DNA-Computing:
Solving The Complementary Bounded Tiling Problem
Using Bionanotechnology

Katherine Virginia Richeson
B.S. Computational Biology
Center for Interdisciplinary Studies Thesis
Advisors: Dr. Laurie Heyer and Dr. Malcolm Campbell
Davidson College
Davidson, NC
April 1, 2011

Abstract

Biology has entered an era of biological mathematics, in which investigators solve mathematical problems using the self-assembling properties of DNA. My research utilizes bionanotechnology to structurally solve the complementary bounded tiling problem. The complementary bounded tiling problem belongs to the complexity class of NP-complete problems, containing difficult problems that cannot be solved efficiently. A solution to the complementary bounded tiling problem would serve as a solution to all other NP-complete mathematical problems with important practical and commercial applications, such as packing optimization, airplane schedule optimization, and finding the shortest path in a network.

In my approach to DNA computation, I engineered DNA nanostructures using a thermal cycler to optimize hybridization of single-stranded oligonucleotides into prescribed nanoscale cruciform shapes known as 4x4 DNA cross tiles. These DNA nanostructures represent individual tiles within the complementary bounded tiling problem and are equipped with five-nucleotide sticky-ends to facilitate the assembly of multi-tile lattices, thereby producing candidate solutions to the complementary bounded tiling problem. Verified through polyacrylamide gel electrophoresis, I have successfully assembled individual tiles. I have modeled and simulated in MATLAB the self-assembly of multiple tiles, and have preliminary evidence to suggest a successfully assembled solution to a complementary bounded tiling problem.

Acknowledgments

The following document is a result of two and a half years of research in pursuit of a Center for Interdisciplinary Bachelor of Science in Computational Biology at Davidson College. During those two years I received guidance, input, monetary support, and help from many different people and organizations. The following is an understated token of gratitude and acknowledgement.

I begin by extending credit and thanks to a fellow student, Linda Kleist. As a one-year international student, Linda aided in the development of the following project as well as the execution of many concepts and experiments. Linda's contributions will be denoted throughout the text using plural pronouns, such as we. Linda contributed to my growth much beyond this project, she taught me how to work in a team. Our conflicting opinions and moments of simultaneous agreement made this project more than it could have been. I will be forever grateful for Linda's help, not only because she aided in the following project development and execution, but because our experiences together exposed both of us to the inner workings, necessity and wonderful outcomes that true collaboration and friendship can lead to in the field of research.

I also extend my deepest gratitude to Dr. Laurie J. Heyer and Dr. A. Malcolm Campbell, my two academic advisors. Their endless support and honest criticism transformed the manner in which I approach science, instilling within me a steadfast vigor to understand the "why" and to believe in the inseparable concepts of learning and fun. From teaching me how to place basic orders to walking me through differentials, both Dr. Campbell and Dr. Heyer have contributed not only to the work below, but also to the training and growth of a scientist who understands that failure is part of the scientific process with great rewards in learning and understanding.

I also thank the Center for Interdisciplinary Studies, including Dr. Denham and Vicki Heitman. I received many questions and probing considerations from faculty and students, shaping the final outcome below. I also am grateful for funding opportunities provided by the Center as well as funding providing by the Biology Department at Davidson College. In addition to the funding received from the Center and the Biology Department, I must also extend gratitude to the Davidson Research Initiative- Mimms Summer Research Fellowship. During the summer of 2010 I received a research grant funded by Larry Mimms in association with the Davidson Research Initiative. Without such monetary support, the research below could not have proceeded.

In addition to the above, I would like to extend thanks to additional students and professors who have aided in the progression of my research, including Dr. Donna Molineck, Dr. Campbell's 2010 Genomics Lab group (Mary Gearing, Pallavi Penumetcha, Bri Pearson and Eric Sawyer) and the 2010 iGEM team from Davidson College.

Finally, I must thank my family for listening to my enthusiasm and attempting to understand my work and passion. Although they did not understand many of my frustrations and successes, they continually supported me.

Without the support of the students, faculty and groups above, the following research would not be possible, I am extremely grateful for their ideas, monetary contributions, and guidance.

Table of Contents

Introduction

The multi-disciplinary interface among mathematics, computer science, and biology entered a new era of biological mathematics in the early 1990s (Kari, 1997). Researchers in the field of DNA computation have turned to DNA, not to unravel the mysteries of life, but to solve extremely difficult mathematical problems, especially NP-complete problems. NP-complete mathematical problems belong to the NP-complete complexity class, for which no efficient algorithms exist. Hence, humans and computers cannot solve large-scale NP-complete problems. As another property of problems belonging to the NP-complete complexity class, once one is solved, all are solved (Dasgupta *et al*., 2006). With an arsenal of over 3,000 NP-complete problems, all possessing particular real-world applications, researchers are on the hunt to discover methods to find solutions (Garey and Johnson, 1979). Difficulties in finding efficient solutions, as well as the potential for wide-ranging practical and commercial applications, motivate researchers in the field of DNA computation to solve NP-complete problems.

The research I conducted for the past two years was motivated by the idea of DNA computation using a structural approach, interweaving the fundamentals of DNA origami and nanotechnology, to solve an NP-complete problem. In particular, my research focused on solving the NP-complete complementary bounded tiling problem. The complementary bounded tiling problem is defined as follows: given a set of k tiles, can the tiles assemble so that a given bounded region (n x m) is filled? For further specification: all tiles need not be used, each tile may be used an unlimited number of times, complementary images on the tiles must match at all adjacent edges, and rotations and reflections are allowed. Although the practical applications of such a tiling problem may not be apparent, the NP-complete problem of packing optimization can be directly tied to a tiling problem such as the complementary bounded tiling problem.

Hence the answers to the complementary bounded tiling problem can serve as answers to other, more practical NP-complete problems, providing real-world utility to my research.

For my thesis I engineered DNA building blocks, known as 4x4 DNA cross tiles. These DNA nanostructures assemble via Watson-Crick base pairing to solve encoded instances of the complementary bounded tiling problem. I produced specific individual tiles that could solve an instance of the complementary bounded tiling problem and analyzed potential candidate solutions.

Chapter I

Mathematical Concepts: NP-Completeness

Mathematical problems can be classified by their degree of complexity. Problems of similar difficulty belong to the same complexity class. NP-complete problems belong to the NP-complete complexity class (Allender *et al.*, 1999). In order to understand complexity classification, one must first understand various basic mathematical concepts.

Mathematical Problems

A mathematical problem is a question possessing unspecified variables. The problem gives a general description of all variables and specifies what properties the solution must satisfy. An instance of a problem specifies particular values for each variable (Garey and Johnson, 1979). For example, consider the traveling salesman problem (TSP), defined by the following:

> Given n vertices (1, 2, . . . , n) and all $n(n-1)/2$ distances between them, find a tour (a sequence of distinct connecting vertices beginning and ending at the same vertex which passes through each vertex exactly once) that minimizes the distance traveled (Dasgupta *et al.*, 2006; Garey and Johnson, 1979; Haris *et al.*, 2008).

TSP is most often modeled using the concept of a traveling salesman, in which the n vertices represent n cities, and the salesman attempts to minimize the distance traveled to visit each city. The variables of TSP therefore consist of a finite set of "cities," $C=\{c_1, c_2, \ldots, c_n\}$, and the distance between each pair of cities, $d(c_i, c_j)$. The solution is an ordering of the cities in which the total distance is minimized from the start city to all other n cities and back (Garey and Johnson, 1979).

One instance of TSP is given by C={c_1, c_2, c_3, c_4} where $d(c_1, c_2)$=17, $d(c_1, c_3)$=7, $d(c_1, c_4)$=18, $d(c_2, c_3)$=10, $d(c_2, c_4)$=9, $d(c_3, c_4)$=26 (Figure 1).

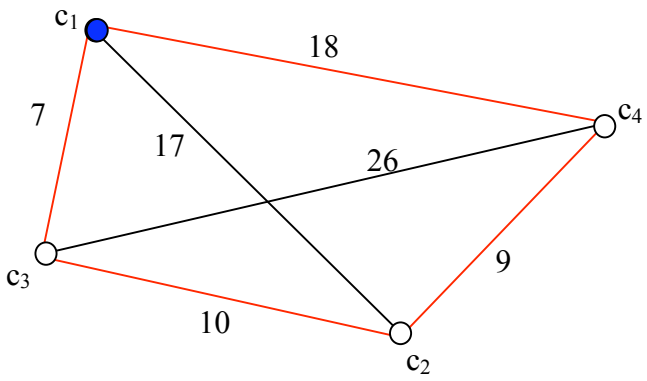


Figure 1. The above TSP problem has the following solution as outlined in red: c_1, c_3, c_2, c_4, c_1 , where c_1 designates the start city in blue.

Algorithms

An algorithm is a series of procedural steps for solving mathematical problems. An algorithm, A, solves a problem if A can be applied to any instance, *I*, of the problem and always produce a solution for that instance (Garey and Johnson, 1979). For example, an algorithm for the traveling salesman problem (TSP) is shown below.

1) Calculate the total distance for every possible tour.

2) Determine which tour minimizes the distance.

In the above instance, the algorithm would perform the following calculations:

Tour	Total Distance
c_1,c_2,c_3,c_4,c_1	71
c_1,c_2,c_4,c_3,c_1	59
c_1,c_3,c_2,c_4,c_1	39
c_1,c_3,c_4,c_2,c_1	59
c_1,c_4,c_3,c_2,c_1	71
c_1,c_4,c_2,c_3,c_1	59

Analyzing the above calculations results in the realized solution of c_1, c_3, c_2, c_4, c_1, where the distance traveled is 39 units, the least of all tours. This solution is highlighted in red in the table and Figure 1. The algorithm above is often referred to as a brute-force algorithm because it calculates the distance of every possible tour. The number of different tours in the above example is six. If an instance of TSP has six cities, then the number of possible tours becomes 120. Using a brute-force algorithm for TSP is considered inefficient, because as n increases by k the number of possible tours increases by $(n+k-1)!/(n-1)!$.

Mathematicians are interested in finding the most efficient algorithms to solve mathematical problems. Usually, the most efficient algorithm is the fastest algorithm, which mathematicians often implement with computer programs. The run time of an algorithm is largely dependent upon the size of the problem, which is reflected by the size of the input, n (Garey and Johnson, 1979). For example, as previously discussed, the traveling salesman problem would take longer to solve if there were six cities, instead of four, because the brute-force algorithm would need to calculate 20 times more possible tours, $(n+2-1)!/(n-1)!$, where $n=4$.

In order to make efficiency comparisons and quantify algorithmic efficiency, mathematics use big-O notation. Big-O notation is a function of the input size of the problem and denotes the growth rate of an algorithm's run time (Carrano and Prichard, 2006). Big-O notation is represented as $O(g(n))$, where $g(n)$ is a function of n, the size of the problem. For example, in TSP the number of tours necessary to investigate is $(n-1)!$, because the starting and ending city is fixed, but the other $n-1$ cities can be visited in any order. In the instance given above, the number of possible tours is six or (4-1)!. Hence the brute-force algorithm for TSP is $O(n!)$. Using dynamic programming algorithm for TSP, which breaks down large problems into

smaller sub-problems, is $O(n^2 2^n)$ (Dasgupta *et al.*, 2006). In both algorithms, as n increases, TSP becomes extremely inefficient to solve. To visualize the inefficiency of both TSP algorithms, we will compare the big-O notation for the two algorithms to an algorithm with a linear run time growth function, $O(n)$. Such a linear big-O notation represents algorithm efficiency to search for a number in an unsorted list (Carrano and Prichard, 2006). Comparing these three big-O notations sheds light on the inefficiency of both algorithms for TSP (Figure 2).

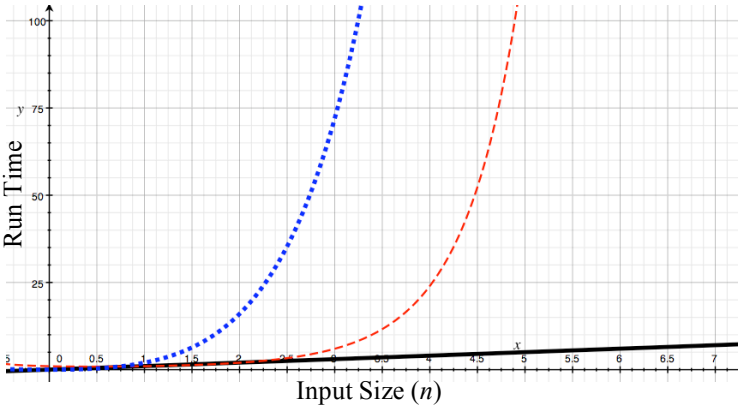


Figure 2. Growth functions for ***O*(*n*!)**, ***O*($n^2 2^n$)**, and ***O*(*n*)**. The x-axis represents the input size, n, and the y-axis represents run time. From slope analysis of the above figure, it is evident that both algorithms for TSP are extremely inefficient when compared to $O(n)$.

Turing Machines

A Turing machine is a theoretical machine used in computer science to identify the abilities and restrictions of computerized algorithms, or computer programs. A Turing machine allows mathematicians to simulate computers. In fact, Turing machines are imagined to be a simple computer, which reads and writes symbols, one at a time, on an infinite piece of tape divided into squares. The symbols are represented as binary digits, 0 and 1, where only one symbol exists in a single square (Barker-Plummer, 2010; Figure 3).

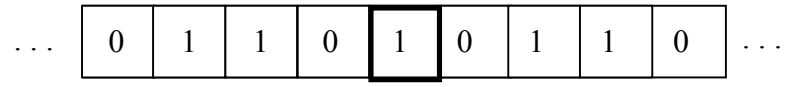


Figure 3. The bolded square represents the tape head of the Turing machine, which is the only symbol that the machine can read until the tape moves.

The Turing machine determines its next action based on rules, dictated by the current symbol and on the current state of the machine. For instance, if the Turing machine reads a 0 and is in state 3, a rule may dictate that the tape move left and change the 0 to a 1. Through the actions of the Turing machine, a computerized algorithm can be traced for various mathematical problems (Barker-Plummer, 2010).

There are three different types of Turing machines: 1) deterministic, 2) non-deterministic and 3) universal Turing machines (Barker-Plummer, 2010). A deterministic Turing machine (DTM) only has one potential action dictated by the current situation (symbol and current state). In other words, each action is clearly defined (Friedman, 2005). A non-deterministic Turing machine (NDTM) may have a set of rules that dictates more than one action for a given situation (Martin, 1997). In other words, the next action of the machine is never known because there can exist more than one action for a given symbol and state. A universal Turing machine (UTM) can simulate any Turing machine, deterministic or non-deterministic. A UTM takes not only the tape as input but also a description of the type of Turing machine (Kamvysselis, 1999). The different types of Turing machines are used to define complexity classes and are therefore important to understanding complexity classification.

