4-way junctions without the double stranded I-Shape.

Existing techniques focused on sequence diversity in their design phase (i.e., sequences that conformed to the scaffolds), to the contrary of the work presented here that introduced the combinatorics of the Tetris shapes into the equation. Therefore, this allowed diversity not only in sequences, but also in the Tetris shapes composition as well (i.e., many sequences to many shapes configurations that conformed to the desired structure). The DNA sequences generated for the five combinations (Set 1-Set 5) were listed in Table 6.5.

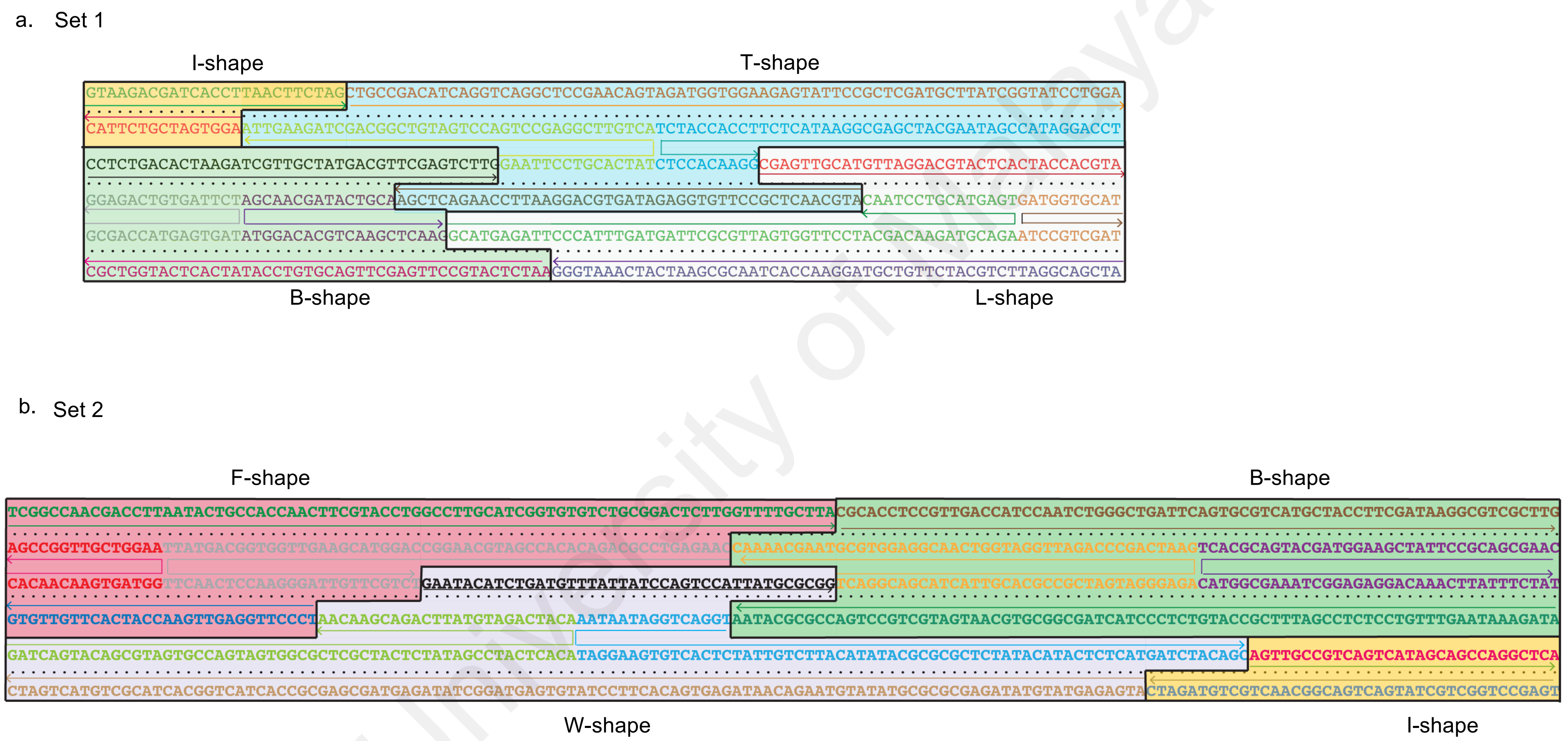
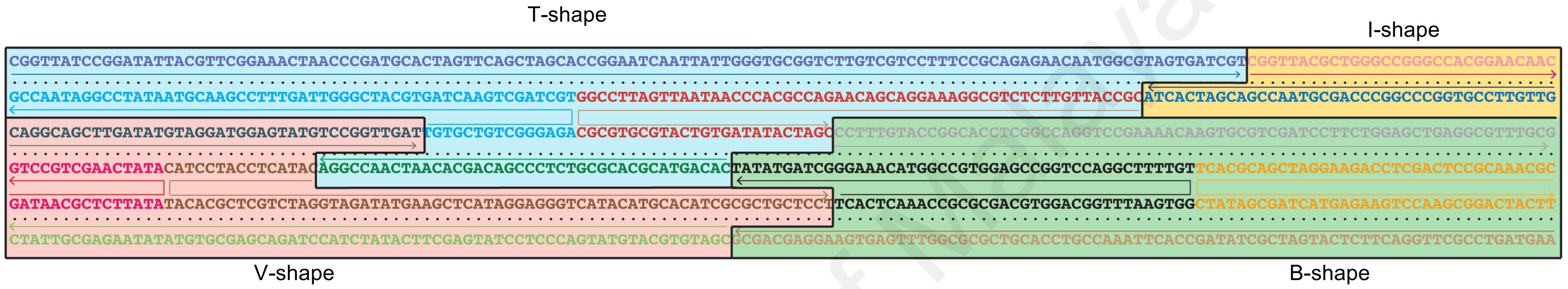


**Figure 6.13:** Nucleotide arrangements of the five combinations

Figure 6.13a-b represent the nucleotide arrangements of DNA Tetris shapes that bound according to 3 × 4 rectangle in Set 1 and Set 2. The arrows represent 5’ to 3’ terminal while the complementary binding is represented by dotted lines.

c. Set 3



d. Set 4

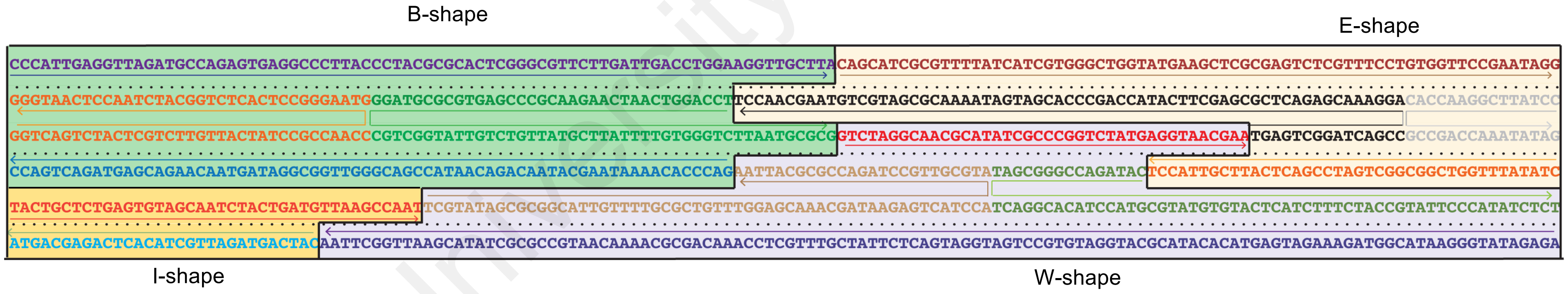


**Figure 6.13 (continued):** Nucleotide arrangements of the five combinations

Figure 6.13c-d represent the nucleotide arrangements of DNA Tetris shapes that bound according to 3 × 4 rectangle in Set 3 and Set 4. The arrows represent 5’ to 3’ terminal while the complementary binding is represented by dotted lines.

e. Set 5

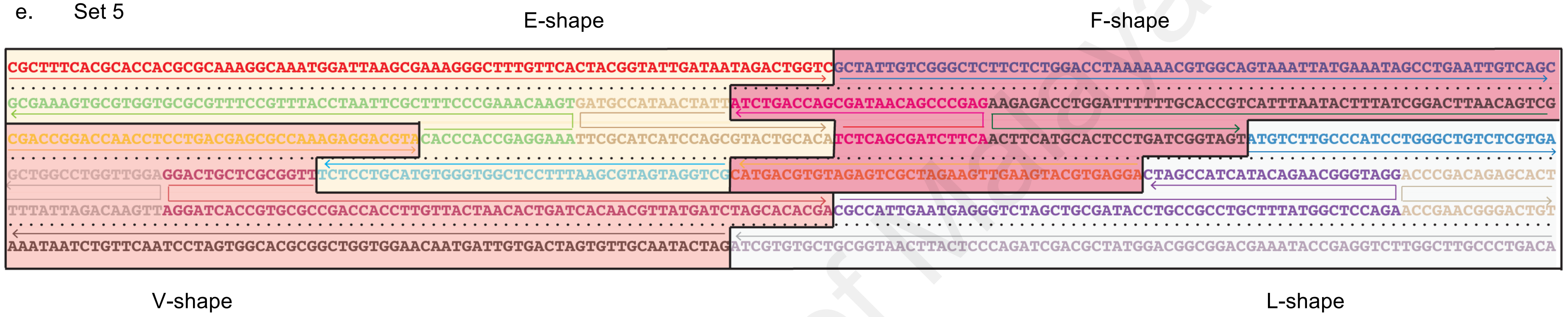

E-shape
F-shape
CGCTTTCACGCACCACGCGCAAAGGCAAATGGATTAAGCGAAAGGGCTTTGTTCACTACGGTATTGATAATAGACTGGTCGCTATTGTCGGGCTCTTCTCTGGACCTAAAAAACGTGGCAGTAAATTATGAAATAGCCTGAATTGTCAGC
GCGAAAGTGCGTGGTGCGCGTTTCCGTTTACCTAATTCGCTTTCCCGAAACAAGTGATGCCATAACTATTATCTGACCAGCGATAACAGCCCGAGAAGAGACCTGGATTTTTGCACCGTCATTTAATACTTTATCGGACTTAACAGTCG
CGACCGGACCAACCTCCTGACGAGCGCCAAAGAGGACGTACACCCACCGAGGAAATTCGCATCATCCAGCGTACTGCACATCTCAGCGATCTTCAACTTCATGCACTCCTGATCGGTAGTATGTCTTGCCCATCCTGGGCTGTCTCGTGA
GCTGGCCTGGTTGGAGGACTGCTCGCGGTTTCTCCTGCATGTGGGTGGCTCCTTTAAGCGTAGTAGGTCGCATGACGTGTAGAGTCGCTAGAAGTTGAAGTACGTGAGGACTAGCCATCATACAGAACGGGTAGGACCCGACAGAGCACT
TTTATTAGACAAGTTAGGATCACCGTGCGCCGACCACCTTGTTACTAACACTGATCACAACGTTATGATCTAGCACACGACGCCATTGAATGAGGGTCTAGCTGCGATACCTGCCGCCTGCTTTATGGCTCCAGAACCGAACGGGACTGT
AAATAATCTGTTCAATCCTAGTGGCACGCGGCTGGTGGAACAATGATTGTGACTAGTGTTGCAATACTAGATCGTGTGCTGCGGTAACTTACTCCCAGATCGACGCTATGGACGGCGGACGAAATACCGAGGTCTTGGCTTGCCCTGACA
V-shape
L-shape

**Figure 6.13 (continued):** Nucleotide arrangements of the five combinations
Figure 6.13e represent nucleotide arrangements of DNA Tetris shapes that bound according to 3 × 4 rectangle in Set 5. The arrows represent 5' to 3' terminal while the complementary binding is represented by dotted lines.

**Table 6.5:** DNA sequences of the five combinations

| Shape | Set 1 | DNA sequences (5' to 3') |
|---|---|---|
| I | CI1 | GTAAGACGATCACCTTAACTTCTAG |
|  | CI2 | AGGTGATCGTCTTAC |
| T | CT1 | CTGCCGACATCAGGTCAGGCTCCGAACAGTAGATGGTGGAAGAGTATTCCGCTCGATGCTTATCGGTATCCTGGA |
|  | CT2 | GAATTCCTGCACTATACTGTTCGGAGCCTGACCTGATGTCGGCAGCTAGAAGTTA |
|  | CT3 | TCCAGGATACCGATAAGCATCGAGCGGAATACTCTTCCACCATCTCTCCACAAGG |
|  | CT4 | ATGCAACTCGCCTTGTGGAGATAGTGCAGGAATTCCAAGACTCGA |
| B | CB1 | CCTCTGACACTAAGATCGTTGCTATGACGTTCGAGTCTTG |
|  | CB2 | GCGACCATGAGTGATTCTTAGTGTCAGAGG |
|  | CB3 | ACGTCATAGCAACGAATGGACACGTCAAGCTCAAG |
|  | CB4 | AATCTCATGCCTTGAGCTTGACGTGTCCATATCACTCATGGTCGC |
| L | CL1 | CGAGTTGCATGTTAGGACGTACTCACTACCACGTA |
|  | CL2 | GCATGAGATTCCCATTTGATGATTCGCGTTAGTGGTTCCTACGACAAGATGCAGATGAGTACGTCCTAAC |
|  | CL3 | TACGTGGTAGATCCGTCGAT |
|  | CL4 | ATCGACGGATTCTGCATCTTGTCGTAGGAACCACTAACGCGAATCATCAAATGGG |
|  |  |  |
| Shape | Set 2 | DNA sequences (5' to 3') |
| F | F1 | TCGGCCAACGACCTTAATACTGCCACCAACTTCGTACCTGGCCTTGCATCGGTGTGTCTGCGGACTCTTGGTTTTGCTTA |
|  | F2 | CACAACAAGTGATGGAAGGTCGTTGGCCGA |
|  | F3 | CAAGAGTCCGCAGACACACCGATGCAAGGCCAGGTACGAAGTTGGTGGCAGTATTTTCAACTCCAAGGGATTGTTCGTCT |
|  | F4 | TCCCTTGGAGTTGAACCATCACTTGTTGTG |
| B | B1 | CGCACCTCCGTTGACCATCCAATCTGGGCTGATTCAGTGCGTCATGCTACCTTCGATAAGGCGTCGCTTG |
|  | B2 | TCAGGCAGCATCATTGCACGCCGCTAGTAGGGAGAGAATCAGCCCAGATTGGATGGTCAACGGAGGTGCGTAAGCAAAAC |
|  | B3 | CAAGCGACGCCTTATCGAAGGTAGCATGACGCACTCATGGCGAAATCGGAGAGGACAAACTTATTTCTAT |
|  | B4 | ATAGAAATAAGTTTGTCCTCTCCGATTTCGCCATGTCTCCCTACTAGCGGCGTGCAATGATGCTGCCTGACCGCGCATAA |

**Table 6.5 (continued):** DNA sequences of the five combinations

| Shape | Set 2 | DNA sequences (5’ to 3’) |
|---|---|---|
| W | W1 | GAATACATCTGATGTTTATTATCCAGTCCATTATGCGCGG |
| | W2 | GATCAGTACAGCGTAGTGCCAGTAGTGGCGCTCGCTACTCTATAGCCTACTCACAACATCAGATGTATTCAGACGAACAA |
| | W3 | TGGACTGGATAATAATAGGAAGTGTCACTCTATTGTCTTACATATACGCGCGCTCTATACATACTCTCATGATCTACAGC |
| | W4 | ATGAGAGTATGTATAGAGCGCGCGTATATGTAAGACAATAGAGTGACACTTCCTATGTGAGTAGGCTATAGAGTAGCGAGCGCCACTA CTGGCACTACGCTGTACTGATC |
| I | I1 | AGTTGCCGTCAGTCATAGCAGCCAGGCTCA |
| | I2 | TGAGCCTGGCTGCTATGACTGACGGCAACTGCTGTAGATC |
| | | |
| Shape | Set 3 | DNA sequences (5’ to 3’) |
| T | T1 | CGGTTATCCGGATATTACGTTCGGAAACTAACCCGATGCACTAGTTCAGCTAGCACCGGAATCAATTATTGGGTGCGGTCTTGTCGTCC TTTCCGCAGAGAACAATGGCGTAGTGATCGT |
| | T2 | TGTGCTGTCGGGAGATGCTAGCTGAACTAGTGCATCGGGTTAGTTTCCGAACGTAATATCCGGATAACCG |
| | T3 | CGCCATTGTTCTCTGCGGAAAGGACGACAAGACCGCACCCAATAATTGATTCCGGCGCGTGCGTACTGTGATATACTAGC |
| | T4 | CACAGTACGCACGCGTCTCCCGACAGCACAATCAACCGGA |
| I | I1 | CGGTTACGCTGGGCCGGGCCACGGAACAAC |
| | I2 | GTTGTTCCGTGGCCCGGCCCAGCGTAACCGACGATCACTA |
| V | V1 | CAGGCAGCTTGATATGTAGGATGGAGTATGTCCGGTTGAT |
| | V2 | GATAACGCTCTTATAATATCAAGCTGCCTG |
| | V3 | CATACTCCATCCTACTACACGCTCGTCTAGGTAGATATGAAGCTCATAGGAGGGTCATACATGCACATCGCGCTGCTCCT |
| | V4 | CGATGTGCATGTATGACCCTCCTATGAGCTTCATATCTACCTAGACGAGCGTGTATATAAGAGCGTTATC |
| B | B1 | CCTTTGTACCGGCACCTCGGCCAGGTCCGAAAACAAGTGCGTCGATCCTTCTGGAGCTGAGGCGTTTGCG |
| | B2 | TCACTCAAACCGCGCGACGTGGACGGTTTAAGTGGTGTTTTCGGACCTGGCCGAGGTGCCGGTACAAAGGGCTAGTATAT |
| | B3 | CGCAAACGCCTCAGCTCCAGAAGGATCGACGCACTCTATAGCGATCATGAGAAGTCCAAGCGGACTACTT |
| | B4 | AAGTAGTCCGCTTGGACTTCTCATGATCGCTATAGCCACTTAAACCGTCCACGTCGCGCGGTTTGAGTGAAGGAGCAGCG |

**Table 6.5 (continued):** DNA sequences of the five combinations

| Shape | Set 4 | DNA sequences (5’ to 3’) |
|---|---|---|
| B | B1 | CCCATTGAGGTTAGATGCCAGAGTGAGGCCCTTACCCTACGCGCACTCGGGCGTTCTTGATTGACCTGGAAGGTTGCTTA |
| | B2 | GGTCAGTCTACTCGTCTTGTTACTATCCGCCAACCGTAAGGGCCTCACTCTGGCATCTAACCTCAATGGG |
| | B3 | TCCAGGTCAATCAAGAACGCCCGAGTGCGCGTAGGCGTCGGTATTGTCTGTTATGCTTATTTGTGGGTCTTAATGCGCG |
| | B4 | GACCCACAAAATAAGCATAACAGACAATACCGACGGGTTGGCGGATAGTAACAAGACGAGTAGACTGACC |
| E | E1 | CAGCATCGCGTTTTATCATCGTGGGCTGGTATGAAGCTCGCGAGTCTCGTTTCCTGTGGTTCCGAATAGG |
| | E2 | TGAGTCGGATCAGCCAGGAAACGAGACTCGCGAGCTTCATACCAGCCCACGATGATAAAACGCGATGCTGTAAGCAACCT |
| | E3 | CCTATTCGGAACCACGCCGACCAAATATAG |
| | E4 | CTATATTTGGTCGGCGGCTGATCCGACTCATTCGTTACCT |
| W | W1 | GTCTAGGCAACGCATATCGCCCGGTCTATGAGGTAACGAA |
| | W2 | TCGTATAGCGCGGCATTGTTTTGCGCTGTTTGGAGCAAACGATAAGAGTCATCCAATGCGTTGCCTAGACCGCGCATTAA |
| | W3 | CATAGACCGGGCGATTCAGGCACATCCATGCGTATGTGTACTCATCTTTCTACCGTATTCCCATATCTCT |
| | W4 | AGAGATATGGGAATACGGTAGAAAGATGAGTACACATACGCATGGATGTGCCTGATGGATGACTCTTATCGTTTGCTCCAAACAGCGC AAAACAATGCCGCGCTATACGAATTGGCTTAA |
| I | I1 | TACTGCTCTGAGTGTAGCAATCTACTGATGTTAAGCCAAT |
| | I2 | CATCAGTAGATTGCTACACTCAGAGCAGTA |
| | | |
| Shape | Set 5 | DNA sequences (5’ to 3’) |
| E | E1 | CGCTTTCACGCACCACGCGCAAAGGCAAATGGATTAAGCGAAAGGGCTTTGTTCACTACGGTATTGATAATAGACTGGTC |
| | E2 | CACCCACCGAGGAAATGAACAAAGCCCTTTCGCTTAATCCATTTGCCTTTGCGCGTGGTGCGTGAAAGCG |
| | E3 | TTATCAATACCGTAGTTCGCATCATCCAGCGTACTGCACA |
| | E4 | GCTGGATGATGCGAATTTCCTCGGTGGGTGTACGTCCTCT |
| F | F1 | GCTATTGTCGGGCTCTTCTCTGGACCTAAAAAACGTGGCAGTAAATTATGAAATAGCCTGAATTGTCAGC |
| | F2 | TCTCAGCGATCTTCAGAGCCCGACAATAGCGACCAGTCTA |

**Table 6.5 (continued):** DNA sequences of the five combinations

| Shape | Set 5 | DNA sequences (5’ to 3’) |
|---|---|---|
| F | F3 | GCTGACAATTCAGGCTATTTCATAATTTACTGCCACGTTTTTAGGTCCAGAGAAACTTCATGCACTCCTGATCGGTAGT |
|  | F4 | AGGAGTGCATGAAGTTGAAGATCGCTGAGATGTGCAGTAC |
| V | V1 | CGACCGGACCAACCTCCTGACGAGCGCCAAAGAGGACGTA |
|  | V2 | TTTATTAGACAAGTTAGGTTGGTCCGGTCG |
|  | V3 | TTGGCGCTCGTCAGGAGGATCACCGTGCGCCGACCACCTTGTTACTAACACTGATCACAACGTTATGATCTAGCACACGA |
|  | V4 | GATCATAACGTTGTGATCAGTGTTAGTAACAAGGTGGTCGGCGCACGGTGATCCTAACTTGTCTAATAAA |
| L | L1 | ATGTCTTGCCCATCCTGGGCTGTCTCGTGA |
|  | L2 | CGCCATTGAATGAGGGTCTAGCTGCGATACCTGCCGCCTGCTTTATGGCTCCAGAGGATGGGCAAGACATACTACCGATC |
|  | L3 | TCACGAGACAGCCCAACCGAACGGGACTGT |
|  | L4 | ACAGTCCCGTTCGGTTCTGGAGCCATAAAGCAGGCGGCAGGTATCGCAGCTAGACCCTCATTCAATGGCGTCGTGTGCTA |

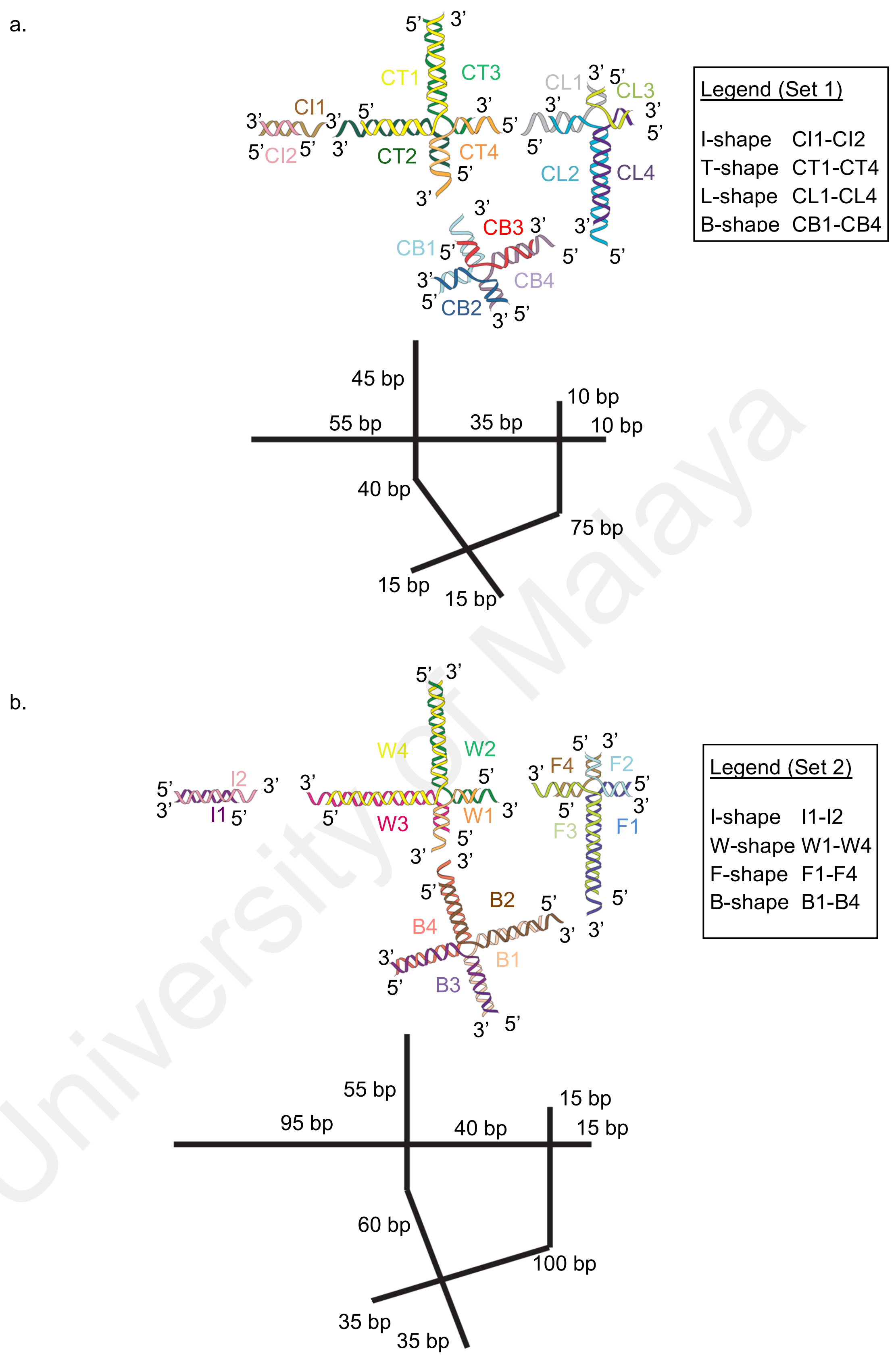
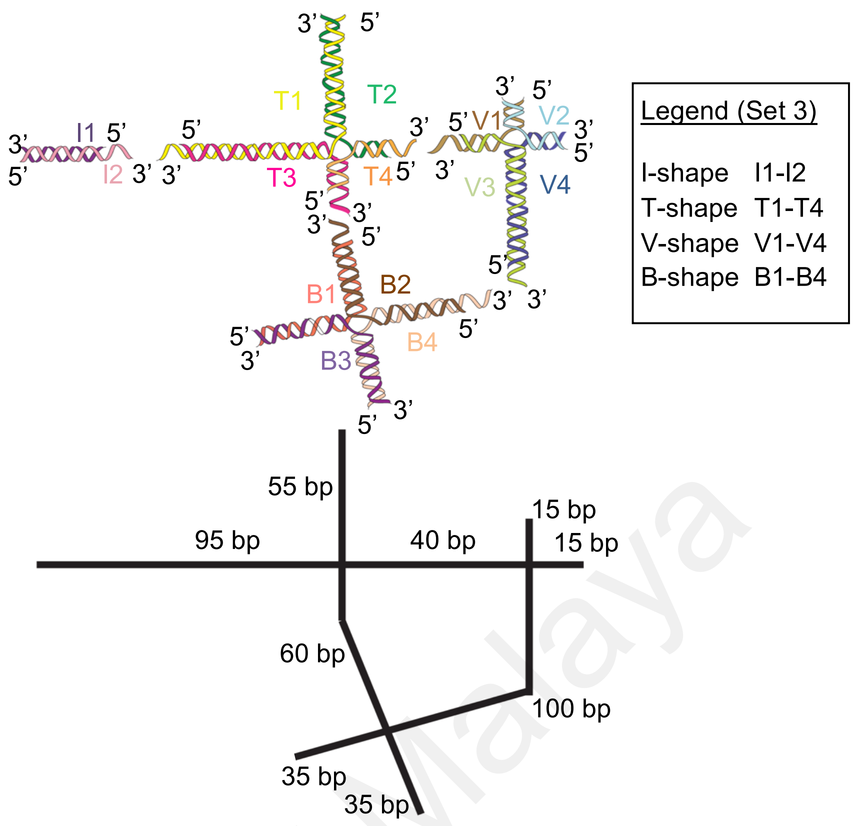


**Figure 6.14:** The predicted structure representation of the five combinations
Figure 6.14a-b represent the DNA Tetris shapes that combined according to 3 x 4 rectangles in Set 1 and Set 2. The numbers indicated on the black line structures in each set was the length of the DNA sequences measured in base pairing (for example, 40 bp denotes 40 base pairing).

c.



d.

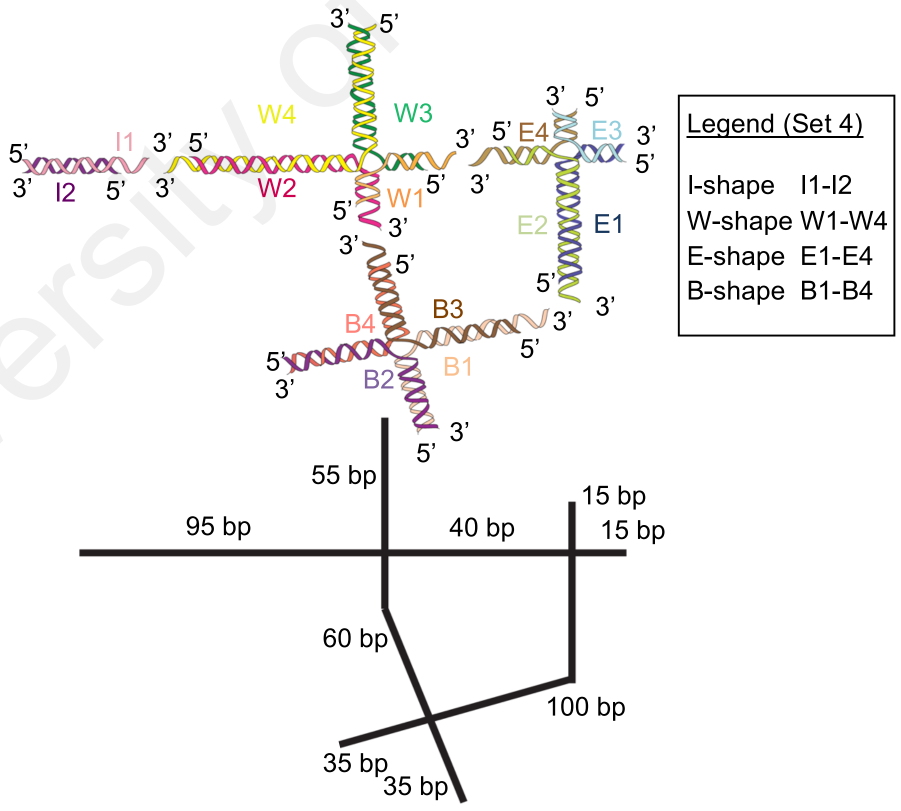
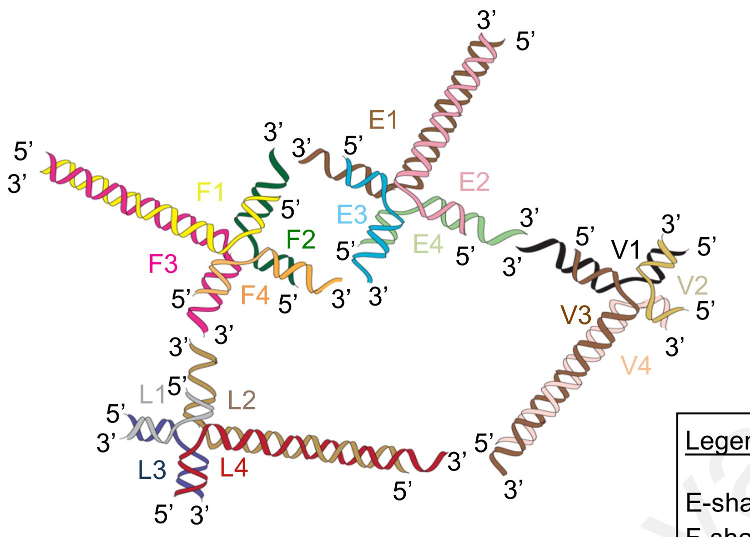
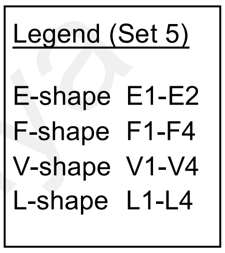


**Figure 6.14 (continued):** The predicted structure representation of the five combinations

Figure 6.14c-d represent the DNA Tetris shapes that combined according to the 3 × 4 rectangles in Set 3 and Set 4. The numbers indicated on the black line structures in each set was the length of the DNA sequences measured in base pairing (For example, 40 bp denotes 40 base pairing).

d.