Four Main Complexity Classes

Mathematicians recognize four main complexity classes of mathematical problems: P, NP, NP-hard and NP-complete, where certain complexity classes interrelate (Aaronson, 2006; Figure 4).

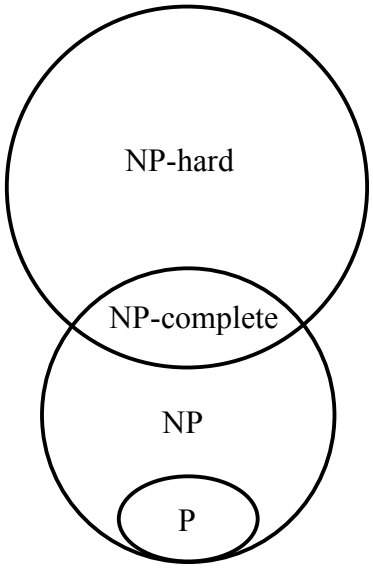


Figure 4. The Venn diagram indicates the relationships among these 4 complexity classes. It is important to note that the relationships above assume that P≠ NP, an assumption that is widely accepted, but unproven (Aaronson, 2006).

P complexity

Problems belonging to the complexity class P can be algorithmically modeled on a deterministic Turing machine (DTM) using a polynomial amount of computation time (Sipser, 2006). Polynomial time refers to an algorithmic run time of $O(n^k)$, where n is the size of the problem's input and k is some constant (Terr, 2010). In other words, the number of steps to find a solution to a problem in P is a polynomial function of the size of the problem, n.

NP complexity

Problems belonging to the complexity class NP are 1) Solvable in polynomial time on a NDTM, and 2) Verifiable in polynomial time on a DTM (Garey and Johnson, 1979). NP stands for nondeterministic polynomial time, indicating that any algorithm solving the problem does not have a uniquely defined next step (because it can only be solved on a NDTM), but rather a

choice between several possible next steps, (*i.e.* a trial and error mechanism; "nondeterministic polynomial time," Dictionary of Computing, 2010). If there existed a NDTM that was able to guess correctly at each decision point where more than one action could be taken, it could solve the problem in polynomial time, since all it would have to do is verify the correctly guessed solution. As a main characteristic of NP, this can be done in polynomial time if such an NDTM existed that could guess correctly. However, such a machine does not exist under the assumption that NP≠ P.

All problems in P, which by definition can be solved in polynomial time using a DTM, can also be verified in polynomial time using a DTM. Also, a NDTM has at least one action for a given situation, but may have more than one. Hence, problems that can be solved on a DTM in polynomial time can also be solved on a NDTM that has only one action, in polynomial time (Garey and Johnson, 1979). In other words, an algorithm modeled on a DTM can be considered a degenerative form of a NDTM algorithm ("nondeterministic polynomial time," Dictionary of Computing, 2010). For these two reasons all problems in P are also in NP ($P \subseteq NP$) as seen in Figure 4.

The distinction between P and NP has caught the attention of mathematicians and computer scientists for years. In fact, one of the Millennium Prize Problems is whether NP=P or NP≠P (Friedman, 2005). In other words, do polynomial algorithms exist using a DTM to solve all problems within NP? The current assumption and belief among the research community is that NP≠P, as depicted in Figure 4 (Friedman, 2005). Currently, problems in NP have only been proven to have at best exponential run time ($O(2^n)$) when solved on a DTM (Garey and Johnson, 1979).

NP-hard complexity

A problem is NP-hard if a polynomial algorithm on a DTM for such a problem would imply a polynomial-time algorithm on a DTM for all other problems in NP (Garey and Johnson, 1979). The previous statement is true because problems belonging to the complexity class NP-hard are "at least as hard as any problem in NP" (De Rezende, 2010). In order to prove that problems are at least as difficult as the hardest problems in any complexity class, one must prove that the problem in question can be reduced from another NP-hard problem.

Mathematically, a **reduction** is the transformation of one problem into another one, indicating that solving a problem is at least as hard as solving another one (Dasgupta *et al.*, 2006). For instance, if a problem Z can be solved using an algorithm for B, Z is no more difficult than B, and we say that Z reduces to B. The idea of reducing one problem to another motivates the idea of a problem being hard for a class of problems. Problem Z is hard for a class of problems if every problem in the class C can be reduced to Z. Hence Z is (one of) the hardest problems in C. An algorithm for Z together with a reduction solves any problem in C. In the case of NP-hard problems, all NP-hard problems can be reduced using a DTM in polynomial time from any problem in NP (Garey and Johnson, 1979).

There are some problems in the complexity class of NP-hard that are undecidable, meaning that it is impossible to construct an algorithm to solve the problem. By definition, NP problems must have candidate solutions that can be checked in polynomial time, but no candidate solutions exist to undecidable problems. Therefore, some problems in NP-hard are at least as hard as any problem in NP, but do not belong in the NP complexity class (Garey and Johnson, 1979).

NP-complete complexity

NP-complete problems have candidate solutions that are verifiable in polynomial time on a DTM (therefore NP-complete ⊆ NP) and are at least as hard any problem in NP (NP-complete ⊆ NP-hard; Dasgupta *et al.*, 2006). In other words, NP-complete problems are represented by NP ∩ NP-hard (Figure 4) and problem C is in the complexity class NP-complete if

1) C is in NP (meaning verifiable in polynomial time on a DTM), and
2) Every problem in NP is reducible in polynomial time to C, meaning that C is at least as hard as the hardest problem in NP, since every problem in NP, even the hardest problems, must be reducible to C. Thus, C is in NP-hard (Garey and Johnson, 1979).

In order to prove that a problem C is NP-complete, one must prove that these two conditions apply to C. To prove condition one, it will suffice to write an algorithm that can be solved in polynomial time, and which verifies a candidate solution. To prove conidtion two, it is sufficient to show a DTM reduction, in polynomial time, from an already existing NP-complete problem to C, the problem in question (Figure 5).

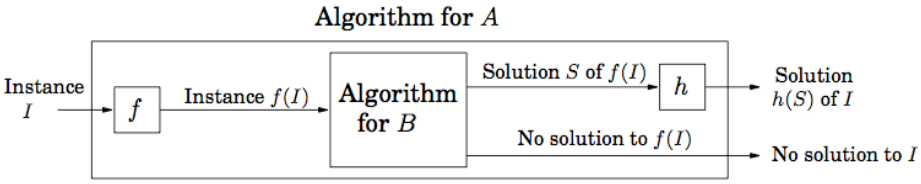
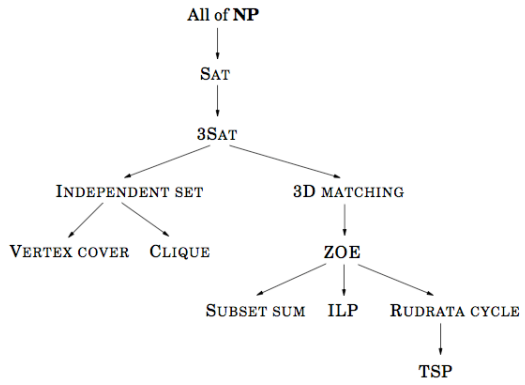


Figure 5. Any NP-complete problem, A, can be reduced to a problem in question, B, by a transformation of the inputs with the function *f*. If such a transformation function exists then B is NP-hard (Dasgupta *et al*., 2006).

Since part of the proof for an NP-complete problem relies on reduction from another NP-complete problem, the question surrounding the proof of the first NP-complete problem often

arises. In 1971, Cook and Levin identified that Boolean satisfiability problem (SAT) as the first problem belonging to the NP-complete complexity class. The Cook-Levin theorem used the logic of NDTM to prove that the SAT problem was NP-complete (Fourman, 2002). Since the theorem was proven, mathematicians have been relating NP-complete problems through reductions directly and indirectly related to the SAT problem (Figure 6A). However, more recently, some mathematicians think that the bounded tiling problem may serve as an "alternative master problem for the theory of NP-completeness" (Van Emde Boas and Savelsbergh, 1984; Figure 6B). The main advantage in using the bounded tiling problem over the SAT problem lies in the "conceptual simplicity of the problem" (Van Emde Boas and Savelsbergh, 1984). The ease with which the bounded tiling problem can be reduced to other NP-complete problems as well as the ease with which a NDTM can be used to model the original proof of NP-completeness allows for a simplistic understanding of NP-complete theory and polynomial time reductions (Van Emde Boas and Savelsbergh, 1984).

A.



B.

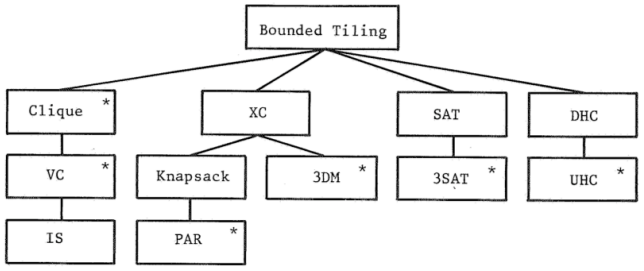


Figure 6. (A) Represents a schematic flow chart of polynomial time reductions for NP-complete problems based on the Cook-Levin theorem (Dasgupta *et al*., 2006). (B). Illustrates another possible schematic of the relationships between NP-complete problems based on polynomial time reductions (Van Emde Boas and Savelsbergh, 1984).

Understanding polynomial time reductions to prove a problem NP-hard is pivotal to understanding proofs concerning NP-complete problems. The following describes the polynomial time reduction of Rudrata cycle to the traveling salesman problem (TSP) as detailed in *Algorithm Design* by Kleinberg and Tardos in 2006 and two class lectures at Durham University and the University of Maryland (Johnson, 2009; Kingsford, 2009). Rudrata cycle is often referred to as the Hamiltonian cycle problem and it is a known NP-complete problem as shown in the flow chart of polynomial time reductions from other NP-complete problems (Figure 6A). The Hamiltonian cycle problem asks: Given a graph, does there exist a cycle that passes through each vertex exactly once (Dasgupta *et al*., 2006).

Logically it is apparent that TSP is at least as hard as the Hamiltonian cycle problem because it is harder to find the shortest cycle than to decide whether or not a cycle even exists. Hence, TSP must be NP-hard because it is at least as hard as another NP-hard problem. In order to formally prove that TSP is NP-hard, one must show a polynomial-time reduction from the Hamiltonian cycle problem to TSP, as seen below.

Theorem (Kingsford, 2009; Kleinberg and Tardos, 2006; Johnson, 2009): The traveling salesman problem (TSP) is in NP-hard because it can be reduced from the Hamiltonian cycle problem.

Proof:

Let G be an instance of the Hamiltonian cycle problem, where G is a set of vertices and edges, $G=(V, E)$.

Now, transform the given instance of the Hamiltonian cycle problem into TSP. For each vertex in G, create a "city" c_v. If an edge, $e=(u, v)$ exists between two vertices in G, then let the distance of e be 1, otherwise let the distance be 2.

Claim: *G* has a Hamiltonian cycle if and only if the instance of the traveling salesman problem constructed above has a distance at most *n*.

Proof:

Forward Direction:

If *G* contains a Hamiltonian cycle, then the cycle forms a tour through the cities of distance *n*, thereby serving as a solution to the TSP.

Reverse Direction:

If there is a tour of distance *n* through *n* cities, then the distance between each city along the route is 1. Thus each pair of cities along the route is adjacent in *G* and the route is a Hamiltonian cycle.

□

One of the most distinctive characteristics of NP-complete problems is the fact that no efficient (polynomial time) algorithms currently exist to solve such problems (Garey and Johnson, 1979). Although all problems in NP are verifiable in polynomial time, only those in P are solvable in polynomial time, at least under the current conviction that NP≠P (Friedman, 2005). Therefore under such an assumption, NP problems not in P, including NP-complete problems, cannot be solved in polynomial time, making them extremely inefficient to solve. As we saw earlier in this chapter, the most efficient algorithm found to solve NP-complete problems, like TSP, is $O(n^2 2^n)$ (Dasgupta *et al.*, 2006). Hence, as the size of the problem increases, the run time of the algorithm increases exponentially, and the problems become extremely time consuming to solve.

NP-complete problems are among the most highly studied mathematical problems not only because of their difficulty, but because of their wide range of applications. For instance,

TSP alone presents a wide variety of practical applications in the world of logistics and planning. TSP has been used to model trucking patterns in order to optimize transport of goods and other items from one city to another, taking into account limiting variables such as cost and time. TSP has also been utilized in minimizing fuel usage for outer space imaging satellites and used by the Worldwide Airport Path Finder to minimize flight distances between airports (Cook, 2007). These applications are not bound to solutions applying only to TSP. Instead, once any NP-complete problem is solved, all NP-complete problems are solved. Hence, answers to these practical applications for TSP can be solved by another NP-complete problem, through the use of a reduction in polynomial time as discussed above (Dasgupta *et al*., 2006).

The relationships among NP-complete problems through polynomial time reductions, their multitude of applications, and their difficulty motivate researchers in various disciplines to search for solutions to NP-complete problems. DNA computation has been a novel approach in such a search, combining the fields of mathematics and biology to find solutions to NP-complete problems. Subsequent chapters illuminate the NP-complete problem studied in this thesis and the methodology used to assemble solutions using DNA computation.

Chapter II

Problem Description: Complementary Bounded Tiling Problem

The purpose of my research is to solve the complementary bounded tiling problem using DNA, more specifically the concepts of bionanotechnology and DNA origami. This chapter is devoted to the description of two NP-complete mathematical problems, the Scramble Square problem and the complementary bounded tiling problem. Linda Kleist and I began our research attempting to solve the Scramble Square problem and transitioned to solving the complementary bounded tiling problem due to biological assembly limitations that I will detail in Chapter V. Both problems are defined below, including proofs confirming that both problems are NP-complete.

The Scramble Square Problem

Scramble Squares® is popular childhood puzzle, consisting of 9 squares and 4 full images. Each edge of a square consists of a half-image of one of the 4 full images. The goal of the puzzle is to arrange all 9 tiles so that all touching edges complete a full image (Brandt *et al.*, 2002; Figure 1).

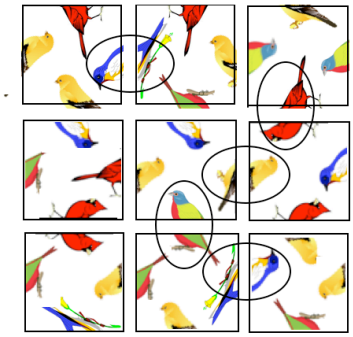


Figure 1. An unsolved 3x3 Scramble Square problem. One must arrange all 9 tiles so that every pair of touching edges brings together complementary edges (*e.g.*, a bird's beak and tail). The circled edges represent matches in the given arrangement. To form a solution, all twelve touching edges need to match.

Formally, the Scramble Square problem is defined as follows:

> Given n^2 tiles and m half-images, where each tile has 4 (not necessarily unique) half-images and $m \geq 2$, is it possible to rotate and arrange all tiles in an n x n grid such that the half-images align with their complements wherever two edges touch?

The above definition dictates that every tile given must be used exactly once in any resulting solution (Brandt *et al.*, 2002). Also, the number of half-images, m, is independent of size of the problem, n. Therefore, for any given instance the only restrictions on m are given in the above definition. In the Scramble Square problem, there are $4^8 \bullet 9! = 23{,}781{,}703{,}680$ possible arrangements of the 9 tiles and more generally, $4^{(n-1)} \cdot n!$ possible arrangements of n tiles (Brandt *et al.*, 2002). A brute-force algorithm to solve the Scramble Square problem, similar to the brute-force algorithm for solving TSP in Chapter I, would require testing all possible tile arrangements and determining which tile arrangements fulfilled the edge-matching criteria. For a Scramble Square problem with n tiles, the big-O notation for a brute-force algorithm would be $O(n!)$, which as demonstrated in Chapter I is inefficient. Hence, as the number of tiles increases, the Scramble Square problem becomes extremely inefficient to solve, especially when utilizing a brute-force algorithmic approach.

In addition to a brute-force algorithm to solve the Scramble Square problem, a backtracking algorithm also exists. The backtracking algorithm places tiles, one at a time, into the 3x3 grid according to the schematic shown in Figure 2 (Brandt *et al.*, 2002).

7	**8**	**9**
6	**1**	**2**
5	**4**	**3**

Figure 2. Schematic representing tile placement using a backtracking algorithm to solve a 3x3 Scramble Square problem (Brandt *et al.*, 2002).

The backtracking algorithm restricts the first tile in position 1 to a specific rotational orientation. Given a partial solution of k tiles, the algorithm tests the remaining 9-k tiles to determine if any of the remaining tiles, taking into account rotations, fit in the $(k+1)^{st}$ position. If a tile does match, then the process continues. If no tiles match, then the k^{th} tile is removed and the other remaining tiles are tested in that k^{th} position. The process of removing tiles is considered to be the backtracking step of the algorithm, where solutions are considered correct until no possible remaining tile can fit and then one tile at a time is removed (Brandt *et al.*, 2002). The backtracking algorithm indicates the non-deterministic nature of the Scramble Square problem because one "guesses" the tile placement and its orientation for position 1 as well as for any remaining positions in which more than one remaining tile fits. Such “guessing” indicates that there is not a unique next action in the backtracking algorithm. Recall from Chapter I that a characteristic of an NP-complete problem depends on a polynomial run time on a non-deterministic Turing machine (NDTM). If a NDTM could guess correctly for each tile placement, then the Scramble Square problem would be considered solvable in polynomial time on a NDTM and P=NP. It is possible for the backtracking algorithm to guess correctly with certain tile placements, thereby providing opportunities for increased efficiency, but such increased efficiency is not guaranteed, and in the worst case, all possible tile arrangements may need to be tested, as in the brute-force algorithm.

Scramble Square is in NP

In order to mathematically prove that the Scramble Square problem is NP-complete, one must prove that the Scramble Square problem is:

1) In NP, by proving that a candidate solution is verifiable in polynomial time, and

2) In NP-hard, by proving that a polynomial time reduction exists from an already existing NP-hard problem.

In order to verify a solution, one would have to check whether or not touching edges in the given candidate solution matched. If every touching edge matched, then the candidate solution would be a true solution to the Scramble Square problem. If not, the candidate solution would not solve the Scramble Square problem. Such a verification algorithm can be completed in polynomial time because for each candidate solution of n^2 tiles there are $2n^2$-$2n$ edges, which must be checked, making the algorithm run time dependent on a polynomial function of n (Figure 3).

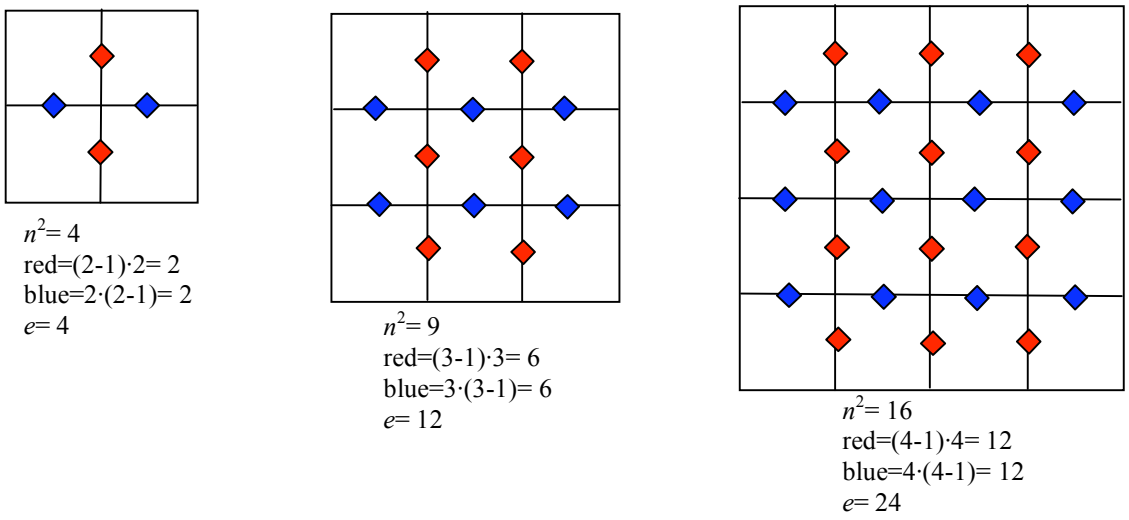


Figure 3. Representation of different number of tiles (n^2) for the Scramble Square problem and the number of edges (e) that must be checked for matching when verifying solutions. Given a tiling problem with n^2 tiles, there are $(n\text{-}1)n + n(n\text{-}1)$, where the first part of the formula represents the edges colored red and the second represents edges colored blue. Hence, when verifying a candidate solution to the Scramble Square problem with n^2 tiles, one need only check $2n^2$-$2n$ edges to determine if a proper matching exists.

In order to mathematically prove that candidate solutions to the Scramble Square problem can be verified in polynomial time, we implemented and wrote a computer program in

MATLAB. The program determines if given candidate solutions are true solutions or not by analyzing matched or unmatched touching edges within the candidate solution. Analyzing the coding structure of this program revealed the run-time of the algorithm and whether or not candidate solutions to the Scramble Square problem can be verified in polynomial time.

For purposes of the program, tiles are identified using a specific numbering system. Each half-image on an edge of a tile is represented by a variable (a, b, c, . . .) or complementary variable (-a,-b,-c, . . .). All tiles are referenced by a vector containing their four edge-associated variables, written starting at the top edge, noted as edge one (Figure 4).

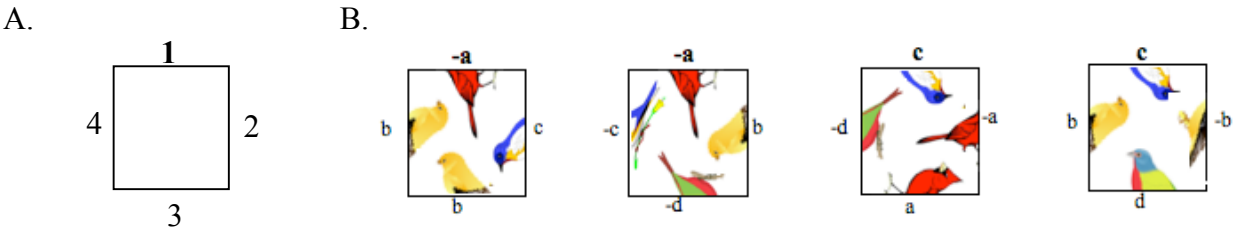
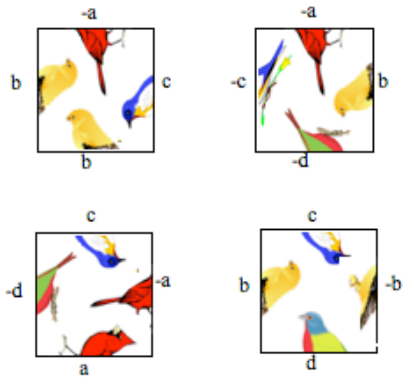


Figure 4. (A). A tile and the schematic for edge numbering. The top edge is referred to as edge 1 and the edge number proceeds clockwise. (B). A diagram depicting different tiles and their associated vectors. Each half-image is associated with a variable, a, b, c, or d and its complement –a, -b, -c, or –d. Starting from left to right, the tiles would have the following associated vectors [a, c, b, b], [-a, b, -d, -c], [c, -a, a, -d], and [c, -b, d, b]. Notice how all vectors begin with the top edge variable, which is bolded above and referred to as edge 1.

The MATLAB program needs 2 inputs based upon the number of tiles, n^2, within the solution:

1) The dimensions of the Scramble Square solution (n x n), and

2) The tiling of the candidate solution in a vector, $\boldsymbol{V}$, of length $4n^2$ (n^2 tiles, each having four sides). The tiles within $\boldsymbol{V}$ should be taken from the candidate solution read left to right and top to bottom (Figure 5).



2 x 2 Scramble Square

$n^2 = 4$

Vector, $\boldsymbol{V}$ = [-a c b b -a b -d -c c -a a -d c -b d b]

Figure 5. Diagram illustrating inputs for MATLAB program, including dimensions of 2 x 2 as well as candidate solution vector, $\boldsymbol{V}$.

In order to check a given solution, one must verify that the half-images match on the touching edges of the candidate solution (see Figure 3). The algorithm computes a matrix A, which stores the pairs of edges that must be checked according to the given dimensions of the candidate solution. Candidate solutions to the Scramble Square problem will always be square and matrix A will be the exact same for Scramble Square problems of the same size, because problems of the same dimensions must check for a match at the same touching edges.

Matrix A is initialized to be a matrix of zeros of size $(2n^2\text{-}2n)$ x $(4n^2)$. The $4n^2$ represents each edge of every tile, starting at top edge, edge 1 for each tile, and rotating clockwise around the tile, ending at edge four (see Figure 4). Each row represents one edge matching and within each row two spots are initialized to 1 based on which tile edges touch in the given matching (Figure 6).

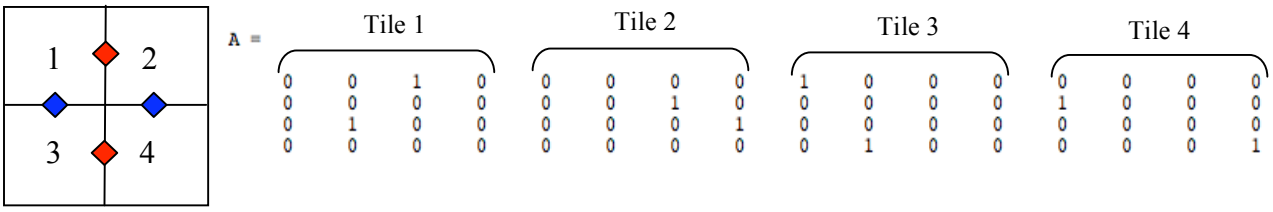


Figure 6. An example of matrix A generated for a general 2 x 2 Scramble Square with n^2=4 tiles.

The algorithm begins by considering the touching edges between the top and bottom of each tile (edges one and three), highlighted in blue for each tile, beginning with edge three of tile 1 and edge one of tile 3. Within matrix *A*, both of these edges are changed to a 1 in the first row. This is the acknowledgment of the first place to test touching edges to determine if there is truly a match. The algorithm continues to change the 0s to 1s within each of the rows, determining the touching edges within each tile. So for instance, in row 2, the touching edges between edge three of tile 2 and edge one of tile 4 must be checked.

For a particular candidate solution, the touching edges indicated by 1s in matrix *A* must be checked to determine if the half-images are complementary. Because matching half pictures add up to 0 (a + (-a) = 0), a candidate solution is a valid solution if and only if AV' is the zero vector. The code for the program in MATLAB can be found in Appendix.

In running this program with proper input, one can determine whether or not a candidate solution is a true solution to the Scramble Square problem based on whether or not $AV'=0$. Also, through algorithm analysis, a candidate solution can be checked in polynomial time, specifically with a big-O notation of $O(n^4)$, where n^2 is the number of tiles in the candidate solution.

Scramble Square is NP-hard

In order to prove that the Scramble Square problem is NP-hard, we must prove that the Scramble Square problem can be reduced from a known NP-hard problem. We will reduce from the Rectangle Packing problem (Demaine, 2007).

Definition of the Rectangle Packing problem:

> Given *n* rectangular pieces of size $1 \times x_1$, $1 \times x_2$, . . . , $1 \times x_n$, where the x_i's are positive integers bounded above by a polynomial in *n*, can these pieces be packed into a specified

rectangle with area $x_1 + \cdots + x_n$ (Figure 7; Demaine, 2007)?

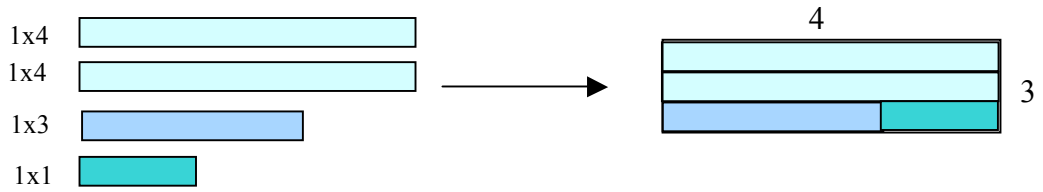


Figure 7. Instance of the Rectangle Packing problem: given 4 rectangular pieces sizes 1 x 4, 1 x 4, 1 x 3 and 1 x 1, can the 4 pieces be packed into a rectangle with an area of 12 unites? The answer is yes, and an illustration of one possible solution is given.

Theorem (Demaine, 2007): The Scramble Square problem is NP-hard because it can be reduced in polynomial time from the NP-complete and therefore NP-hard Rectangle Packing problem.

Proof:

If we are given an instance of the rectangular packing problem, then we must show that a transformation exists to change the inputs of the rectangular packing problem to an instance of the Scramble Square problem. With such a transformation, the solution to the instance of the Scramble Square problem must also be a solution to the original Rectangule Packing problem. This concept of reductions was described in Chapter I Figure 5, and this proof follows that of Demaine for the signed-edge matching problem (Demaine, 2007).