Legend (Set 5)

| | |
|---|---|
| E-shape | E1-E2 |
| F-shape | F1-F4 |
| V-shape | V1-V4 |
| L-shape | L1-L4 |

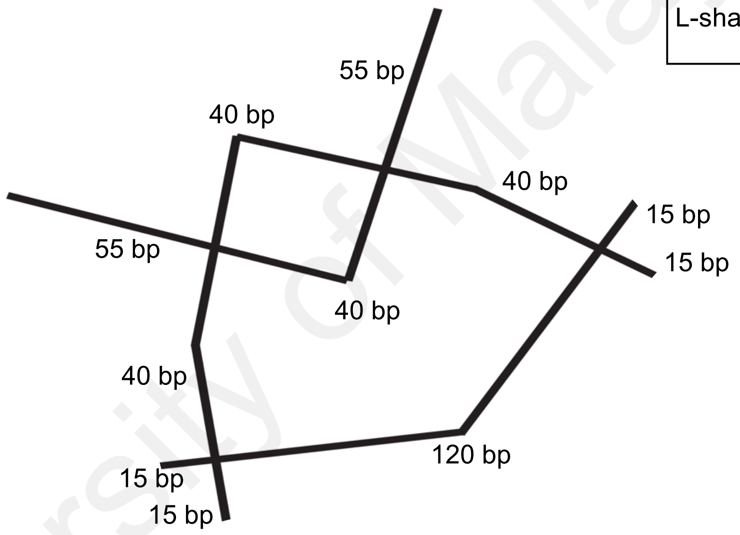


**Figure 6.14 (continued):** The predicted structure representation of the five combinations

Figure 6.14e represent the DNA Tetris shapes that combined according to the 3 × 4 rectangles in Set 5. The numbers indicated on the black line structures in each set was the length of the DNA sequences measured in base pairing (For example, 40 bp denotes 40 base pairing).

## 6.4 Experimental Validation Protocol

### 6.4.1 DNA Annealing

As mentioned in the previous chapter, DNA annealing is a process for single stranded DNA/RNA to hybridize through hydrogen bond and forms double stranded polynucleotides. DNA sequences (Oligonucleotides) were purchased from Integrated DNA Technologies Pte. Ltd. (USA). The complexes were formed by mixing stoichiometric quantities of DNA in an annealing buffer (40 mM Tris base, 2.5 mM EDTA, and 13 mM $MgCl_2$) and the annealing process from 90 °C to 40 °C for three hours using an Eppendorf Mastercycler Pro S thermocycler (Eppendorf, Hamburg, Germany). To form individual shape of DNA tetrominoes, four different oligonucleotides were mixed stoichiometrically in an annealing buffer and the final concentration was set to 0.5 μM.

### 6.4.2 Gel Electrophoresis

The results of the annealing reactions were analyzed using non-denaturing gel electrophoresis containing 4%, 5% and 8% polyacrylamide gel (29:1 acrylamide:bisacrylamide), 0.75 mm thick and run at approximately 12V/cm-1 for 2 hours at 4 °C. The running buffer contained 10 mM $MgCl_2$ and 1X TBE (89 mM Tris base, 89 mM Boric acid and 2 mM EDTA pH8.3) and the loading buffer contained 0.25% Bromophenol blue tracking dye and 30% glycerol. GelRed$^{TM}$ Nucleic Acid gel stain (Biotium, US) was used to stain the gel. As mentioned in the previous chapter, gel electrophoresis is a method to separate biological molecules (DNA, RNA and proteins) based on their size and charge. It uses electric current to separate DNA fragments by size as they migrate through the gel matrix (e.g. polyacrylamide gel). Smaller size DNA move further away through the gel pores than the larger size DNA.

### 6.4.3 Sample Preparation and Atomic force microscopy (AFM) Imaging

Atomic force microscopy (AFM) is a high-resolution microscopy that is used to capture images of DNA structures at nanometer scale (Andersen et al., 2008; Brown et al., 2015; Marchi, Saaem, Vogen, Brown, & LaBean, 2014; Zhang et al., 2015). All DNA samples from Set 1 - Set 5 were subjected to a few preparation steps before the images of the nanostructures could be captured.

**Preparation of Mica Surface:** A 0.1% APTES ((3-aminopropyl) triethoxysilane) solution was prepared in ultrapure water. A drop (2 µL) of 0.1% APTES solution was deposited onto the freshly cleaved mica surface and the surface was rinsed with ultrapure water (20 µL) after 5 minutes incubation at room temperature.

**Sample Preparation for AFM Imaging:** The samples were diluted to 0.2 ng/µL with buffer (40 mM Tris-HCl (pH 7.6), 13 mM $MgCl_2$, 2.5 mM EDTA). 2 µL of the sample solution was placed onto the APTES-treated mica surface for 5 minutes and the surface was later rinsed with the buffer (20 µL) to remove unbound molecules.

**Atomic Force Microscopy (AFM) Imaging:** The AFM images were captured using high-speed AFM (Nano Live Vision, Research Institute of Biomolecules Metrology Co., Tsukuba, Japan) and cantilevers with dimensions (L×W×H) 10×2×0.1 $\mu m^3$ (BL-AC10EGS, Olympus Corporation, Tokyo, Japan) were used. The cantilevers had a spring constant of 0.1-0.2 N/m with a resonant frequency of 400 - 1,000 kHz in water. The 320 · 240 pixels images were collected in tapping mode.

## 6.5 Analysis of Gel Electrophoresis Result and AFM Images

DNA sequences for each shape were added sequentially during the gel electrophoresis procedure (Figure 6.15). In the well 6 and 7 of Set 4 (Figure 6.15d), there was no significant increment of the band size when I-shape was added to the well 7. This may due to the lower molecular size of the two DNA sequences that made up the I-shape (I1=40 nucleotides and I2=30 nucleotides). However, when E-shape was added into the well 8, smear was observed. This may due to the existent of many random pairing, since DNA connectivity tool showed that it (Set 4) had the highest competition between desired and undesired base pairing based on the list of binding affinities (will be describe in details under section 6.6).

Although gel electrophoresis was used to detect the structural formation based on the band size increment, it was not sufficient to elucidate the formation of the complete DNA nanostructures. Therefore, AFM images of the structure were crucial to ascertain the successful structural formation. Comparison between the AFM images (Figure 6.16) and the predicted designed structures (Figure 6.14) were conducted. This led us to believe that the formations of DNAs that resembled the designed structures had successfully been observed for all the five combinations (Set 1 - Set 5).

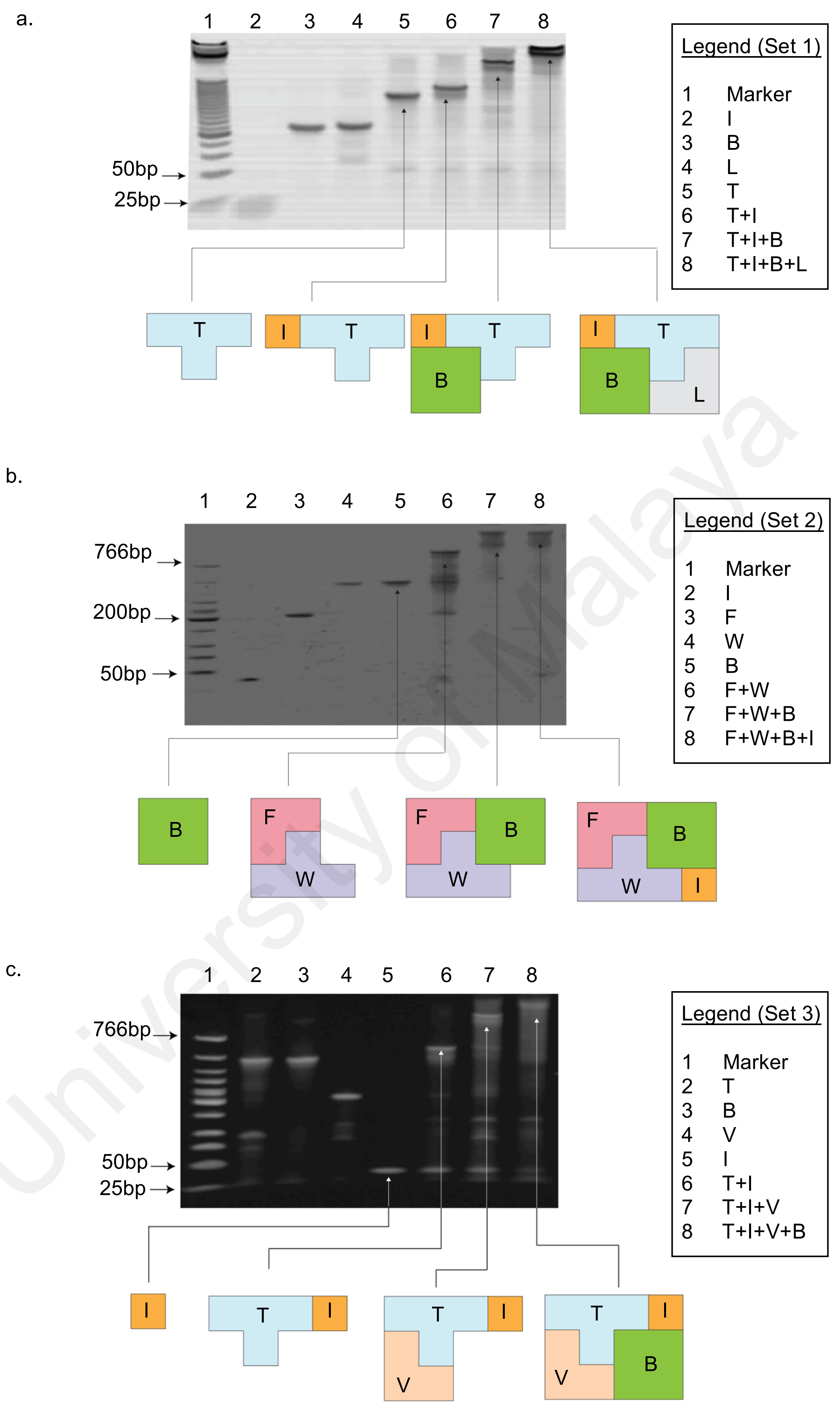
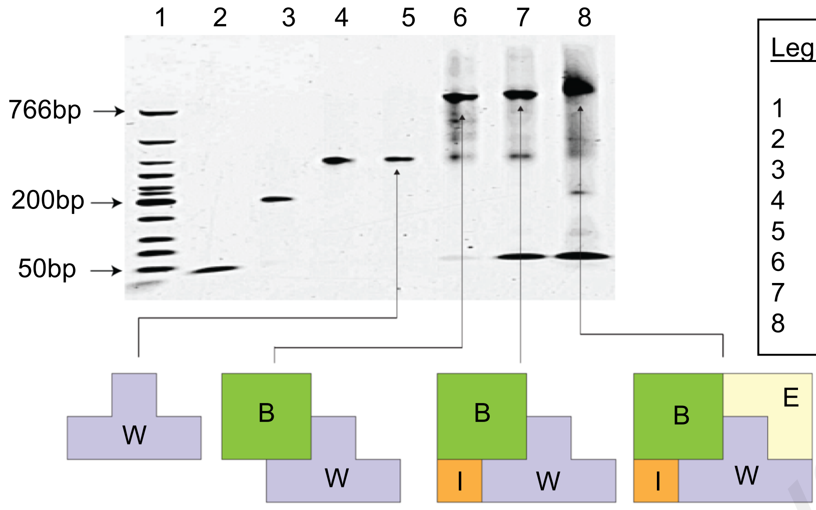


**Figure 6.15:** Gel electrophoresis results of the five combinations
Figure 6.15a represents the gel electrophoresis of Set 1 conducted on 8% non-denaturing PAGE gel. Figure 6.15b-c represent the gel electrophoresis of Set 2 and Set 3 conducted on 5% non-denaturing PAGE gel.

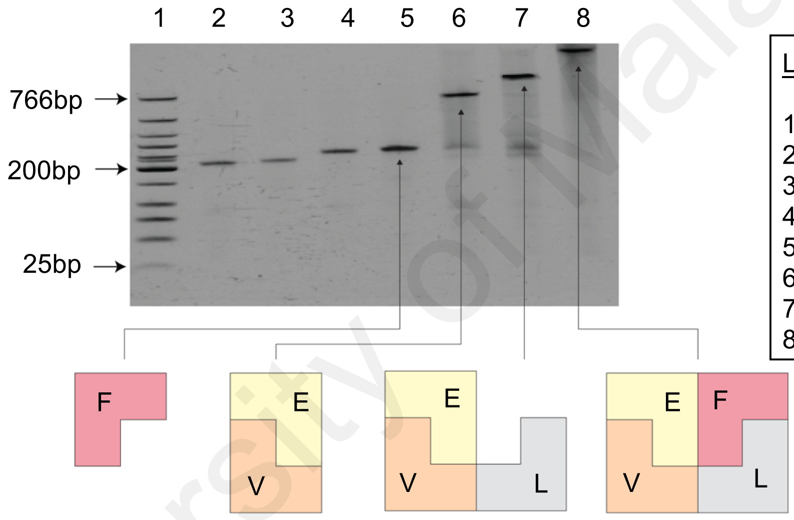
d.



Legend (Set 4)

| | |
|---|---|
| 1 | Marker |
| 2 | I |
| 3 | E |
| 4 | B |
| 5 | W |
| 6 | B+W |
| 7 | B+W+I |
| 8 | B+W+I+E |

e.

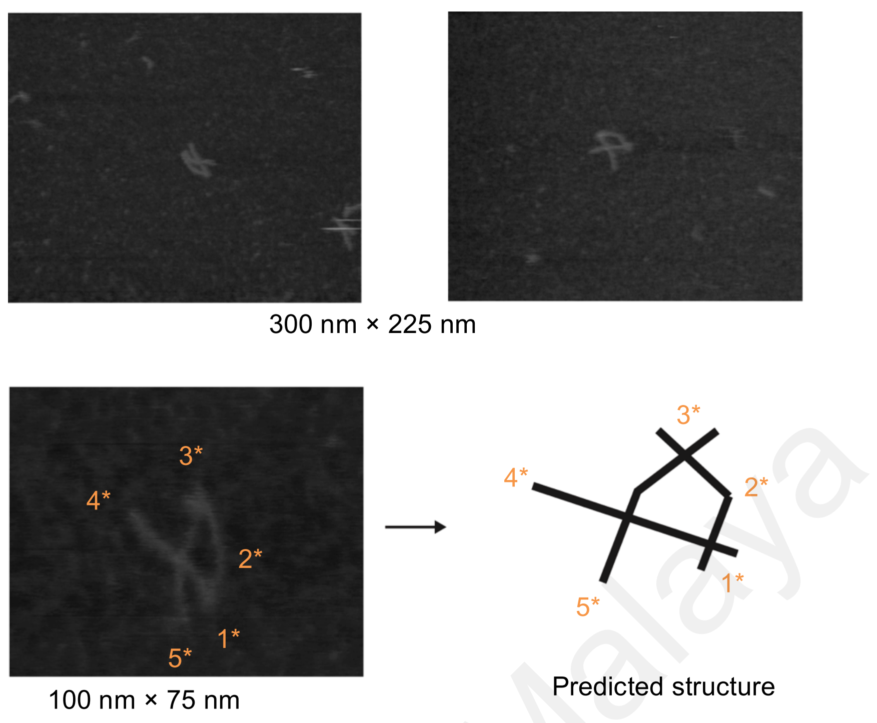


Legend (Set 5)

| | |
|---|---|
| 1 | Marker |
| 2 | V |
| 3 | L |
| 4 | E |
| 5 | F |
| 6 | E+V |
| 7 | E+V+L |
| 8 | E+V+L+F |

**Figure 6.15 (continued):** Gel electrophoresis results of the five combinations
Figure 6.15d-e represent the gel electrophoresis results of Set 4 and Set 5 conducted on 4% and 5% non-denaturing PAGE gel.

a. Set 1



b. Set 2

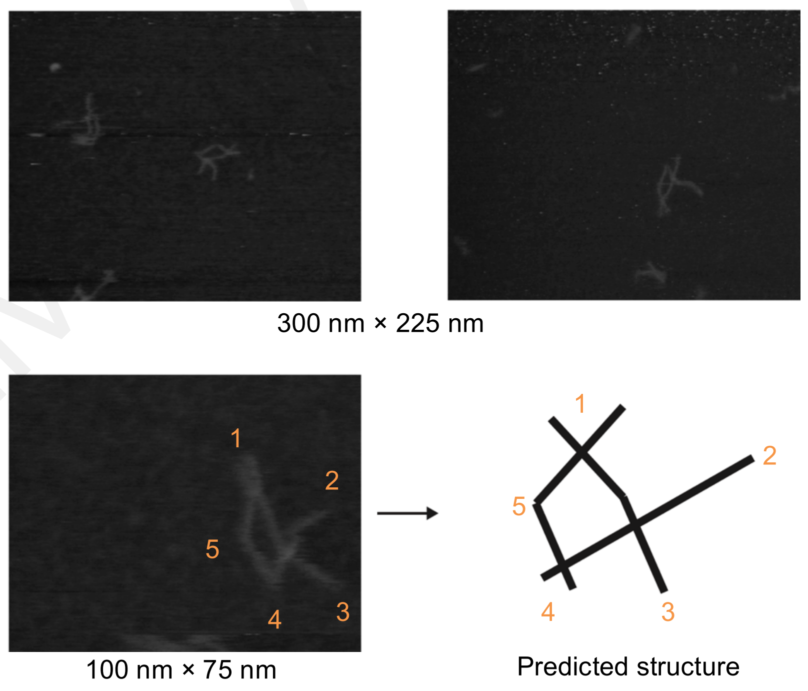
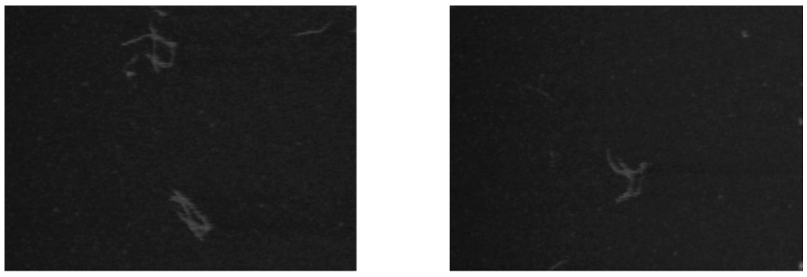
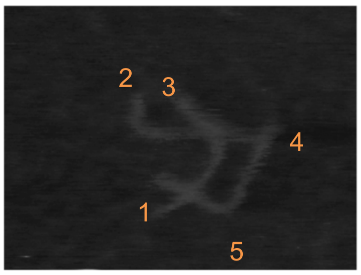


**Figure 6.16:** AFM images of the five combinations
Figure 6.16a represents Set 1 while Figure 6.16b represents Set 2. The images were taken at 300 nm × 225 nm and 100 nm × 75 nm. They were compared to the predicted structures and each region was labelled with numerical number.

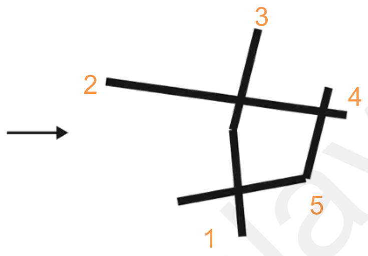
c. Set 3



300 nm x 225 nm

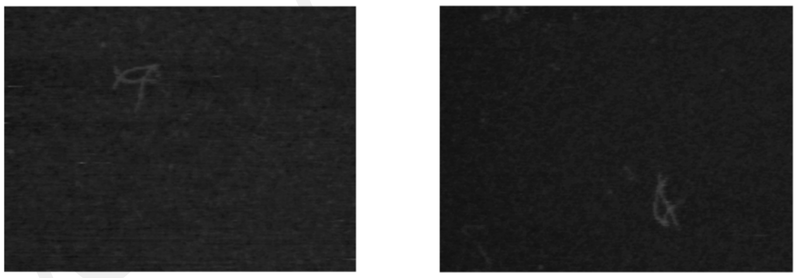


100 nm x 75 nm

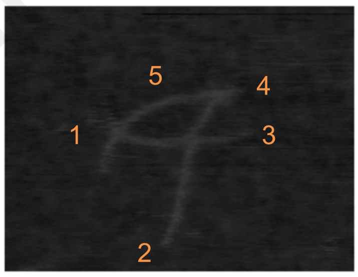


Predicted structure

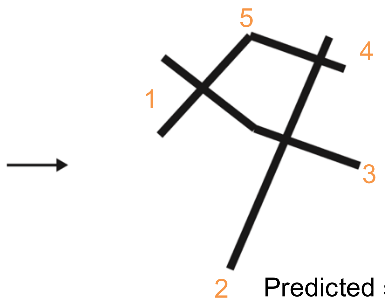
d. Set 4



300 nm × 225 nm



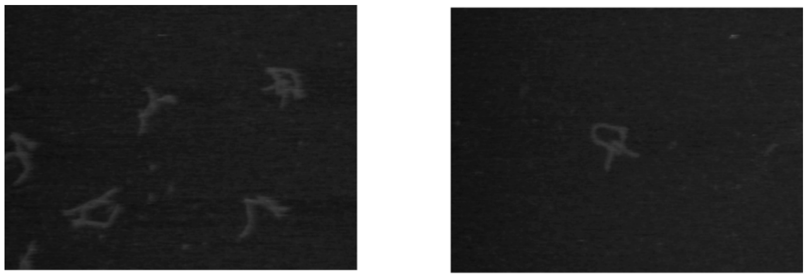
100 nm × 75 nm



Predicted structure

**Figure 6.16 (continued):** AFM images of the five combinations
Figure 6.16c represents Set 3 while Figure 6.16d represents Set 4. The images were taken at 300 nm × 225 nm and 100 nm × 75 nm. They were compared to the predicted structures and each region was labelled with numerical number.

e. Set 5



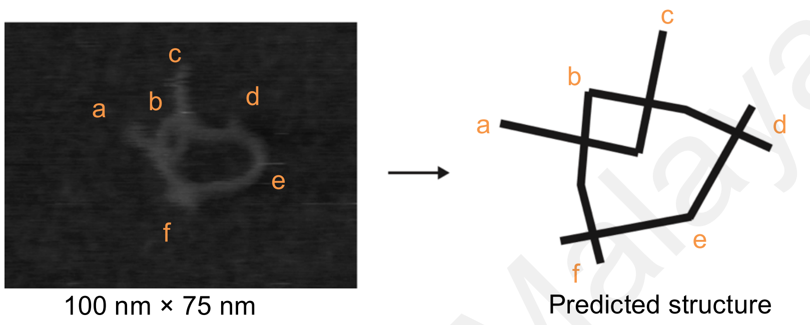


**Figure 6.16 (continued):** AFM images of the five combinations
Figure 6.16e represents Set 5. The images were taken at 300 nm × 225 nm and 100 nm × 75 nm. They were compared to the predicted structures and each region was labelled with alphabetical letter.

## 6.6 DNA Connectivity Map

The DNA connectivity tool developed in prior chapter was used to study the interactions between the DNA through the binding affinity matrix. This had resulted in the generation of graphs and the relative binding affinity between different nodes or DNA segments (Table 6.6). The number of graphs was equivalent to the number of potential structures that could be generated from a set of DNA strands. This number included both the desired and misfolded structures. For example, Set 5 produced 31 different graphs with only 21 graphs indicating the formation of the desired structure. Thus, there were 10 misleading paths that were biased towards unfavourable folding leading to the formation of mismatch structures. The number of occurrences for binding affinity close to 1.0 indicated the level of competition between the unintended nodes (i.e., not design to form base pair). The higher the competition, the more number of graphs would be produced. Our search revealed that Set 4 had the highest number of graphs generated, followed by Set 3, 2, 1 and 5 respectively. This was due to the higher number of binding affinities that had value near to 1.0 existed in the sets. Set 4 had the highest competition based on the list of binding affinities and this result was consistent with the fact that the existence of smear on well 8 gel electrophoresis.

**Table 6.6:** Summary of the graphs generated through the searches

| Combinations | Number of correct graphs | Number of graphs | Binding affinity |
|---|---|---|---|
| Set 1 | 17 | 200 | 0.74, 0.76 |
| Set 2 | 16 | 469 | 0.71, 0.72, 0.75, 0.77 |
| Set 3 | 16 | 605 | 0.72, 0.72, 0.72, 0.73, 0.75 |
| Set 4 | 12 | 757 | 0.72, 0.72, 0.72, 0.73, 0.77, 0.79, 0.84 |
| Set 5 | 21 | 31 | 0.73 |

The value $P_{i,j}$ represents the relative binding affinity between each DNA segment estimated using the free energy from program *Duplexfold* (Reuter & Mathews, 2010). $P_{i,j}$ will have the value of 1.0, if the intended binding between nodes is a perfect complementary pair. In the calculation, partially complement ($P_{i,j}$ < 1.0) DNA segments are still included. However these partially complement segments have the tendency to create false routes (causing the emergence of sticky ends) and eventually resulted in false structures or miscellaneous aggregates. The threshold for binding affinity was set at 0.7, therefore $P_{i,j}$ > 0.7 will be included. This is to ensure that the graph (Figure 6.17) is restricted to only display strong estimation values (i.e., representative of preferable binding interactions). Lower assignment of threshold generates convoluted paths full of weak interactions, which will complicate the search process. The correct graphs are represented with all the nodes visited exactly once and the edges taken by each node are correctly linked as designed, regardless of the starting points. The order of the completed routes will provide a blueprint for the DNA sequences to form the desired structures.

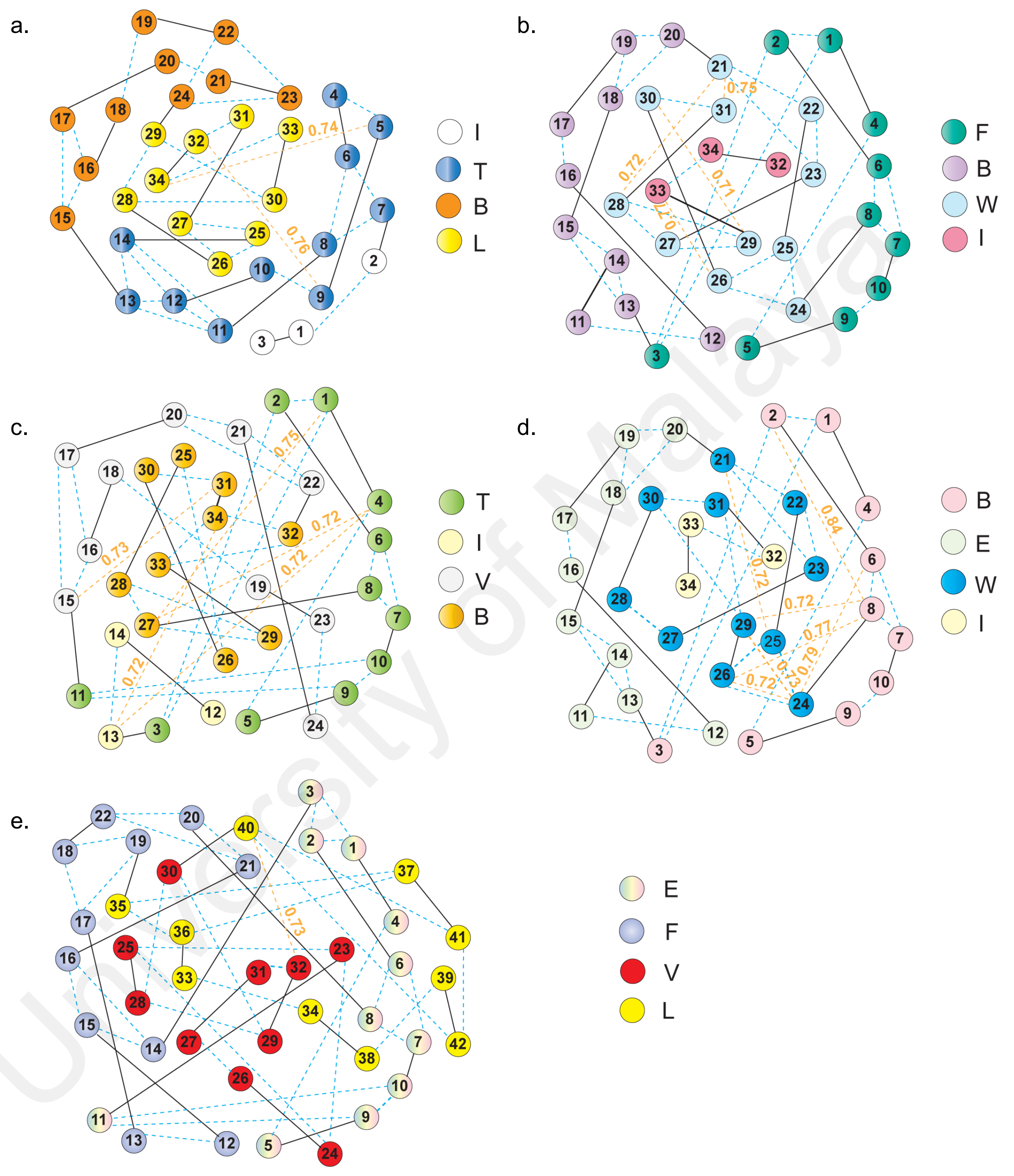


**Figure 6.17:** The connectivity map for the five combinations
Figure 6.17a-e represent connectivity map for Set 1, Set 2, Set 3, Set 4 and Set 5. Black lines indicate the binding affinity between the respective nodes, whi ch was equals to 1.0. Blue dashed lines indicate nodes that were derived from the same DNA strands, which were then used to decide on the emergence of potential sticky ends binding region. Orange lines reveal the nodes with the binding affinity value of $0.7 < P_{i,j} < 1.0$. The colour legends represent the type of DNA Tetris shapes involved in the configuration of the rectangles.

## 6.7 Summary

This chapter presented the successful construction of five distinct conformations (Set 1 to Set 5) of DNA nanostructure to address the ability of DNA sequences to self-organize themselves into predefined configurations. The DNA sequences for these combinations were designed using our newly developed sequence design tool while the study of the interactions between the generated DNAs was done using the developed DNA connectivity tool. The attempt for the structural construction was deemed successful following the formations of DNAs that resembled the designed structures been successfully observed for all five combinations (Set 1- Set 5).

# CHAPTER 7: CONCLUSION

The preceding chapters began by introducing the hypotheses of implementing a new schema to construct DNA nanostructures, which derived from the principle of self-organization and the popular computer game, Tetris. It then continued with the literature reviews on the research subjects and previous works before commencing with the development of the computational tools to support the proposed schema. Finally, this chapter presents the summary of the entire research work and further concludes the work. Several recommendations for future work on this research topic are also included in the last section of this chapter.

## 7.1 Research Summary

This research proposed a new schema for constructing self-assembled DNA nanostructures to create a dynamic and flexible structure design method. It further promotes a complete outlook of the shapes involved and the sequence landscape necessary in designing DNA sequences. Although there are a number of computational tools (Andronescu, Aguirre-Hernández, Condon, & Hoos, 2003; Hofacker, 2003; Reuter & Mathews, 2010) available to do structure prediction and analysis, the application of these tools are broad and they are not specifically developed to support the objectives of this DNA Tetrominoes concept. In order to achieve this, it is crucial to utilize the principles in computer science to compute the algorithm and manipulate it to do prediction, simplify the search space and finally to find solutions to build the targeted structure.

It is also not feasible to build DNA nanostructures manually from scratch since it involves large search space and combinatorics issue. This has eventually led to the development of computational methods to resolve this issue. Therefore, three main research objectives have been outlined, i.e. to propose a new schema that simplifies the design search space, to develop an autonomous tool that support the schema and to explore the plausibility of implementing self-organization principle in DNA nanostructure fabrication.

To resolve the first research objective, the use of heterogeneous DNA Tetris shapes constructed based on the Tetris game shape was introduced. Eight Tetris shapes that could be used to derive from DNA sequences were identified. The DNA nanostructures acted as search space for these shapes to self-assemble. These shapes were constructed by allowing the process of block stacking and merging to form four long continuous DNA sequences (except two DNA sequences for I-shape). The used of heterogeneous Tetris shapes and the dynamicity in the search space based on the Tetris board, allowed multiple conformations to be derived.