Given an instance of the Rectangle Packing problem, one can produce an equivalent instance of the Scramble Square problem in two steps. First, create a square frame around a rectangle of area $x_1 + \cdots + x_n$ and dimensions l x w (Figure 8). All exterior edges of this frame use the same half-image. This half-image does not exist elsewhere in the puzzle and its complement does not exist anywhere in the puzzle. These restrictions force these half-images to be on the border of the square. The shared edge between every two adjacent tiles

brings two complementary image halves together. Every image only appears once in the puzzle. The horizontal edges facing the interior of the rectangle are complementary images; the vertical edges facing the interior of the rectangle represent another complementary image pair. Now the frame is complete (Figure 8).

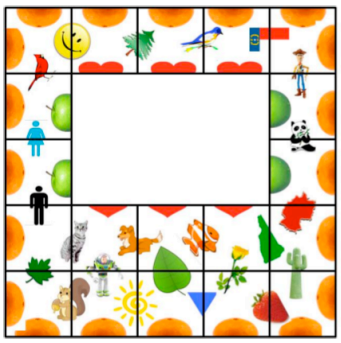


Figure 8. An example of a frame around a 3x2 rectangle.

Secondly, create analogues to the rectangular packing pieces: Each packing piece $1 \times x_i$ is represented by x_i tiles. The tiles are created such that they only align horizontally, because the adjacent edges share complementary images unique to the rectangular packing piece. The top edges are complementary to the bottom edges of the interior of the square frame. The bottom edges are complementary to the bottom edges of the interior of the square frame. The left and right outer edges are complementary to each other and to the vertical interior frame. All of the tiles are identical but the two ends (Figure 9).

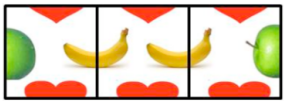


Figure 9. Example of a rectangular packing piece realized through the Scramble Square problem.

Now, we have transformed an instance of the Rectangle Packing problem into an instance of the Scramble Square problem. Using an algorithm to solve the Scramble Square problem will also result in a solution to the Rectangle Packing problem. Hence, we have proven that the Scramble Square problem is in NP-hard.

□

We previously proved that the Scramble Square problem is in NP and hence now we have finished the proof that the Scramble Square problem is NP-complete.

Complementary Bounded Tiling Problem

The complementary bounded tiling problem is a modification to the Scramble Square problem and to my knowledge is not discussed in the literature. Therefore, I define the complementary bounded tiling problem as follows:

> Given a set of *k* tiles, can the tiles assemble so that a given bounded region (*n* x *m*) is filled and complementary images match?

All tiles need not be used, and each tile may be used an unlimited number of times. These are both deviations from the Scramble Square problem, in which each tile must be used exactly once. Also, individual tile rotations and reflections are allowed (Figure 10). Figure 11 illustrates the complementary bounded tiling problem for a given tile set.

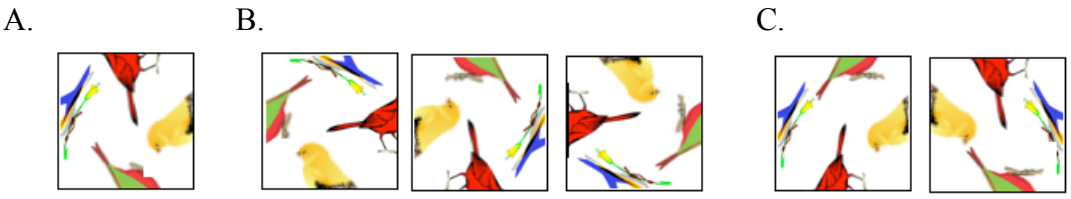


Figure 10. (A) Represents the original tile in one rotational state. (B) The other 3 rotational states of the original tile. Note that the original tile was simply turned. (C) Two examples of reflection for the original tile, where two sides were "swapped". To notice the difference, pick an image and look at the image to the right and left of it, and notice that this arrangement does not match the original tile.

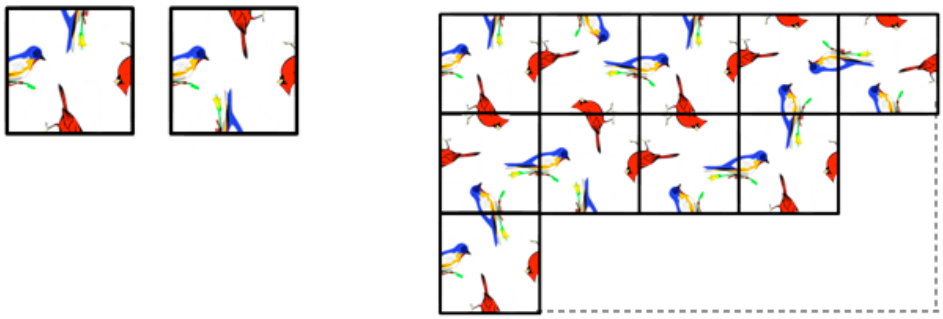


Figure 11. An example of the complementary bounded tiling problem, where two tiles are given. Those two tiles can be used infinitely to fill the bounded region as seen on the right.

In order to prove that the complementary bounded tiling problem is NP-complete we must prove that it is in NP and that it is NP-hard.

Complementary Bounded Tiling Problem is in NP

The same program detailed above for the Scramble Square problem can be used to verify solutions to the complementary bounded tiling problem. In the case of the complementary bounded tiling problem, the dimensions given for the suggested solution do not need to be equal. In other words, solutions to the complementary bounded tiling problem must fill a region, not necessarily a square, bounded by given dimensions. The only change to the MATLAB program detailed above for the Scramble Square problem is in the size of matrix A. Recall that matrix A represents all of the edges for every tile in the columns and all of the possible touching edges in the rows. Since the dimensions for the complementary bounded tiling problem may be rectangular, the number of possible touching edges changes and hence the size of matrix A changes as well as the touching edges (Figure 12). The code for verifying candidate solutions to the complementary bounded tiling problem can be found in the Appendix.

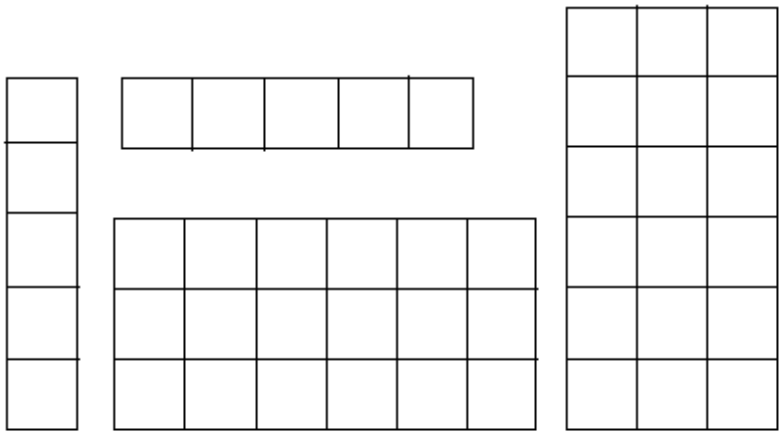


Figure 12. Possible configurations for the complementary bounded tiling problem for which different touching edges and testing algorithms must be implemented in the creation of matrix A - chains for which the x dimension or y dimension is longer, rectangles where the x dimension is larger than the y dimension, and rectangles where the y dimension is larger than the x dimension. Due to these new possibilities the number of touching edges must change.

Despite the changes in matrix *A* configuration, the algorithm continues identically to the Scramble Square program, where a candidate solution is considered valid if *AV'* is the zero vector. Hence, with the proper input, candidate solutions to the complementary bounded tiling problem can be verified in polynomial time, specifically $O(k^2)$, where k is equal to the number of tiles in the suggested solution. Therefore, the complementary bounded tiling problem is in NP.

Complementary Bounded Tiling Problem is NP-hard

In order to prove that the complementary bounded tiling problem is NP-hard, we will reduce it from the bounded tiling problem, which is already known to be NP-complete and thus NP-hard.

Definition of Bounded Tiling Problem:

Given a set of *k* tiles, can the tiles assemble, without rotations or reflections, so that a given bounded region (*n* x *m*) is filled and edges with the same color match (Figure 13; Boas and Savelsbergh, 1984)?

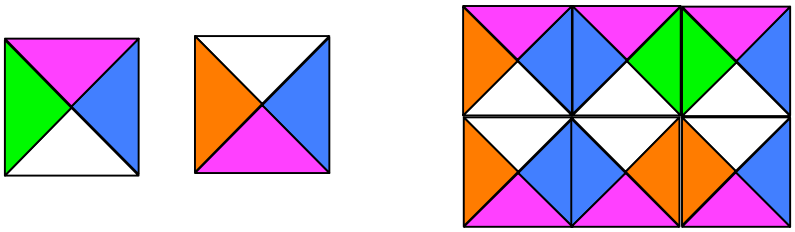


Figure 13. Bounded tiling problem, where instead of complementary images matching, edges with the same color must match. The given tile set is seen to the left and the successful tiling of a 2x3 region is shown to the left. The tiles pictured to the left are often referred to as Wang tiles (Gopalkrishnan, 2008).

Theorem: The complementary bounded tiling problem with rotations is NP-hard because it can be reduced from the NP-hard bounded tiling problem.

Proof:

If we are given an instance of the bounded tiling problem, known to be NP-complete, we must show that a transformation exists to change the inputs of the bounded tiling problem to an instance of the complementary bounded tiling problem. With such a transformation, the solution to the instance of the complementary bounded tiling problem must also be a solution to the original bounded tiling problem.

Given an instance of the bounded tiling problem (Figure 14A), one can transform it into an equivalent instance of the complementary bounded tiling problem in two steps. First, rename the vertical edges such that horizontal and vertical edges have no common color (Figure 14B). In other words, in a given tile set, no color should appear on both the

vertical and horizontal plane. If a color does exist on both planes, change the vertical edges of that color to a color not currently in the puzzle. Second, replace each color that appears more than once by a pair of complementary colors (Figure 14C; Gopalkrishnan, 2008).

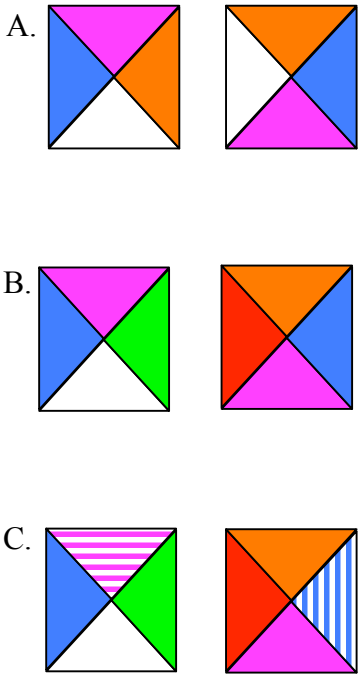


Figure 14. (A). Tile set for a particular instance of the bounded tiling problem. Notice that the color blue is restricted to only horizontal edges, whereas both the color orange and white exists on the horizontal and vertical plane. (B). Step one for problem transformation is to rename the vertical edges such that the shared colors on both planes no longer exist. Here, the vertical white was replaced by red and the vertical orange was replaced by green. (C) Step two is to replace each color with complementary colors. Since pink appears twice, pink is replaced with complementary colors denoted by the pink and the pink stripes. Similarly, blue is replaced by blue and blue stripes.

The bounded tiling problem, which does not account for rotations, can therefore be transformed into an instance of the complementary bounded tiling problem, which allows for rotations. Using an assembly algorithm for the complementary bounded tiling problem will now result in the same output as an algorithm for the bounded tiling problem and the output can be transformed back into the instance of the bounded tiling problem using the reverse of the steps above (Figure 15).

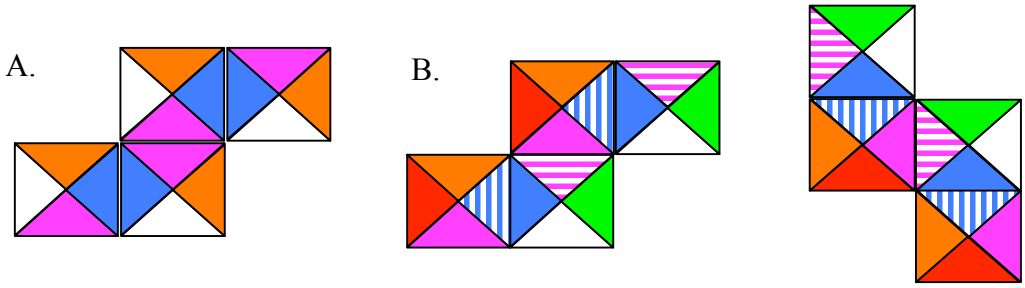


Figure 15. (A). The only tiling pattern that could result from the tile set given in Figure 14A because rotations are not allowed. (B). The only two tiling patterns that can occur after transforming the problem. These patterns are identical to the original tiling pattern without rotations. Therefore, an algorithm for the complementary bounded tiling problem with rotations produces a solution to the bounded tiling problem without rotations.

Hence, the complementary bounded tiling problem with rotations can be reduced from the NP-hard bounded tiling problem without rotations using the above transformation. Thus, the complementary bounded tiling problem with rotations is NP-hard.

□

The complementary bounded tiling problem that I want to solve with DNA allows for rotations and reflections. I already proved that the complementary bounded tiling problem with rotations is NP-hard, but I also must prove that the complementary bounded tiling problem with reflections is NP-hard.

Theorem: The complementary bounded tiling problem with reflections is NP-hard because it can be reduced from the NP-hard complementary bounded tiling problem with rotations.

For this proof, it is important to realize that for complementary pictures there is a specific orientation needed for the edges to actually match. For instance, in the tiling problems above it is important that the bird's beck and tail match in the correct orientation (Figure 16).

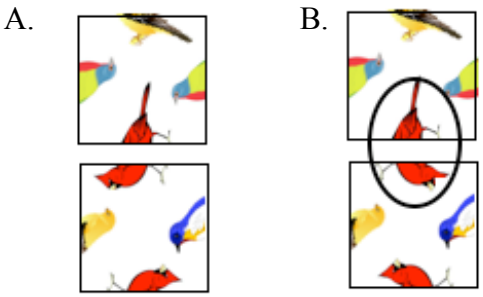


Figure 16. The bottom tile of A is a reflection of the bottom tile in B. Circled in black is a mismatch, indicating that it is important that the complementary images meet in the correct orientation.

Unlike the representation of complementary colors, the bird half-images allow more control over the orientation of matching, which we see as a necessity in Figure 11.

Proof:

Given an instance of the complementary bounded tiling problem with rotations (Figure 17A), one can transform it into an equivalent instance of the complementary bounded tiling problem with reflections in one step. Introduce complementary non-symmetric half-images for each set of colors (Gopalkrishnan, 2008; Figure 17B).

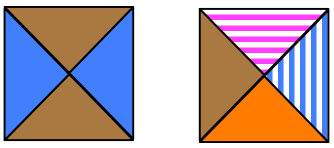


Figure 17. (A) Instance of the complementary bounded tiling problem with rotations.


(B) In order to transform the instance in part A to an instance of the complementary bounded tiling problem with reflections, we introduce the notches to represent the complementarity. The colors that were shaded with lines are the receiving ends of the notches, whereas the solid colors are the protruding ends. Introducing these notches does not allow for reflections to change the problem given in A and therefore a solution to B will be the same solution to A even though reflections are allowed.

As you can see above, an instance of the complementary bounded tiling problem with rotations can be transformed into an instance of complementary bounded tiling problem with reflections as seen through a notched tiling problem. Using an algorithm to solve the notched tiling problem with reflections will result in a solution to the complementary bounded tiling problem without reflections. Therefore, we have transformed an instance of the complementary bounded tiling problem with rotations but without reflections into an instance of the complementary bounded tiling problem with reflections (the notched tiling problem) (Gopalkrishnan, 2008; Figure 18).

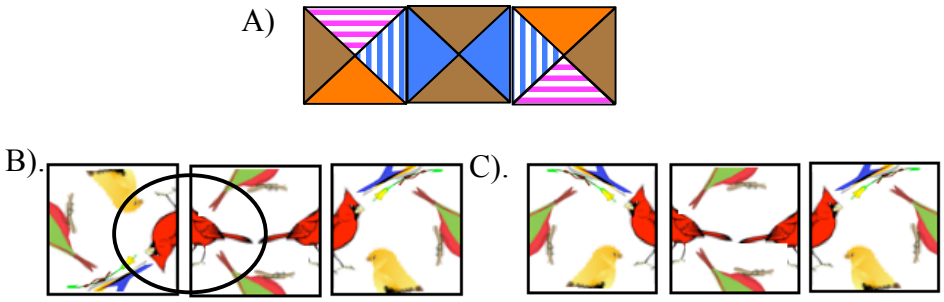


Figure 18. (A) Solution to an instance of the complementary bounded tiling problem with rotations shown in Figure 17A. (B) Using an algorithm to solve the problem with

complementary images without reflections will not result in a solution to the complementary bounded tiling problem with rotations but without reflections. (C). One needs to reflect one tile in order for an algorithm from the complementary bounded tiling problem with reflections to solve an instance of the complementary bounded tiling problem with rotations.

Hence, we have reduced the complementary bounded tiling problem with reflections from the complementary bounded tiling problem with rotations, but without reflections. Therefore the complementary bounded tiling problem with reflections is NP-hard.

□

We have proven that the complementary bounded tiling problem with rotations and the complementary bounded tiling problem with reflections is NP-hard. To prove that the complementary bounded tiling problem with rotations and reflections is NP-hard we would need to apply the same transformation steps, for both rotations and reflections, to the bounded tiling problem. Combining the above proofs into one proof shows that the complementary bounded tiling problem with rotations and reflections is NP-hard. We have already shown that the complementary bounded tiling problem with rotations and reflections is in NP and now we have shown that it is NP-hard. Thus, the complementary bounded tiling problem with rotations and reflections is NP-complete. A solution to the complementary bounded tiling problem can be reduced to a solution to the bounded tiling problem, which can be reduced to any other NP-complete problem, such as the traveling salesman problem, for which many applications exist, as detailed in Chapter I.

Chapter III

Biological Mathematics

Given the complexity of NP-complete problems as discussed in Chapter I, researchers have turned to the field of biological mathematics in order to find solutions. My research harnesses the inherent properties of DNA to assemble and solve the NP-complete complementary bounded tiling problem, which I described in Chapter II. The concept of using DNA, to solve mathematical problems relies on the field of biological mathematics.

This chapter focuses on a particular topic in biological mathematics, known as DNA computation, and its applications in solving NP-complete mathematical problems. To understand how DNA computation can be used to solve the complementary bounded tiling problem, one must first grasp the background of biological mathematics and DNA computation, as well as the advantages and disadvantages of using such biological models.

Biological Mathematics

The classical intersection between biology and mathematics applies mathematical concepts to decipher biological data and model biological systems. Until recently, the interdisciplinarity of these two fields had been one directional, using math to understand biology. It was not until 1994 that researchers began to use biology to understand math, specifically harnessing biological molecules, such as DNA, to model and solve mathematical problems (Deaton, *et al*., 1997). The intersection of math and biology has become bi-directional, using mathematics to understand biological systems and data and using biological systems and molecules to understand mathematical problems.

Within the field of biological mathematics, various biological molecules can be used to study and model mathematical problems. Yet, since the field's inception, DNA has been used

frequently as a stable, predictable biological model for mathematical problems (Kari , 1997). DNA computation specifically uses DNA to implement computational algorithms in order to solve mathematical problems (Amos, 2008). The basis of DNA computation and the relationship between DNA and mathematics relies on the fact that the two processes below, one biological and one mathematical, are analogous (Kari , 1997):

1) The complex structure of a living being is the direct ". . .result of applying simple operations (copying, spicing, etc.) to initial information encoded in a DNA sequence."

2). The result of a mathematical function $f(x)$ "can be obtained by applying a combination of basic simple functions to x."

The acknowledgement that the above statements are parallel led to the idea that DNA sequences can be used to encode mathematical information, in the form of 4 bases (A, T, G and C), while biological operations, carried out by enzymes and other proteins, can be used to manipulate such information in accordance with a computational algorithm. Examples of biological operations include cutting with restriction enzymes and lengthening with DNA ligase (Amos, 2008).

Encoding information within DNA strands and manipulating such strands with simple biological operations was first used in a 1994 publication (Adleman, 1994). This paper was the first to show that computational algorithms could be implemented using DNA to solve a particular type of mathematical problem, an NP-complete problem. Since 1994, the field of biological mathematics, particularly DNA computation, has been widely explored as a mode of solving mathematical problems, in particular NP-complete problems.

Adleman's 1994 Publication

As discussed in Chapter I, no efficient algorithm exists to solve NP-complete problems. In the absence of efficient algorithms, the only way to decrease the run time for finding a solution is

to break down the problem and let more hardware conquer it simultaneously. Unfortunately, hardware is expensive. A fairly recent idea, introduced by Leonard Adleman in 1994, is to solve NP-complete problems using DNA computation, allowing DNA to solve the problem. Adleman proposed a DNA sequence based approach to solve the NP-complete directed Hamiltonian Path problem (HPP).

Directed Hamiltonian Path Problem:

> Given a directed graph, *G*, with designated in and out vertices (v_{in} and v_{out}), does there exist a path, beginning at v_{in} and ending at v_{out}, that touches every vertex in *G* exactly once (Dasgupta *et al.*, 2006; Figure 1)?

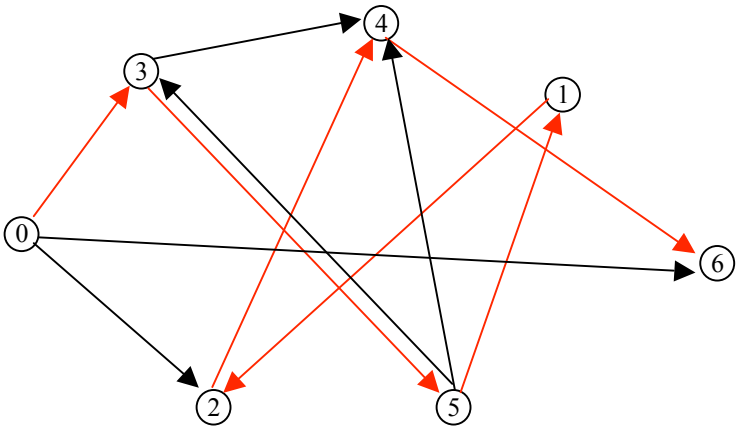


Figure 1. Given the directed graph, *G*, where v_{in}=0 and v_{out}=6, there does exist a Hamiltonian Path in red: 0 →3, 3 →5, 5 →1, 1→2, 2 →4, 4 →6.

A brute-force algorithm exists to solve the directed Hamiltonian Path problem. The algorithm, as seen below, generates all possible random paths with exactly *n*-1 edges (*n* is the number of vertices) and determines whether one of these paths fulfills the qualifications of being a directed Hamiltonian Path.

> Step 1. Generate all random paths through the graph.
>
> Step 2. Keep only those paths that begin with v_{in} and end with v_{out}.
>
> Step 3. If the graph has *n* vertices, then keep only those paths that enter exactly *n* vertices.

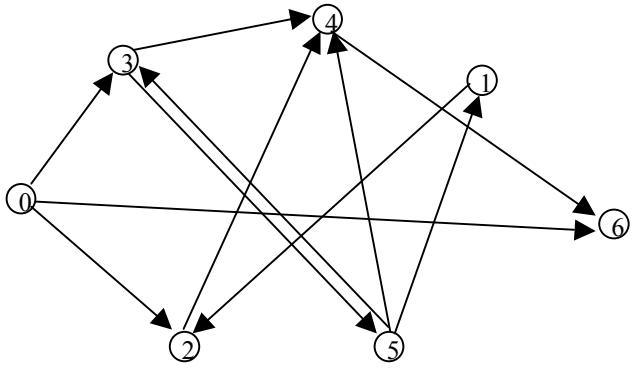
Step 4. Keep only those paths that enter all of the vertices of the graph at least once.

Step 5. If any paths remain, say "YES", meaning that a Hamiltonian Path does exist; otherwise say "NO" (Adleman, 1994).

Even though an algorithm exists to solve the directed Hamiltonian Path problem, it is a brute-force algorithm and therefore is inefficient.

Adleman modeled the brute-force algorithm shown above using DNA. He encoded a particular instance of the problem into DNA sequences. Harnessing the inherent Watson-Crick base pairing properties of DNA and utilizing various biological techniques, he was able to determine a solution to the Hamiltonian Path problem (Adleman, 1994).

Step 1: To implement step 1 from the brute-force algorithm biologically, Adleman randomly associated each vertex, *i*, in the directed graph with a single stranded DNA sequence that was 20 nucleotides long, a 20-mer, labeled O_i. In order to produce the random paths, Adleman first engineered a new 20-mer sequence composed of part of one vertex and part of another vertex ($O_{i \to j}$), dependent upon which vertices had connecting edges. In other words, for each pair of adjacent vertices a 20-mer nucleotide sequence was produced to represent the edge connecting those two vertices, where the 3' 10-mer of O_i was annealed with the 5' 10-mer of O_j (Figure 2). If i=0 in $O_{i \to j}$ then the whole 20-mer was simply the O_1 DNA sequence and if j=6 then the whole 20-mer sequence was the O_6 DNA sequence. Next, Adleman mixed every $O_{i \to j}$ with the complement of each vertex strand, indicated by $\bar{O}_i$. The complementary strand annealed via Watson-Crick base pairing, bringing together potential edges to produce a potential path in accordance with the directed edges (Figure 2).



Vertices: Engineered 20-mer Sequence:

O_1 5' ATGCCTAGCCATGAGTACCT 3'
O_2 5' TATCGGATCGGTATATCCGA 3'
O_3 5' GCTATTCGAGCTTAAAGCTA 3'
O_4 5' GGCTAGGTACCAGCATGCTT 3'

Complement of Vertices:

$\bar{O}_1$ 5' TACGGATCGGTACTCATGGA 3'
$\bar{O}_2$ 5' ATAGCCTAGCCATATAGGCT 3'
$\bar{O}_3$ 5' CGATAAGCTCGAATTTCGAT 3'
$\bar{O}_4$ 5' CCGATCCATGGTCGTACGAA 3'

Possible Edges within *G*: Engineered 20-mer Sequence:

$O_{1\to2}$
ATGAGTACCTTATCGGATCG
$O_{2\to4}$
GTATATCCGAGGCTAGGTAC
$O_{3\to4}$
CTTAAAGCTAGGCTAGGTAC

Possible Partial Paths:

$O_{1\to2}$ $O_{2\to4}$
ATGAGTACCTTATCGGATCGGTATATCCGAGGCTAGGTAC
ATAGCCTAGCCATATAGGCT
$\bar{O}_2$

Figure 2. An example of step 1 for Adleman's DNA computation model to solve an instance of the directed Hamiltonian Path problem. The above details a few possible 20-mer vertex sequences as well as a few possible edge 20-mer sequences and shows how assembly into possible paths may occur. The colors help show how the vertex 20-mer sequences are used to produce possible edge sequences and eventually partial paths.

Step 2: Once Adleman produced random potential paths, he utilized PCR with v_{in} and v_{out} specific primers to amplify only the random paths that started with the v_{in} sequence and ended with the v_{out} sequence.

Step 3: To implement step 3 of the brute-force algorithm, Adleman used gel electrophoresis to separate the randomly generated paths beginning with v_{in} and v_{out} isolated in step 2. The paths of appropriate length (indicating that the path touched n vertices) were isolated. In this particular case, the DNA strands of length 140 base pairs indicates that exactly 7 edges were included in the path (20-mers · 7 edges).

Step 4: In order to ensure that the product from step 3's gel purification included each vertex exactly once, Adleman used a process known as affinity purification. He generated single-stranded DNA from the purified product in step 3 and mixed the strands with the complement to the 20-mer of vertex 1, $\bar{O}_1$ attached to a magnetic bead. Therefore, if vertex 1 was included in the random path of size n, then the complementary sequence to vertex 1 would bind, along with the magnetic bead, and the random path containing vertex 1 would be retained. Adleman repeated this process of affinity purification with the retained random paths from the past purification using the complementary sequences to the other 6 vertices. In the end, only the paths containing all 7 vertices were left.

Step 5: To ensure that a solution existed, the product from step 4 was amplified using PCR and run on a gel. If a 140 base pair band appeared on the gel, then a path touching all 7 vertices, beginning and ending at v_{in} and v_{out} existed, and the answer to a particular instance of the directed Hamiltonian Path problem is yes. Otherwise, no Hamiltonian Path existed within the given directed graph, *G*.

Adleman harnessed the properties of DNA and laboratory techniques often used in biology to solve particular instances of the directed Hamiltonian Path problem. He encoded the instance of the problem in DNA and allowed Watson-Crick base pairing to find a solution. Adleman was the first to show that basic algorithms could be encoded within DNA sequences

and used to solve NP-complete mathematical problems. Since Adleman's successful demonstration of DNA computation to solve NP-complete problems, many other researchers have taken sequence based approached to solving other NP-complete problems, such as the 3SAT problem (Amos, 2008).