Secondly, after proposing the DNA Tetrominoes concept, the next step was to develop a computational method for structural construction that was used to support the newly introduced schema (objective 2). The computational tool comprised of two modules; module 1 was the sequence design tool and the module 2 was the DNA connectivity tool. These tools included the incorporation of optimization algorithm with fitness evaluation criteria to mutate DNA so that it could autonomously form the prescribed structures. Three external programs (*RNAstructure* (Reuter & Mathews, 2010), *UNAFold* (Markham & Zuker, 2008) and *DNA sequence generator* (Feldkamp, Saghafi, & Rauhe, 2001)) were also subsequently manipulated to aid in the development of the computational tools. As DNA self-assembly process is asynchronous by which correctly formed DNA strands will compete with the partially correct DNA strands, it was crucial to map the interaction that occurred between a set of DNA when forming the DNA nanostructures (Module 2). The connectivity map was used to analyse and map the level of competition between each DNA prior to forming the end-structures. It was used to indicate the relative binding strength between DNA sequences, so that sequence alteration could be used to minimize the ill-formed structures. DNA sequences that formed the structure were subsequently being analyzed using graph method. The analysis had deciphered the interactions between DNA sequences through the calculation of binding affinity of five different sets of combinations. As such, Set 4 had the highest degree of competition to form the end structure, followed by Set 3, 2, 1 and 5.

The third objective focused on exploring the plausibility to uphold the principle of self-organization during the structural design. To implement this, each piece of Tetris shape was mutually exclusive and considered as an entity. Each DNA Tetris shape would self-assemble to form structures once the sticky end of its shape met with its complementary sticky end from another shape. It enabled each component to be included, excluded or replaced without affecting the entire structures and the resulted components were modular. Meanwhile, in order to proof the ability of DNA Tetrominoes concept to address the many shapes to many combinations relationship, a hypothetical application with search space of 3 rows and 4 columns had been conducted. A total of five distinct combinations (Set 1 – Set 5) had been utilized and the successful formation of these DNA nanostructures had been validated using Atomic Force Microscopy (AFM) imaging.

This work started with the goal to promote an alternative yet simpler schema to aid in the construction of DNA nanostructures. In this dissertation, the contributions that have been proposed and developed: A simpler schema to support the feasibility of integrating Tetris representation into constructing DNA nanostructures. This schema created modular DNA Tetris shapes by which each shape was considered as an entity. These shapes were mutually exclusive and independent from the overall structural design. The shapes were programmed in the ways that they could bind in different type of combinations when they met with matching sticky ends. Tetris representation was chosen since it had heterogeneous property to show that different configurations (many shapes to many combinations) could be used to form the same nanostructure. The core principle of having multiple configurations was to let the most preferred shape and sequence combinations to take precedence and in the event when any set of the structure collapsed, the remaining sets could still be formed. Therefore, this concept was used to address the stability issue od the resulting nanostructures against degradations (Mei et al., 2011; Shen et al., 2012).

The development of the computational tools was used to support the dynamic structure design method. The tools were used to introduce the combinatorics of Tetris shapes into the equation. This allowed diversity not only in sequences, but also in the Tetris shapes composition as well (i.e., many sequences to many shapes configurations that conformed to the desired structure).

Despite the development of computational methods to uphold the research objectives, there are several limitations in this study. For instance, this work does not include the quantification on the total number of DNA nanostructures that are correctly and incorrectly formed. The quantity of the well-formed structures is speculated to be relatively lower than the conventional approach due to the use of less stringent parameters. Even though the correct structures are still formed (images of these nanostructures were successfully captured using AFM), it is speculated to have more ill-formed structures using this approach.

In addition, the sequence design method introduced the existence of forbidden region in the DNA sequences. This will limit the total number of nucleotide positions available for mutation. The forbidden region in the DNA sequences is the region that is refrained from mutations process. Hence, the longer the length of the forbidden region, the less nucleotide positions that are available for mutation.

Aside from this, the hypothetical application that was presented here has the search space designed to be planar (being “glued” to a board). However, it is worth to note that the resulted DNA nanostructures would not remain in planar arrangement and therefore the images observed under AFM will vary. This shows the need to have a proper structural interpretation so that the desired nanostructure end products could be converted into the planar search space.

## 7.2 Conversion of the Framework into Web Service

The website, DNATetris was developed using Adobe Dreamweaver CS5 and involved the setting up of a local testing server using MAMP 3.5 software (Mac OS X, Apache, MySQL, PHP) (Figure 7.1). The server-side scripting language used PHP version 5.6.10. Since the scripts were written in Perl and Tool Command Language (TCL), the local server was required to have both packages installed. The address used to access the website is http://localhost:8888/index.html. Safari, Google Chrome and Firefox browsers have been used during the server-testing phase.

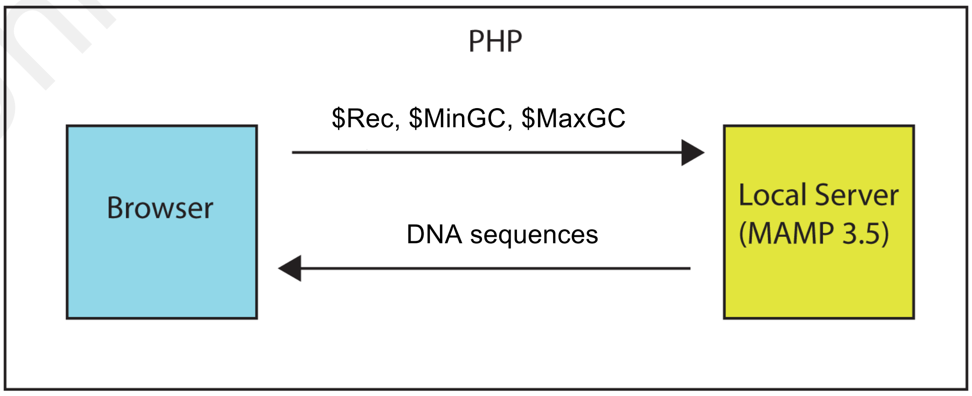
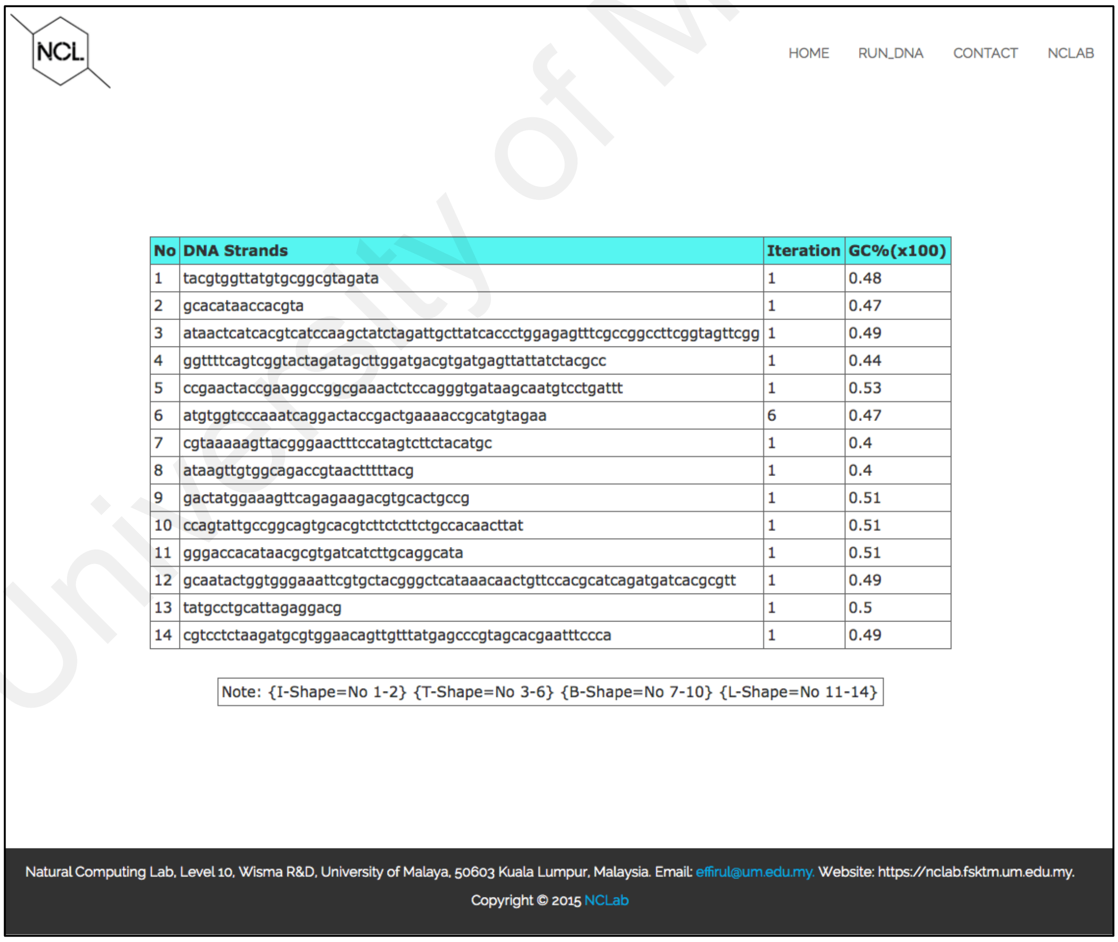


**Figure 7.1:** The flow between the browser and local server

DNATetris takes a set of random DNA sequences and further optimizes the DNA so that it can form the end configuration according to the 3 x 4 rectangle framework. The website receives user parameters such as percentage of guanine-cytosine content of the DNA and the type of configurations. At one time, it generates one set of sequences based on the type of configuration that the user selected, which will eventually generate 14 DNA strands for Set 1, Set 2, Set 3 and Set 4 or 16 DNA strands for Set 5 conformations. Upon clicking on the Submit button, the webpage will send the user specified parameters to index2.php (Appendix R), which will then parse the parameters to the Main_RunDNATetris.tcl. It then produces the output page as illustrated in Figure 7.2.



| No | DNA Strands | Iteration | GC%(x100) |
|---|---|---|---|
| 1 | tacgtggttatgtgcggcgtagata | 1 | 0.48 |
| 2 | gcacataaccacgta | 1 | 0.47 |
| 3 | ataactcatcacgtcatccaagctatctagattgcttatcaccctggagagtttcgccggccttcggtagttcgg | 1 | 0.49 |
| 4 | ggttttcagtcggtactagatagcttggatgacgtgatgagttattatctacgcc | 1 | 0.44 |
| 5 | ccgaactaccgaaggccggcgaaactctccagggtgataagcaatgtcctgattt | 1 | 0.53 |
| 6 | atgtggtcccaaatcaggactaccgactgaaaaccgcatgtagaa | 6 | 0.47 |
| 7 | cgtaaaaagttacgggaactttccatagtcttctacatgc | 1 | 0.4 |
| 8 | ataagttgtggcagaccgtaactttttacg | 1 | 0.4 |
| 9 | gactatggaaagttcagagaagacgtgcactgccg | 1 | 0.51 |
| 10 | ccagtattgccggcagtgcacgtcttctcttctgccacaacttat | 1 | 0.51 |
| 11 | gggaccacataacgcgtgatcatcttgcaggcata | 1 | 0.51 |
| 12 | gcaatactggtgggaaattcgtgctacgggctcataaacaactgttccacgcatcagatgatcacgcgtt | 1 | 0.49 |
| 13 | tatgcctgcattagaggacg | 1 | 0.5 |
| 14 | cgtcctctaagatgcgtggaacagttgtttatgagcccgtagcacgaatttccca | 1 | 0.49 |

Note: {I-Shape=No 1-2} {T-Shape=No 3-6} {B-Shape=No 7-10} {L-Shape=No 11-14}

Natural Computing Lab, Level 10, Wisma R&D, University of Malaya, 50603 Kuala Lumpur, Malaysia. Email: effirul@um.edu.my. Website: https://nclab.fsktm.um.edu.my.

Copyright © 2015 NCLab

**Figure 7.2:** The output page for the generated DNA sequences
The iterations are the number of iteration used to mutate the sequences and the GC% is the percentage of guanine-cytosine content.

## 7.3 Future Work

Given the high impact of the structural design on DNA nanotechnology field, this work hereby enlists several recommendations for the future work in order to improve the existing research work. It is being proposed towards the direction of industrial applications in DNA nanotechnology. Since this work has successfully demonstrated the use of the proposed schema to construct simple structures, additional works are needed to construct 3-dimensional and functional DNA nanostructure. For instance, the schema could potentially be applied towards the generation of multiple conformations of functional DNA nanostructures that have ability to carry functional payloads within its structure.

This cutting edge application can be seen as crucial especially in the field of medicine whereby, a real life application of delivering DNA nanorobots carrying load has been demonstrated in a living environment (Amir et al., 2014). Another future proposed work to improve this version would be to have multiple combinations of DNA structures, which have different precedence in delivering the therapeutic agents to the targeted cell for treatment. Recent advancements on the emergence of DNA nanotechnology has seen its potential applications in nanomedicine such as drug delivery and disease therapy (Sekhon, 2012). This field has put forward the attempt in diagnostic, treatment and to destroy cancer cells.

Current cancer therapies involved the use of high concentration of chemotherapeutic agents targeting the tumour site to destroy the cancerous cells. However, at the same time it also causes injury to the healthy cells. Thereby, in order to prevent the loss of healthy cells, recent work in nanomedicine has focused on the development of technologies such as ligand targeted delivery of therapeutic drugs and nanocarriers. These nanocarriers can be of liposomes or albumin-based nanoparticules and have been approved for clinical trials by the Food and Drug administration (FDA) in the United States in year 2009 (Bharali, Khalil, Gurbuz, Simone, & Mousa, 2009; Sparreboom et al., 2005). The compositions of lipid in liposomes allowed them to move across cell membranes and deliver therapeutic product to the targeted cells (Figure 7.3). Given the importance of the DNA nanorobots, future work on exploring into quantifying the percentage of well-formed DNA structures that can be significantly utilized is being proposed. This will solve the dispute on the amount of nanostructures that can work efficiently, especially when the nanostructures are used to deliver substances. The quantification work can hopefully lead to the successful delivery of the constructed structure into the cellular environment.

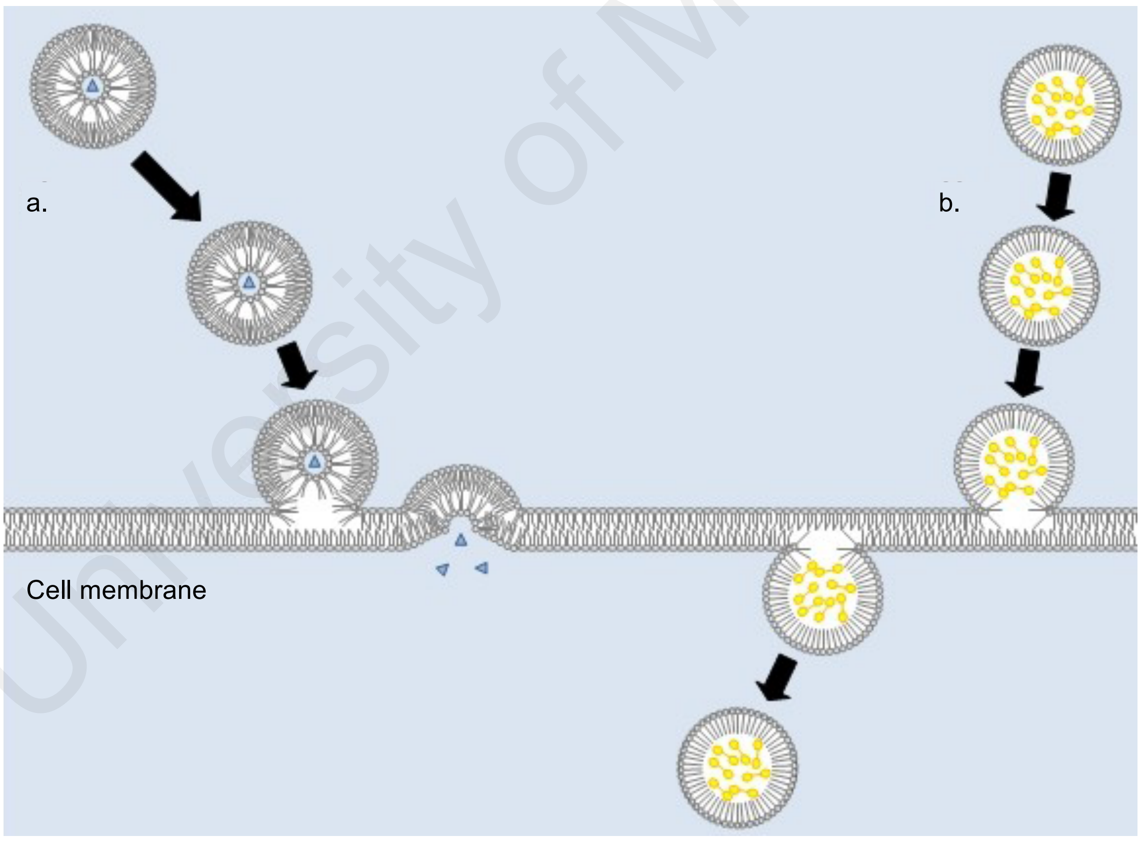


**Figure 7.3:** Drug delivery mechanism
Figure 7.3a represents drug delivery mechanism using liposomes. Figure 7.3b represents the emulsions to cross a cell membrane. Retrieved from (Zahid, Kim, Hussain, Amin, & Park, 2013).

Aside from the nanomedicine applications, the DNA nanostructures can also be developed for implementation in electrical components and circuits. At present, nanoelectronic depends on the complementary-symmetry metal-oxide semiconductor (CMOS) technology as it is currently being used for circuits such as image sensor, microcontroller and microprocessor (Baker, 2008). However, as the demand for further miniaturization and increasing of processing speeds, CMOS has gradually being substituted. In fact, nanoscale size electronic has been developed at the molecular level and this devices are identified as molecular electronics (Petty, Bryce, & Bloor, 1995) (Figure 7.4).

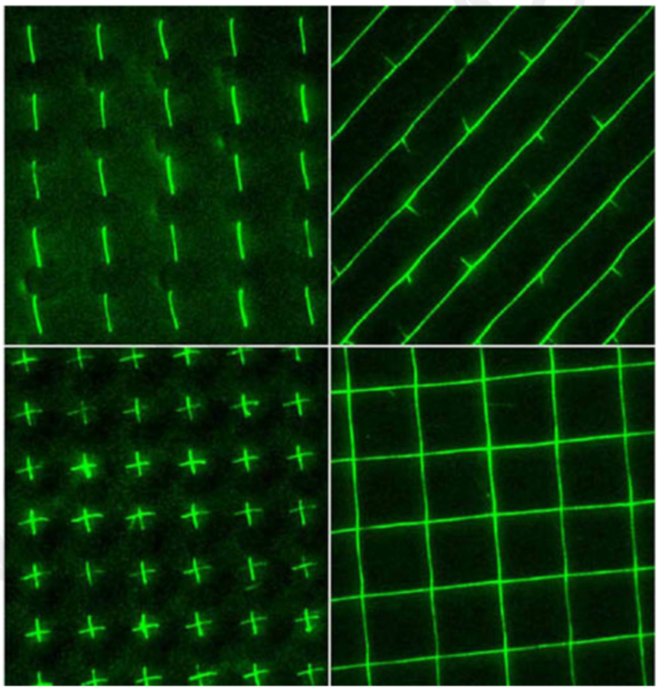


**Figure 7.4:** DNA nanostrand array
The DNA nanostrand array (Guan & Lee, 2005) serves as an important feature to biological based electronic and medical devices. Retrieved from (Zahid et al., 2013).

As DNA offers solutions to the current CMOS issue, an extension of this current work into forming electronic devices such as nanowires devices and transistors (Bachtold, Hadley, Nakanishi, & Dekker, 2001; DeHon, 2003) that behave like CMOS circuit is highly recommended. Besides being efficient in terms of power consumption, the main advantage of nanodevices made of DNA is that it can accumulate in a larger density compared to a normal circuit in an electronic system (Patwardhan, Dwyer, Lebeck, & Sorin, 2004).

# REFERENCES

Adleman, L. M. (1994). Molecular computation of solutions to combinatorial problems. *Science, 266*(5187), 1021-1024.

Aldaye, F. A., Palmer, A. L., & Sleiman, H. F. (2008). Assembling materials with DNA as the guide. *Science, 321*(5897), 1795–1799.

Allawi, H. T., & SantaLucia, J. J. (1997). Thermodynamics and NMR of internal G.T mismatches in DNA. *Biochemistry, 36*, 10581 – 10594.

Allawi, H. T., & SantaLucia, J. J. (1998a). Nearest-neighbor thermodynamics of internal A.C mismatches in DNA: sequence dependence and pH effects. *Biochemistry, 37*(26), 9435-9444.

Allawi, H. T., & SantaLucia, J. J. (1998b). NMR solution structure of a DNA dodecamer containing single G-T mismatches. *Nucleic Acids Research, 26*(21), 4925-4934.

Allawi, H. T., & SantaLucia, J. J. (1998b). Thermodynamics of internal C.T mismatches in DNA. *Nucleic Acids Research, 26*(11), 2694-2701.

Amir, Y., Ben-Ishay, E., Levner, D., Ittah, S., Abu-Horowitz, A., & Bachelet, I. (2014). Universal computing by DNA origami robots in a living animal. *Nature Nanotechnology, 9*, 353–357.

Andersen, E. S., Dong, M., Nielsen, M. M., Jahn, K., Lind-Thomsen, A., Mamdouh, W., . . . Kjems, J. (2008). DNA origami design of dolphin-shaped structures with flexible tails. *ACS Nano, 2*(6), 1213-1218.

Andersen, E. S., Dong, M., Nielsen, M. M., Jahn, K., Subramani, R., Mamdouh, W., . . . Kjems, J. (2009). Self-assembly of a nanoscale DNA box with a controllable lid. *Nature, 459*, 73–76.

Andronescu, M., Aguirre-Hernández, R., Condon, A., & Hoos, H. H. (2003). RNAsoft: A suite of RNA secondary structure prediction and design software tools. *Nucleic Acids Research, 31*(13), 3416-3422.

Avery, O. T., Macleod, C. M., & McCarty, M. (1944). Studies on the chemical nature of the substance inducing transformation of Pneumococcal types : induction of transformation by a desoxyribonucliec acid fraction isolated from Pneumococcus type III. *The Journal of Experimental Medicine, 79*(2), 137-158.

Bachtold, A., Hadley, P., Nakanishi, T., & Dekker, C. (2001). Logic circuits with carbon nanotube transistors. *Science, 294*(5545), 1317–1320.

Back, T., Fogel, D. B., & Michalewicz, Z. (1997). *Handbook of Evolutionary Computation*. Bristol, UK: IOP Publishing Ltd.

Baker, R. J. (2008). *CMOS: Circuit Design, Layout, and Simulation. 2nd edition*. New York: Wiley.

Bandyopadhyay, A., Pati, R., Sahu, S., Peper, F., & Fujita, D. (2010). Massively parallel computing on an organic molecular layer. *Nature Physics, 6*, 369-375.

Beisel, C. L., Bayer, T. S., Hoff, K. G., & Smolke, D. (2008). Model-guided design of ligand-regulated RNAi for Programmable control of gene expression. *Molecular Systems Biology, 4*(224), 1–14.

Ben-Dor, A., Karp, R., Schwikowski, B., & Yakhini, Z. (2000). Universal DNA Tag Systems: A Combinatorial Design Scheme. *Journal of Computational Biology, 7*, 503-551.

Bharali, D. J., Khalil, M., Gurbuz, M., Simone, T. M., & Mousa, S. A. (2009). Nanoparticles and cancer therapy: A concise review with emphasis on dendrimers. *International Journal of Nanomedicine, 4*, 1-7.

Bhatia, D., Surana, S., Chakraborty, S., Koushika, S. P., & Krishnan, Y. (2011). A synthetic icosahedral DNA-based host–cargo complex for functional in vivo imaging. *Nature Communications, 2*(339).

Biggs, N. L., Lloyd, E. K., & Wilson, R. J. (1986). *Graph Theory 1736-1936*. New York: Oxford University Press.

Bolewska, K., Zielenkiewicz, A., & Wierzchowski, K. L. (1984). Deoxydodecanucleotide heteroduplex d(TTTTATAATAAA). d(TTTATTATAAAA) containing the promoter Pribnow sequence TATAAT. I. Double-helix stability by UV spectrophotometry and calorimetry. *Nucleic Acids Research, 12*(7), 3245-3256.

Bolshoy, A., McNamara, P., Harrington, R. E., & Trifonov, E. N. (1991). Curved DNA without A-A: experimental estimation of all 16 DNA wedge angles. *Proceedings of the National Academy of Sciences, 88*(6), 2312–2316.

Bommarito, S., Peyret, N., & SantaLucia, J. J. (2000). Thermodynamic parameters for DNA sequences with dangling ends. *Nucleic Acids Research, 28*(9), 1929-1934.

Bray, W. (1921). A periodic reaction in homogeneous solution and its relation to catalysis. *Journal of the American Chemical Society, 43*, 1262–1267.

Brenner, S., & Lerner, R. A. (1992). Encoded combinatorial chemistry. *Proceedings of the National Academy of Sciences USA, 89*(12), 5381–5383.

Breslauer, K. J., Frank, R., Blocker, H., & Marky, L. A. (1986). Predicting DNA duplex stability from the base sequence. *Proc. Natl Acad. Sci. USA, 83*, 3746 – 3750.

Breukelaar, R., Demaine, E. D., Hohenberger, S., Hoogeboom, H. J., Kosters, W. A., & Liben-Nowell, D. (2004). Tetris is hard, even to approximate. *International Journal of Computational Geometry and Applications, 14*, 41–68.

Bumcrot, D., Manoharan, M., Koteliansky, V., & Sah, D. W. Y. (2006). RNAi therapeutics: a potential new class of pharmaceutical drugs. *Nature Chemical Biology, 2*, 711–719.

Cacchione, S., Santis, P. D., Foti, D., Palleschi, A., & Savino, M. (1989). Periodical polydeoxynucleotides and DNA curvature. *Biochemistry, 28*(22), 8706–8713.

Calladine, C. R., Drew, H. R., & McCall, M. J. (1988). The intrinsic curvature of DNA in solution. *201*(1), 127-137.

Camazine, S., Deneubourg, J., Franks, N. R., Sneyd, J., Theraulaz, G., & Bonabeau, E. (2001). *Self-organization in Biological Systems*: Princeton University Press.

Castets, V. V., Dulos, E., Boissonade, J., & De Kepper, P. (1990). Experimental evidence of a sustained standing Turing-type nonequilibrium chemical pattern. *Physical Review Letters, 64*, 2953–2956.

Chandrasekhar, K., & Malathhi, R. (2003). Non-Watson Crick base pairs might stabilize RNA structural motifs in ribozymes -- a comparative study of group-I intron structures. *Journal of Biosciences, 28*(5), 547-555.

Chen, R. H., Korotkov, A. N., & Likharev, K. K. (1996). Single-electron transistor logic. *Applied Physics Letters, 68*, 1954 -1956.

Cheng, J., Cheng, W., & Nagpal, R. (2005). *Robust and Self-Repairing Formation Control for Swarms of Mobile Agents.* Paper presented at the Proceeding 20th National Conference on Artificial Intelligence (AAAI 05).

Conrad, M. (1972). Information processing in molecular systems. *Currents in Modern Biology (now BioSystems), 5*, 1–14.

Das, J., Mukherjee, S., Mitra, A., & Bhattacharyya, D. (2006). Non-Canonical Base Pairs and Higher Order Structures in Nucleic Acids: Crystal Structure Database Analysis. *Journal of Biomolecular Structure and Dynamics, 24*(2), 149-161.

DeHon, A. (2003). Array-based architecture for FET-based, nanoscale electronics. *IEEE Transactions on Nanotechnology, 2*(1), 23–32.

Dickerson, R. E. (1989). Definitions and nomenclature of nucleic acid structure components. *Nucleic Acids Research, 17*(5), 1797-1803.

Dirks, R. M., Lin, M., Winfree, E., & Pierce, N. A. (2004). Paradigms for computational nucleic acid design. *Nucleic Acids Res, 32*(4), 1392-1403.

Douglas, S. M., Bachelet, I., & Church, G. M. (2012). A logic-gated nanorobot for targeted transport of molecular payloads. *Science, 335*, 831.

Douglas, S. M., Dietz, H., Liedl, T., Högberg, B., Graf, F., & Shih, W. M. (2009). Self-assembly of DNA into nanoscale three-dimensional shapes. *Nature, 459*, 414-418.

Douglas, S. M., Marblestone, A. H., Teerapittayanon, S., Vazquez, A., Church, G. M., & Shih, W. M. (2009). Rapid prototyping of 3D DNA-origami shapes with caDNAno. *Nucleic Acids Research, 37*(15), 5001–5006.

Duan, S., Mathews, D. H., & Turner, D. H. (2006). Interpreting oligonucleotide microarray data to determine RNA secondary structure: application to the 3' end of Bombyx mori R2 RNA. *Biochemistry, 45*(32), 9819–9832.

Elbashir, S. M., Harborth, J., Lendeckel, W., Yalcin, A., Weber, K., & Tuschl, T. (2001). Duplexes of 21-nucleotide RNAs mediate RNA interference in cultured mammalian cells. *Nature, 411*(6836), 494–498.

Elliott, D., & Ladomery, M. (2011). *Molecular Biology of RNA*: Oxford University Press.

Feldkamp, U., & Niemeyer, C. M. (2006). Rational design of DNA nanoarchitectures. *Angewandte Chemie International Edition in English, 45*, 1856-1876.

Feldkamp, U., Rauhe, H., & Banzhaf, W. (2003). Software tools for DNA sequence design. *Genet Programming Evolvable Machines, 4*(2), 153–171.

Feldkamp, U., Saghafi, S., & Rauhe, W. H. (2001). DNA sequence generator: A program for the construction of DNA sequences. *Springer LNCS, 2340*, 23-32.

Frutos, A. G., Liu, Q., Thiel, A. J., Sanner, A. M. W., Condon, A. E., Smith, L. M., & Corn, R. M. (1997). Demonstration of a word design strategy for DNA computing on surfaces. *Nucleic Acids Research, 25*(23), 4748-4757.

Geggier, S., & Vologodskii, A. (2010). Sequence dependence of DNA bending rigidity. *Proceedings of the National Academy of Sciences, 107*(35), 15421–15426.

Gerry, N. P., Witowski, N. E., Day, J., Hammer, R. P., Barany, G., & Barany, F. (1999). Universal DNA Microarray Method for Multiplex Detection of Low Abundance Point Mutations. *Journal of Molecular Biology, 292*, 251-262.

Goodsell, D. S., & Dickerson, R. E. (1994). Bending and curvature calculations in B-DNA. *Nucleic Acids Research, 22*(24), 5497-5503.

Gothelf, K. V., & LaBean, T. H. (2005). DNA-programmed assembly of nanostructures. *Org. Biomol. Chem, 3*, 4023–4037.

Grunbaum, B., & Shephard, G. C. (1986). Tilings and Patterns. *New York: Freeman*.

Guan, J., & Lee, L. J. (2005). Generating highly ordered DNA nanostrand arrays. *Proceedings of the National Academy of Sciences of the United States of America, 102*(51), 18321–18325.

Han, D., Pal, S., Nangreave, J., Deng, Z., Liu, Y., & Yan, H. (2011). DNA origami with complex curvatures in three-dimensional space. *Science, 332*, 342–346.

He, Y., Ye, T., Su, M., Zhang, C., Ribbe, A. E., Jiang, W., & Mao, C. (2008). Hierarchical self-assembly of DNA into symmetric supramolecular polyhedra. *Nature, 452*, 198-201.

Hermann, T., & Westhof, E. (1999). Non-Watson-Crick base pairs in RNA-protein recognition. *Chemistry & Biology, 6*(12), R335-R343.

Heylighen, F., & Gershenson, C. (2003). The Meaning of Self-organization in Computing. *IEEE Intelligent Systems, section Trends & Controversies - Self-organization and Information Systems*.

Ho, P. S., & Carter, M. (2011). DNA Structure: Alphabet Soup for the Cellular Soul. In H. Seligmann (Ed.), *DNA Replication-Current Advances* (pp. 708): InTech.

Hofacker, I. L. (2003). Vienna RNA secondary structure server. *Nucleic Acids Research, 31*(13), 3429-3431.

Hogan, M., & Austin, R. H. (1987). Importance of DNA stiffness in protein–DNA binding specificity. *Nature, 329*, 263–266.

Hogan, M., LeGrange, J., & Austin, R. H. (1983). Dependence of DNA helix flexibility on base composition. *Nature, 304*, 752–754.

Holliday, R. (1964). A mechanism for gene conversion in fungi. *Genetical Research, 5*, 282–304.

Hormeño, S., Ibarra, B., Carrascosa, J. L., Valpuesta, J. M., Moreno-Herrero, F., & Arias-Gonzalez, J. R. (2011). Mechanical Properties of High-G·C Content DNA with A-Type Base-Stacking. *Biophysical Journal, 100*(8), 1996–2005.

Hsu, J. Y. (2002). *Computer Logic: Design Principles and Applications*: Springer.

Jaeger, L., & Leontis, N. B. (2000). Tecto-RNA: one-dimensional self-assembly through tertiary interactions. *Angewandte Chemie International Edition in English, 39*, 2521-2524.

Jaeger, L., Westhof, E., & Leontis, N. B. (2001). TectoRNA: modular assembly units for the construction of RNA nano-objects. *Nucleic Acids Research, 29*, 455-463.

Jungmann, R., Liedl, T., Sobey, T. L., Shih, W., & Simmel, F. C. (2008). Isothermal assembly of DNA origami structures using denaturing agents. *Journal of the American Chemical Society, 130*, 10062–10063.