Advantages and Disadvantages: DNA Computation to Solve NP-complete Problems

While the field of DNA computation to solve NP-complete problems has grown, researchers have uncovered various advantages and disadvantages to using DNA computation in solving such difficult problems. DNA computing is generally considered a clever alternative for solving NP-complete problems and exhibits several advantages over silicon-based computers. For instance, DNA computing is a form of parallel computing, because it takes advantage of the millions of different copies of DNA molecules. In doing so, DNA computers try many different possibilities at once, making the parallel power of DNA many times faster than that of traditional machines. For example, a mix of 1,018 strands of DNA could operate at 10,000 times the speed of today's advanced supercomputers (Parker, 2003). Another advantage is the high potential for information storage. One gram of DNA can store as much information as approximately 1 trillion CDs (Vigliaturo, 2010). Additionally, DNA computing is remarkably energy efficient. For instance, considering the ligation of two DNA molecules to be one operation, 1J is sufficient for approximately $2 \cdot 10^{19}$ operations. In comparison, a 2006 super computer executed at most $2 \cdot 10^{9}$ operations per Joule (Adleman, 1994; Reddy, 2010). Finally, although initial programming of DNA computers software is expensive, hardware costs are negligible. For the above four reasons, many researchers think that DNA computing can fill in the gap where the capacity of traditional technology ends; for instance, the gap in which NP-complete problems cannot be

solved efficiently using modern silicon-based computers.

The main challenge of using DNA to solve mathematical problems involves uncertainty. The chemistry of DNA hybridization according to Watson-Crick base pairing is not 100% precise. As a consequence, algorithms encoded within DNA may not produce precise results when compared to a computer algorithm. In order to try and avoid errors within DNA computers, researchers must ensure that the initial sequences are of good quality and are free of contaminations. Contaminations often arise from incomplete oligonucleotides synthesis. Researchers are currently conducting tests to attempt to minimize error introduction during DNA computing by understanding and controlling DNA hybridization more closely and utilizing different synthesis methods (Deaton *et al.*, 1997). A second practical challenge involves scaling DNA computers to work on much larger problems. The most challenging aspects of NP-complete problems is the lack of efficient algorithms to achieve solutions for problems with large input. Hence, the concern of scaling DNA computers can prove a hurdle to solving NP-complete problems on a large scale. Specifically, Juris Hartmanis estimates that in order to solve such interesting and large mathematical problems using DNA computation it ". . .would require a volume of DNA that would fill the Pacific ocean or weigh as much as the earth" (Deaton *et al.*, 1997). Although, these two obstacles present challenges for DNA computing, research to overcome these obstacles is currently being conducted. My research utilizes the concepts of DNA computation, while attempting to minimize and understand these two obstacles.

Chapter VI

Bionanotechnology

In order to model the complementary bounded tiling problem biologically, we needed a structural building block, representing an individual tile, which could self-assemble to form larger structures, representing solutions to the complementary bounded tiling problem. For such building blocks, we turned to the field of bionanotechnology, specifically the sub-disciplines of DNA origami and DNA nanotechnology.

Nanotechnology

In the late 1950s, Richard Feynman founded the field of nanotechnology and introduced the idea of harnessing biological systems to act as nanoscale information processors (Amos, 2008). The roots of the word nanotechnology define its meaning. The prefix, nano-, refers to a nanometer. A nanometer is one billionth of meter (Figure 1). Nanoscience is built around the study of objects on the nanoscale. The suffix of the word, -technology, suggests the application of nanoscientific knowledge for practical purposes (Brucale, 2006).

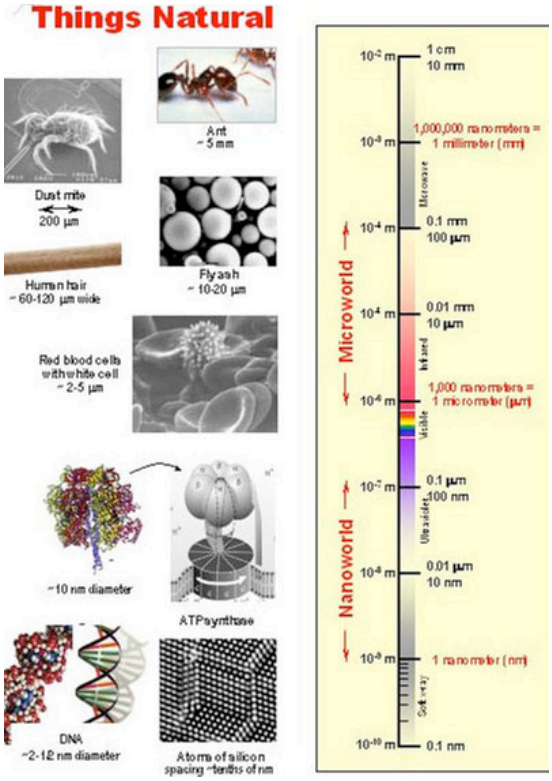


Figure 1. An illustration depicting size relationships amongst various naturally occurring organisms and biological molecules. The schematic shows a relative size comparison for objects on a nanoscale and microscale (Los Angles Harbour College, 2009).

There are two fabrication techniques for developing nanoscale objects: top-down and bottom-up. Top-down assembly "seeks to 'engineer' nanoscale structures or devices by using larger, externally-controlled tools to direct their assembly" (Brucale, 2006). Examples of top-down assembly include etching and lithography. Bottom-up assembly methods start from a structural subunit and allow self-assembly to build a final desired structure (Brucale, 2006). Living organisms posses innate mechanisms for bottom-up assembly of nanoscale structures. For example, organisms repeatedly build complex nanoscale proteins through protein folding and modification. G.M. Whitesides, a professor of chemistry at Harvard, recognizes the advantage of studying biological bottom-up assembly:

> . . . start with biology, which offers a cornucopia of designs and strategies that have been successful at the highest levels of sophistication. In tackling a difficult subject, it is sensible to start by studying at the feet of an accomplished master. Even if they are flagella, not feet. (Brucale, 2006)

The field of bionanotechnology harnesses biological processes of bottom-up fabrication for nanotechnology applications. Compared to other biological bottom-up processes, DNA stands out as the most adaptable nanoscale building block (Brucale, 2006). DNA nanotechnology capitalizes on the Watson-Crick base pairing of DNA, manipulating DNA strands to assemble into a variety of shapes. There are four unique features that make DNA an ideal molecule for bottom-up assembly of nanoscale objects.

1. DNA has a nanoscale structural geometry with a 3.4 nm helical repeat and approximately a 2 nm diameter (Figure 2).

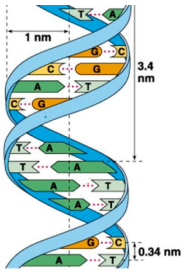


Figure 2. Schematic of DNA structural geometry, showing that DNA fulfills the nanoscale requirements for bottom-up assembly (©1998 Addison Wesley Longman).

2. DNA has predictable intramolecular interactions, specifically the conserved Watson-Crick base pairings (A-T and C-G; Figure 2). Hence, DNA's ability to store information within the nucleotide bases can be exploited in the predictable self-assembly process.
3. In addition to the predictable base pairings, the hybridization energies between such base pairings are known. For this reason, it is important to recall that a DNA molecule assumes the shape of its intrinsic lowest energy state. In other words, based upon the nucleotide sequence of the given DNA strand(s), the DNA molecule will assemble to minimize the energy utilized. Knowing the hybridization energies of each base pairing aids in understanding how DNA will assemble to form nanoscale structures. More detail on Watson-Crick hybridization energies and thermodynamics within DNA nanostructures will be given in Chapter VI.
4. DNA is relatively easy to synthesize and is physically and chemically stable. Therefore, DNA nanostructures are easy to handle and store. Methods for DNA purification and structural characterization are readily available (Brucale, 2006).

For the above reasons, we turned to DNA for bottom-up assembly of individual tiles within the complementary bounded tiling problem. Understanding the field of DNA nanotechnology is pivotal in selecting the DNA nanostructure to represent individual tiles within the complementary bounded tiling problem.

DNA Nanotechnology

DNA nanotechnology exploits the self-assembling ability of DNA to study and control the bottom-up assembly of objects at the nanoscale. There are three categories used to classify existing DNA nanostructure design approaches: 1) the scaffolded design, 2) the single-stranded design, and 3) the multi-stranded design. These three design techniques divide the field of DNA nanotechnology into two subfields: DNA origami and structural DNA nanotechnology. Paul Rothemund, the first to use DNA origami, classifies the first two approaches, scaffolded and single-stranded designs, as DNA origami because one long scaffold strand is folded. The third approach, multi-stranded, is used in structural DNA nanotechnology to construct regular and repeating DNA building blocks with specific geometric properties (Kuzuya *et al.*, 2009).

DNA Origami

DNA origami is the nanoscale folding of DNA to produce arbitrary two-dimensional and three-dimensional shapes. DNA origami utilizes a unique design approach, differentiating it from other forms of DNA nanotechnology. All DNA origami design approaches utilize one long strand of DNA, called the scaffold strand (Kuzuya *et al.*, 2009; Rothemund, 2006). In fact, there are generally two design approaches in DNA origami, single-stranded and scaffolded (Figure 3):

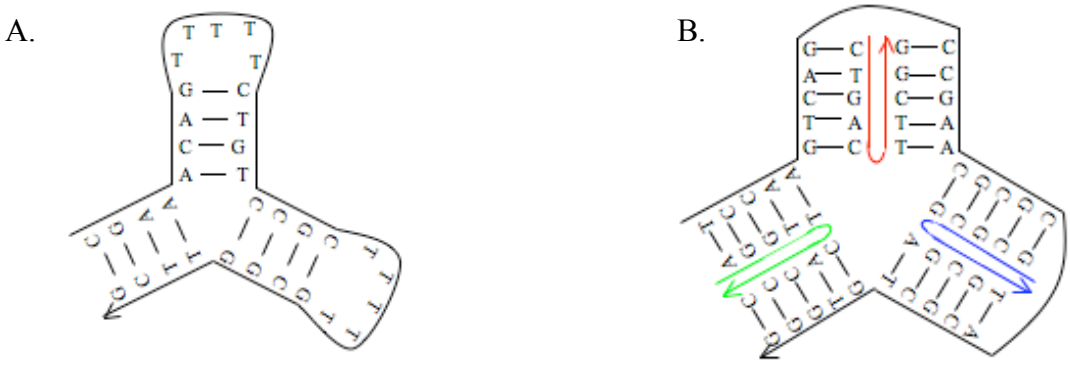


Figure 3. Two approaches to DNA origami design: (A) The single-stranded design uses one long

scaffold strand relying heavily on its ability to bind to itself (B) The scaffolded design is composed of one long scaffold single strand of DNA and multiple short staple strands of single-stranded DNA, known as oligonucleotides. The staple strands bind to the scaffold strand to determine shape. (Kuzuya *et al.*, 2009; Rothemund, 2006).

Currently, most DNA origami structures utilize the scaffolded design technique, which Paul Rothemund developed in 2006. Before Rothemund's innovation, shapes were assembled using the single-stranded design technique. In the single-stranded design technique, one long single-stranded DNA scaffold binds to itself. The most successful example the single-stranded design technique is the DNA octahedron (Kuzuya *et al.*, 2009; Figure 4).

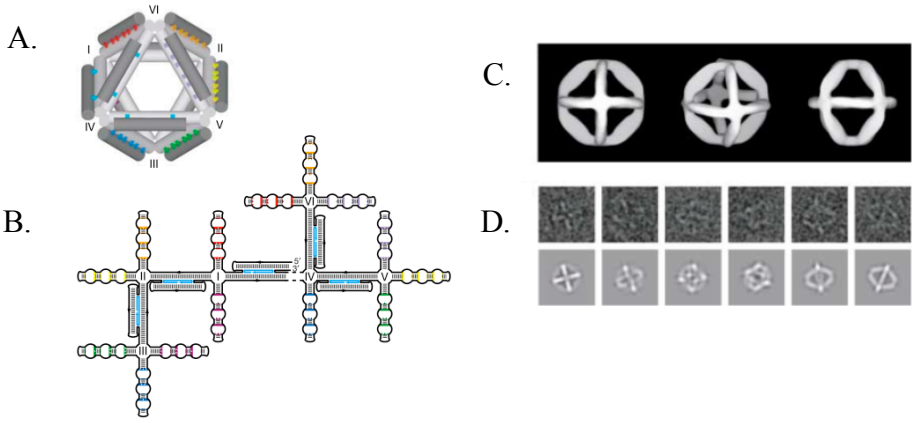


Figure 4. Nanoscale DNA octahedron assembled using single-stranded design (A) Schematic three-dimensional structure of the DNA octahedron. (B) Two-dimensional image of how the single-strand scaffold folds on itself to form a branched structure. (C) Three-dimensional reconstruction of microscopy images representing the DNA octahedron. (D) Images from an AFM of the DNA octahedron and the corresponding reconstruction diagrams (Kuzuya *et al.*, 2009).

The scaffolded design uses one long scaffold strand of a single-stranded DNA, usually from a viral genome. The folding is controlled by the addition of engineered oligonucleotides (Rothemund, 2006; Figure 5).

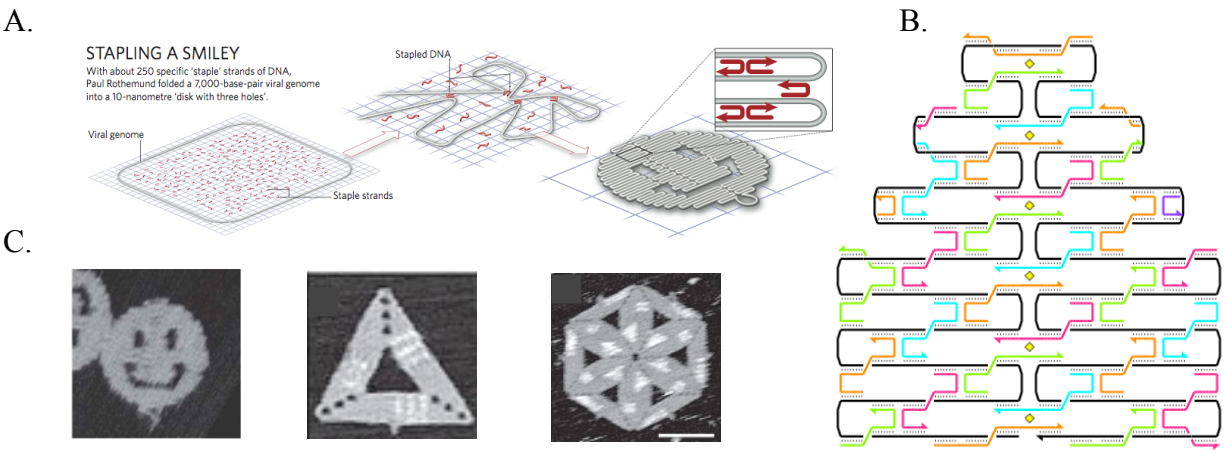


Figure 5. DNA origami scaffolded design (A) Scaffolded design to construct a DNA origami smiley face. Single-stranded oligonucleotides (shown in red) were added to the viral genome scaffold strand to determine smiley face structure (Sanderson, 2010). (B) Scaffold single-stranded piece of DNA, in black, runs through the entire structure. Staple oligonucleotides, multicolor, bind to various regions of the scaffold via Watson-Crick base pairings to form the desired shape. (C) Atomic force microscope (AFM) pictures of DNA origami shapes produced using the scaffolded design.