Karsenti, E. (2008). Self-organization in cell biology: a brief history. *Nature Reviews Molecular Cell Biology, 9*(3), 255-262.

Ke, Y., Douglas, S. M., Liu, M., Sharma, J., Cheng, A., Leung, A., . . . Yan, H. (2009). Multilayer DNA Origami Packed on a Square Lattice. *J Am Chem Soc, 131*(43), 15903.

Ke, Y., Ong, L. L., Shih, W. M., & Yin, P. (2012). Three-Dimensional Structures Self-Assembled from DNA Bricks. *Science, 338*(6111), 1177-1183

Kowalczyk, S. W., Tuijtel, M. W., Donkers, S. P., & Dekker, C. (2010). Unraveling Single-Stranded DNA in a Solid-State Nanopore. *Nano Letters, 10*(4), 1414-1420.

Kuzuya, A., & Komiyama, M. (2010). DNA origami: Fold, stick, and beyond. *Nanoscale. Review., 2*, 310-322.

Le, J. D., Pinto, Y., Seeman, N. C., Musier-Forsyth, K., Taton, T. A., & Kiehl, R. A. (2004). DNA-templated self-assembly of metallic nanocomponent arrays on a surface. *Nano Letters, 4*, 2343–2347.

Lee, H., Lytton-Jean, A. K. R., Chen, Y., Love, K. T., Park, A. I., Karagiannis, E. D., . . . Anderson, D. G. (2012). Molecularly self-assembled nucleic acid nanoparticles for targeted in vivo siRNA delivery. *Nature Nanotechnology, 7*, 389–393.

Lehn, J. M. (2004). Supramolecular chemistry: from molecular information towards selforganization and complex matter. *Reports on Progress in Physics, 67*, 249-265.

Leonard, G. A., Thomson, J., Watson, W. P., & Brown, T. (1990). High-resolution structure of a mutagenic lesion in DNA. *Proceedings of the National Academy of Sciences, 87*(24), 9573-9576.

Leontis, N. B., & Westhof, E. (2003). Analysis of RNA motifs. *Current Opinion in Structural Biology, 13*(3), 300-308.

Leu, K., Obermayer, B., Rajamani, S., Gerland, U., & Chen, I. A. (2011). The prebiotic evolutionary advantage of transferring genetic information from RNA to DNA. *Nucleic Acids Research, 39*(18), 8135–8147.

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

Lewis, M., Chang, G., Horton, N. C., Kercher, M. A., Pace, H. C., Schumacher, M. A., . . . Lu, P. (1996). Crystal Structure of the Lactose Operon Repressor and Its Complexes with DNA and Inducer. *Science, 271*(5253), 1247-1254.

Li, F., & Stormo, G. D. (2001). Selection of optimal DNA oligos for gene expression arrays. *Bioinformatics, 17*, 1067-1076.

Liu, D., Park, S.-H., Reif, J. H., & LaBean, T. H. (2004). DNA nanotubes self-assembled from TX tiles as templates for conductive nanowires. *Proc. Nat. Acad. Sci., 101*, 717-722.

Lu, Z. J., Turner, D. H., & Mathews, D. H. (2006). A set of nearest neighbor parameters for predicting the enthalpy change of RNA secondary structure formation. *Nucleic Acids Research, 34*, 4912-4924.

Mameia, M., Menezesb, R., Tolksdorfc, R., & Zambonelli, F. (2006). Case studies for self-organization in computer science. *Journal of Systems Architecture, 52*(8–9), 443–460.

Marathe, A., Condon, A. E., & Corn, R. M. (2001). On combinatorial DNA word design. *Journal of Computational Biology, 8*(3), 201-219.

Marchi, A. N., Saaem, I., Vogen, B. N., Brown, S., & LaBean, T. H. (2014). Toward Larger DNA Origami. *Nano Letter, 14*(10), 5740–5747.

Markham, N. R., & Zuker, M. (2008). UNAFold: software for nucleic acid folding and hybridization. *Methods Molecular Biology, 453*, 3-31.

Masquida, B., & Westhof, E. (2000). On the wobble GoU and related pairs. *RNA, 6*(1), 9-15.

Mathews, D. H., Burkard, M. E., Freier, S. M., Wyatt, J. R., & Turner, D. H. (1999). Predicting oligonucleotide affinity to nucleic acid targets. *RNA, 5*(11), 1458-1469.

Mathews, D. H., Disney, M. D., Childs, J. L., Schroeder, S. J., Zuker, M., & Turner, D. H. (2004). Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. *Proceedings of the National Academy of Sciences, 101*, 7287-7292.

Mathews, D. H., & Turner, D. H. (2006). Prediction of RNA secondary structure by free energy minimization. *Current Opinion in Structural Biology, 16*(3), 270-278.

Mei, Q., Wei, X., Su, F., Liu, Y., Youngbull, C., Johnson, R., . . . Meldrum, D. (2011). Stability of DNA Origami Nanoarrays in Cell Lysate. *Nano Letter, 11*, 1477–1482.

Misteli, T. (2001). The concept of self-organization in cellular architecture. *The Journal of Cell Biology, 155*, 181–185.

Moody, E. M., & Bevilacqua, P. C. (2003). Thermodynamic coupling of the loop and stem in unusually stable DNA hairpins closed by CG base pairs. *Journal of the American Chemical Society, 125*(8), 2032-2033.

Moore, G. E. (1965). Cramming More Components onto Integrated Circuits. *Electronics*, 114–117.

Nagpal, R., Zambonelli, F., Sirer, E. G., Chaouchi, H., & Smirnov, M. (2006). Interdisciplinary Research: Roles for Self-Organization. *Intelligent Systems, IEEE, 21*(2), 50-58.

Nakano, M., Moody, E. M., Liang, J., & Bevilacqua, P. C. (2002). Selection for thermodynamically stable DNA tetraloops using temperature gradient gel electrophoresis reveals four motifs: d(cGNNAg), d(cGNABg), d(cCNNGg), and d(gCNNGc). *Biochemistry, 41*(48), 14281-14292.

Nenni, D. (2015, 05-22-2015). Samsung Foundry Debuts 10nm Wafer!   , from https://http://www.semiwiki.com/forum/content/4662-samsung-foundry-debuts-10nm-wafer.html

Nicolis, G., & Prigogine, I. (1977). *Self-organization in nonequilibrium systems: from dissipative structures to order through fluctuation*. New York: Wiley.

Niehaus, J. (1998). *DNA Computing: Bewertung und Simulation.* (Diploma thesis ), University of Dortmund.

Parker, J. (2003). Computing with DNA. *EMBO Reports, 4*(1), 7-10.

Patwardhan, J. P., Dwyer, C., Lebeck, A. R., & Sorin, D. J. (2004). *Circuit and system architecture for DNA-guided self-assembly of nanoelectronics.* Paper presented at the Foundations of Nanoscience: Self-Assembled Architectures and Devices.

Pazhitnov, A., Gerasimov, V., & Pavlovsky, D. (1985). Tetris. A video game. *Academy of Sciences. Moscow, Russia*.

Pettinaro, G. C., Kwee, I., Gambardella, L. M., Mondada, F., Floreano, D., Nolfi, S., . . . Dorigo, M. (2002). *SWARM Robotics: A Different Approach to Service Robotics.* Paper presented at the Proceedings of the 33rd International Symposium on Robotics, Stockholm, Sweden.

Petty, M. C., Bryce, M. R., & Bloor, D. (1995). *An Introduction to Molecular Electronics. 1st edition*. London: Oxford University Press.

Peyret, N., Seneviratne, P. A., Allawi, H. T., & SantaLucia, J. J. (1999). Nearest-Neighbor Thermodynamics and NMR of DNA Sequences with Internal A.A, C.C, G.G, and T.T Mismatches. *Biochemistry, 38*, 3468-3477.

Piekna-Przybylska, D., DiChiacchio, L., Mathews, D. H., & Bambara, R. A. (2009). A sequence similar to tRNA3Lys gene is embedded in HIV-1 U3/R and promotes minus strand transfer. *Nature Structural & Molecular Biology, 17*(1), 83–89.

Pinheiro, A. V., Han, D., Shih, W. M., & Yan, H. (2011). Challenges and opportunities for structural DNA nanotechnology. *Nature Nanotechnology, 6*, 763–772.

Plum, G. E., Grollman, A. P., Johnson, F., & Breslauer, K. J. (1995). Influence of the oxidatively damaged adduct 8-oxodeoxyguanosine on the conformation, energetics, and thermodynamic stability of a DNA duplex. *Biochemistry, 34*(49), 16148-16160.

Poole, A. M., & Ranganathan, R. (2006). Knowledge-based potentials in protein design. *Current Opinion in Structural Biology, 16*(4), 508-513.

Pray, L. A. (2008). Discovery of DNA Structure and Function: Watson and Crick. *Nature Eduction, 1*(1), 100.

Ramlan, E. I., & Zauner, K.-P. (2013). In-silico design of computational nucleic acids for molecular information processing. *Journal of Cheminformatics, 5*, 22.

Reuter, J. S., & Mathews, D. H. (2010). RNAstructure: software for RNA secondary structure prediction and analysis. *BMC Bioinformatics, 11*, 129.

Rhodes, D., & Klug, A. (1980). Helical periodicity of DNA determined by enzyme digestion. *Nature, 286*, 573–578.

Richmond, T. J., & Davey, C. A. (2003). The structure of DNA in the nucleosome core. *Nature, 423*, 145–150.

Rinaudo, K., Bleris, L., Maddamsetti, R., Subramanian, S., Weiss, R., & Benenson, Y. (2007). A universal RNAi-based logic evaluator that operates in mammalian cells. *Nature Biotechnology, 25*(7), 795–801.

Rothemund, P. W. K. (2006). Folding DNA to create nanoscale shapes and patterns. *Nature, 440*, 297–302.

Rothemund, P. W. K., Papadakis, N., & Winfree, E. (2004). Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biol., 2*, e424.

Rozenberg, G., Back, T., & Kok, J. (2012). *Handbook of Natural Computing*: Springer.

Sahin, E., Labella, T., Trianni, V., Deneubourg, J.-L., Rasse, P., Floreano, D., . . . Dorigo, M. (2002). *SWARM-BOTS: Pattern Formation in a Swarm of Self-Assembling Mobile Robots.* Paper presented at the Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Hammamet, Tunisia.

SantaLucia, J., Allawi, H. T., & Seneviratne, A. (1996). Improved nearest-neighbor parameters for predicting DNA duplex stability. *Biochemistry, 35*, 3555–3562.

SantaLucia, J. J., & Hicks, D. (2004). The thermodynamics of DNA structural motifs. *Annu Rev Biophys Biomol Struct, 33*, 415-440.

Satchwell, S. C., Drew, H. R., & Travers, A. A. (1986). Sequence periodicities in chicken nucleosome core DNA. *Journal of Molecular Biology, 191*(4), 659-675.

Schultz, S. C., Shields, G. C., & Steitz, T. A. (1991). Crystal structure of a CAP-DNA complex: the DNA is bent by 90 degrees. *Science, 253*(5023), 1001-1007.

Schumacher, M. A., Choi, K. Y., Zalkin, H., & Brennan, R. G. (1994). Crystal structure of LacI member, PurR, bound to DNA: minor groove binding by alpha helices. *Science, 266*, 763-770.

Seeman, N. C. (1982). Nucleic-acid junctions and lattices. *J Theor Biol, 99*, 237–247.

Seeman, N. C. (1990). De novo design of sequences for nucleic acid structural engineering. *J Biomol Struct Dyn, 8*, 573–581.

Seeman, N. C. (1999). DNA engineering and its application to nanotechnology. *Trends Biotechnol, 17*(11), 437-443.

Seeman, N. C. (2005). Structural DNA nanotechnology: an overview. *Methods Mol Biology, 303*, 143-166.

Seeman, N. C. (2007). An overview of structural DNA nanotechnology. *Mol Biotechnol, 37*(3), 246-257. doi: 10.1007/s12033-007-0059-4

Seeman, N. C. (2010). Nanomaterials based on DNA. *Annual Review of Biochemistry, 79*, 65–87.

Seeman, N. C., & Lukeman, P. S. (2005). Nucleic Acid Nanostructures: Bottom-Up Control of Geometry on the Nanoscale. *Reports on Progress in Physics, 68*(1), 237-270.

Seeman, N. C., Wang, H., Yang, X., Liu, F., Mao, C., Sun, W., . . . Chen, J. (1999). New motifs in DNA nanotechnology. *Nanotechnology, 9*(3), 257–273.

Sekhon, B. S. (2012). Nanobiotechnology: An overview of drug discovery, delivery and development. *RGUHS Journal of Pharmaceutical Sciences, 2*(1), 14-23.

Sen, D., & Gilbert, W. (1988). Formation of parallel four-stranded complexes by guanine rich motifs in DNA and its implications for meiosis. *Nature, 334*(6180), 364–366.

Shen, X., Jiang, Q., Wang, J., Dai, L., Zou, G., Wang, Z. G., . . . Ding, B. (2012). Visualization of the intracellular location and stability of DNA origami with a label-free fluorescent probe. *48*(92), 11301-11303.

Shih, W., Quispe, J., & Joyce, G. (2004). A 1.7-kilobase single-stranded DNA that folds into a nanoscale octahedron. *Nature, 427*, 618–621.

Shoemaker, D. D., Lashkari, D. A., Morris, D., Mittmann, M., & Davis, R. W. (1996). Quantitative phenotypic analysis of yeast deletion mutants using a highly parallel molecular bar-coding strategy. *Nature Genetics, 14*, 450 - 456.

Sinden, R. R. (1994). *DNA Structure and Function*. California: Academic Press, Inc.

Sparreboom, A., Scripture, C. D., Trieu, V., Williams, P. J., De, T., Yang, A., . . . Desai, N. (2005). Comparative preclinical and clinical pharmacokinetics of a cremophor-free, nanoparticle albumin-bound paclitaxel (ABI-007) and paclitaxel formulated in Cremophor (Taxol). *Clinical Cancer Research, 11*(11), 4136–4143.

Sun, S., Brem, R., Chan, H. S., & Dill, K. A. (1995). Designing amino acid sequences to fold with good hydrophobic cores. *Protein Engineering, 8*(12), 1205-1213.

Surana, S., Bhat, J. M., Koushika, S. P., & Krishnan, Y. (2011). An autonomous DNA nanomachine maps spatiotemporal pH changes in a multicellular living organism. *Nature Communications, 2*, 340.

Surana, S., Shenoy, A. R., & Krishnan, Y. (2015). Designing DNA nanodevices for compatibility with the immune system of higher organisms. *Nature Nanotechnology, 10*, 741–747.

Tabony, J. (2006). Historical and conceptual background of self-organization by reactive processes. *The Journal of Cell Biology, 98*, 589–560.

Team, R. D. C. (2005). R: a language and environment for statistical computing. *R Foundation for Statistical Computing, Vienna, Austria,* http://www.R-project.org.

Toffoli, T., & Margolus, N. H. (1991). Programmable matter: Concepts and realization. *Physica D: Nonlinear Phenomena, 47*(1-2), 263-272.

Travers, A. A. (2004). The structural basis of DNA flexibility. *Philos Trans A Math Phys Eng Sci, 362*(1820), 1423-1438.

Trikha, J., Filman, D. J., & Hogle, J. M. (1999). Crystal structure of a 14 bp RNA duplex with non-symmetrical tandem GxU wobble base pairs. *Nucleic Acids Research, 27*(7), 1728-1739.

Varani, G., & McClain, W. H. (2000). The G x U wobble base pair. A fundamental building block of RNA structure crucial to RNA function in diverse biological systems. *EMBO Reports, 1*(1), 18-23.

Von Neumann, J. (1958). *The computer and the brain*. New Haven, CT: Yale University Press.

Wang, H. (1961). Proving theorems by pattern recognition I. *Bell System Tech. Journal, 40*(40), 1.

Wang, J. C. (1979). Helical repeat of DNA in solution. *Proceedings of the National Academy of Sciences, 76*(1), 200-203.

Watson, J. D., & Crick, F. H. (1953). Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature, 171*(4356), 737–738.

Wei, B., Dai, M., & Yin, P. (2012). Complex shapes self-assembled from single-stranded DNA tiles. *Nature, 485*(7400), 623-626.

Wetmur, J. G. (1991). DNA Probes: Applications of the Principles of Nucleic Acid Hybridization. *Critical Reviews in Biochemistry and Molecular Biology, 26*(3-4), 227-259.

Williams, S., Lund, K., Lin, C., Wonka, P., Lindsay, S., & Yan, H. (2008). *Tiamat: a three-dimensional editing tool for complex DNA structures.* Paper presented at the The 14th International Meeting on DNA Computing Proceedings, Czech Republic: Silesian University in Opava.

Win, M. N., & Smolke, C. D. (2008). Higher-order cellular information processing with synthetic RNA devices. *Science, 322*, 456–460.

Winfree, E. (1996). On the computational power of DNA annealing and ligation. In R. J. Lipton & E. B. Baum (Eds.), *DNA-based computers* (pp. 199–221). Providence, Rhode Island: American Mathematical Society.

Winfree, E. (1998). *Algorithmic self-assembly of DNA.* (Ph.D. thesis), California Institute of Technology.

Winfree, E., Sun, W., & Seeman, N. C. (1998). Design and self-assembly of two-dimensional DNA crystals. *Nature, 394*, 539-544.

Wolfram, S. (1983). Statistical mechanics of cellular automata. *Reviews of Modern Physics, 55*(3), 601-644.

Wu, P., Nakano, S., & Sugimoto, N. (2002). Temperature dependence of thermodynamic properties for DNA/DNA and RNA/DNA duplex formation. *Eur. J. Biochem, 269*, 2821–2830.

Wuchty, S., Fontana, W., Hofacker, I. L., & Schuster, P. (1999). Complete suboptimal folding of RNA and the stability of secondary structures. *Biopolymers, 49*(2), 145–165.

Xia, T., SantaLucia, J. J., Burkard, M. E., Kierzek, R., Schroeder, S. J., Jiao, X., . . . Turner, D. H. (1998). Thermodynamic parameters for an expanded nearest-neighbor model for formation of RNA duplexes with Watson–Crick base pairs. *Biochemistry, 37*, 14719 – 14735.

Yakovchuk, P., Protozanova, E., & Frank-Kamenetskii, M. D. (2006). Base-stacking and base-pairing contributions into thermal stability of the DNA double helix. *Nucleic Acids Research, 34*(2), 564-574.

Yin, P., Hariadi, R. F., Sahu, S., Choi, H. M. T., Park, S. H., LaBean, T. H., & Reif, J. H. (2008). Programming DNA Tube Circumferences. *Science, 321*, 824-826.

Yue, K., & Dill, K. A. (1992). Inverse protein folding problem: designing polymer sequences. *Proceedings of the National Academy of Sciences, 89*(9), 4163-4167.

Zahid, M., Kim, B., Hussain, R., Amin, R., & Park, S. H. (2013). DNA nanotechnology: a future perspective. *Nanoscale Research Letters, 8*(1), 119.

Zambonelli, F. (2006). Self-management-and the many facets of "nonself". *Intelligent Systems, IEEE, 21*(2), 53-55.

Zauner, K. P. (2005). From Prescriptive Programming of Solid-State Devices to Orchestrated Self-organisation of Informed Matter. *Unconventional Programming Paradigms, 3566*, 47-55.

Zhabotinsky, A. M., & Zaikin, A. N. (1973). Autowave processes in a distributed chemical system. *Journal of Theoretical Biology, 40*, 45–61.

Zhang, F., Jiang, S., Wu, S., Li, Y., Mao, C., Liu, Y., & Yan, H. (2015). Complex wireframe DNA origami nanostructures with multi-arm junction vertices. *Nature Nanotechnology, 10*, 779–784.

Zhou, C., Luo, H., Feng, X., Li, X., Zhu, J., He, L., & Li, C. (2012). FOLDNA, a Web Server for Self-Assembled DNA Nanostructure Autoscaffolds and Autostaples. *Journal of Nanotechnology, 2012*, 453953.

Zhu, J., Wei, B., Yuan, Y., & Mi, Y. (2009). UNIQUIMER 3D, a software system for structural DNA nanotechnology design, analysis and evaluation. *Nucleic Acids Research, 37*(7), 2164-2175.

Zwicke, G. L., Mansoori, G. A., & Jeffery, C. J. (2012). Utilizing the folate receptor for active targeting of cancer nanotherapeutics. *Nano Reviews, 3*.

## LIST OF PUBLICATION AND PAPER PRESENTED

### Publication

- Ong HS, Syafiq-Rahim M, Kasim NH, Firdaus-Raih M, Ramlan EI (2016) Self-assembly programming of DNA polyominoes. J Biotechnol 236: 141-51. doi: 10.1016/j.jbiotec.2016.08.017

- Ong HS, Rahim MS, Firdaus-Raih M, Ramlan EI (2015) DNA Tetrominoes: The Construction of DNA Nanostructures Using Self-Organised Heterogeneous Deoxyribonucleic Acids Shapes. *PLoS ONE* 10(8): e0134520. doi:10.1371/journal.pone.0134520

### Paper presented

- Ong HS. (2015) DNA tetrominoes: Programmable heterogeneous deoxyribonucleic acids as self-organized information processors. Postgraduate Research Excellence Symposium (PgRES), Faculty of Computer Science and Information Technology, University of Malaya

## **APPENDIX A:** Main_RunDNATetris.tcl

```
#! /usr/local/bin/tclsh
# SignalRegenerateInitialSeq  1 = need to regenerate initial seq again
#SignalRegenerateInitialSeq  2 = solution found

# Get current working directory
proc getScriptDirectory {} {
   set dispScriptFile [file normalize [info script]]
   set scriptFolder [file dirname $dispScriptFile]
   return $scriptFolder
}
set path [getScriptDirectory]
set originalPath $path
set Max_Iteration 500

set OutputMax_Iteration [open "${path}/Max_Iteration.txt" w]
puts $OutputMax_Iteration "$Max_Iteration"
close $OutputMax_Iteration

set SignalRegenerateInitialSeq 1
while {$SignalRegenerateInitialSeq < 2} {
set path $originalPath
exec ${path}/Main2GenerateSeq.tcl

            set ReadSignal [open ${path}/Signal.txt r]
            while {[gets $ReadSignal lineReadSignal] >=0} {
                        set Signal $lineReadSignal
            }
            close $ReadSignal

            if {$Signal == "Need to Regenerate Initial seq"} {
            set SignalRegenerateInitialSeq 1
            } else {
            set SignalRegenerateInitialSeq 2
            }
}
```