Structural DNA Nanotechnology

The concept of structural DNA nanotechnology is similar to DNA origami: harnessing the Watson-Crick base pairing of DNA to form shapes. The two fields differ in design approaches. DNA origami utilizes scaffolded and single-stranded design techniques, whereas structural DNA nanotechnology utilizes a multi-stranded design (Kuzuya *et al.*, 2009). The multi-stranded design approach uses only small single-stranded oligonucleotides that self-assemble into higher

order structures (Rothemund, 2006; Figure 6).

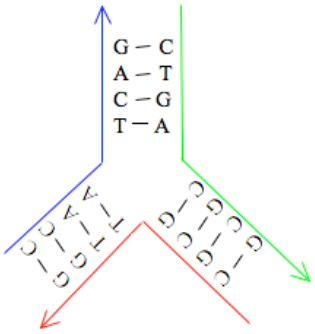
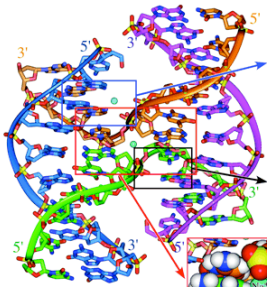


Figure 6. Multi-stranded design technique for structural DNA nanotechnology. Three oligonucleotides assembled into a triangular shape. There is no larger scaffold strand in this design approach, only smaller oligonucleotides (Rothemund, 2006).

The main goal of structural DNA nanotechnology is to "provide an operatively simple way to control the positioning of matter in the nanoscale through the bottom-up self-assembly of DNA strands" (Brucale, 2006). More than twenty years ago, Ned Seeman used a multi-stranded design to engineer the first instance of structural DNA nanotechnology: the 4-way branched DNA junction (Seeman, 1983; Brucale, 2006). The 4-way branched DNA junction was equipped with sticky-ends allowing assembly into larger DNA lattice structures (Seeman *et al.*, 1998; Figure 7B). The idea capitalized on a naturally occurring state of DNA known as the Holliday Junction (Aldaye *et al.*, 2008; Figure 7A). A Holliday junction is a mobile junction between four DNA strands, which occurs as a form of genetic exchange between homologous sequences. The shape of the Holliday junction is highly conserved between prokaryotes and eukaryotes, including mammals (Hays *et al.*, 2003).

A. 

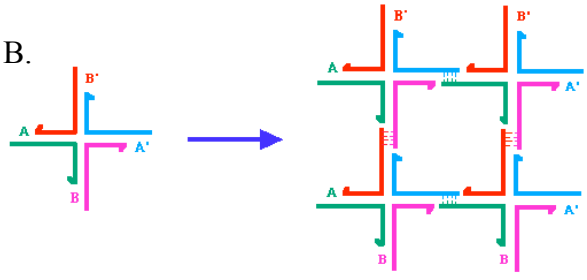
B. 

Figure 7. (A) Schematic of a DNA Holliday junction, where the four different colors (orange, purple, blue, and green) represent the different DNA strands (Hays *et al.*, 2003). (B) Schematics of the first DNA structural building block modeled after the Holliday junction, where the four colors represent the four DNA strands (A, B, A', B'). The lattice to the right shows how four copies of one DNA subunit can self-assemble into a larger DNA lattice (Seeman *et al.*, 1998).

Scientists eventually discovered that the 4-way branched DNA junction (Figure 7B) was not as stable as originally presumed. The Holiday junctions were mobile (Figure 8A) and flexible (Figure 8B), threatening the uniform nature of the structural 4-way branched DNA building block. Mobility of the Holiday junction occurs because the intersection point between the four strands is flanked by homologous symmetric sequences. Due to these similar sequences, the branching point between the four strands can migrate back and forth (Brucale, 2006; Seeman, 2003; Figure 8A). The inability of the DNA nanostructure to maintain its ideal shape makes it an unstable building block. Flexibility of the Holiday junction refers to the inability of the structure to maintain the 90º angles between branching points (Figure 7A). In other words, the intersection point can bend in a scissoring motion (Figure 7B). The strands are flexible because DNA segments have a natural curvature and there is no force dictating that the branching points remain at a 90º angle (Liu *et al.*, 2003; Figure 8B).

A. B.

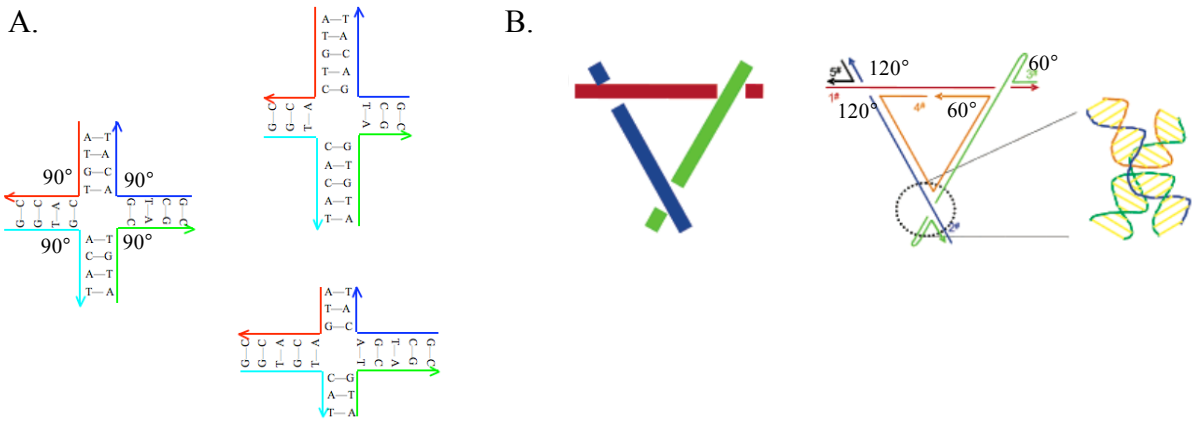


Figure 8. (A) Experimentation showed that the 4 arms on the DNA branched junction were mobile, meaning that they could slide. (B) Multiple schematics of the flexible DNA junction, where triangles (with 120° and 60° angles) can form due to the flexibility of the intersection point between all 4 arms (Liu *et al.*, 2003).

4x4 DNA Cross Tile

The 4x4 DNA cross tile, a multi-stranded design, consists of nine single-stranded DNA strands: four shell strands, four arm strands and one core strand (Park *et al*., 2006; Figure 9A). The sequence design of all nine strands, as well as controlled assembly conditions, allows for uniform assembly of the given cross tile structure. To allow the 90º cross tile assembly, the core strand contains four T4 loops, four thymines in a row. The thymine residue is known to preferably bend and the repeated thymines allow the junction to form with the desired 90º angles (Park *et al*., 2006).

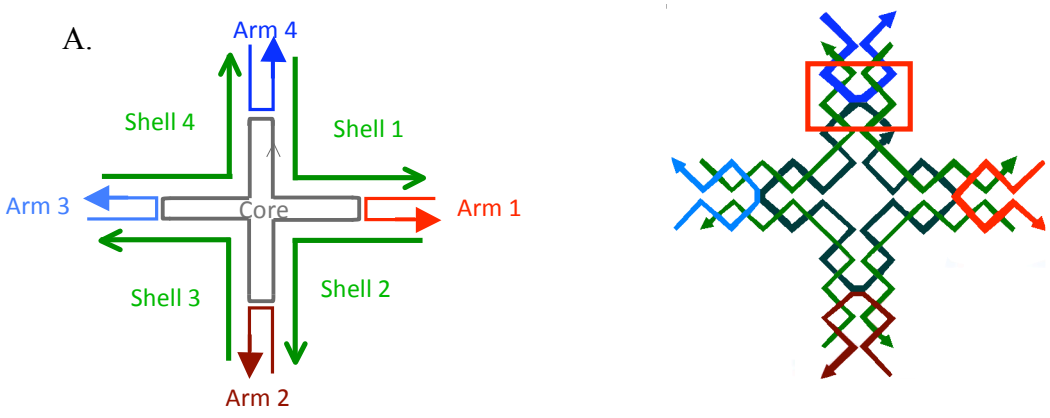


Figure 9. 4x4 DNA cross tile (A) Nine DNA strands assemble to make the 4x4 DNA cross tile: 1 core, 4 shell and 4 arm single-strands of DNA (Park *et al*., 2006). (B). Visual schematic of the DNA helices of the 4x4 DNA cross tile. Note the core strand (dark grey) interacts with all 4 shells and the red box highlighting the crossover event (Park *et al*., 2006).

There are three main differences between the structure of the 4x4 DNA cross tile and the 4-way DNA junction.

1) Sequence similarity and complementarity in all nine strands is much lower in the 4x4 DNA cross tile. Within the DNA sequences of the 4x4 DNA cross tile, no 5 nucleotides appear twice in the same order (Lin *et al.*, 2009). Minimizing sequence homology ensures less branch mobility and therefore greater uniformity.
2) The core strand connects all four arms, stabilizing the DNA nanostructure (Figure 9B).
3) Interactions between helices are strengthened by the existence of a reciprocal exchange in each arm strand (Lin *et al.*, 2009; Figure 9B). The reciprocal exchange between each arm and shell strand is known as a crossover event (Brun *et al.*, 2004; Figure 10). Each crossover event adds stability to the arm strands and helps maintain the 90º angles in each shell strand.

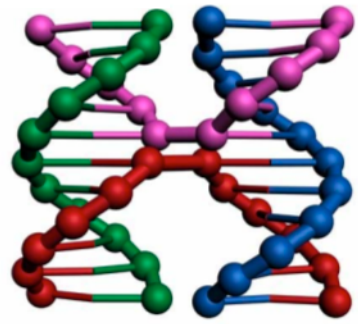


Figure 10. Diagram of a crossover event, which increases the rigidity, immobility, and stability of the 4x4 DNA cross tile (Brun *et al.*, 2004).

As in the 4-way branched DNA junction, there are overhanging sticky-ends on each arm strand. These sticky-ends allow individual 4x4 DNA cross tiles to form DNA lattices (Figure 11) through Watson-Crick base pair binding. The 4x4 DNA cross tile offers more control between tile interactions when compared to the 4-way branched DNA junction because there are two sticky-ends per arm. In the 4-way branched DNA junction only one 5-nucleotide DNA sequence must anneal to bring together two tiles. In the 4x4 DNA cross tiles there are two 5-

nucleotide sequences that must anneal. The existence of 2 sets of 5-nuceltoides sticky-ends increases the number of different half-images that can be encoded within the complementary bounded tiling problem.

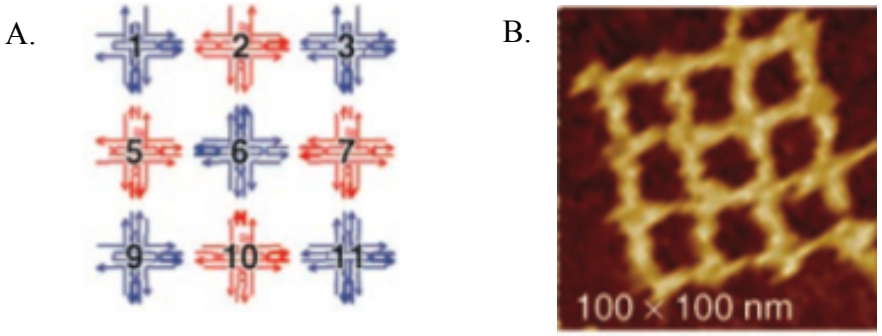


Figure 11. DNA tile lattice, in which multiple 4x4 DNA cross tiles anneal through Watson-Crick base pairings of the sticky-ends. (A) Schematic of a 3x3 lattice (B) AFM image of a 3x3 DNA lattice made with cross tiles (Park *et al.*, 2006).

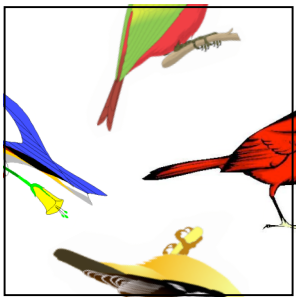
Because of its stability and rigidity, we chose to use the 4x4 DNA cross tile to represent a tile in the complementary bounded tiling problem. Bottom-up assembly, or self-assembly, of multiple tiles forms DNA lattices that may tile a given bounded region. The following section provides more details for the implementation of the 4x4 DNA cross tile in the complementary bounded tiling problem.

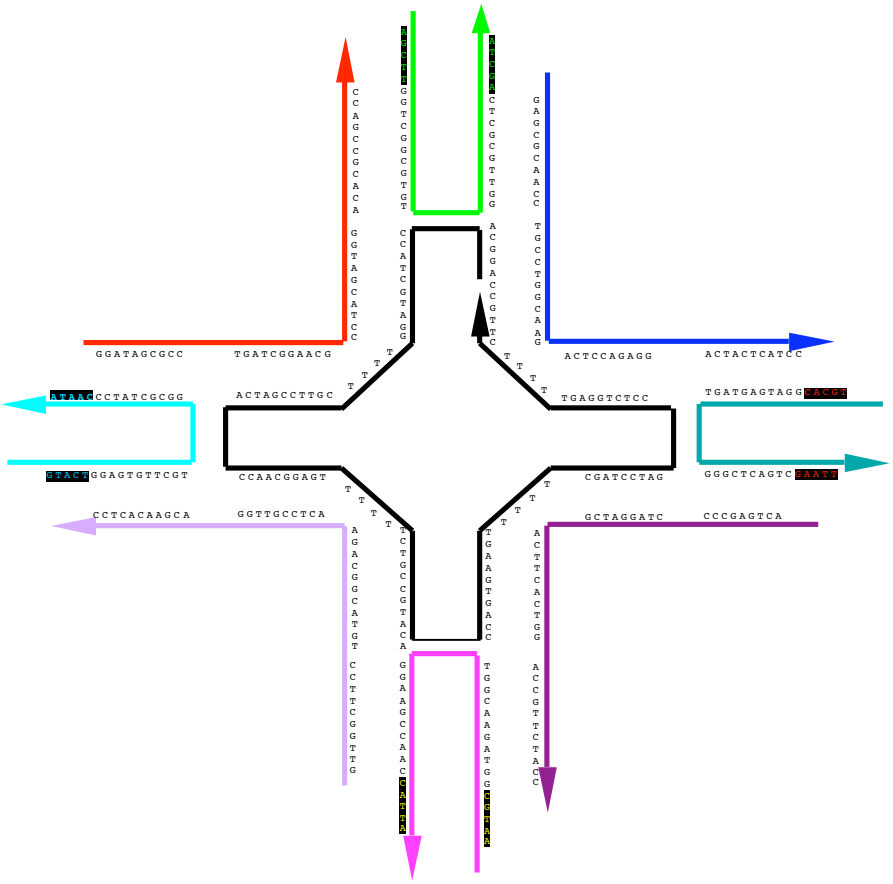
Implementation of 4x4 DNA Cross Tiles in the Complementary Bounded Tiling Problem

The following illustrations serve as a model to introduce how a particular instance of the complementary bounded tiling problem can be modeled using 4x4 DNA cross tiles.