## **APPENDIX B:** Main2GenerateSeq.tcl

```
#! /usr/local/bin/tclsh
#cylenumber is the completion of seq 1 = seq1 etc

# Get current working directory
proc getScriptDirectory {} {
   set dispScriptFile [file normalize [info script]]
   set scriptFolder [file dirname $dispScriptFile]
   return $scriptFolder
}

set path [getScriptDirectory]
set originalPath $path
set ReadSquareType [open ${path}/SquareType.txt r]
        while {[gets $ReadSquareType lineReadSquareType] >=0} {
                set SquareType $lineReadSquareType
        }
close $ReadSquareType

exec ${path}/GenerateRandomSeq.pl
exec ${path}/Adjust_RandomSeq.pl

if {[file exists ${path}/b.txt] == 1} {
        file delete -force ${path}/b.txt
}
set ReadMinCG [open ${path}/MinCG.txt r]
        while {[gets $ReadMinCG lineMinCG] >=0} {
        set MinCG $lineMinCG
        }
close $ReadMinCG

set ReadMaxCG [open ${path}/MaxCG.txt r]
        while {[gets $ReadMaxCG lineMaxCG] >=0} {
        set MaxCG $lineMaxCG
        }
close $ReadMaxCG
set LimitGACycle 500
set DefineSeqFilename "${path}/${SquareType}DefineSeq.txt"
set DefineSeq2Filename "${path}/${SquareType}DefineSeq2.txt"
set TotalSeqNum {}

# Generate DefineSeq2.txt, get total seqNo involved
set InDefine [open "$DefineSeqFilename" r]
while {[gets $InDefine lineDefineSeq] >=0} {
set SeqNum [lindex $lineDefineSeq 0]
lappend TotalSeqNum $SeqNum
}
close $InDefine

set OutDefineSeq2 [open "$DefineSeq2Filename" w]
# Get all SeqNo involved in the DefineSeq.txt
set AllSeqNo [lsort -real -unique $TotalSeqNum ]
set TotalStrand [lindex $AllSeqNo end]

foreach itemb $AllSeqNo {
        set TotalDLength {}
        set TotalStartD {}
        set TotalEndD {}
        set InDefine [open "$DefineSeqFilename" r]
        while {[gets $InDefine lineDefineSeq] >=0} {
        set SeqNoD [lindex $lineDefineSeq 0]
        set SeqLengthD [lindex $lineDefineSeq 1]
        set StartD [lindex $lineDefineSeq 2]
        set EndD [lindex $lineDefineSeq 3]
                if {$itemb ==  $SeqNoD} {
                lappend TotalDLength $SeqLengthD
                lappend TotalStartD $StartD
                lappend TotalEndD $EndD
                }
        }
                close $InDefine
                if {$TotalStartD != "NIL"} {
                set SeqDLength_Sort [lsort -integer $TotalDLength ]
                set StartD_Sort [lsort -real $TotalStartD ]
                set EndD_Sort [lsort -real $TotalEndD ]
                set SeqInDLength [lindex $SeqDLength_Sort 0]
                set StartD_Lowest [lindex $StartD_Sort 0]
                set EndD_Highest [lindex $EndD_Sort end]
                puts $OutDefineSeq2 "$itemb     $SeqInDLength $StartD_Lowest
        $EndD_Highest"
                }
                if {$TotalStartD == "NIL"} {
                set StartD_Lowest NIL
                set EndD_Highest NIL
                puts $OutDefineSeq2 "$itemb     $TotalDLength  $StartD_Lowest
        $EndD_Highest"
                }
}
close $OutDefineSeq2

# split DefineSeq.txt into 1BEWIDefineSeq.txt, 2BEWIDefineSeq.txt , 3BEWIDefineSeq.txt etc
set CountFileNo 1
foreach itemS $AllSeqNo {
        set InDefine [open "$DefineSeqFilename" r]
        set OutSplitDefineSeq [open "${path}/${CountFileNo}_${SquareType}DefineSeq.txt" w]
        while {[gets $InDefine lineDefineSeq] >=0} {
        set SeqNum [lindex $lineDefineSeq 0]
                            if { $SeqNum <= $itemS} {
                puts $OutSplitDefineSeq "$lineDefineSeq"
                }
        }
        incr CountFileNo
close $InDefine
close $OutSplitDefineSeq
}

# split DefineSeq2.txt into 1BEWIDefineSeq2.txt, 2BEWIDefineSeq2.txt  etc.
set CountFile2No 1
foreach itemS $AllSeqNo {
        set InDefine [open "$DefineSeq2Filename" r]
        set OutSplitDefineSeq2 [open "${path}/${CountFile2No}_${SquareType}DefineSeq2.txt"
        w]
        while {[gets $InDefine lineDefineSeq2] >=0} {
        set SeqNum [lindex $lineDefineSeq2 0]
                if { $SeqNum <= $itemS} {
                puts $OutSplitDefineSeq2 "$lineDefineSeq2"
                }
        }
        incr CountFile2No
close $InDefine
close $OutSplitDefineSeq2
}
```

```
set countfinalseq 0
set in [open "${path}/UnOptimized_Seq.txt" r]
set strand 0
  while {[gets $in linesplit] >=0} {
          if {[string length $linesplit] != 0} {
          set SplitFile [open ${path}/FinalSeq_${strand}.txt w]
          puts $SplitFile "$linesplit"
          close $SplitFile
          incr strand
          incr countfinalseq
          }
  }
  close $in

file delete -force ${path}/Summary_AfterOptimized_Seq.txt
set SummaryFinalSeq [open "${path}/Summary_AfterOptimized_Seq.txt" a+]
set CycleNumber 1
set SSType [split "$SquareType" {}]
set SeqNo 1
set SeqNoInterpretation {}
foreach item $SSType {
          if {$item != "I"} {
          set EndNo "[expr $SeqNo+3]"
          lappend SeqNoInterpretation "$item=SequenceNo $SeqNo-$EndNo"
          incr SeqNo 4
          }
          if {$item == "I"} {
          set EndNo "[expr $SeqNo+1]"
          lappend SeqNoInterpretation "$item=SequenceNo $SeqNo-$EndNo"
          incr SeqNo 2
          }
}

# $z is no of dna strands
for { set z 0 } { $z < $TotalStrand  } { incr z } {
set value 1
set CycleGA 0
set PrintCycleNumber [open "${path}/cycle.txt" w]
puts $PrintCycleNumber $CycleNumber
set CycleNoMinus1 [expr $CycleNumber-1]
close $PrintCycleNumber
set grabb [open "${path}/cycle.txt" r]
set 1stElement 0
while {[gets $grabb linegrab] >=0} {
          if {$1stElement == 0} {
                          set CycleNumber [lindex $linegrab 0]
          }
          incr 1stElement
}
close $grabb
set fit [open "${path}/FinalSeq_${CycleNoMinus1}.txt" r]
set lines [split [read $fit] "\n"]
close $fit
set listseq {}
for { set u 0 } { $u < 1 } { incr u } {
set listseq [lindex $lines $u]
}
while {$value == 1} {
if {[file exists ${path}/query1.txt] == 1} {
file delete -force ${path}/query1.txt
}
if {[file exists ${path}/target1.txt] == 1} {
file delete -force ${path}/target1.txt
```

```
}

incr CycleGA
set number $CycleNumber

if {$CycleNumber ==1 } {
if {$CycleGA == 1} {
lappend ComSeq $listseq
set NoLine 1
set Score 0
set number $CycleNumber
set t 1
set SplitFile [open ${path}/$t.fasta w+]
puts $SplitFile ">DNA $t"
puts $SplitFile "$listseq"
flush $SplitFile
close $SplitFile
set AfterAdjSeq $listseq
}

if {$CycleGA != 1} {
set SplitFile [open ${path}/$CycleNumber.fasta w+]
puts $SplitFile ">DNA $CycleNumber"
puts $SplitFile $new
flush $SplitFile
close $SplitFile
set AfterAdjSeq $new
}
}

# cz 1st seq doesnot have static region
if {$CycleNumber > 1} {
set NoLine 1
set Score 0
if {$CycleGA == 1} {
set NewSeqAfterMute [open "${path}/Sequence${SquareType}.txt" r]
set lines [split [read $NewSeqAfterMute] "\n"]
for { set i 0 } { $i <= $CycleNumber-1  } { incr i } {
set t [expr {$i + 1}]
set SplitFile [open ${path}/$t.fasta w+]
puts $SplitFile ">DNA $t"
puts $SplitFile [lindex $lines $i]
flush $SplitFile
close $SplitFile
}
close $NewSeqAfterMute

lappend ComSeq $listseq
set countAdjust 1
set infileDefineSeq [open "${path}/${CycleNumber}_${SquareType}DefineSeq.txt" r]
while {[gets $infileDefineSeq lineDefineSeq] >=0} {
set SeqNum [lindex $lineDefineSeq 0]
set ComplemSeq [lindex $lineDefineSeq 4]
set ComSeqStart [lindex $lineDefineSeq 5]
set ComSeqEnd [lindex $lineDefineSeq 6]
set CurrSeqStart [lindex $lineDefineSeq 2]
set CurrSeqEnd [lindex $lineDefineSeq 3]

if {$SeqNum == $CycleNumber && ![regexp "NIL" $ComplemSeq]} {
if {$countAdjust == 1} {
set CurrSeq [lindex $ComSeq $SeqNum-1]
}
set NewAdj {}
set c [expr $ComplemSeq-1]
set SeqGetCom [lindex $ComSeq $c]
set RegToRev [string range $SeqGetCom $ComSeqStart-1 $ComSeqEnd-1]
set Rev [string reverse $RegToRev]
for {set b 0} {$b < [string length $Rev]} {incr b} {
set Com [string index $Rev $b]
        if {$Com == "c"} {
        lappend NewAdj "g"
        }
        if {$Com == "g"} {
        lappend NewAdj "c"
        }
        if {$Com == "a"} {
        lappend NewAdj "t"
        }
        if {$Com == "t"} {
        lappend NewAdj "a"
        }
incr countAdjust
}
set A [join $NewAdj ""]
set CurrSeq [string replace $CurrSeq $CurrSeqStart-1 $CurrSeqEnd-1 $A]
}
if {$SeqNum == $CycleNumber && [regexp "NIL" $ComplemSeq]} {
set CurrSeq [lindex $ComSeq $SeqNum-1]
}
}
set AfterAdjSeq $CurrSeq
close $infileDefineSeq

set SplitFile [open ${path}/$CycleNumber.fasta w+]
puts $SplitFile ">DNA $CycleNumber"
puts $SplitFile $AfterAdjSeq
flush $SplitFile
close $SplitFile
}

if {$CycleGA != 1} {
set SplitFile [open ${path}/$CycleNumber.fasta w+]
puts $SplitFile ">DNA $CycleNumber"
puts $SplitFile $new
flush $SplitFile
close $SplitFile
}
}

set QuerySeq {}
set TargetSeq {}
set a {}
set b {}
set infileDefineSeq [open "${path}/${CycleNumber}_${SquareType}DefineSeq.txt" r]
while {[gets $infileDefineSeq lineDefineSeq] >=0} {
  set QuerySeq [lindex $lineDefineSeq 0]
  set TargetSeq [lindex $lineDefineSeq 4]
  if {$QuerySeq == $CycleNumber} {
  lappend b $TargetSeq
  }
}
close $infileDefineSeq
set a $CycleNumber
if {$CycleNumber ==1} {
                cd ${path}/RNAstructure/data_tables/
                exec chmod u+rwx ${path}/1.fasta
```

```
                    exec  [auto_execok ${path}/RNAstructure/exe/AllSub] ${path}/1.fasta
                    ${path}/AllSub.ct --DNA

                    set f [open "${path}/AllSub.ct"]
                    set AllSub {}
                    while {[gets $f lineSub] >= 0} {
                    if {[regexp "ENERGY" $lineSub]} {
                    set AllSub $lineSub
                    }
                    }
                    close $f

                    set EnergyAllSub [string range $AllSub 16 20]
                    if {[string length $EnergyAllSub] == 0} {
                    set EnergyAllSub 0
                    }
                    set EnergyAllDuplex 0

                    if {$EnergyAllSub > $EnergyAllDuplex} {
                    }
                    if {$EnergyAllSub < $EnergyAllDuplex} {
                    set Score 0
                    incr Score
                    }
set FinalOutAllSub $EnergyAllSub
set FinalSubDuplex $Score
}

if {$CycleNumber > 1 && ![string equal NIL $b]} {
cd ${path}/RNAstructure/data_tables/
exec chmod u+rwx ${path}/$a.fasta
exec  [auto_execok ${path}/RNAstructure/exe/AllSub] ${path}/$a.fasta ${path}/AllSub.ct --DNA
set f [open "${path}/AllSub.ct"]
          set AllSub {}
          while {[gets $f lineSub] >= 0} {
          if {[regexp "ENERGY" $lineSub]} {
          set AllSub $lineSub
          }
          }
          close $f
          set EnergyAllSub [string range $AllSub 16 20]
          if {[string length $EnergyAllSub] == 0} {
                    set EnergyAllSub 0
          }
set FinalOutAllSub $EnergyAllSub
cd ${path}/RNAstructure/data_tables/
set Tot [llength $b]
set AllDuplex {}
set NoListAllDuplex {}

foreach itemb $b {
exec  [auto_execok ${path}/RNAstructure/exe/DuplexFold] ${path}/$a.fasta ${path}/$itemb.fasta
${path}/DuplexFold.ct --DNA
set d [open "${path}/DuplexFold.ct"]

          while {[gets $d lineduplex] >= 0 } {
          if {[regexp "ENERGY" $lineduplex]} {
          set AllDuplex $lineduplex
          set EnergyAllDuplex [string range $AllDuplex 16 20]
          lappend NoListAllDuplex "$EnergyAllDuplex"
                    }
          }
          close $d
```

```
            if {[string length $EnergyAllDuplex] == 0} {
            set EnergyAllSub 0
            }
            set numberlistDuplex [lsort -real $NoListAllDuplex]
}
            if {$EnergyAllSub > [lindex $numberlistDuplex 0]} {
            }
            if {$EnergyAllSub < [lindex $numberlistDuplex 0]} {
            incr Score
            }
set FinalOutDuplex [lindex $numberlistDuplex 0]
set FinalSubDuplex $Score
set NumberListAllDuplex {}
}
set FinalSubDuplex $Score
if {$CycleNumber ==1} {
set FinalFalseBinding 0
}

if {$CycleNumber != 1} {
set NewSeqAfterMute [open "${path}/Sequence${SquareType}.txt" r]
set linesNewSeq [split [read $NewSeqAfterMute] "\n"]
close $NewSeqAfterMute
set SplitFileTarget [open ${path}/target1.txt a+]

for { set t 0 } { $t <= $CycleNumber-1  } { incr t } {
            puts $SplitFileTarget [lindex $linesNewSeq $t]
            flush $SplitFileTarget
}
close $SplitFileTarget

            if {$CycleGA > 1} {
            set AfterAdjSeq $new
            }
set CStart {}
set CEnd {}

  set infileDefineSeq [open "${path}/${CycleNumber}_${SquareType}DefineSeq.txt" r]
  while {[gets $infileDefineSeq lineDefineSeq] >=0} {
  set QuerySeq [lindex $lineDefineSeq 0]
  set CurrSeqMatchStart [lindex $lineDefineSeq 5]
  set CurrSeqMatchEnd [lindex $lineDefineSeq 6]

            if {$QuerySeq == $CycleNumber && ![string equal NIL $CurrSeqMatchStart]} {
            lappend CStart $CurrSeqMatchStart
            lappend CEnd $CurrSeqMatchEnd
            }
}
close $infileDefineSeq

if { [string length $CStart] != 0} {
set CCycle $CycleNumber
set TotalListCStart "[llength $CStart]"
set RetrieveSeq $AfterAdjSeq
            for { set t 0 } { $t < $TotalListCStart } { incr t } {
            set IntermeRetrieveSeq [string replace $RetrieveSeq [expr [lindex $CStart $t]-1] [expr
            [lindex $CEnd $t]-1]]
            set RetrieveSeq $IntermeRetrieveSeq
            }
}
if { [string length $RetrieveSeq] != 0} {
    set SplitFileQuery [open ${path}/query1.txt w]
```

```
        puts $SplitFileQuery $RetrieveSeq
        flush $SplitFileQuery
        close $SplitFileQuery
        set path $originalPath
        cd ${path}/
        exec ${path}/FindStartPosition.pl
        exec ${path}/CleanEmptyPosition.pl
        exec ${path}/GetLongestComplement.pl
        set grabFinalScoreFalseBinding [open "${path}/FinalScoreFalseBinding.txt" r]
        while {[gets $grabFinalScoreFalseBinding linegrabFalseBinding] >=0} {
        set FinalFalseBinding [lindex $linegrabFalseBinding 0]
        }
        close $grabFinalScoreFalseBinding
}
        if { [string length $RetrieveSeq] == 0} {
        set FinalFalseBinding 0
        }
}

set G4Pattern gggg
set C4Pattern cccc
set NoCytosine c
set NoGuanine g
set scoreCG 0
set scoreCG_G4 0
set tcl_precision 2

proc FindG4 {G4Pattern DnaSeq} {
   return [regexp -all $G4Pattern $DnaSeq]
}
proc FindC4 {C4Pattern DnaSeq} {
   return [regexp -all $C4Pattern $DnaSeq]
}

set GrepG4 [FindG4 $G4Pattern $AfterAdjSeq]
set GrepC4 [FindC4 $C4Pattern $AfterAdjSeq]
set TotalC [regexp -all -- $NoCytosine $AfterAdjSeq]
set lengDna [string length $AfterAdjSeq]
set TotalG [regexp -all -- $NoGuanine $AfterAdjSeq]
set TotalCG [expr $TotalC + $TotalG]
set percentageCG [expr {double($TotalCG)/$lengDna}]

if {$MinCG<=$percentageCG && $percentageCG<=$MaxCG && $GrepG4 == 0} {
}

if {$percentageCG < $MinCG || $percentageCG > $MaxCG || $GrepG4 > 0 || $GrepC4} {
        if {$percentageCG < $MinCG || $percentageCG > $MaxCG} {
        incr scoreCG
        }
set scoreCG_G4 [expr {$scoreCG + $GrepG4 + $GrepC4}]
}


set FinalOutPercentageCG $percentageCG
set CombineStructureFalseBinding_CG_G4 [expr $FinalSubDuplex + $FinalFalseBinding +
$scoreCG_G4]
set infileDefineSeq2 [open "${path}/${CycleNumber}_${SquareType}DefineSeq2.txt" r]

if {$CombineStructureFalseBinding_CG_G4 != 0 && $CycleGA <= $LimitGACycle} {

set no 1
set value 1
```

```
proc range {from to} {
   if {$to>$from} {concat [range $from [incr to -1]] $to}
 }
proc RegionToMutate {WholeRange ForbidListPos} {
   set diff {}
   foreach i $WholeRange {
      if { [lsearch -exact $ForbidListPos $i]==-1} {
         lappend diff $i
      }
   }
   return $diff
}
proc PickMutatePosition PositionList {
   lindex $PositionList [expr {int(rand()*[llength $PositionList])}]
}
proc PickNucleoForReplace Nucleotide {
   lindex $Nucleotide [expr {int(rand()*[llength $Nucleotide])}]
}

while {[gets $infileDefineSeq2 lineDefineSeq2] >=0} {
          set SeqNo [lindex $lineDefineSeq2 0]
          set ForbidStartPos [lindex $lineDefineSeq2 2]
          set ForbidEndPos [lindex $lineDefineSeq2 3]
          set WholeSeqEndPos [lindex $lineDefineSeq2 1]
          set WholeSeqStartPos 1
          if {$SeqNo == $CycleNumber} {

                    if {$SeqNo == $number && [string equal NIL $ForbidEndPos] != 1} {
                    set ForbidEndPosPlus1 [expr $ForbidEndPos + 1 ]
                    set WholeSeqEndPosPlus1 [expr $WholeSeqEndPos + 1 ]
                    set ForbidListPos [range $ForbidStartPos $ForbidEndPosPlus1]
                    set WholeRange [range $WholeSeqStartPos $WholeSeqEndPosPlus1]
                    set PosToMutate [RegionToMutate $WholeRange $ForbidListPos]

                    if  {[ string length $PosToMutate ] == 0 } {
                    set new $AfterAdjSeq
                    set old $AfterAdjSeq
                    }

                    if { [ string length $PosToMutate ] != 0 } {
                    set OriSeq $AfterAdjSeq
                    set PositionList $PosToMutate
                    set MutatePosition [PickMutatePosition $PositionList]
                    if {[string index $OriSeq $MutatePosition] == {a}} {
                    set Nucleotide {c g t}
                    }
                    if {[string index $OriSeq $MutatePosition] == {c}} {
                    set Nucleotide {a g t}
                    }
                    if {[string index $OriSeq $MutatePosition] == {g}} {
                    set Nucleotide {a c t}
                    }
                    if {[string index $OriSeq $MutatePosition] == {t}} {
                    set Nucleotide {a c g}
                    }
                    set NewNucleotide [PickNucleoForReplace $Nucleotide]
                    set NewSeqAfterMutated [string replace $OriSeq $MutatePosition-1
                    $MutatePosition-1 $NewNucleotide]
                    set new $NewSeqAfterMutated
                    set old $AfterAdjSeq
                    }
                    }
                    if {$SeqNo == $number && [string equal NIL $ForbidEndPos] == 1} {
                    set WholeSeqEndPosPlus1 [expr $WholeSeqEndPos + 1 ]
                    set PosToMutate [range $WholeSeqStartPos $WholeSeqEndPosPlus1]

                    if { [ string length $PosToMutate ] != 0 } {
                    set OriSeq $AfterAdjSeq
                    set PositionList $PosToMutate
                    set MutatePosition [PickMutatePosition $PositionList]

                    if {[string index $OriSeq $MutatePosition] == {a}} {
                    set Nucleotide {c g t}
                    }
                    if {[string index $OriSeq $MutatePosition] == {t}} {
                    set Nucleotide {a c g}
                    }
                    if {[string index $OriSeq $MutatePosition] == {c}} {
                    set Nucleotide {a g t}
                    }
                    if {[string index $OriSeq $MutatePosition] == {g}} {
                    set Nucleotide {a c t}
                    }
                    set NewNucleotide [PickNucleoForReplace $Nucleotide]
                    set NewSeqAfterMutated [string replace $OriSeq $MutatePosition-1
                    $MutatePosition-1 $NewNucleotide]
                    set new $NewSeqAfterMutated
                    set old $AfterAdjSeq
                    }
                    }
                    if {$SeqNo != $number && $WholeSeqStartPos == $ForbidStartPos &&
                    $WholeSeqEndPos == $ForbidEndPos} {
                    set new $AfterAdjSeq
                    set old $AfterAdjSeq
                    }
}
}
close $infileDefineSeq2
}
if {$CombineStructureFalseBinding_CG_G4 != 0 && $CycleGA > $LimitGACycle} {
puts "Exceed Threshold: CycleGA=$CycleGA. No solution found at Strand $CycleNumber"
puts "Program Terminated"
set OutputSignal [open "${path}/Signal.txt" w]
puts $OutputSignal "Need to Regenerate Initial seq"
close $OutputSignal
exit
}

if {$CombineStructureFalseBinding_CG_G4 == 0} {
set value 0
set new $AfterAdjSeq
}

if {$CycleNumber > 0} {
set IntermComSeq [lreplace $ComSeq $CycleNumber-1 $CycleNumber-1 $new]
set ComSeq $IntermComSeq
}
}

if {$CycleNumber == 1} {
puts $SummaryFinalSeq "$CycleNumber\t$new\t$CycleGA\t$FinalOutPercentageCG"
}
if {$CycleNumber >1} {
puts $SummaryFinalSeq "$CycleNumber\t$new\t$CycleGA\t$FinalOutPercentageCG"
}
```

```
set FinalSeqAfterMutate [open "${path}/Sequence${SquareType}.txt" a+]
puts $FinalSeqAfterMutate "$new"
close $FinalSeqAfterMutate
incr CycleNumber
}
close $SummaryFinalSeq
        # clean up all existing temporary files run before this
        file delete -force ${path}/output
        file delete -force ${path}/position_query
        file delete -force ${path}/position_query_1
        file delete -force ${path}/position_target
        file delete -force ${path}/position_target_1
        file delete -force ${path}/query
        file delete -force ${path}/query_1
        file delete -force ${path}/query1.txt
        file delete -force ${path}/target
        file delete -force ${path}/target_1
        file delete -force ${path}/target1.txt
        file delete -force ${path}/FinalScoreFalseBinding.txt
        file delete -force ${path}/DuplexFold.ct
        file delete -force ${path}/cycle.txt
        file delete -force ${path}/AllSub.ct
        eval file delete [glob ${path}/FinalSeq_*.txt]
        eval file delete [glob ${path}/*.fasta]
        eval file delete [glob ${path}/*_${SquareType}DefineSeq.txt]
        eval file delete [glob ${path}/*_${SquareType}DefineSeq2.txt]
        eval file delete [glob ${path}/${SquareType}DefineSeq2.txt]


set OutputSignal [open "${path}/Signal.txt" w]
puts $OutputSignal "Done"
close $OutputSignal

puts "Output File for Optimised Sequence:       ${path}/Sequence${SquareType}.txt"
puts "Output File for Sequence Summary:         ${path}/Summary_AfterOptimized_Seq.txt"
```

## **APPENDIX C:** GenerateRandomSeq.pl

```
#!/usr/bin/perl
# Generate random DNA
#  using a random number generator to randomly select bases

use strict;
use warnings;

use Cwd;
use Cwd qw();
# Extract current working directory and pass to the script
my $CurrentDirectory = Cwd::cwd();
my $path = $CurrentDirectory;

my @Length = @_;
my $ColumnSeqNo = @_;
my @SeqNo = @_;

open (READ_SQUARETYPE, "${path}/SquareType.txt") || die "couldn't open the file
(${path}/SquareType.txt)!";
#open THEFILE, "<filename.txt";
my $first_line = <READ_SQUARETYPE>;
my @extractWithoutNewline = split(/\n/, $first_line);
close READ_SQUARETYPE;
my $SquareType = $extractWithoutNewline[0];
print "SquareType= --$SquareType---\n";


open (READ_DEFINE, "${path}/${SquareType}DefineSeq.txt") || die "couldn't open the file
(${path}/${SquareType}DefineSeq.txt)!";
while (my $DefineSeq = <READ_DEFINE>)  {
my @Column = split(/\t/,$DefineSeq);
my $ColumnLength = $Column[1];
my $ColumnSeqNo = $Column[0];
push(@Length,$ColumnLength);
push(@SeqNo,$ColumnSeqNo);
}
close READ_DEFINE;

my @sortedLength = sort { $a <=> $b } @Length;
my @sortedSeqNo = sort { $a <=> $b } @SeqNo;
my $MaxLengthDefineSeq = $sortedLength[-1];
my $MaxSeqNo = $sortedSeqNo[-1];

my $size_of_set = $MaxSeqNo;
my $maximum_length = $MaxLengthDefineSeq;
my $minimum_length = $MaxLengthDefineSeq;

# An array, initialized to the empty list, to store the DNA in
my @random_DNA = (  );

# Seed the random number generator.
# time|$$ combines the current time with the current process id
srand(time|$$);

# And here's the subroutine call to do the real work
@random_DNA = make_random_DNA_set( $minimum_length, $maximum_length,
$size_of_set );
```

```
open (OUTPUTSEQ, ">${path}/RandomSeq.txt");
foreach my $dna (@random_DNA) {
    print OUTPUTSEQ "$dna\n";
}
close OUTPUTSEQ;
exit;

#### Subroutines
#   Accept parameters setting the maximum and minimum length of
#     each string of DNA, and the number of DNA strings to make

sub make_random_DNA_set {
    my($minimum_length, $maximum_length, $size_of_set) = @_;
    my $length;
    my $dna;
    my @set;
    for (my $i = 0; $i < $size_of_set ; ++$i) {
        $length = randomlength ($minimum_length, $maximum_length);
        $dna = make_random_DNA ( $length );
        push( @set, $dna );
    }
return @set;
}

# randomlength
# A subroutine that will pick a random number from
# $minlength to $maxlength, inclusive.

sub randomlength {

    # Collect arguments, declare variables
    my($minlength, $maxlength) = @_;

    # Calculate and return a random number within the
    #  desired interval.
    # Notice how we need to add one to make the endpoints inclusive,
    #  and how we first subtract, then add back, $minlength to
    #  get the random number in the correct interval.
    return ( int(rand($maxlength - $minlength + 1)) + $minlength );
}

# Make a string of random DNA of specified length.
sub make_random_DNA {
    my($length) = @_;
    my $dna;
    for (my $i=0 ; $i < $length ; ++$i) {
        $dna .= randomnucleotide(  );
    }
    return $dna;
}

# Select at random one of the four nucleotides
sub randomnucleotide {
    my(@nucleotides) = ('a', 'c', 'g', 't');
    return randomelement(@nucleotides);
}

# randomly select an element from an array
sub randomelement {
    my(@array) = @_;
    return $array[rand @array];
}
```

# **APPENDIX D:** Adjust_RandomSeq.pl

```
#!/usr/bin/perl
use warnings;
use strict;
use Cwd qw();
my $CurrentDirectory = Cwd::cwd();
my $path = $CurrentDirectory;

open (READ_SQUARETYPE, "${path}/SquareType.txt") || die "couldn't open the file
(${path}/SquareType.txt)!";
#open THEFILE, "<filename.txt";
my $first_line = <READ_SQUARETYPE>;
my @extractWithoutNewline = split(/\n/, $first_line);
close READ_SQUARETYPE;
my $SquareType = $extractWithoutNewline[0];
print "SquareType= --$SquareType---\n";

my $dsg_output = "${path}/RandomSeq.txt";
my $record ;
my @arrfilename;
my $count = 0;
my $DefSeqFilename = "${path}/${SquareType}DefineSeq.txt";
my $recordDef;
my $countDef = 0;
my @arrfilenameDef;

# read raw seq from DSG
open (READFILES, "$dsg_output") || die "couldn't open the file ($dsg_output)!";
while ($record = <READFILES>){
 $arrfilename[$count] = substr($record,0,length($record)-1);
 $count++;
}
close(READFILES);

# Read DefineFile
open (READDEF, "$DefSeqFilename") || die "couldn't open the file ($DefSeqFilename)!";
while ($recordDef = <READDEF>){
 $arrfilenameDef[$countDef] = substr($recordDef,0,length($recordDef)-1);
 $countDef++;
}
close(READDEF);

open (TEMPO_SEQ, ">${path}/Tempo_seq.txt");
# Extract DSGseq with specified length in defineSeq.txt
foreach (my $s=0; $s<$count; $s++) {
my $SPlus1 = $s+1;
                foreach (my $d=0; $d<$countDef; $d++) {
                my $arrayDef = $arrfilenameDef[$d];
                my @colDef = split('      ', $arrayDef);
                        if($SPlus1 == $colDef[0]) {
                        my $SeqLength = $colDef[1];
                        my $ExtractSeqDefLength =  substr ($arrfilename[$s],0,$SeqLength);
                        print TEMPO_SEQ "$ExtractSeqDefLength\n";
                        $d=$countDef;
                        }
                }
}
close(TEMPO_SEQ);

my $Tempo = "${path}/Tempo_seq.txt";
my @arrTempo;
my $recordT;
my $t = 0;
my $countTempo = 0;
my $Seqnt;

# Read seq with the specified length
open (READTEMPO, "$Tempo") || die "couldn't open the file ($Tempo)!";
while ($recordT = <READTEMPO>){
 $arrTempo[$countTempo] = substr($recordT,0,length($recordT)-1);
 $countTempo++;
}
close(READTEMPO);

open (SEQ_AFTER_ADJUSTCOMPLEM, ">${path}/UnOptimized_Seq.txt");

foreach (my $t=0; $t<$countTempo; $t++) {
my $Seqnt = $arrTempo[$t];
my $tPlus1 = $t+1;
          foreach (my $e=0; $e<$countDef; $e++) {
          my $arrayDef = $arrfilenameDef[$e];
          my @colDef = split('      ', $arrayDef);
                    if($tPlus1 == $colDef[0] ) {
                              my $CurrentSeqStartPos = $colDef[2];
                              my $CurrentSeqEndPos = $colDef[3];
                              my $PairSeqNo = $colDef[4];
                              my $PairSeqStartPos = $colDef[5];
                              my $PairSeqEndPos = $colDef[6];
                              ## if NII exist, move to next cz NII cannot extract position
                                        if ($CurrentSeqStartPos =~ /^[+-]?\d+$/ ) {
                                        my $PairSeqNoMinus1 = $PairSeqNo-1;
                                        my $ComplemPairSegment;
                                        my $revComplemPairSegment;
                                        my $DifferencesNtsPairPlus1 = $PairSeqEndPos-
                                        $PairSeqStartPos+1;
                                        my $Pairsegment =  substr
($arrTempo[$PairSeqNoMinus1],$PairSeqStartPos-1,$DifferencesNtsPairPlus1);
                                        my $revcomplemdsegment =
reverse_complement($Pairsegment);
                                        my $DifferencesNts = $colDef[3]-$colDef[2];
                                        my $DifferencesNtsPlus1 = $DifferencesNts+1;
                                        my $ColDef2Minus1 = $colDef[2]-1;
                                        # To replace current strong position
                                        substr($Seqnt, $ColDef2Minus1, $DifferencesNtsPlus1)
                              = "$revcomplemdsegment";
                                        }
                                        }
                    }
                    print SEQ_AFTER_ADJUSTCOMPLEM "$Seqnt\n";
}

close(SEQ_AFTER_ADJUSTCOMPLEM);

sub reverse_complement {
      my $dna = shift;
          # reverse the DNA sequence
          my $revcomp = reverse($dna);
          # complement the reversed DNA sequence
          $revcomp =~ tr/ACGTacgt/TGCAtgca/;
          return $revcomp;
}
unlink glob ("${path}/Tempo_seq.txt");
```

# **APPENDIX E:** FindStartposition.pl

```
#!/usr/bin/perl
use strict;
use warnings;

use Cwd qw();
my $CurrentDirectory = Cwd::cwd();
my $path = $CurrentDirectory;
my $FBS = 6;
my $query_file = "${path}/query1.txt";
my $target_file = "${path}/target1.txt";
my $recordq;
my @arrfilenameq;
my $q=0;
my $recordt;
my @arrfilenamet;
my $t=0;
my $i=0;
my $target;
my @values_target = @_;
my @rev_complem_query2 = @_;

open (READQUERY, "$query_file") || die "couldn't open the file ($query_file)!";
while ($recordq = <READQUERY>){
 $arrfilenameq[$q] = substr($recordq,0,length($recordq)-1);
 $q++;
}
close(READQUERY);

open (READTARGET, "$target_file") || die "couldn't open the file ($target_file)!";
while ($recordt = <READTARGET>){
 $arrfilenamet[$t] = substr($recordt,0,length($recordt)-1);
 $t++;
}
close(READTARGET);

open (QUERY_SEQ, ">${path}/query");
open (TARGET_SEQ, ">${path}/target");
open (POS_QUERY, ">${path}/position_query");
open (POS_TARGET, ">${path}/position_target");

# search the beginner starting sequence within minimum 2 nucleotides
for ($i=0; $i<scalar @arrfilenameq; $i++) {
my $rev_query;
my $rev_complem_query;
my $query = $arrfilenameq[$i];
$rev_query = reverse($query);
$rev_complem_query = complement($rev_query) ;

        for (my $j=0; $j<scalar @arrfilenamet; $j++) {
        my $target = $arrfilenamet[$j];
        my @values_target = split(//,$target);
        my $len_target = scalar(@values_target);
        my @rev_complem_query2 = split(//,$rev_complem_query);
        my $len_query = scalar (@rev_complem_query2);

                for (my $position_query=0; $position_query<$len_query-2; $position_query++) {
                my @start_target = ();
                my @start_query = ();
                print QUERY_SEQ "$rev_complem_query\n";
print TARGET_SEQ "$target\n";
print POS_QUERY "$position_query\n"; #start of query position from left to right, 1 nt at a time
for (my $position=0; $position < scalar (@values_target) ; $position++) {
        if(substr($rev_complem_query,$position_query,$FBS) eq substr($target,$position,$FBS)) {
            push (@start_target,$position);
        }
            }
print POS_TARGET "@start_target\n";
}
}
}

close QUERY_SEQ ;
close TARGET_SEQ ;
close POS_QUERY ;
close POS_TARGET ;

# subroutine for complement
sub complement {
   $_[0] =~ y/CGATcgat/GCTAgcta/;
   return $_[0];
}
```

## **APPENDIX F:** CleanEmptyPosition.pl

```
#!/usr/bin/perl
use strict;
use warnings;
use Cwd qw();
my $CurrentDirectory = Cwd::cwd();
my $path = $CurrentDirectory;
my $query_file = "${path}/query";
my $target_file = "${path}/target";
my $query_position = "${path}/position_query";
my $target_position = "${path}/position_target";
my $recordq;
my @arrfilenameq;
my $q = 0;
my @arrfilenamet;
my $t = 0;
my $recordt;
my @arrfilenamea;
my $recorda;
my $a=0;
my $recordb;
my @arrfilenameb;
my $b=0;

open (READQUERY, "$query_file") || die "couldn't open the file ($query_file)!";
while ($recordq = <READQUERY>){
 $arrfilenameq[$q] = substr($recordq,0,length($recordq)-1);
 $q++;
}
close(READQUERY);

open (READTARGET, "$target_file") || die "couldn't open the file ($target_file)!";
while ($recordt = <READTARGET>){
 $arrfilenamet[$t] = substr($recordt,0,length($recordt)-1);
 $t++;
}
close(READTARGET);

open (READPOSQUERY, "$query_position") || die "couldn't open the file ($query_position)!";
while ($recorda = <READPOSQUERY>){
$arrfilenamea[$a] = substr($recorda,0,length($recorda)-1);
$a++;
}
close(READPOSQUERY);

open (READPOSTARGET, "$target_position") || die "couldn't open the file ($target_position)!";
while ($recordb = <READPOSTARGET>){
$arrfilenameb[$b] = substr($recordb,0,length($recordb)-1);
$b++;
}
close(READPOSTARGET);

open (QUERY_SEQ, ">${path}/query_1");
open (TARGET_SEQ, ">${path}/target_1");
open (POS_QUERY, ">${path}/position_query_1");
open (POS_TARGET, ">${path}/position_target_1");

for (my $i=0; $i<scalar @arrfilenameb; $i++) {

        if ($arrfilenameb[$i] ne '') {
```

```
		print TARGET_SEQ "$arrfilenamet[$i]\n";
		print QUERY_SEQ "$arrfilenameq[$i]\n";
		print POS_QUERY "$arrfilenamea[$i]\n";
		print POS_TARGET "$arrfilenameb[$i]\n";
}
}

close QUERY_SEQ ;
close TARGET_SEQ ;
close POS_QUERY ;
close POS_TARGET ;
```

## **APPENDIX G:** GetLongestComplement.pl

```perl
#!/usr/bin/perl
use strict;
use warnings;
use Cwd qw();
my $CurrentDirectory = Cwd::cwd();
my $path = $CurrentDirectory;
use List::Util qw(min max);
my $query_file = "${path}/query_1";
my $target_file = "${path}/target_1";
my $query_position = "${path}/position_query_1";
my $target_position = "${path}/position_target_1";
my $recordq;
my @arrfilenameq;
my $q=0;
my $recordt;
my @arrfilenamet;
my $t=0;
my @arrfilenamea;
my $recorda;
my $a=0;
my $recordb;
my @arrfilenameb;
my $b=0;
my $start_pointt;
my @EndTarget;
my @EndQuery;
my @cycle_within_same_querypos;
my $end_query_range;
my $ori_query;
my $end_target_range;
my $get_oriquery;
my $lengthQuery;
my $BackOriQuery;
my $linesOutput = 0 ;

open (READQUERY, "$query_file") || die "couldn't open the file ($query_file)!";
while ($recordq = <READQUERY>){
 $arrfilenameq[$q] = substr($recordq,0,length($recordq)-1);
 $q++;
}
close(READQUERY);

open (READTARGET, "$target_file") || die "couldn't open the file ($target_file)!";
while ($recordt = <READTARGET>){
 $arrfilenamet[$t] = substr($recordt,0,length($recordt)-1);
 $t++;
}
close(READTARGET);

open (READPOSQUERY, "$query_position") || die "couldn't open the file ($query_position)!";
while ($recorda = <READPOSQUERY>){
$arrfilenamea[$a] = substr($recorda,0,length($recorda)-1);
$a++;
}
close(READPOSQUERY);

open (READPOSTARGET, "$target_position") || die "couldn't open the file ($target_position)!";
while ($recordb = <READPOSTARGET>){
$arrfilenameb[$b] = substr($recordb,0,length($recordb)-1);
```

```
$b++;
}
close(READPOSTARGET);
open (OUTPUT, ">${path}/output");
my $i;
my $j;
my $u;
for ($i=0; $i<scalar @arrfilenameq; $i++) {
my $target = $arrfilenamet[$i];
my $start_point_query = $arrfilenamea[$i];
my $query = $arrfilenameq[$i];

if ($arrfilenamea[$i] eq '0') {
my @cycle_within_same_querypos = ();
my @everyelement = ();
my @EndTarget = ();
my @EndQuery =();
} else {}
my @len_target = split //, $target ;
my $length_target = scalar @len_target;
my $split_pos_target = $arrfilenameb[$i];
my @after_split_position = split / /, $split_pos_target ;
my @split_position = @after_split_position;

#each position in the array target i.e 0 9 18 48
foreach ($j=0; $j<scalar @split_position; $j++) {
my $start_point_target = $split_position[$j];
my @match_target = @_;
my @match_query = @_;
my @start_point_target = @_;
my @start_point_counter = @_;
my $counter = 1;
my @start_point_query = @_;

for ($u=0; $u<$length_target;$u++) {
 if(substr($query,$start_point_query,$counter) eq substr($target,$start_point_target,$counter)) {
                        push (@match_query,substr($query,$start_point_query,$counter));
                        push (@match_target,substr($target,$start_point_target,$counter));
                        push (@start_point_query,$start_point_target);
                        push (@start_point_target,$start_point_target);
                        push (@start_point_counter,$counter);
                        $counter++;
                        }
                        else {
                        }
}
my $size = (scalar @match_target) - 1;
#get the total number of match nts between query and target
my $total_sequence_target_match = $match_target[$size];
my $LengthTotalSeqTargetMatch = length($total_sequence_target_match);
my $size_pointt = (scalar @start_point_target) - 1;
$start_pointt = $start_point_target[$size_pointt];
my $size_counter = (scalar @start_point_counter) - 1;
my $last_counter = $start_point_counter[$size_counter];
$end_query_range = $start_point_query+$LengthTotalSeqTargetMatch;
$end_target_range = $start_pointt+$LengthTotalSeqTargetMatch;
$get_oriquery = reverse($query);
$ori_query = complement($get_oriquery);
$lengthQuery = length($query);
$BackOriQuery = $lengthQuery-$LengthTotalSeqTargetMatch;
my @lo_coun = @_;
push (@lo_coun,$counter);
my $max_value = max(@lo_coun);
my $match_range = $max_value-1;
my @everyelement = ();
push (@everyelement,($query,',', $target,',',$total_sequence_target_match,',', $start_pointt,',',
$end_target_range,',',$start_point_query,',',$end_query_range,',','Range=',$match_range,';'));
push (@cycle_within_same_querypos,$match_range);
push (@EndTarget, $end_target_range);
push (@EndQuery, $end_query_range) ;
}
my $max_cycle_within_same_querypos = max @cycle_within_same_querypos;
my $numelements = @cycle_within_same_querypos;

if ($numelements == 1) {
}
else {
my $LastTarget = $EndTarget[-1];
my $SecondLastTarget = $EndTarget[-2];
my $LastQuery = $EndQuery[-1];
my $SecondLastQuery = $EndQuery[-2];
print OUTPUT "$ori_query\t$target\t$start_pointt\t$end_target_range\t$start_point_query\t$end_query_range\n";
}
}
my $linesScore0 = 0;
my $NoOfLinesScore0 = "${path}/output";
open (OUTPUTFINALSCORE, ">${path}/FinalScoreFalseBinding.txt");
open (FILESCORE, $NoOfLinesScore0) or die "Can't open '$NoOfLinesScore0'";
$linesScore0++ while (<FILESCORE>);
close FILESCORE;

#So to get the total score: line in false binding sites (filename: output) is given score 1
my $NoOfLinesOutput = "${path}/output";
open (FILEOUTPUT, $NoOfLinesOutput) or die "Can't open '$NoOfLinesOutput'";
$linesOutput++ while (<FILEOUTPUT>);
close FILEOUTPUT;
my $TotalPenaltyForEachRound = $linesOutput-$linesScore0;
print OUTPUTFINALSCORE "$TotalPenaltyForEachRound\n";
close OUTPUT;
close OUTPUTFINALSCORE;

sub complement {
  $_[0] =~ y/CGATcgat/GCTAgcta/;
  return $_[0];
}
```

**APPENDIX H:** MainGraph_theory.tcl

```
#! /usr/local/bin/tclsh
# need to 2 input file eg. 1) SeqDefineSeq.txt, 2)Sequence.txt, SquareType is $Rec
# output file: List_CorrectGraph.txt"

# Get current working directory
proc getScriptDirectory {} {
   set dispScriptFile [file normalize [info script]]
   set scriptFolder [file dirname $dispScriptFile]
   return $scriptFolder
}
set path [getScriptDirectory]
set ReadSquareType [open ${path}/SquareType.txt r]
         while {[gets $ReadSquareType lineReadSquareType] >=0} {
                  set SquareType $lineReadSquareType
         }
close $ReadSquareType

if {[file exists ${path}/FinalLevel] == 1} {
         file delete -force ${path}/FinalLevel
}

set OutputSquareType [open "${path}/SquareType.txt" w]
puts $OutputSquareType "$SquareType"
close $OutputSquareType
set OutputTM [open "${path}/MeltingTemperature.txt" w]
puts $OutputTM "$ThresholdMeltingTemperature"
close $OutputTM
set OutputDNATakeUpRate [open "${path}/DNAUptakeRate.txt" w]
puts $OutputDNATakeUpRate "$DNAUptakeRate"
close $OutputDNATakeUpRate
set OutputThresholdProbabilityEnergy [open "${path}/ThresholdProbabilityEnergy.txt" w]
puts $OutputThresholdProbabilityEnergy "$ThresholdProbabilityEnergy"
close $OutputThresholdProbabilityEnergy
set OutputMax_Iteration [open "${path}/Max_Iteration.txt" w]
puts $OutputMax_Iteration "$Max_Iteration"
close $OutputMax_Iteration

set infileDefineSeq [open "${path}/${SquareType}DefineSeq.txt" r]
set SeqFragPSSM [open "${path}/ListSeqFragPSSM.txt" w]

# LineInMatrix.txt is the representative of each column & row in the matrix
set EachLineinMatrix [open "${path}/LineInMatrix${SquareType}.txt" w]

# get total no of seqs involved
while {[gets $infileDefineSeq lineDefineSeq] >=0} {
set counter 1
set SeqNum [lindex $lineDefineSeq 0]
set ComplemSeq [lindex $lineDefineSeq 4]
lappend CheckSeqList $SeqNum
set PreviousSeq [lindex $CheckSeqList end-1]

         if { $PreviousSeq != $SeqNum} {
         lappend AllSeq $SeqNum
         }
incr counter
}
close $infileDefineSeq
set TotalSeq [lindex $AllSeq end]
# start: get the occurances/ no of fragment for each seqs
```

```
for { set p 0 } { $p < $TotalSeq } { incr p } {
set NoFragmentInASeq 0
set Seq [lindex $AllSeq $p]
set SeqNumStartEnd {}
set ComplemSeqStartEnd {}
set infileDefineSeq2 [open "${path}/${SquareType}DefineSeq.txt" r]
while {[gets $infileDefineSeq2 lineDefineSeq2] >=0} {

set SeqNum [lindex $lineDefineSeq2 0]
set SeqNumStart [lindex $lineDefineSeq2 2]
set SeqNumEnd [lindex $lineDefineSeq2 3]
set ComplemSeq [lindex $lineDefineSeq2 4]
set ComplemSeqStart [lindex $lineDefineSeq2 5]
set ComplemSeqEnd [lindex $lineDefineSeq2 6]

if {![string equal NIL $ComplemSeq]} {
        if {$Seq == $SeqNum} {
        lappend SeqNumStartEnd $SeqNumStart
        lappend SeqNumStartEnd $SeqNumEnd
        incr NoFragmentInASeq
        }
        if {$Seq == $ComplemSeq} {
        lappend ComplemSeqStartEnd $ComplemSeqStart
        lappend ComplemSeqStartEnd $ComplemSeqEnd
        incr NoFragmentInASeq
        }
}
}
}
close $SeqFragPSSM
close $infileDefineSeq

set ExtractSeqFrag [open "${path}/ListSeqFragPSSM.txt" r]
set Matrix [open "${path}/Matrix${SquareType}.txt" w]
set TotalMatrix [open "${path}/TotalMatrix${SquareType}.txt" w]
set FreeEnergy [open "${path}/Tempo1FreeEnergy${SquareType}.txt" w]
set FragmentSeq [open "${path}/FragmentSeq.txt" w]
set InSeq [open "${path}/Sequence${SquareType}.txt" r]
set LineSeq [split [read $InSeq] "\n"]

close $InSeq
set No 1
while {[gets $ExtractSeqFrag lineExtractSeq] >=0} {
set SeqNmber [lindex $lineExtractSeq 0]
set NoOccuranceFragment [lindex $lineExtractSeq 1]
set NoOccuranceFragmentTimes2 [expr $NoOccuranceFragment*2]
set SeqNumberMinus1 [expr $SeqNumber-1]
set SequenceLine [lindex $LineSeq $SeqNumberMinus1]
for { set u 2 } { $u <= $NoOccuranceFragmentTimes2+1 } { incr u} {
set StartFragment [lindex $lineExtractSeq $u]
set EndFragment [lindex $lineExtractSeq $u+1]
set ExtractedFragment [string range $SequenceLine $StartFragment-1 $EndFragment-1]
puts $FragmentSeq "$No $SeqNumber $ExtractedFragment"
puts $EachLineinMatrix "$SeqNumber    $StartFragment $EndFragment"
incr No
incr u
}
}
close $FragmentSeq
close $EachLineinMatrix
set TotalRatio 0
set InFragmentSeq [open "${path}/FragmentSeq.txt" r]
set QueryNo 1
set CountPerQueryRun 0
while {[gets $InFragmentSeq lineInFragmentSeq] >=0} {
set InFragmentSeq2 [open "${path}/FragmentSeq.txt" r]
set TargetNo 1
        while {[gets $InFragmentSeq2 lineInFragmentSeq2] >=0} {
        set LineFragment [lindex $lineInFragmentSeq 2]
        set LineFragment2 [lindex $lineInFragmentSeq2 2]
        set query [open ${path}/query.fasta w+]
        puts $query ">DNA $QueryNo"
        puts $query "$LineFragment"
        flush $query
        close $query
        set target [open ${path}/target.fasta w+]
        puts $target ">DNA $TargetNo"
        puts $target "$LineFragment2"
        flush $target
        close $target
        cd ${path}/RNAstructure/data_tables/
        exec  [auto_execok ${path}/RNAstructure/exe/DuplexFold] ${path}/query.fasta
${path}/target.fasta ${path}/DuplexFold.ct --DNA


        set d [open "${path}/DuplexFold.ct"]

        set NoListAllDuplex {}
        while {[gets $d lineduplex] >= 0 } {
        if {[regexp "ENERGY" $lineduplex]} {
        set AllDuplex $lineduplex
        set EnergyAllDuplex [string range $AllDuplex 16 20]
        lappend NoListAllDuplex "$EnergyAllDuplex"
                }
        }
        close $d

        if {[string length $EnergyAllDuplex] == 0} {
        set EnergyAllDuplex 0
        }
        set numberlistDuplex [lsort -real $NoListAllDuplex]
        set LowestInADuplexRun [lindex $numberlistDuplex 0]

        # check if the lowest value duplex is a negative or not
        if {$LowestInADuplexRun < 0} {
        lappend PerQueryRun "$LowestInADuplexRun"
        incr CountPerQueryRun
        }

        if {$LowestInADuplexRun > 0} {
        lappend PerQueryRun "0"
        puts "Error message !! Positive value found $QueryNo $TargetNo"
        incr CountPerQueryRun
        }
        incr TargetNo
        }
        close $InFragmentSeq2
        #$LowestPerQueryRun is the wanted hybridization
        #$IndividuListPerQueryRun is the duplexfold value for all unwanted and wanted
        set SortPerQueryRun [lsort -real $PerQueryRun]
        set LowestPerQueryRun [lindex $SortPerQueryRun 0]
        puts $FreeEnergy "$PerQueryRun"
        for { set a 0 } { $a < $CountPerQueryRun } { incr a } {
        set IndividuListPerQueryRun [lindex $PerQueryRun $a]
        set RatioProbability [expr double($IndividuListPerQueryRun)/$LowestPerQueryRun]
        set RoundedRatioProb [expr [format "%.2f" $RatioProbability]]
        set TargetNumber [expr $a+1]
        set TotalRatio [expr $TotalRatio+$RatioProbability]
        lappend ListMatrix "$RoundedRatioProb"
        }
        incr QueryNo
        set TotalRatio 0
        set ListMatrix {}
        set PerQueryRun {}
        set CountPerQueryRun 0
}
close $InFragmentSeq
close $TotalMatrix
close $Matrix
close $ExtractSeqFrag
close $FreeEnergy

# START : To remove { , } because List {-6.6 }  so need to become -6.6
exec /bin/bash -c "sed -e 's/\{//g' ${path}/Tempo1FreeEnergy${SquareType}.txt >
${path}/Tempo2FreeEnergy${SquareType}.txt"
exec /bin/bash -c "sed -e 's/\}//g' ${path}/Tempo2FreeEnergy${SquareType}.txt >
${path}/FreeEnergy${SquareType}.txt"

cd ${path}/
eval exec ${path}/Run_Tm.pl
cd ${path}/
eval exec ${path}/Combine_TmMatchPair.tcl
cd ${path}/
eval exec ${path}/FindSegment_TmMatchPair.tcl
cd ${path}/
exec ${path}/Generate_StartNodes.tcl
cd ${path}/
exec ${path}/Run_Level1.tcl
cd ${path}/
exec ${path}/Run_Level2ToN.tcl
cd ${path}/
exec ${path}/Remove_redundacy.tcl
cd ${path}/
exec ${path}/count.tcl
cd ${path}/
exec ${path}/Find_CorrectMatch.tcl

puts "Final output at List_CorrectGraph.txt"
cd ${path}/
file delete -force ${path}/Concentration.txt
file delete -force ${path}/DuplexFold.ct
file delete -force ${path}/Filtered_UnaFold.txt
file delete -force ${path}/FinalePairTm_PSSM.txt
file delete -force ${path}/FinaleToUse.txt
file delete -force ${path}/FinaleToUseTempo1.txt
file delete -force ${path}/FinaleToUseTempo2.txt
file delete -force ${path}/FinalMatchingPairsTm.txt
file delete -force ${path}/FragmentSeq.txt
file delete -force ${path}/LastLevelNoFile.txt
file delete -force ${path}/OutFinaleWithBracket.txt
file delete -force ${path}/OutTmColumn_Part1.txt
file delete -force ${path}/PreviousNodesConcen.txt
file delete -force ${path}/query.fasta
file delete -force ${path}/StartNode${SquareType}.txt
file delete -force ${path}/target.fasta
file delete -force ${path}/ListSeqFragPSSM.txt
file delete -force ${path}/MatchingPair.txt
file delete -force ${path}/MeltingTemperature.txt
file delete -force ${path}/RedundantList.txt
file delete -force ${path}/SquareType.txt
file delete -force ${path}/TempoLineInMatrix${SquareType}.txt
file delete -force ${path}/Tempo1FreeEnergy${SquareType}.txt
file delete -force ${path}/Tempo1StartNode${SquareType}.txt
file delete -force ${path}/Tempo2FreeEnergy${SquareType}.txt
file delete -force ${path}/TotalFile
file delete -force ${path}/TotalMatrix${SquareType}.txt
file delete -force ${path}/Matrix${SquareType}.txt
file delete -force ${path}/Temporary
eval file delete -force [glob ${path}/Level*]
file delete -force ${path}/2ndStepProcessResult
file delete -force ${path}/ProcessResult
file delete -force ${path}/RemovedRepeatedRows
file delete -force ${path}/List_CorrectPair
file delete -force ${path}/MaxCG.txt
file delete -force ${path}/MinCG.txt
file delete -force ${path}/Max_Iteration.txt
file delete -force ${path}/RandomSeq.txt
file delete -force ${path}/Signal.txt
file delete -force ${path}/ThresholdProbabilityEnergy.txt
file delete -force ${path}/DNAUptakeRate.txt
```

## APPENDIX I: Run_Tm.pl

```
#!/usr/bin/perl
#use strict;
#use warnings;
# Output Filename: CompletedSortTmDecreasingOrder.txt

use Cwd;
use Cwd qw();
# Extract current working directory and pass to the script
my $CurrentDirectory = Cwd::cwd();
my $path = $CurrentDirectory;
my $originalpath = $path;

open (READ_SQUARETYPE, "${path}/SquareType.txt") || die "couldn't open the file
(${path}/SquareType.txt)!";
my $first_line = <READ_SQUARETYPE>;
my @extractWithoutNewline = split(/\n/, $first_line);
close READ_SQUARETYPE;
my $SquareType = $extractWithoutNewline[0];
my $file = "${path}/OutUnaFoldCompleted.txt";
my $removed = unlink($file);

# split all sequences in SequenceTILB.txt into different files for each line, 1.txt, 2.txt...14.txt
open (INSEQ, "${path}/Sequence${SquareType}.txt") or die;
my $SeqNo = 1;
while (my $line = <INSEQ>)  {
    my @eachline = split(/\n/, $line);
            open (SPLITSEQ, ">${path}/unafold-3.8/$SeqNo.txt");
    print SPLITSEQ "$eachline[0]\n";
    close SPLITSEQ;
   $SeqNo++;
}
close INSEQ;

# with self hybridization eg. 1-1, 2-2 homodimer
open (OUTPUTMATCHINGPAIR, ">${path}/MatchingPair.txt") || die "couldn't output to file
(MatchingPairTm)!";
my $file2 = "${path}/unafold-3.8/OutUnaFoldCompleted.txt";
my $removed2 = unlink($file2);
my $RemovedOutunafold = unlink("${path}/OutUnaFoldCompleted.txt");
chdir("${path}/unafold-3.8/") or die "cannot change: $!\n";

# run melt.pl
for (my $a=1; $a< $SeqNo; $a++) {
            for (my $b=$a; $b< $SeqNo; $b++) {
            system "chmod", "u+x", "${path}/";
            system ("melt.pl --NA DNA --sodium 1 --C 0.00001 $a.txt $b.txt >>
OutUnaFoldCompleted.txt");
            print OUTPUTMATCHINGPAIR "$a        $b\n";
            }
}
close OUTPUTMATCHINGPAIR;
# END run melt.pl
my $FilteredUnaFold = "${path}/Filtered_UnaFold.txt";
# Start Checking
system ("cp -i ${path}/unafold-3.8/OutUnaFoldCompleted.txt ${path}/");
system ("awk 'NR % 3 == 0' ${path}/OutUnaFoldCompleted.txt > ${path}/Filtered_UnaFold.txt");
my $Seq_MatchingPair = "${path}/MatchingPair.txt";
my $countMatchigPairLine = 0;
my @arrfilename;
my $recordm;

open (READMATCH, "$Seq_MatchingPair") || die "couldn't open the file ($Seq_MatchingPair)!";
while ($recordm = <READMATCH>){
 $arrfilename[$countMatchigPairLine] = substr($recordm,0,length($recordm)-1);
 $countMatchigPairLine++;
}
close(READMATCH);

my $countTmLine = 0;
my @arrfilenameTm;
my $record;

open (READTM, "$FilteredUnaFold") || die "couldn't open the file ($FilteredUnaFold)!";
while ($record = <READTM>){
 $arrfilenameTm[$countTmLine] = substr($record,0,length($record)-1);
 $countTmLine++;
}
close(READTM);
my $countMatchigPairLin;

if ($countMatchigPairLine == $countTmLine) {
}
else {
print "Error in matching pairs and extracted Tm lines, Please check back original file Filename:
OutUnaFoldCompleted.txt\n";
}
# delete all temporary files generated during unafold
for (my $s=1; $s< $SeqNo; $s++) {
unlink glob ("${path}/unafold-3.8/$s.txt*");
}
```

## **APPENDIX J:** Combine_TmMatchpair.tcl

```
#! /usr/local/bin/tclsh
# combine Matching Pair & Tm into 1 file, Output file : CompletedSortTmDecreasingOrder.txt
proc getScriptDirectory {} {
    set dispScriptFile [file normalize [info script]]
    set scriptFolder [file dirname $dispScriptFile]
    return $scriptFolder
}
set Tpath [getScriptDirectory]
set path "${Tpath}/"
set TmList {}
set Pair1List {}
set Pair2List {}
set inputTm [open ${path}Filtered_UnaFold.txt r]
while {[gets $inputTm lineinputTm] >=0} {
            set Tm [lindex $lineinputTm 3]
            lappend TmList $Tm
            }
close $inputTm
set inputMatchPair [open ${path}MatchingPair.txt r]
while {[gets $inputMatchPair lineinputMatchPair] >=0} {
            set Pair1 [lindex $lineinputMatchPair 0]
            set Pair2 [lindex $lineinputMatchPair 1]
            lappend Pair1List $Pair1
            lappend Pair2List $Pair2
}
close $inputMatchPair
set TotalList "[llength $Pair1List]"
set OutCombinedPairTm [open ${path}FinalMatchingPairsTm.txt w]
for { set t 0 } { $t <= $TotalList-1  } { incr t } {
            puts $OutCombinedPairTm "[lindex $Pair1List $t][lindex $Pair2List $t][lindex $TmList $t]" }
close $OutCombinedPairTm
set TmDecreasingOrder {}
set ToSortTmList {}
# Start: Sort column according to Tm on descending order
set inputToSortTm [open ${path}FinalMatchingPairsTm.txt r]
while {[gets $inputToSortTm lineToSortTm] >=0} {
            set ToSortTm [lindex $lineToSortTm 2]
            lappend ToSortTmList $ToSortTm }
close $inputToSortTm
set TmDecreasingOrder [lsort -real -decreasing $ToSortTmList]
puts "$TmDecreasingOrder";

# Step 2: Find the pairs correspond to the Tm value ond descending order
set OutTmColumn_Part1 [open ${path}OutTmColumn_Part1.txt w]
foreach itemb $TmDecreasingOrder {
set inputToSortTm [open ${path}FinalMatchingPairsTm.txt r]
            while {[gets $inputToSortTm lineToSortTm] >=0} {
            set PPair1 [lindex $lineToSortTm 0]
            set PPair2 [lindex $lineToSortTm 1]
            set TTm [lindex $lineToSortTm 2]
                        if {$itemb == $TTm} {
                        puts $OutTmColumn_Part1 "$PPair1      $PPair2$TTm" }
            }
close $inputToSortTm }
close $OutTmColumn_Part1

# step 3: remove redundant line from sorting work.
exec /bin/bash -c {awk '{if (++dup[$0] == 1) print $0;}' "$path"OutTmColumn_Part1.txt >
"$path"CompletedSortTmDecreasingOrder.txt}
```

## **APPENDIX K:** FindSegment_TmMatchPair.tcl

```
#! /usr/local/bin/tclsh
# Output : $db

# Get current working directory
proc getScriptDirectory {} {
   set dispScriptFile [file normalize [info script]]
   set scriptFolder [file dirname $dispScriptFile]
   return $scriptFolder
}
set Tpath [getScriptDirectory]
set path "${Tpath}/"

set ReadSquareType [open ${path}SquareType.txt r]
          while {[gets $ReadSquareType lineReadSquareType] >=0} {
                    set SquareType $lineReadSquareType
          }
close $ReadSquareType


set FileNameLineInMatrix "${path}LineInMatrix${SquareType}.txt"
set FileNameFreeEnergy "${path}FreeEnergy${SquareType}.txt"

if {[file exists ${path}FinalePairTm_PSSM.txt] == 1} {
file delete -force ${path}FinalePairTm_PSSM.txt
}
if {[file exists ${path}OutFinaleWithBracket.txt] == 1} {
file delete -force ${path}OutFinaleWithBracket.txt
}
set FinaleFinale [open ${path}FinalePairTm_PSSM.txt w+]

# get total lines in LineInMatrix.txt to get n number of rows and column in Free Energy
set countline 0
          set inputcount [open $FileNameLineInMatrix r]
          while {[gets $inputcount linecountline] >=0} {
          incr countline
          }

# Step A1: foreach hybridizaton pairs, start with the  highest Tm, go by row..
set inputMeltingTemp [open ${path}CompletedSortTmDecreasingOrder.txt r]
while {[gets $inputMeltingTemp lineMeltingTemp] >=0} {
set AllFinale {}
          set SPair1 [lindex $lineMeltingTemp 0]
          set SPair2 [lindex $lineMeltingTemp 1]
          set STm [lindex $lineMeltingTemp 2]
          set FindRow {}
          set FindColumn {}

          # Step A2: and check which row & column it situated in FreeEnergy Matrix
          set line 0
          set inputFindLine [open $FileNameLineInMatrix r]
          while {[gets $inputFindLine lineinputFindLine] >=0} {
          incr line
                    if {[lindex $lineinputFindLine 0] == $SPair1} {
                    lappend FindRow $line
                    }
                    if {[lindex $lineinputFindLine 0] == $SPair2} {
                    lappend FindColumn $line
                    #puts "FindColumn=$FindColumn"
                    }
```

```
}
set TotalRow [llength $FindRow]
set TotalColumn [llength $FindColumn]
set RepresentvFromEachRowLowest {}
# Get free energy according to columnNo and rowNo in FreeEnergy matrix table
set MatrixRow 0
set FreeEnergyMatrix [open $FileNameFreeEnergy r]
while {[gets $FreeEnergyMatrix lineFreeEnergyMatrix] >=0} {
incr MatrixRow
            for { set y 0 } { $y < $TotalRow} { incr y } {
            set RowNo [lindex $FindRow $y]
            set ListEnergy {}
            set References {}
            set Last [expr $TotalRow-1]
            set LastRow [lindex $FindRow $Last]
                  if { $MatrixRow == $RowNo } {
                  for { set z 0 } { $z < $TotalColumn} { incr z } {
                  set ColumnNo [lindex $FindColumn $z]
                  set ColumnNoMinus1 [expr $ColumnNo-1]
                  set ExtractedFreeEnergy [lindex $lineFreeEnergyMatrix
                  $ColumnNoMinus1]
                  lappend References "$RowNo"
                  lappend References "$ColumnNo"
                  lappend References "$ExtractedFreeEnergy"
                  lappend ListEnergy $ExtractedFreeEnergy
                  }

                  # get Lowest Energy (in different column but same Row
                  set SortEnergyRow [lsort -real $ListEnergy]
                  set LowestEnergyWithinRow [lindex $SortEnergyRow 0]
                  set TotalReferences [llength $References]
                  for { set r 2 } { $r < $TotalReferences} { incr r } {
                        if {[lindex $References $r] ==
                        $LowestEnergyWithinRow} {
                        set RMinus2 [expr $r-2]
                        set RMinus1 [expr $r-1]
                        lappend RepresentvFromEachRowLowest "[lindex
                        $References $RMinus2]"
                        lappend RepresentvFromEachRowLowest "[lindex
                        $References $RMinus1]"
                        lappend RepresentvFromEachRowLowest "[lindex
                        $References $r]"

                        if { $LastRow == [lindex $References $RMinus2] } {

                        # Get Unique Column To Compare In Next Step
                        set CompareSameColumn
                        $RepresentvFromEachRowLowest
                        set LengthCompareSameColumn [llength
                        $RepresentvFromEachRowLowest]
                        set ToSortElementColumn {}
                        for { set s 1 } { $s < $LengthCompareSameColumn} {
                        incr s } {
                        lappend ToSortElementColumn [lindex
                        $RepresentvFromEachRowLowest $s]
                        incr s
                        incr s
                        }
                        set UniqElementInColumn [lsort -unique
                        $ToSortElementColumn]
# START : compare Columns, after get lowest Tm within Rows

 set TotalUniq [llength $UniqElementInColumn]
           foreach itemU $UniqElementInColumn {
           set count 0
           set ListLowColumn {}

if { $itemU == [lindex $RepresentvFromEachRowLowest $u]} {
incr count
           if {$count == 1} {

           lappend ListLowColumn [lindex $RepresentvFromEachRowLowest [expr $u-1]]
           lappend ListLowColumn [lindex $RepresentvFromEachRowLowest $u]
           lappend ListLowColumn [lindex $RepresentvFromEachRowLowest [expr $u+1]]
           }
           if {$count != 1} {
if { [lindex $ListLowColumn 2] < [lindex $RepresentvFromEachRowLowest [expr $u+1]]} {
}
{ [lindex $ListLowColumn 2] > [lindex $RepresentvFromEachRowLowest [expr $u+1]]} {
set ListLowColumn {}
lappend ListLowColumn [lindex $RepresentvFromEachRowLowest [expr $u-1]]
lappend ListLowColumn [lindex $RepresentvFromEachRowLowest $u]
lappend ListLowColumn [lindex $RepresentvFromEachRowLowest [expr $u+1]]
}
if { [lindex $ListLowColumn 2] == [lindex $RepresentvFromEachRowLowest [expr $u+1]] &&
[lindex $ListLowColumn 0] != [lindex $RepresentvFromEachRowLowest [expr $u-1]]} {
lappend ListLowColumn [lindex $RepresentvFromEachRowLowest [expr $u-1]]
lappend ListLowColumn [lindex $RepresentvFromEachRowLowest $u]
ListLowColumn [lindex $RepresentvFromEachRowLowest [expr $u+1]]
                                                                  }
                                                           }
                                                    }
                                             incr u
                                             incr u
                                             }
                                       lappend AllFinale $ListLowColumn
                                       }
                                      }
                                      }
                                      incr r
                                      incr r
                               }
                        }
                 }
           }
           close $FreeEnergyMatrix
           # end of get free energy

# START: Use the FinaleToUse.txt consist of filtered lowest energy of Row,Column
set OutFinaleBracket [open ${path}OutFinaleWithBracket.txt w]
puts $OutFinaleBracket "$AllFinale"
close $OutFinaleBracket
exec /bin/bash -c "sed -e 's/\{//g' ${path}OutFinaleWithBracket.txt
>${path}FinaleToUseTempo1.txt"
exec /bin/bash -c "sed -e 's/\}//g' ${path}FinaleToUseTempo1.txt >
${path}FinaleToUseTempo2.txt"
exec /bin/bash -c {awk '{ printf NF?$0:"\n" }' "$path"FinaleToUseTempo2.txt >
"$path"FinaleToUse.txt}
set in [open ${path}FinaleToUse.txt r]
while {[gets $in linein] >=0} {
}
close $in
set in [open ${path}FinaleToUse.txt r]
```

```
            set LineSeq [split [read $in] " "]
            close $in
            set TotalItem [llength $LineSeq]
            set db {}
            for { set t 0 } { $t < $TotalItem  } { incr t } {
            set TPlus1 [expr $t+1]
            set TPlus2 [expr $t+2]
            if { $t == 0} {
            lappend db [lindex $LineSeq $t]
            lappend db [lindex $LineSeq $TPlus1]
            lappend db [lindex $LineSeq $TPlus2]
            if { $t != 0} {
            set TotalDB [llength $db]
            set Incoming1 [lindex $LineSeq $t]
            set Incoming2 [lindex $LineSeq $TPlus1]
            Incoming3 [lindex $LineSeq $TPlus2]
            for { set d 0 } { $d < $TotalDB } { incr d } {
            if { $Incoming1 != [lindex $db $d] || $Incoming2 != [lindex $db [expr $d+1]] || $Incoming3
!= [lindex $db [expr $d+2]]} {
            if {$d == $TotalDB-3} {
            lappend db $Incoming1
            lappend db $Incoming2
            lappend db $Incoming3
            }
            }
if { $Incoming1 == [lindex $db $d] && $Incoming2 == [lindex $db [expr $d+1]] && $Incoming3
== [lindex $db [expr $d+2]]} {
set d [expr $TotalDB-1]
}
incr d
incr d
}
}
incr t
incr t
}
            puts $FinaleFinale "$SPair1 $SPair2 $STm $db"
            close $inputFindLine
            }
close $inputMeltingTemp
close $FinaleFinale
```

# **APPENDIX L:** Generate_StartNodes.tcl

```
#! /usr/local/bin/tclsh

# Get current working directory
proc getScriptDirectory {} {
   set dispScriptFile [file normalize [info script]]
   set scriptFolder [file dirname $dispScriptFile]
   return $scriptFolder
}
set path [getScriptDirectory]

set ReadSquareType [open ${path}/SquareType.txt r]
         while {[gets $ReadSquareType lineReadSquareType] >=0} {
                  set SquareType $lineReadSquareType
         }
close $ReadSquareType
set ReadDNAUptakeRate [open ${path}/DNAUptakeRate.txt r]
         while {[gets $ReadDNAUptakeRate lineDNAUptakeRate] >=0} {
                  set DNAUptakeRate $lineDNAUptakeRate
         }
close $ReadDNAUptakeRate

set FileNameStartNode "${path}/StartNode${SquareType}.txt"
set FileNameLineInMatrix "${path}/LineInMatrix${SquareType}.txt"

# To change: first Node of DNA start binding is 0.1. Original concentration is 1.0
if {[file exists ${path}/Concentration.txt] == 1} {
file delete -force ${path}/Concentration.txt
}
if {[file exists ${path}/TempoDNAConcentration.txt] == 1} {
file delete -force ${path}/TempoDNAConcentration.txt
}
if {[file exists ${path}/StartNode${SquareType}.txt] == 1} {
file delete -force ${path}/StartNode${SquareType}.txt
}
if {[file exists ${path}/PreviousNodesConcen.txt] == 1} {
file delete -force ${path}/PreviousNodesConcen.txt
}
set StartNode [open $FileNameStartNode w+]
set DNAConcentration [open ${path}/Concentration.txt w]
set PreviousNodeConcen [open ${path}/PreviousNodesConcen.txt w+]

# Step: Set initial DNA concentration to 1.0
set ExtractLine [open $FileNameLineInMatrix r]
while {[gets $ExtractLine lineExtractLine] >=0} {
puts $DNAConcentration  "$lineExtractLine 1.0000"
}
close $ExtractLine
close $DNAConcentration
exec /bin/bash -c "sed -e 's/	/ /g' ${path}/Concentration.txt > ${path}/TempoConcen.txt"
exec /bin/bash -c "mv ${path}/TempoConcen.txt ${path}/Concentration.txt"

# Get the Temperature from Tm to use as Initial start node, ThresholdMeltingTemp is set by
user at list.tcl
set ExtractTm [open ${path}/MeltingTemperature.txt r]
while {[gets $ExtractTm lineExtractTm] >=0} {
set ThresholdMeltingTemp $lineExtractTm
}
close $ExtractTm
puts "ThresholdMeltingTemp = $ThresholdMeltingTemp"
```

```
set ExtractLineTM [open ${path}/CompletedSortTmDecreasingOrder.txt r]
set LineLimitTm 0
while {[gets $ExtractLineTM lineExtractLineTM] >=0} {
set ColumnLineTM [split $lineExtractLineTM "     "]
set MeltingTemp [lindex $ColumnLineTM 2]
puts "MeltingTemp = $MeltingTemp "
        if {$MeltingTemp >= $ThresholdMeltingTemp} {
        incr LineLimitTm
        }
}
close $ExtractTm

set LimitLine $LineLimitTm
set CountLine 1
set ExtractFinal [open ${path}/FinalePairTm_PSSM.txt r]
while {[gets $ExtractFinal lineExtractFinal] >=0} {

if { $CountLine <= $LimitLine } {
        set ColumnFinal [split $lineExtractFinal " "]
        set TotalColumnFinal [llength $ColumnFinal]
        set ListLine {}
        set ExistingPair {}
        set BoundSeg {}
        set SEndFree {}
        for { set y 0 } { $y < $TotalColumnFinal} { incr y } {
        set ToGetLowestEnergy {}
        set ColNo [lindex $lineExtractFinal $y]
                set ExtractLine [open $FileNameLineInMatrix r]
                set LineNo 1
                while {[gets $ExtractLine lineExtractLine] >=0} {
                set Col0 [lindex $lineExtractLine 0]
                # STEP 1: Get The Database LineInMatrix for that corresponded SeqNo
                # Step 1a: Get The Seq No of Pair 1
                if { $y == 0 && $ColNo == $Col0} {
                lappend ListLine $LineNo
                }
                ## Step 1b: Get Seq No of Pair 2
                if { $y == 1 && $ColNo == $Col0} {
                lappend ListLine $LineNo
                }
                incr LineNo
                }
                close $ExtractLine
                }
                for { set c 3 } { $c < $TotalColumnFinal} { incr c } {
                        lappend ExistingP1 [lindex $lineExtractFinal $c]
                        lappend ToGetLowestEnergy "[lindex $lineExtractFinal $c]"
                        lappend ToGetLowestEnergy "[lindex $lineExtractFinal [expr $c+1]]"
                        lappend ToGetLowestEnergy "[lindex $lineExtractFinal [expr $c+2]]"
                        incr c
                        incr c
                }
                # STEP 3: select the lowest energy pair only
                        set SortLowest [lsort -real $ToGetLowestEnergy]
                        set LowestE [lindex $SortLowest 0]
                        set TotalToGetLowestEnergy [llength $ToGetLowestEnergy]
                        if { $TotalToGetLowestEnergy == 3 } {
                                lappend ExistingPair [lindex $ToGetLowestEnergy 0]
                                lappend ExistingPair [lindex $ToGetLowestEnergy 1]
                                }
                        if { $TotalToGetLowestEnergy > 3 } {
                        for { set m 2 } { $m < $TotalToGetLowestEnergy } { incr m } {
                                if { [lindex $ToGetLowestEnergy $m] == $LowestE } {
                                        lappend ExistingPair [lindex $ToGetLowestEnergy [expr $m-2]]
                                        lappend ExistingPair [lindex $ToGetLowestEnergy [expr $m-1]]
                                        }
                                        incr m
                                        incr m
                                        }
                                        }
                                        puts "ListLine = $ListLine"
                                        puts "ExistingPair = $ExistingPair"

                        # STEP 4 : Get the free sticky end and Bound Seq
                        foreach itemL $ListLine {
                                if  { [ lsearch $ExistingPair $itemL] >= 0 }   {
                                lappend BoundSeg $itemL
                                } else { lappend SEndFree $itemL }
                                }
                                set SEndFreeUnique [lsort -unique $SEndFree]
                                lappend ListNode_StickyEnds "$BoundSeg"
                                lappend ListNode_StickyEnds "; $SEndFreeUnique"
                                puts $StartNode "$ListNode_StickyEnds"
                                puts $PreviousNodeConcen "$BoundSeg"
                                set ListNode_StickyEnds {}
# START: Adjust Concentration for BOUNDED SEQ - $BoundSegUnique
set LineNo 1
set DNAConcentration [open ${path}/Concentration.txt r]
set TempoDNAConcentration [open ${path}/TempoDNAConcentration.txt w+]
while {[gets $DNAConcentration lineConcen] >=0} {
        set ConCols [split $lineConcen " "]
        set TotalElementCon [llength $ConCols]
        set TotalConCols [llength $ConCols]
        # Step 3: foreach PairsToReduceConcentration, get the CurrConcentration
        set CurrCon [lindex $ConCols $TotalConCols-1]

        if { $itemBound != $LineNo } {
        #  Step 4: if the PairsToReduceConcentration NO need to reduce concen, therefore
LatestConcentration = CurrConcentration output in TempoConcentration.txt file
        puts $TempoDNAConcentration "$ConCols" }
        if { $itemBound == $LineNo } {
        set LatestConcen [expr $CurrCon-$DNAUptakeRate]
        set RoundedLatestConcen [expr [format "%.4f" $LatestConcen]]

# Step 5: LatestConcentration=CurrConcentration-DNAUptakeRate
        puts $TempoDNAConcentration "$ConCols $RoundedLatestConcen"
        puts "($LineNo) $ConCols $RoundedLatestConcen"
        }
        incr LineNo
        }
        close $DNAConcentration
close $TempoDNAConcentration
exec /bin/bash -c "mv ${path}/TempoDNAConcentration.txt ${path}/Concentration.txt"
}
}
incr CountLine
}
close $ExtractFinal
close $StartNode
close $PreviousNodeConcen
exec /bin/bash -c "sed -e 's/\{//g' ${path}/StartNode${SquareType}.txt >
${path}/Tempo1StartNode${SquareType}.txt"
exec /bin/bash -c "sed -e 's/\}//g' ${path}/Tempo1StartNode${SquareType}.txt >
${path}/Tempo2StartNode${SquareType}.txt"
exec /bin/bash -c "mv ${path}/Tempo2StartNode${SquareType}.txt
${path}/StartNode${SquareType}.txt"
```

## **APPENDIX M:** Run_Level1.tcl

```
#! /usr/local/bin/tclsh
# Get current working directory
proc getScriptDirectory {} {
    set dispScriptFile [file normalize [info script]]
    set scriptFolder [file dirname $dispScriptFile]
    return $scriptFolder
}
set path [getScriptDirectory]

set ReadSquareType [open ${path}/SquareType.txt r]
        while {[gets $ReadSquareType lineReadSquareType] >=0} {
                set SquareType $lineReadSquareType
        }
close $ReadSquareType
set FileNameStartNodes "${path}/StartNode${SquareType}.txt"
set FileNameProbabiMatrix "${path}/Matrix${SquareType}.txt"
set FileNameLineInMatrix "${path}/LineInMatrix${SquareType}.txt"
set FileNameTempo "${path}/Temporary/Tempo.txt"
set FileNameTotalFile "${path}/TotalFile"
set DNAConcentrationFile "${path}/Concentration.txt"
set TempoDNAConcentrationFile "${path}/TempoDNAConcentration.txt"
set DirectoryTempo "${path}/Temporary"
set DirectoryLevel1 "${path}/Level1/"

set ReadDNAUptakeRate [open ${path}/DNAUptakeRate.txt r]
        while {[gets $ReadDNAUptakeRate lineReadDNAUptakeRate] >=0} {
                set DNAUptakeRate $lineReadDNAUptakeRate
        }
close $ReadDNAUptakeRate
set ReadProbabilityEnergy [open ${path}/ThresholdProbabilityEnergy.txt r]
        while {[gets $ReadProbabilityEnergy lineReadProbabilityEnergy] >=0} {
                set ThresholdProbability $lineReadProbabilityEnergy
        }
close $ReadProbabilityEnergy

proc listcomp {a b} {
  set diff {}
  foreach i $a {
    if {[lsearch -exact $b $i]==-1} {
      lappend diff $i
    }
  }
  return $diff
 }

set InitialStartNode [open $FileNameStartNodes r]
set OutNo 0
set TotalLineInLevel1 2

if {[file exists $DirectoryLevel1 ] == 1} {
        file delete -force $DirectoryLevel1
}
file mkdir $DirectoryLevel1

while {[gets $InitialStartNode lineInitialStartNode] >=0} {
set DifferentNodeNo $OutNo
        if {[file exists $DirectoryTempo ] == 1} {
        file delete -force $DirectoryTempo
        }
```

```
file mkdir $DirectoryTempo
set TempoCombinedse {}
set Group [split $lineInitialStartNode ";" ]
set PairGroup [lindex $Group 0]
set SEFree [lindex $Group 1]
set TotalSE [llength $SEFree]
set Pairr {}
for { set x 0 } { $x < [expr [llength $PairGroup]-1]} { incr x } {
lappend Pairr [lindex $PairGroup $x]
}
set CombineAllPrevious [concat $Pairr $SEFree]
if { $TotalSE != 0 } {
# STEP 1 : read Probability for the row corresponding to the Sticky Ends
foreach itemSE $SEFree {
        # Get Energy from Matrix
        set MatrixRow 0
        set ProbabilityMatrix [open $FileNameProbabiMatrix r]
        while {[gets $ProbabilityMatrix lineProbabilityMatrix] >=0} {
        incr MatrixRow
        set TotalColumnProbabilityMatrix [llength $lineProbabilityMatrix]
                if { $itemSE == $MatrixRow} {
                # STEP 2: To get each Column
                for { set c 0 } { $c < $TotalColumnProbabilityMatrix} { incr c } {
                set CPlus1 [expr $c+1]
                # STEP 3: Get THe new Free Sticky Ends resulted from $CPlus1
                set LineNo 0
                set ListLineNo {}
                set SearchLineInMatrix [open $FileNameLineInMatrix r]
                while {[gets $SearchLineInMatrix lineSearchLineInMatrix] >=0} {
                incr LineNo
                set SeqNo [lindex $lineSearchLineInMatrix 0]
                if { $CPlus1 == $LineNo } {
                set SeqNo2 $SeqNo
                }
                                }
                close $SearchLineInMatrix
                set LineNo 0
                set SEndFree {}

                # STEP 4 : get lINES NO that have the same SeqNo
                set SearchLineInMatrix [open $FileNameLineInMatrix r]
                while {[gets $SearchLineInMatrix lineSearchLineInMatrix] >=0} {
                incr LineNo
                if { [lindex $lineSearchLineInMatrix 0] == $SeqNo2 } {
                lappend ListLineNo $LineNo
                }
                }
                close $SearchLineInMatrix

                        foreach itemL $ListLineNo {
                        if  {  $itemL != $CPlus1 }   {
                        lappend SEndFree $itemL
                        }
                        }

        # if probability of occurance > $ThresholdProbability
        if { [lindex $lineProbabilityMatrix $c] > $ThresholdProbability } {
        if { $TotalSE == 1} {
        set BoundedDNA {}
        incr OutNo
        lappend BoundedDNA $itemSE
        lappend BoundedDNA $CPlus1
```

```
                set NewUpcomingBindingDNAToCheck [listcomp $BoundedDNA $SEFree]
                set ListCurrentSolutionConcen {}
                foreach itemToCheck $NewUpcomingBindingDNAToCheck {
                # CHECK IF the current dna concentration if it's enough to bind
                set LineNoConcen 0
                set DNAConcentration [open $DNAConcentrationFile r]
                while {[gets $DNAConcentration LineDNAConcentration] >=0} {
                incr LineNoConcen
                if { $LineNoConcen == $itemToCheck} {
                set SplitConcen [split $LineDNAConcentration " " ]
                set CurrentSolutionConcen [lindex $SplitConcen end]
                lappend ListCurrentSolutionConcen $CurrentSolutionConcen
                }
                }
                close $DNAConcentration
                set OrderListCurrentSolutionConcen [lsort -real $ListCurrentSolutionConcen]
                }
                set LowestCurrentConcenSolution [lindex $OrderListCurrentSolutionConcen 0]

                # if current [dna] > 0, or enough to bind, else do not proceed to binding
                if { $LowestCurrentConcenSolution > 0} {
                set Finale [open ${DirectoryLevel1}${OutNo}.txt w]
                puts $Finale "$lineInitialStartNode"
                puts $Finale "$itemSE $CPlus1 [lindex $lineProbabilityMatrix $c] ; $SEndFree"
                flush $Finale
                close $Finale
                # if used DNA got bind, update total solution concentration
                foreach itemToCheck $NewUpcomingBindingDNAToCheck {
                set LineNoConcen 0
                set DNAConcentration [open $DNAConcentrationFile r]
                set TempoDNAConcentration [open $TempoDNAConcentrationFile w]
                while {[gets $DNAConcentration LineDNAConcentration] >=0} {
                incr LineNoConcen

                if { $LineNoConcen != $itemToCheck} {
                puts $TempoDNAConcentration "$LineDNAConcentration"
                }
                if { $LineNoConcen == $itemToCheck} {
                set SplitConcen [split $LineDNAConcentration " " ]
                set CurrentSolutionConcen [lindex $SplitConcen end]
                set LatestConcen [expr $CurrentSolutionConcen-$DNAUptakeRate]
                RoundedLatestConcen [expr [format "%.4f" $LatestConcen]]
                puts $TempoDNAConcentration
        "$LineDNAConcentration$RoundedLatestConcen"
                }
                }
        close $TempoDNAConcentration
        close $DNAConcentration
        exec /bin/bash -c "mv ${path}/TempoDNAConcentration.txt ${path}/Concentration.txt"
        }
        } else {
                set Finale [open ${DirectoryLevel1}${OutNo}.txt w]
                puts $Finale "$lineInitialStartNode"
                flush $Finale
                close $Finale
                }
                }
                if { $TotalSE > 1} {
                lappend TempoCombinedse "$itemSE"
                lappend TempoCombinedse "$CPlus1"
                lappend TempoCombinedse "[lindex $lineProbabilityMatrix $c]"
                }
                }
```

```
                                        }
                              }
                    }
          close $ProbabilityMatrix
          }
          }

          ## no SE so no need to change concentration of solution.
          if { $TotalSE == 0 } {
          incr OutNo
          set Finale [open ${DirectoryLevel1}${OutNo}.txt w]
          puts $Finale "$lineInitialStartNode"
          close $Finale
}

# STEP 5: output tempo file
if { $TotalSE > 1} {
set TotalTempoCombinedse [llength $TempoCombinedse]
foreach itemSEFr $SEFree {
set List($itemSEFr) {}
          for { set l 0 } { $l < $TotalTempoCombinedse} { incr l } {
                    if { $itemSEFr == [lindex $TempoCombinedse $l]} {
                    lappend  List($itemSEFr) "[lindex $TempoCombinedse $l] [lindex
          $TempoCombinedse [expr $l+1]] [lindex $TempoCombinedse [expr $l+2]]"
                              }
                    incr l
                    incr l
                    }
}


if { $TotalSE == 2} {
set itemSEFr [lindex $SEFree 0]
foreach a $List($itemSEFr) {
          set itemSEFr [lindex $SEFree 1]
          foreach b $List($itemSEFr) {
                    lappend ListForFindse $a
                    lappend ListForFindse $b
                    set Tempo [open $FileNameTempo w]
                    puts $Tempo "$ListForFindse"
                    close $Tempo
                    exec /bin/bash -c "sed -e 's/\{//g' ${path}/Temporary/Tempo.txt >
          ${path}/Temporary/TempoTempo1.txt"
          /bin/bash -c "sed -e 's/\}//g' ${path}/Temporary/TempoTempo1.txt >
${path}/Temporary/TempoTempo.txt"
          exec /bin/bash -c "mv ${path}/Temporary/TempoTempo.txt
${path}/Temporary/Tempo.txt"
                    set TempoIn [open $FileNameTempo r]
                              while {[gets $TempoIn lineTempoIn] >=0} {
                              set StickyendsList $lineTempoIn
                              }
                              close $TempoIn
                              set ListForFindse {}
                              set LookForSeqNo {
                    for { set j 0 } { $j < [llength $StickyendsList]} { incr j } {
                    lappend LookForSeqNo [lindex $StickyendsList $j]
                    lappend LookForSeqNo [lindex $StickyendsList [expr $j+1]]
                    incr j
                    incr j
          }
          set ListLineNoLook {}
          foreach itemO $LookForSeqNo {
```

```
                    set LineNoLook 0
                    set SearchLineInMatrix [open $FileNameLineInMatrix r]
                    while {[gets $SearchLineInMatrix lineSearchLineInMatrix] >=0} {
                    incr LineNoLook
                    if { $itemO == $LineNoLook} {
                    lappend ListLineNoLook [lindex $lineSearchLineInMatrix 0]
                    }
}
close $SearchLineInMatrix
}
set UniqListLineNoLook [lsort -unique $ListLineNoLook]
set ListLi {}

          foreach itemLi $UniqListLineNoLook {
          set LineNoLi 0
          set SearchLineInMatrix [open $FileNameLineInMatrix r]
          while {[gets $SearchLineInMatrix lineSearchLineInMatrix] >=0} {
          incr LineNoLi
          if { [lindex $lineSearchLineInMatrix 0] == $itemLi } {
                    lappend ListLi $LineNoLi
          }
}
$SearchLineInMatrix
}
set UniqListLineNoLook [lsort -unique $LookForSeqNo]
set CombineAllPreviousCurrBound [concat $UniqListLineNoLook $CombineAllPrevious]
set UniqCombineAllPreviousCurrBound [lsort -unique $CombineAllPreviousCurrBound]
set NewStickyEnd [listcomp $ListLi $UniqCombineAllPreviousCurrBound]
set NewUpcomingBindingDNAToCheck [listcomp $UniqListLineNoLook $SEFree]
set ListCurrentSolutionConcen {}
foreach itemToCheck $NewUpcomingBindingDNAToCheck {
          # Check the current dna concentration if it's enough to bind
                    set LineNoConcen 0
                    set DNAConcentration [open $DNAConcentrationFile r]
                    while {[gets $DNAConcentration LineDNAConcentration] >=0} {
                              incr LineNoConcen
                              if { $LineNoConcen == $itemToCheck} {
                              set SplitConcen [split $LineDNAConcentration " " ]
                              set CurrentSolutionConcen [lindex $SplitConcen end]
                              lappend ListCurrentSolutionConcen $CurrentSolutionConcen
                              }
                              }
                    close $DNAConcentration
                    set OrderListCurrentSolutionConcen [lsort -real $ListCurrentSolutionConcen]
                    }
                    set LowestCurrentConcenSolution [lindex $OrderListCurrentSolutionConcen 0]
                    incr OutNo
                    ## If concentration is enough, proceed else print till previous line
                              if { $LowestCurrentConcenSolution > 0} {
                              set Finale [open ${DirectoryLevel1}${OutNo}.txt w]
                                        puts $Finale "$lineInitialStartNod
                                        puts $Finale "$StickyendsList ; $NewStickyEnd"
                                        flush $Finale
                                        close $Finale
                    foreach itemToCheck $NewUpcomingBindingDNAToCheck {
                              set LineNoConcen 0
                              set DNAConcentration [open $DNAConcentrationFile r]
                              set TempoDNAConcentration [open $TempoDNAConcentrationFile w]
                    while {[gets $DNAConcentration LineDNAConcentration] >=0} {
                              incr LineNoConcen
                              if { $LineNoConcen != $itemToCheck} {
                              puts $TempoDNAConcentration "$LineDNAConcentration"
                              }
                              if { $LineNoConcen == $itemToCheck} {
                              set SplitConcen [split $LineDNAConcentration " " ]
                              set CurrentSolutionConcen [lindex $SplitConcen end]
                              set LatestConcen [expr $CurrentSolutionConcen-$DNAUptakeRate]
                              set RoundedLatestConcen [expr [format "%.4f" $LatestConcen]]
                              puts $TempoDNAConcentration "$LineDNAConcentration
                              $RoundedLatestConcen"
                              }
                              }
                              close $TempoDNAConcentration
                              close $DNAConcentration
                              exec /bin/bash -c "mv ${path}/TempoDNAConcentration.txt
                              ${path}/Concentration.txt"
                              }
                              } else {
                              set Finale [open ${DirectoryLevel1}${OutNo}.txt w]
                              puts $Finale "$lineInitialStartNode"
                              flush $Finale
                              close $Finale
                              }
                              }
                    }
          }


if { $TotalSE == 3} {
          set itemSEFr [lindex $SEFree 0]
                    foreach a $List($itemSEFr) {
                              set itemSEFr [lindex $SEFree 1]
                              foreach b $List($itemSEFr) {
                                        set itemSEFr [lindex $SEFree 2]
                                        foreach c $List($itemSEFr) {
                                        lappend ListForFindse $a
                                        lappend ListForFindse $b
                                        lappend ListForFindse $c
                                        set Tempo [open $FileNameTempo w]
                                        puts $Tempo "$ListForFindse"
                                        close $Tempo

exec /bin/bash -c "sed -e 's/\{//g' ${path}/Temporary/Tempo.txt >
${path}/Temporary/TempoTempo1.txt"
exec /bin/bash -c "sed -e 's/\}//g' ${path}/Temporary/TempoTempo1.txt >
${path}/Temporary/TempoTempo.txt"
exec /bin/bash -c "mv ${path}/Temporary/TempoTempo.txt ${path}/Temporary/Tempo.txt"
set TempoIn [open $FileNameTempo r]
          while {[gets $TempoIn lineTempoIn] >=0} {
                    set StickyendsList $lineTempoIn
                    }
                    close $TempoIn
                    set ListForFindse {}
                    set LookForSeqNo {}
                    for { set j 0 } { $j < [llength $StickyendsList]} { incr j } {
                    lappend LookForSeqNo [lindex $StickyendsList $j]
                    lappend LookForSeqNo [lindex $StickyendsList [expr $j+1]]
                    incr j
                    incr j
                    }
                    set ListLineNoLook {}
                    foreach itemO $LookForSeqNo {
                    set LineNoLook 0
                    set SearchLineInMatrix [open $FileNameLineInMatrix r]
```

```
while {[gets $SearchLineInMatrix lineSearchLineInMatrix] >=0} {
        incr LineNoLook
        if { $itemO == $LineNoLook} {
        lappend ListLineNoLook [lindex $lineSearchLineInMatrix 0]
        }
        }
        close $SearchLineInMatrix
        }
        set UniqListLineNoLook [lsort -unique $ListLineNoLook]
        set ListLi {}
        foreach itemLi $UniqListLineNoLook {
        set LineNoLi 0
        set SearchLineInMatrix [open $FileNameLineInMatrix r]
        while {[gets $SearchLineInMatrix lineSearchLineInMatrix] >=0} {
        incr LineNoLi
        if { [lindex $lineSearchLineInMatrix 0] == $itemLi } {
        lappend ListLi $LineNoLi
        }
        }
        close $SearchLineInMatrix
        }
        set UniqListLineNoLook [lsort -unique $LookForSeqNo]
        set CombineAllPreviousCurrBound [concat $UniqListLineNoLook
        $CombineAllPrevious]
        set UniqCombineAllPreviousCurrBound [lsort -unique
        $CombineAllPreviousCurrBound]
        set NewStickyEnd [listcomp $ListLi
        $UniqCombineAllPreviousCurrBound]
        set NewUpcomingBindingDNAToCheck [listcomp $UniqListLineNoLook
        $SEFree]
        set ListCurrentSolutionConcen {}

foreach itemToCheck $NewUpcomingBindingDNAToCheck {
        set LineNoConcen 0
        set DNAConcentration [open $DNAConcentrationFile r]
        while {[gets $DNAConcentration LineDNAConcentration] >=0} {
        incr LineNoConcen
                if { $LineNoConcen == $itemToCheck} {
                set SplitConcen [split $LineDNAConcentration " " ]
                set CurrentSolutionConcen [lindex $SplitConcen end]
                lappend ListCurrentSolutionConcen $CurrentSolutionConcen
                }
        }
close $DNAConcentration
set OrderListCurrentSolutionConcen [lsort -real $ListCurrentSolutionConcen]
}
set LowestCurrentConcenSolution [lindex $OrderListCurrentSolutionConcen 0]
incr OutNo

if { $LowestCurrentConcenSolution > 0} {
        set Finale [open ${DirectoryLevel1}${OutNo}.txt w]
        puts $Finale "$lineInitialStartNode"
        puts $Finale "$StickyendsList ; $NewStickyEnd"
        flush $Finale
        close $Finale

foreach itemToCheck $NewUpcomingBindingDNAToCheck {
        set LineNoConcen 0
        set DNAConcentration [open $DNAConcentrationFile r]
        set TempoDNAConcentration [open $TempoDNAConcentrationFile w]

        while {[gets $DNAConcentration LineDNAConcentration] >=0} {
```

```
                                incr LineNoConcen
                                if { $LineNoConcen != $itemToCheck} {
                                        $TempoDNAConcentration "$LineDNAConcentration"
                                        }
                                if { $LineNoConcen == $itemToCheck} {
                                set SplitConcen [split $LineDNAConcentration " " ]
                                set CurrentSolutionConcen [lindex $SplitConcen end]
                                set LatestConcen [expr $CurrentSolutionConcen-$DNAUptakeRate]
                                set RoundedLatestConcen [expr [format "%.4f" $LatestConcen]]
                                puts $TempoDNAConcentration "$LineDNAConcentration
                                $RoundedLatestConcen"
                                }
                                }
                close $TempoDNAConcentration
                close $DNAConcentration
                exec /bin/bash -c "mv ${path}/TempoDNAConcentration.txt ${path}/Concentration.txt"
                }
                } else {
                set Finale [open ${DirectoryLevel1}${OutNo}.txt w]
                puts $Finale "$lineInitialStartNode"
                flush $Finale
                close $Finale
                }
                }
                }
                }
}

# (until sticky end 5) END of get no of lines in each nodes
}
}
close $InitialStartNode

set NoFile [open $FileNameTotalFile w]
puts $NoFile "$OutNo"
close $NoFile
```

## **APPENDIX N:** Run_Level2ToN.tcl

```
#! /usr/local/bin/tclsh

proc getScriptDirectory {} {
    set dispScriptFile [file normalize [info script]]
    set scriptFolder [file dirname $dispScriptFile]
    return $scriptFolder
}
set path [getScriptDirectory]
set OriginalPath $path

set ReadSquareType [open ${path}/SquareType.txt r]
        while {[gets $ReadSquareType lineReadSquareType] >=0} {
                set SquareType $lineReadSquareType
        }
close $ReadSquareType
set FileNameTotalFile "${path}/TotalFile"
set FileNameMatrixProb "${path}/Matrix${SquareType}.txt"
set FileNameLineInMatrix "${path}/LineInMatrix${SquareType}.txt"
set FileNameDirectoryTempo "${path}/Temporary"
set DNAConcentrationFile "${path}/Concentration.txt"
set TempoDNAConcentrationFile "${path}/TempoDNAConcentration.txt"

set ReadDNAUptakeRate [open ${path}/DNAUptakeRate.txt r]
        while {[gets $ReadDNAUptakeRate lineReadDNAUptakeRate] >=0} {
                set DNAUptakeRate $lineReadDNAUptakeRate
        }
close $ReadDNAUptakeRate

proc listcomp {a b} {
  set diff {}
  foreach i $a {
    if {[lsearch -exact $b $i]==-1} {
      lappend diff $i
    }
  }
  return $diff
 }
set ReadProbabilityEnergy [open ${path}/ThresholdProbabilityEnergy.txt r]
        while {[gets $ReadProbabilityEnergy lineReadProbabilityEnergy] >=0} {
                set ThresholdProbability $lineReadProbabilityEnergy
        }
close $ReadProbabilityEnergy
set ReadMax_Iteration [open ${path}/Max_Iteration.txt r]
        while {[gets $ReadMax_Iteration lineReadMax_Iteration] >=0} {
                set Max_Iteration $lineReadMax_Iteration
        }
close $ReadProbabilityEnergy
set RoundedTotalPenalty 0.0000

for { set e 2 } { $e <= $Max_Iteration } { incr e } {

if { $DepletedDNA == 0} {
set NoFile [open $FileNameTotalFile r]
while {[gets $NoFile lineNoFile] >=0} {
set TotalFile $lineNoFile
}
close $NoFile
set LevelMinus1 [expr $e-1]
if {[file exists ${path}/Level${e}] == 1} {
        file delete -force ${path}/Level${e}
        }
file mkdir ${path}/Level${e}
set OutNo 0
for { set a 1 } { $a <= $TotalFile} { incr a } {
        set PreviousLevel [open ${path}/Level${LevelMinus1}/${a}.txt r]
        while {[gets $PreviousLevel linePreviousLevel] >=0} {
        set LastLine "$linePreviousLevel"
        }
        close $PreviousLevel
        if {[file exists $FileNameDirectoryTempo ] == 1} {
        file delete -force $FileNameDirectoryTempo
        }
        file mkdir $FileNameDirectoryTempo
        set TempoCombinedse {}
        set SplitLine [split $LastLine ";" ]
        set StickyEnd [lindex $SplitLine 1]
        set TotalSENo [llength $StickyEnd]
        set PairGroup [lindex $SplitLine 0]
        set SEFree [lindex $SplitLine 1]
        set TotalSE [llength $SEFree]
        set Pairr {}
        for { set x 0 } { $x < [expr [llength $PairGroup]-1]} { incr x } {
        lappend Pairr [lindex $PairGroup $x]
        lappend Pairr [lindex $PairGroup [expr $x+1]]
        incr x
        incr x
        }
        set CombineAllPrevious [concat $Pairr $SEFree]
if { $TotalSENo != 0 } {
        foreach itemSE $StickyEnd {
        set MatrixRow 0
        set ProbabilityMatrix [open $FileNameMatrixProb r]
                while {[gets $ProbabilityMatrix lineProbabilityMatrix] >=0} {
                incr MatrixRow
                set TotalColumnProbabilityMatrix [llength $lineProbabilityMatrix]
                        if { $itemSE == $MatrixRow} {
                                for { set c 0 } { $c < $TotalColumnProbabilityMatrix} { incr c } {
                                set CPlus1 [expr $c+1]
                                set LineNo 0
                                set ListLineNo {}
                                set SearchLineInMatrix [open $FileNameLineInMatrix r]
                        while {[gets $SearchLineInMatrix lineSearchLineInMatrix] >=0} {
                        incr LineNo
                        set SeqNo [lindex $lineSearchLineInMatrix 0]
                                if { $CPlus1 == $LineNo } {
                                set SeqNo2 $SeqNo
                                }
                                }
                                close $SearchLineInMatrix
                                set LineNo 0
                                set SEndFree {}
                                set SearchLineInMatrix [open $FileNameLineInMatrix r]
                        while {[gets $SearchLineInMatrix lineSearchLineInMatrix] >=0} {
                        incr LineNo
                        if { [lindex $lineSearchLineInMatrix 0] == $SeqNo2 } {
                                        lappend ListLineNo $LineNo
                                        }
                        }
                        close $SearchLineInMatrix

                        foreach itemL $ListLineNo {
                                if {  $itemL != $CPlus1 }   {
                                        lappend SEndFree $itemL
                                        }
                                        }
if { [lindex $lineProbabilityMatrix $c] > $ThresholdProbability } {
                    if { $TotalSE >= 1} {
                    lappend TempoCombinedse "$itemSE"
                    lappend TempoCombinedse "$CPlus1"
                    lappend TempoCombinedse "[lindex $lineProbabilityMatrix $c]"
                    }
}
}
}
}
close $ProbabilityMatrix
}
}
if { $TotalSENo == 0 } {
incr OutNo
          set FinaleL [open ${path}/Level${e}/${OutNo}.txt w]
          set PreviousLevel [open ${path}/Level${LevelMinus1}/${a}.txt r]
          while {[gets $PreviousLevel linePreviousLevel] >=0} {
                    puts $FinaleL "$linePreviousLevel"
          }
          close $PreviousLevel
          close $FinaleL
}

if { $TotalSENo >= 1} {
set TotalTempoCombinedse [llength $TempoCombinedse]
foreach itemSEFr $SEFree {
set List($itemSEFr) {}
          for { set l 0 } { $l < $TotalTempoCombinedse} { incr l } {
                    if { $itemSEFr == [lindex $TempoCombinedse $l]} {
                    lappend  List($itemSEFr) "[lindex $TempoCombinedse $l] [lindex
                    $TempoCombinedse [expr $l+1]] [lindex $TempoCombinedse [expr $l+2]]"
                              }
                    incr l
                    incr l
                    }
}
if { $TotalSENo == 1} {
set SeqToReduceConcentration {}
set itemSEFr [lindex $SEFree 0]
          foreach u $List($itemSEFr) {
          lappend ListForFindse $u
          set Tempo [open ${FileNameDirectoryTempo}/Tempo.txt w]
          puts $Tempo "$ListForFindse"
          close $Tempo

exec /bin/bash -c "sed -e 's/\{//g' ${path}/Temporary/Tempo.txt >
${path}/Temporary/TempoTempo1.txt"
exec /bin/bash -c "sed -e 's/\}//g' ${path}/Temporary/TempoTempo1.txt >
${path}/Temporary/TempoTempo.txt"
exec /bin/bash -c "mv ${path}/Temporary/TempoTempo.txt ${path}/Temporary/Tempo.txt"
set TempoIn [open ${FileNameDirectoryTempo}/Tempo.txt r]
          while {[gets $TempoIn lineTempoIn] >=0} {
                    set NewPairs $lineTempoIn
          }
          close $TempoIn
                    set ListForFindse {}
                    set LookForSeqNo {}
                    for { set j 0 } { $j < [llength $NewPairs]} { incr j } {
                              lappend LookForSeqNo [lindex $NewPairs $j]
```

```
                        lappend LookForSeqNo [lindex $NewPairs [expr $j+1]]
                        lappend SeqToReduceConcentration [lindex $NewPairs [expr $j+1]]
                        incr j
                        incr j
                        }
                        set ListCurrentSolutionConcen {}
                        foreach itemToCheck $SeqToReduceConcentration {
                                set LineNoConcen 0
                                set DNAConcentration [open $DNAConcentrationFile r]
                                while {[gets $DNAConcentration LineDNAConcentration] >=0} {
                                incr LineNoConcen
                                if { $LineNoConcen == $itemToCheck} {
                                set SplitConcen [split $LineDNAConcentration " " ]
                                set CurrentSolutionConcen [lindex $SplitConcen end]
                                lappend ListCurrentSolutionConcen $CurrentSolutionConcen
                                }
                                }
                                close $DNAConcentration
                set OrderListCurrentSolutionConcen [lsort -real $ListCurrentSolutionConcen]
                }
                set LowestCurrentConcenSolution [lindex $OrderListCurrentSolutionConcen 0]
                set ListLineNoLook {}
                        foreach itemO $LookForSeqNo {
                        set LineNoLook 0
                        set SearchLineInMatrix [open $FileNameLineInMatrix r]
                        while {[gets $SearchLineInMatrix lineSearchLineInMatrix] >=0} {
                                incr LineNoLook
                                if { $itemO == $LineNoLook} {
                                lappend ListLineNoLook [lindex $lineSearchLineInMatrix 0]
                                }
                                }
                                close $SearchLineInMatrix
                                }
                                set UniqListLineNoLook [lsort -unique $ListLineNoLook]
                                set ListLi {}
                        foreach itemLi $UniqListLineNoLook {
                        set LineNoLi 0
                        set SearchLineInMatrix [open $FileNameLineInMatrix r]
                        while {[gets $SearchLineInMatrix lineSearchLineInMatrix] >=0} {
                                incr LineNoLi
                        if { [lindex $lineSearchLineInMatrix 0] == $itemLi } {
                                lappend ListLi $LineNoLi
                                }
                                }
                                close $SearchLineInMatrix
                                }
set CombineAllPreviousUsedLineNo [concat $LookForSeqNo $CombineAllPrevious]
set UniqCombineAllPreviousUsedLineNo [lsort -unique $CombineAllPreviousUsedLineNo]
set NewStickyEnd [listcomp $ListLi $UniqCombineAllPreviousUsedLineNo]
incr OutNo
set Finale [open ${path}/Level${e}/${OutNo}.txt w]
set PreviousLevel [open ${path}/Level${LevelMinus1}/${a}.txt r]
        while {[gets $PreviousLevel linePreviousLevel] >=0} {
        puts $Finale "$linePreviousLevel"
        }
        close $PreviousLevel
                if { $LowestCurrentConcenSolution > 0 } {
                        puts $Finale "$NewPairs ; $NewStickyEnd"
                        close $Finale
                        foreach itemToCheck $SeqToReduceConcentration {
                        set LineNoConcen 0
                        set DNAConcentration [open $DNAConcentrationFile r]
                        set TempoDNAConcentration [open $TempoDNAConcentrationFile w]
```

```
                              while {[gets $DNAConcentration LineDNAConcentration] >=0} {
                                        incr LineNoConcen
                              if { $LineNoConcen != $itemToCheck} {
                                        puts $TempoDNAConcentration "$LineDNAConcentration"
                                                  }
                                        if { $LineNoConcen == $itemToCheck} {
          set SplitConcen [split $LineDNAConcentration " " ]
          set CurrentSolutionConcen [lindex $SplitConcen end]
          set LatestConcen [expr $CurrentSolutionConcen-$DNAUptakeRate]
          set RoundedLatestConcen [expr [format "%.4f" $LatestConcen]]
          puts $TempoDNAConcentration "$LineDNAConcentration $RoundedLatestConcen"
          }
          }
          close $TempoDNAConcentration
          close $DNAConcentration
          exec /bin/bash -c "mv ${path}/TempoDNAConcentration.txt ${path}/Concentration.txt"
          }
          } else {
          close $Finale
          }
          }
          }
if { $TotalSENo == 2} {
          SeqToReduceConcentration {}
          itemSEFr [lindex $SEFree 0]
          foreach k $List($itemSEFr) {
                    itemSEFr [lindex $SEFree 1]
                    foreach l $List($itemSEFr) {
                              lappend ListForFindse $k
                              lappend ListForFindse $l
                              set Tempo [open ${FileNameDirectoryTempo}/Tempo.txt w]
                              puts $Tempo "$ListForFindse"
                              close $Tempo
exec /bin/bash -c "sed -e 's/\{//g' ${path}/Temporary/Tempo.txt >
${path}/Temporary/TempoTempo1.txt"
exec /bin/bash -c "sed -e 's/\}//g' ${path}/Temporary/TempoTempo1.txt >
${path}/Temporary/TempoTempo.txt"
exec /bin/bash -c "mv ${path}/Temporary/TempoTempo.txt ${path}/Temporary/Tempo.txt"
set TempoIn [open ${FileNameDirectoryTempo}/Tempo.txt r]
          while {[gets $TempoIn lineTempoIn] >=0} {
          set NewPairs $lineTempoIn
          }
          close $TempoIn
          set ListForFindse {}
          set LookForSeqNo {}
          for { set j 0 } { $j < [llength $NewPairs]} { incr j } {
                    lappend LookForSeqNo [lindex $NewPairs $j]
                    lappend LookForSeqNo [lindex $NewPairs [expr $j+1]]
                    lappend SeqToReduceConcentration [lindex $NewPairs [expr $j+1]]
                    incr j
                    incr j
                    }

                    set ListCurrentSolutionConcen {}
                    foreach itemToCheck $SeqToReduceConcentration {
                    set LineNoConcen 0
                    set DNAConcentration [open $DNAConcentrationFile r]
                    while {[gets $DNAConcentration LineDNAConcentration] >=0} {
                              incr LineNoConcen
                              if { $LineNoConcen == $itemToCheck} {
                                        set SplitConcen [split $LineDNAConcentration " " ]
                                        set CurrentSolutionConcen [lindex $SplitConcen end]
                                        lappend ListCurrentSolutionConcen $CurrentSolutionConcen
                                        }
                                        }
                                        close $DNAConcentration
            set OrderListCurrentSolutionConcen [lsort -real $ListCurrentSolutionConcen]
            }
            set LowestCurrentConcenSolution [lindex $OrderListCurrentSolutionConcen 0]
            set ListLineNoLook {}
                        foreach itemO $LookForSeqNo {
                        set LineNoLook 0
                        set SearchLineInMatrix [open $FileNameLineInMatrix r]
                                    while {[gets $SearchLineInMatrix lineSearchLineInMatrix] >=0} {
                                    incr LineNoLook
                                                if { $itemO == $LineNoLook} {
                                                lappend ListLineNoLook [lindex $lineSearchLineInMatrix 0]
                                                }
                                                }
                                                close $SearchLineInMatrix
                                                }
                                                set UniqListLineNoLook [lsort -unique $ListLineNoLook]
                                                set ListLi {}
                        foreach itemLi $UniqListLineNoLook {
                                    set LineNoLi 0
                                    set SearchLineInMatrix [open $FileNameLineInMatrix r]
                                    while {[gets $SearchLineInMatrix lineSearchLineInMatrix] >=0} {
                                                incr LineNoLi
                                                if { [lindex $lineSearchLineInMatrix 0] == $itemLi } {
                                                            lappend ListLi $LineNoLi
                                                            }
                                                            }
                                                            close $SearchLineInMatrix
                                                            }
set CombineAllPreviousUsedLineNo [concat $LookForSeqNo $CombineAllPrevious]
set UniqCombineAllPreviousUsedLineNo [lsort -unique $CombineAllPreviousUsedLineNo]
set NewStickyEnd [listcomp $ListLi $UniqCombineAllPreviousUsedLineNo]
incr OutNo
set Finale [open ${path}/Level${e}/${OutNo}.txt w]
set PreviousLevel [open ${path}/Level${LevelMinus1}/${a}.txt r]
            while {[gets $PreviousLevel linePreviousLevel] >=0} {
            puts $Finale "$linePreviousLevel"
            }
            close $PreviousLevel
            if { $LowestCurrentConcenSolution > 0 } {
                        puts $Finale "$NewPairs ; $NewStickyEnd"
                        close $Finale
            foreach itemToCheck $SeqToReduceConcentration {
            set LineNoConcen 0
            set DNAConcentration [open $DNAConcentrationFile r]
            set TempoDNAConcentration [open $TempoDNAConcentrationFile w]
                        while {[gets $DNAConcentration LineDNAConcentration] >=0} {
                        incr LineNoConcen
                        if { $LineNoConcen != $itemToCheck} {
                        puts $TempoDNAConcentration "$LineDNAConcentration"
                        }
                        if { $LineNoConcen == $itemToCheck} {

            set SplitConcen [split $LineDNAConcentration " " ]
            set CurrentSolutionConcen [lindex $SplitConcen end]
            set LatestConcen [expr $CurrentSolutionConcen-$DNAUptakeRate]
            set RoundedLatestConcen [expr [format "%.4f" $LatestConcen]]
            puts $TempoDNAConcentration "$LineDNAConcentration $RoundedLatestConcen"
            }
}
close $TempoDNAConcentration
close $DNAConcentration
exec /bin/bash -c "mv ${path}/TempoDNAConcentration.txt ${path}/Concentration.txt"
}
# END : if used DNA got bind, update total solution concentration
} else {
close $Finale
}
}
}
}
if { $TotalSENo == 3} {
        set SeqToReduceConcentration {}
        set itemSEFr [lindex $SEFree 0]
        foreach k $List($itemSEFr) {
        set itemSEFr [lindex $SEFree 1]
                foreach l $List($itemSEFr) {
                        set itemSEFr [lindex $SEFree 2]
                        foreach m $List($itemSEFr) {
                        lappend ListForFindse $k
                        lappend ListForFindse $l
                        lappend ListForFindse $m
                        set Tempo [open ${FileNameDirectoryTempo}/Tempo.txt w]
                        puts $Tempo "$ListForFindse"
                        close $Tempo
        exec /bin/bash -c "sed -e 's/\{//g' ${path}/Temporary/Tempo.txt >
${path}/Temporary/TempoTempo1.txt"
exec /bin/bash -c "sed -e 's/\}//g' ${path}/Temporary/TempoTempo1.txt >
${path}/Temporary/TempoTempo.txt"
exec /bin/bash -c "mv ${path}/Temporary/TempoTempo.txt ${path}/Temporary/Tempo.txt"
set TempoIn [open ${FileNameDirectoryTempo}/Tempo.txt r]
        while {[gets $TempoIn lineTempoIn] >=0} {
        set NewPairs $lineTempoIn
        }
        close $TempoIn
set ListForFindse {}
set LookForSeqNo {}
        for { set j 0 } { $j < [llength $NewPairs]} { incr j } {
        lappend LookForSeqNo [lindex $NewPairs $j]
                lappend LookForSeqNo [lindex $NewPairs [expr $j+1]]
                lappend SeqToReduceConcentration [lindex $NewPairs [expr $j+1]]
                incr j
                incr j
                }
                set ListCurrentSolutionConcen {}
                        foreach itemToCheck $SeqToReduceConcentration {
                        set LineNoConcen 0
                        set DNAConcentration [open $DNAConcentrationFile r]
                        while {[gets $DNAConcentration LineDNAConcentration] >=0} {
                        incr LineNoConcen
                                if { $LineNoConcen == $itemToCheck} {
                                set SplitConcen [split $LineDNAConcentration " " ]
                                set CurrentSolutionConcen [lindex $SplitConcen end]
                                lappend ListCurrentSolutionConcen $CurrentSolutionConce
                                }
                                }
                close $DNAConcentration
                set OrderListCurrentSolutionConcen [lsort -real $ListCurrentSolutionConcen]
                }
                set LowestCurrentConcenSolution [lindex $OrderListCurrentSolutionConcen 0]
                set ListLineNoLook {}
foreach itemO $LookForSeqNo {
                set LineNoLook 0
                set SearchLineInMatrix [open $FileNameLineInMatrix r]
```

```
                    while {[gets $SearchLineInMatrix lineSearchLineInMatrix] >=0} {
                    incr LineNoLook
                    if { $itemO == $LineNoLook} {
                              lappend ListLineNoLook [lindex $lineSearchLineInMatrix 0]
                              }
                              }
                              close $SearchLineInMatrix
                              }
                              set UniqListLineNoLook [lsort -unique $ListLineNoLook]
                              set ListLi {}
                              foreach itemLi $UniqListLineNoLook {
                                        set LineNoLi 0
                                        set SearchLineInMatrix [open $FileNameLineInMatrix r]
                              while {[gets $SearchLineInMatrix lineSearchLineInMatrix] >=0} {
                              LineNoLi
                                        if { [lindex $lineSearchLineInMatrix 0] == $itemLi } {
                                                  lappend ListLi $LineNoLi
                                                  }
                                                  }
                                                  close $SearchLineInMatrix
                                                  }
set CombineAllPreviousUsedLineNo [concat $LookForSeqNo $CombineAllPrevious]
set UniqCombineAllPreviousUsedLineNo [lsort -unique $CombineAllPreviousUsedLineNo]
set NewStickyEnd [listcomp $ListLi $UniqCombineAllPreviousUsedLineNo]
incr OutNo
set Finale [open ${path}/Level${e}/${OutNo}.txt w]
set PreviousLevel [open ${path}/Level${LevelMinus1}/${a}.txt r]
          while {[gets $PreviousLevel linePreviousLevel] >=0} {
          puts $Finale "$linePreviousLevel"
          }
          close $PreviousLevel
                    if { $LowestCurrentConcenSolution > 0 } {
                    puts $Finale "$NewPairs ; $NewStickyEnd"
                    close $Finale
foreach itemToCheck $SeqToReduceConcentration {
          set LineNoConcen 0
          set DNAConcentration [open $DNAConcentrationFile r]
          set TempoDNAConcentration [open $TempoDNAConcentrationFile w]
                              while {[gets $DNAConcentration LineDNAConcentration] >=0} {
                              incr LineNoConcen
                              if { $LineNoConcen != $itemToCheck} {
                                        puts $TempoDNAConcentration "$LineDNAConcentration"
                              }
          if { $LineNoConcen == $itemToCheck} {
          set SplitConcen [split $LineDNAConcentration " " ]
          set CurrentSolutionConcen [lindex $SplitConcen end]
          set LatestConcen [expr $CurrentSolutionConcen-$DNAUptakeRate]
          set RoundedLatestConcen [expr [format "%.4f" $LatestConcen]]
          puts $TempoDNAConcentration "$LineDNAConcentration $RoundedLatestConcen"
          }
          }
          close $TempoDNAConcentration
          close $DNAConcentration
          exec /bin/bash -c "mv ${path}/TempoDNAConcentration.txt ${path}/Concentration.txt"
          }
          } else {
          close $Finale
          }
          }
          }
          }
}
}
```

```
# (until stickyend == 10)START: check overall [DNA] if any of the [DNA strand]< 0.0000 stop
set CurrentSolutionConcen 0
set LineNoCheck 0
set DepletedDNA 0
set DNAConcentration [open $DNAConcentrationFile r]
        while {[gets $DNAConcentration LineDNAConcentration] >=0} {
        incr LineNoCheck
        set SplitConcen [split $LineDNAConcentration " " ]
        set CurrentSolutionConcen [lindex $SplitConcen end]
                if { $CurrentSolutionConcen > 0.0000 }  {
                set LastLevel $e
                } else {
                incr DepletedDNA
                }
        }
close $DNAConcentration
set NoFile [open $FileNameTotalFile w]
puts $NoFile "$OutNo"
close $NoFile
} else { }
}
incr OutNo
set Finale [open ${path}/Level${LastLevel}/${OutNo}.txt w]
puts $Finale "at least 1 identical files"
close $Finale
set NoFile [open $FileNameTotalFile w]
puts $NoFile "$OutNo"
close $NoFile

# get the last generated level folder
set LastLevelFilename [open ${path}/LastLevelNoFile.txt w]
puts $LastLevelFilename "$LastLevel"
close $LastLevelFilename
```

## APPENDIX O: Remove_redundancy.tcl

```
#! /usr/local/bin/tclsh
# remove redundancy of consecutive rows and identical files

# Get current working directory
proc getScriptDirectory {} {
   set dispScriptFile [file normalize [info script]]
   set scriptFolder [file dirname $dispScriptFile]
   return $scriptFolder
}
set path [getScriptDirectory]
set OriginalPath $path

file delete -force ${path}/RemovedRepeatedRows

set ReadTotalLastLevel [open ${path}/TotalFile r]
         while {[gets $ReadTotalLastLevel lineReadTotalLastLevel] >=0} {
                  set TotalFileInLastLevel $lineReadTotalLastLevel
         }
close $ReadTotalLastLevel
set ReadLastLevel [open ${path}/LastLevelNoFile.txt r]
         while {[gets $ReadLastLevel lineReadLastLevel] >=0} {
                  set LastLevel $lineReadLastLevel
         }
close $ReadLastLevel
file mkdir ${path}/RemovedRepeatedRows
set Step1FileNo 0
for { set g 1 } { $g <= $TotalFileInLastLevel } { incr g } {
set Line 0
set ConsecutiveOccurance 0
set ListOne {}
set ListTwo {}
                  set ReadLastLevel [open ${path}/Level${LastLevel}/$g.txt r]
                  while {[gets $ReadLastLevel lineReadLastLevel] >=0} {
                  lappend ListOne $lineReadLastLevel
                  incr Line
                  }
                  close $ReadLastLevel
                  set ListTwo $ListOne
                  # Loop into that particular 1.txt, 2.txt etc...
                  for { set a 0 } { $a < $Line } { incr a } {
                  set IdenticalRow 0
                  for { set b 0 } { $b < $Line } { incr b } {
                           if {[lindex $ListOne $a] == [lindex $ListTwo $b] && [lindex $ListOne
                           $a+1] == [lindex $ListTwo $b+1]} {
                           incr IdenticalRow
                           }
                           }
                           if {$IdenticalRow >= 2} {
                           incr ConsecutiveOccurance
                           }
                  }
if { $ConsecutiveOccurance >= 1} {
} else {
incr Step1FileNo
file copy -force ${path}/Level${LastLevel}/$g.txt
${path}/RemovedRepeatedRows/$Step1FileNo.txt
}
}
# Remove the redundant of repeated row
```

```
cd ${path}/RemovedRepeatedRows
exec /bin/bash -c {find . \! -type d -exec cksum {} \; | sort | tee /tmp/f.tmp | cut -f 1,2 -d ' ' | uniq -d
| grep -hif - /tmp/f.tmp > RedundantList.txt}
file rename -force ${path}/RemovedRepeatedRows/RedundantList.txt
${OriginalPath}/RedundantList.txt
cd ${OriginalPath}
set path $OriginalPath
if {[file exists ${path}/FinalLevel] == 1} {
        file delete -force ${path}/FinalLevel
}

# Remove identical files/redundant files and output after filtered is at Folder ${path}/FinalLevel
set ListCol0 {}
set input_RedundantList [open ${path}/RedundantList.txt r]
        while {[gets $input_RedundantList lineinput_RedundantList] >=0} {
        set SplitLineRedun [split $lineinput_RedundantList " " ]
        set RCol0 [lindex $SplitLineRedun 0]
        lappend ListCol0 $RCol0
        }
close $input_RedundantList
set UniqueList [lsort -real -unique $ListCol0]
set TotalElement "[llength $UniqueList]"

# Output redundant files Name into RedundantList_ToRemove.txt
set CountToRemoveFile 0
set Outremove {}
for { set r 0 } { $r < $TotalElement } { incr r } {
set UCol0 [lindex $UniqueList $r]
set countList 0
        set input_RedundantList2 [open ${path}/RedundantList.txt r]
        while {[gets $input_RedundantList2 lineinput_RedundantList2] >=0} {
        set SplitLineRedun2 [split $lineinput_RedundantList2 " " ]
        set R2Col0 [lindex $SplitLineRedun2 0]
        set R2Col2 [lindex $SplitLineRedun2 2]
                        if { $UCol0 == $R2Col0} {
                        incr countList
                                if {$countList != 1} {
                                set SplitDot [split $R2Col2 "/" ]
                                lappend Outremove "[lindex $SplitDot 1]"
                                incr CountToRemoveFile
                                }
                        }
        }
        close $input_RedundantList2
}

set LastLevelFileNo [open ${path}/LastLevelNoFile.txt r]
                while {[gets $LastLevelFileNo lineLastLevelFileNo] >=0} {
                set LastLevelNo $lineLastLevelFileNo
                }
close $LastLevelFileNo
file mkdir ${path}/FinalLevel
set NewFileNo 1
for { set c 1 } { $c < $Step1FileNo-1 } { incr c } {
set OldFilename "$c.txt"
        if {[lsearch $Outremove $OldFilename] >= 0} {
        } else {
        file copy -force ${path}/RemovedRepeatedRows/$c.txt
${path}/FinalLevel/$NewFileNo.txt
        incr NewFileNo
        }
}
```

## **APPENDIX P:** count.tcl

```
#! /usr/local/bin/tclsh

# Get current working directory
proc getScriptDirectory {} {
   set dispScriptFile [file normalize [info script]]
   set scriptFolder [file dirname $dispScriptFile]
   return $scriptFolder
}
set path [getScriptDirectory]
set ReadSquareType [open ${path}/SquareType.txt r]
         while {[gets $ReadSquareType lineReadSquareType] >=0} {
                  set SquareType $lineReadSquareType
         }
close $ReadSquareType
set DirectoryFinalLevel "${path}/FinalLevel"
set FinalFilesLastLevel [glob "${DirectoryFinalLevel}/*.txt"]
set TotalFileNo [llength $FinalFilesLastLevel]
set DirectoryProcessResult "${path}/ProcessResult/"
if {[file exists ${DirectoryProcessResult}] == 1} {
         file delete -force ${DirectoryProcessResult}
}
file mkdir $DirectoryProcessResult
for { set a 1 } { $a <= $TotalFileNo } { incr a } {
set LineNo 1
         set LastFile [open "${DirectoryFinalLevel}/$a.txt" r]
         set InProcessResult [open "${DirectoryProcessResult}/$a.txt" w+]

         while {[gets $LastFile LineLastFile] >=0} {
         set SplitBoundedSegSE [split $LineLastFile ";" ]
         set BoundedSegList [lindex $SplitBoundedSegSE 0]
         set TotalBoundedSegList [llength $BoundedSegList]
         for { set b 0 } { $b < $TotalBoundedSegList } { incr b } {
         if { $LineNo == 1 } {
         puts $InProcessResult "[lindex $BoundedSegList $b] [lindex $BoundedSegList [expr
$b+1]] 1.0"
         }
         if { $LineNo != 1 } {
         puts $InProcessResult "[lindex $BoundedSegList $b] [lindex $BoundedSegList [expr
$b+1]] [lindex $BoundedSegList [expr $b+2]]"
         }
                  incr LineNo
                  incr b
                  incr b
                  }
         }
         close $LastFile
         close $InProcessResult

         incr LineNo
}

# START: 2nd Step Filter redundancy
set DirectoryProcessResult "${path}/ProcessResult/"
set FileProcessResult [glob "${DirectoryProcessResult}/*.txt"]
set TotalFileNoProcessResult [llength $FileProcessResult]
set FilteredDirectory "${path}/2ndStepProcessResult/"

if {[file exists ${FilteredDirectory}] == 1} {
         file delete -force ${FilteredDirectory}
}

file mkdir $FilteredDirectory
for { set w 1 } { $w <= $TotalFileNoProcessResult } { incr w } {
                  set Reference {}
                  set  FileNoProcessResult [open ${DirectoryProcessResult}$w.txt r]
                  while {[gets $FileNoProcessResult lineFileNoProcessResult] >=0} {
                  set SplitCols [split $lineFileNoProcessResult " " ]
                  set Col0 [lindex $SplitCols 0]
                  set Col1 [lindex $SplitCols 1]
                  set Forward "$Col0 $Col1"
                  set Reverse "$Col1 $Col0"
                            if { $Reference == {} } {
                            lappend Reference $Forward
                            }
                            set NotSame 0
                            if { $Reference != {}} {
                                                foreach item $Reference {
                                                set TotalRef [llength $Reference]

                                                if { $Forward != $item && $Reverse != $item} {
                                                incr NotSame
                                                }
                                                if { $NotSame == $TotalRef} {
                                                lappend Reference $Forward
                                                }
                                                if { $NotSame == $TotalRef} {
                                                }

                                                }
                            }
                  }
close $FileNoProcessResult

                  set  FilteredFileName [open ${FilteredDirectory}/$w.txt w+]
                  foreach itemR $Reference {
                  set Found 0
                            set  FileNoProcessResult [open "${path}/ProcessResult/$w.txt" r]
                            while {[gets $FileNoProcessResult lineFileNoProcessResult] >=0} {

                            set SplitCols [split $lineFileNoProcessResult " " ]
                            set ColR0 [lindex $SplitCols 0]
                            set ColR1 [lindex $SplitCols 1]
                            set ColR2 [lindex $SplitCols 2]
                            set ForwardR "$ColR0 $ColR1"
                                      if { $Found == 0 && $itemR == $ForwardR } {
                                      puts $FilteredFileName "$ColR0 $ColR1 $ColR2"
                                      incr Found
                                      }
                            }
                            close $FileNoProcessResult
                  }
                  close $FilteredFileName
}
```

## **APPENDIX Q:** Find_CorrectMatch.tcl

```
#! /usr/local/bin/tclsh
# final output of list of correct match is at $List_CorrectGraph

# Get current working directory
proc getScriptDirectory {} {
   set dispScriptFile [file normalize [info script]]
   set scriptFolder [file dirname $dispScriptFile]
   return $scriptFolder
}
set path [getScriptDirectory]
set ReadSquareType [open ${path}/SquareType.txt r]
          while {[gets $ReadSquareType lineReadSquareType] >=0} {
                    set SquareType $lineReadSquareType
          }
close $ReadSquareType
set ExtractTm [open ${path}/MeltingTemperature.txt r]
while {[gets $ExtractTm lineExtractTm] >=0} {
set ThresholdMeltingTemp $lineExtractTm
}
close $ExtractTm

set ReadDNATakeUpRate [open ${path}/DNAUptakeRate.txt r]
while {[gets $ReadDNATakeUpRate lineReadDNATakeUpRate] >=0} {
set DNATakeUpRate $lineReadDNATakeUpRate
}
close $ReadDNATakeUpRate

set ReadThresholdProbability [open ${path}/ThresholdProbabilityEnergy.txt r]
while {[gets $ReadThresholdProbability lineThresholdProbability] >=0} {
set ThresholdProbability $lineThresholdProbability
}
close $ReadThresholdProbability

set FileNameLineInMatrix "${path}/LineInMatrix${SquareType}.txt"
set ListCorrectGraph {}
set TrueMatchPairFile "${path}/TrueMatchPair.txt"
if {[file exists ${TrueMatchPairFile}] == 1} {
          file delete -force ${TrueMatchPairFile}
}

set TotalCorrect 0
set TrueMatchPair [open ${path}/TrueMatchPair.txt w+]
file rename -force  ${path}/LineInMatrix${SquareType}.txt
${path}/TempoLineInMatrix${SquareType}.txt
exec /bin/bash -c "sed -e 's/        / /g' ${path}/TempoLineInMatrix${SquareType}.txt >
${path}/LineInMatrix${SquareType}.txt"
set DefineSeq [open ${path}/${SquareType}DefineSeq.txt r]
while {[gets $DefineSeq lineDefineSeq] >=0} {
set SplitColumn [split $lineDefineSeq "    " ]
set Column0 [lindex $SplitColumn 0]
set Column1 [lindex $SplitColumn 1]
set Column2 [lindex $SplitColumn 2]
set Column3 [lindex $SplitColumn 3]
set Column4 [lindex $SplitColumn 4]
set Column5 [lindex $SplitColumn 5]
set Column6 [lindex $SplitColumn 6]
set MatchPair1 {}
set MatchPair2 {}
if { $Column2 != "NIL" } {
```

```
          lappend MatchPair1 $Column0
          lappend MatchPair1 $Column2
          lappend MatchPair1 $Column3
          lappend MatchPair2 $Column4
          lappend MatchPair2 $Column5
          lappend MatchPair2 $Column6
          set TrueMatch {}
          set  LineInMatrix [open ${path}/LineInMatrix${SquareType}.txt r]
          set LineNo 1
          while {[gets $LineInMatrix lineLineInMatrix] >=0} {
                    if { $lineLineInMatrix == $MatchPair1 || $lineLineInMatrix == $MatchPair2 } {
                    lappend TrueMatch "$LineNo"
                    } else {
                    }
          incr LineNo
          }
          close $LineInMatrix
          }

}
close $DefineSeq
close $TrueMatchPair


set TrueMatchPair [open ${path}/List_CorrectPair w+]

# 2nd Procedure: compare TrueMatch/CorrectPair to the processresult folder
set FilteredDirectory "${path}/2ndStepProcessResult"
set FilesProcessResults [glob "${FilteredDirectory}/*.txt"]
set TotalFileNo [llength $FilesProcessResults]
set List_CorrectGraph [open "${path}/List_CorrectGraph.txt" w+]
puts $List_CorrectGraph "Parameter: Square Type = $SquareType"
puts $List_CorrectGraph "Parameter: Define Filename = ${SquareType}DefineSeq.txt"
puts $List_CorrectGraph "ThresholdMeltingTemp = $ThresholdMeltingTemp degree celcius"
puts $List_CorrectGraph "DNATakeUpRate = $DNATakeUpRate"
puts $List_CorrectGraph "ThresholdProbability = $ThresholdProbability"
puts $List_CorrectGraph "Total number of Graph generated = $TotalFileNo"
          # Total Lines in TrueMatch
          set MatchFile [open "${path}/TrueMatchPair.txt" r]
          set LineSeq [split [read $MatchFile] "\n"]
          close $MatchFile
          set Len [expr [llength $LineSeq]-1]

          # Get all True pairs into a List
          set Pair12 {}
          set TotalLineInTrueMatchPair 0
          set MatchFile [open "${path}/TrueMatchPair.txt" r]
          while {[gets $MatchFile lineMatchFile] >=0} {
          set SplitCols [split $lineMatchFile " " ]
          lappend Pair12 [lindex $SplitCols 0]
          lappend Pair12 [lindex $SplitCols 1]
          incr TotalLineInTrueMatchPair
          }
          close $MatchFile
          set TotalPair12 [llength $Pair12]
          for { set a 1 } { $a <= $TotalFileNo } { incr a } {

          # Total Lines In Process Results
          set  InProcessresult [open ${FilteredDirectory}/$a.txt r]
          set LineProcessResult [split [read $InProcessresult] "\n"]
          close $InProcessresult
          set LenP [expr [llength $LineProcessResult]-1]
```

```
        if { $Len == $LenP} {
        set CountMatch 0
        for { set x 0 } { $x < $TotalPair12} { incr x } {
                set  FileNoProcessResult [open ${FilteredDirectory}/$a.txt r]
                while {[gets $FileNoProcessResult lineFileNoProcessResult] >=0} {
                set SplitCols [split $lineFileNoProcessResult " " ]
                set Col0 [lindex $SplitCols 0]
                set Col1 [lindex $SplitCols 1]
                if { [lindex $Pair12 $x] == $Col0 && [lindex $Pair12 [expr $x+1]] == $Col1} {
                incr x
                incr CountMatch
                }
                if { [lindex $Pair12 $x] == $Col1 && [lindex $Pair12 [expr $x+1]] == $Col0} {
                incr x
                incr CountMatch
                }
                }
                close $FileNoProcessResult
        }
        if { $CountMatch == $TotalLineInTrueMatchPair} {
        lappend ListCorrectGraph "${FilteredDirectory}/$a.txt"
        incr TotalCorrect
        }
}
}
puts $List_CorrectGraph "Total number of Correct Graph = $TotalCorrect"
close $List_CorrectGraph
```

## **APPENDIX R**: Index2.php

```
<html>
   <body>
      <?php
          $MinGC = $_POST['mingc'];
          $MaxGC = $_POST['maxgc'];
          $Tm = $_POST['meltingtemp'];
          $Prob = $_POST['probability'];
          $Uptake = $_POST['uptakerate'];
          $SquareType = $_POST['square'];
          $Email = $_POST['email'];
    echo "<p><font face='Arial'>MinGC: ".$MinGC. "</font></p>";
     echo "<p><font face='Arial'>MaxGC: ".$MaxGC. "</font></p>";
      echo "<p><font face='Arial'>Tm: ".$Tm. "</font></p>";
       echo "<p><font face='Arial'>Prob: ".$Prob. "</font></p>";
        echo "<p><font face='Arial'>Uptake: ".$Uptake. "</font></p>";
        echo "<p><font face='Arial'>Rectangular: ".$SquareType. "</font></p>";
         echo "<p><font face='Arial'>Email: ".$Email. "</font></p>";

file_put_contents("/Applications/MAMP/htdocs/MinCG.txt", ($MinGC / 100));
file_put_contents("/Applications/MAMP/htdocs/MaxCG.txt", ($MaxGC / 100));
file_put_contents("/Applications/MAMP/htdocs/MeltingTemperature.txt", $Tm);
file_put_contents("/Applications/MAMP/htdocs/ThresholdProbabilityEnergy.txt", $Prob);
file_put_contents("/Applications/MAMP/htdocs/DNAUptakeRate.txt", $Uptake);
file_put_contents("/Applications/MAMP/htdocs/SquareType.txt", $SquareType);
function print_procedure ($arg) {
  echo shell_exec("/Applications/MAMP/htdocs/Main_RunDNATetris.tcl");
}
$script_name='Main_RunDNATetris.tcl';
echo "<table border='1' cellpadding='1' cellspacing='2'><tr><th><font
face='Arial'>SequenceNo</th><th><font face='Arial'>DNA Strands</th><th><font
face='Arial'>No of Iteration</th><th><font face='Arial'>Thermodynamics Free Energy(AllSub,
kcal/mol)<th><font face='Arial'>Thermodynamics Free Energy(DuplexFold,
kcal/mol)</th><th><font face='Arial'>Percentage of CG content</th></font></tr>";
?>
</body>
</html>
```