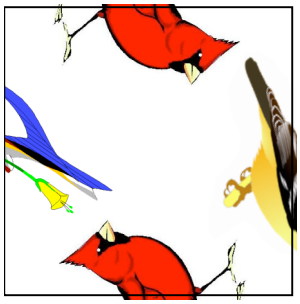
In order to model a particular tile set for the complementary bounded tiling problem, we must first encode the equivalent individual 4x4 DNA cross tiles.

Tile One





Tile Two



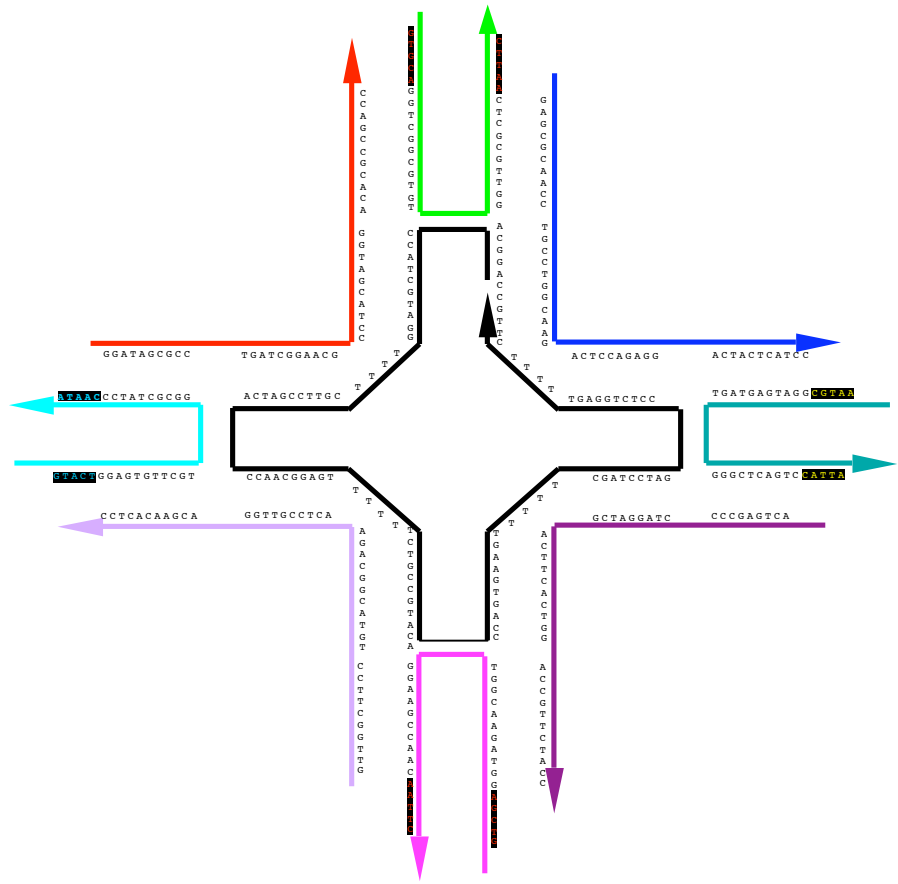


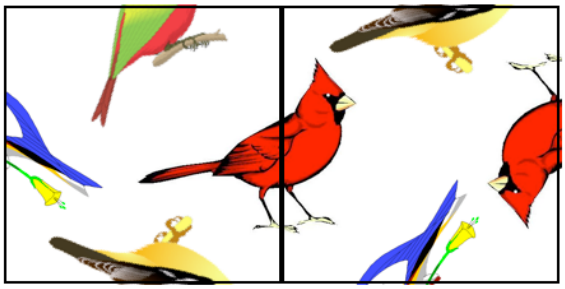
Figure 12. The tile set given above can be modeled by two analogous 4x4 DNA cross tiles. Observe the sticky-end sequences highlighted in red in comparison to the half-images on the tiles.

Sticky-end	Corresponding Image	5'→ 3' Sequence
A	Beak of Red Bird	GTGCA ... AATTC
-a	Tail	TGCAC ... GAATT
B	Beak of Green Bird	AAGCT ... TAGCT
-b	Tail	AGCTT ... AGCTA
C	Beak of Blue Bird	AGTAC ... TATTG
-c	Tail	GTACT ... CAATA
D	Beak of Yellow Bird	GCATT ... TAATG
-d	Tail	AATGC ... CATTA

Figure 13. Table representing the corresponding sticky-ends for all of the images in the above tile set. Cross-referencing the sticky-ends provided in this table with those seen in Figure 12 will correspond accordingly.

A particular solution to the given complementary bounded tiling problem in Figure 12 is shown in Figure 14. Notice the rotation of the text of the second tile. Tile two was rotated 90 degrees counterclockwise in order to give the particular solution to the complementary bounded tiling problem shown below.

A.



B.

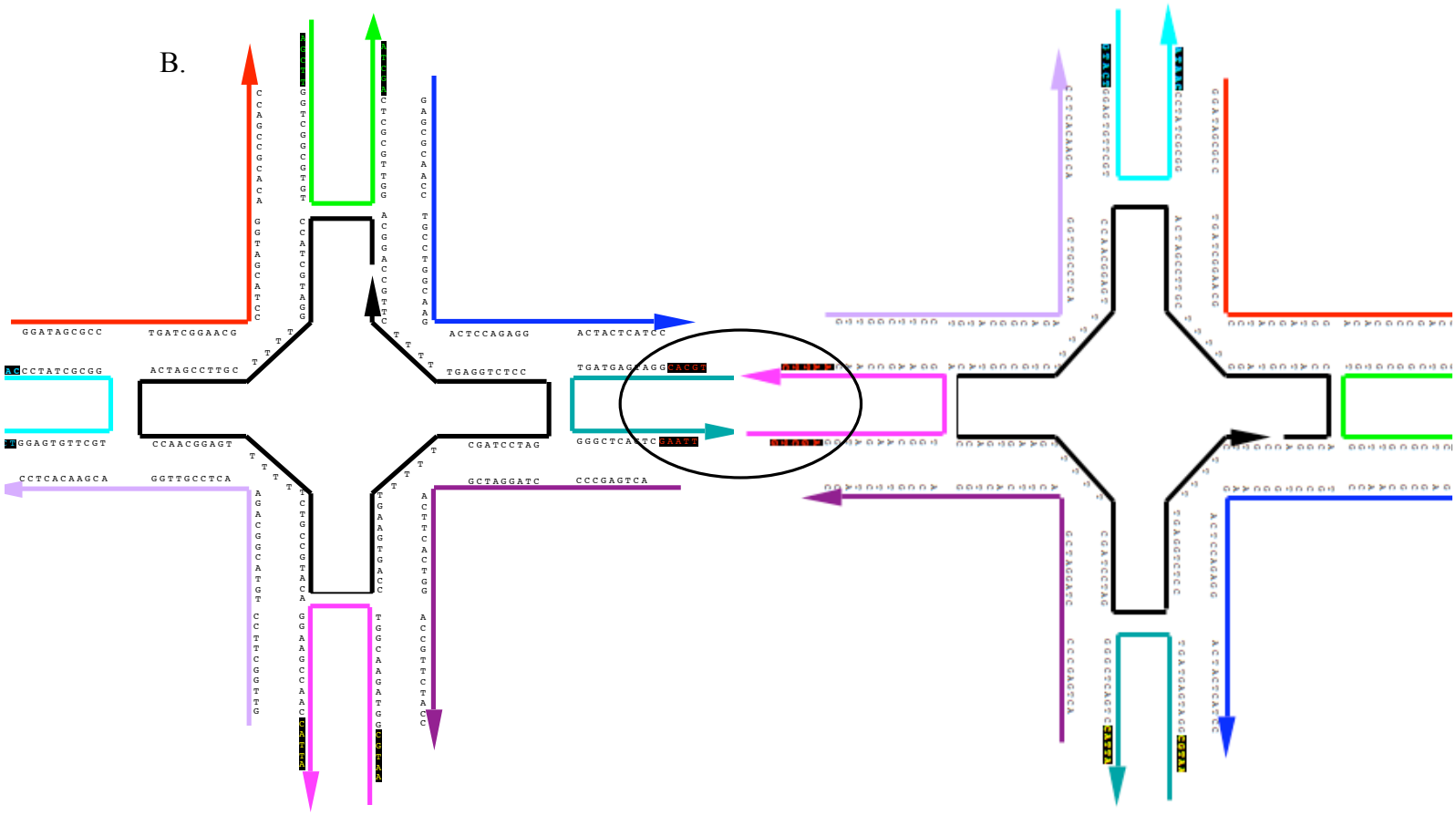


Figure 14. (A) A particular solution to the complementary bounded tiling problem presented in Figure 12. (B) Tile two was rotated 90 degrees counterclockwise in order for the red bird's beak to match with the red bird's tail.

Zooming in to where the two 4 x4 DNA cross tiles meet (the red bird's beak and tail) exposes some unique and challenging issues of larger lattice structures. The section below represents the sticky-ends, which overhang from the circled section in Figure 14. Recall that these two sets of sticky-ends should bind the two 4x4 DNA cross tiles together.

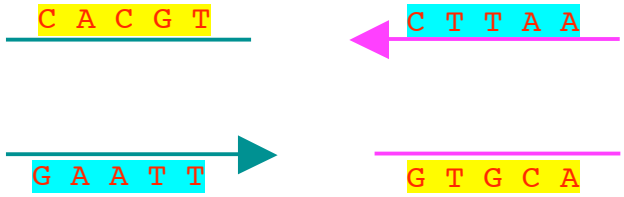


Figure 15. Depicts the sticky-end relationship between the two 4x4 DNA cross tiles pictured in Figure 14.

Studying the relationship between the sticky-ends above sheds light on how I designed the 4x4 DNA cross tiles to interact and form larger DNA lattices. Observe the two sequences highlighted with blue and the two sequences highlighted in yellow. The two blue sequences and the two yellow sequences are complementary when aligned properly, indicating that the 4x4 DNA cross tiles interact by crisscrossing DNA strands (Figure 16).

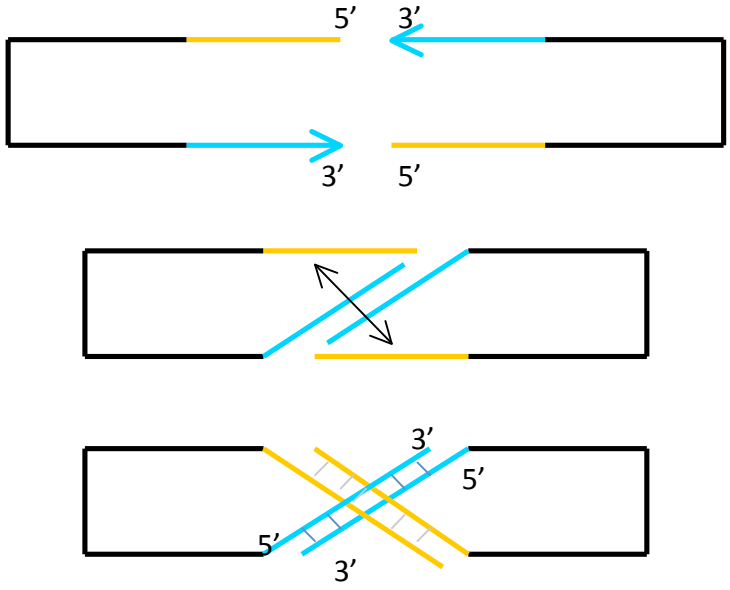


Figure 16. A step-wise schematic indicating how arm strands of two 4x4 DNA cross tiles interact. The strands highlighted in blue and yellow correspond to 5-nucleotide sticky-ends shown in Figure 15. The black strands represent the rest of the arm strands. For example, in the solution given in Figure 14, the interacting arms are arm one from tile one and arm three from tile two. In order to match the anti-parallel DNA strands, the DNA sequences from respective tiles must crisscross as shown in the third diagram.

Realizing that the sticky-ends between two 4x4 DNA tiles crisscross leads to two hypotheses concerning intermolecular relationships between multiple 4x4 DNA tiles:

1) The DNA strands may remain crossed as in Figure 16, OR

2) One whole 4x4 DNA cross tile may twist, or flip (reflect), to unwind the DNA strands, producing a structure like the one shown in Figure 18.

Researchers think that when two 4x4 DNA cross tiles interact and bind via the 5-nucleotide sticky-ends, one of the two tiles flips, or reflects, in order to untwist the DNA strands (Park, 2005; Figure 18 and Figure 11). Scientists think that the untwisting of the DNA strands is inherent in the structure due to the natural helical periodicity of DNA. Double-stranded DNA naturally assumes a double helical shape and the helical periodicity is approximately 10.5 base pairs per turn in the DNA. The binding between sticky-ends occurs in a region where the DNA naturally should turn, therefore providing chemical support for the untwisting of the DNA strands, and allowing the DNA to maintain its normal helical periodicity. Researchers have also observed planarity when analyzing larger lattices of 4x4 DNA cross tiles under microscopes, indicating that one of the tiles involved in an intermolecular interaction flips. Naturally, an individual 4x4 DNA cross tile is not planar; the tile has slight curvature due to the interacting

forces within the helical DNA molecules (Figure 17). If multiple 4x4 DNA cross tiles bind, the resulting structure would not be planar, but instead curved. Hower, under the assumption that at least one tile reflects to unwind the DNA strands, the resulting structure is closer to planar because the curvatures of the two interacting 4x4 DNA cross tiles cancel out.

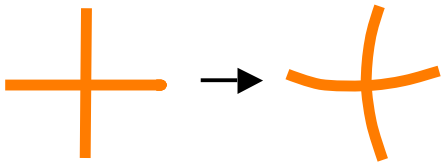


Figure 17. Diagram showing a sketch of the 4x4 DNA cross tile and its naturally occurring non-planar shape (Park, 2005).

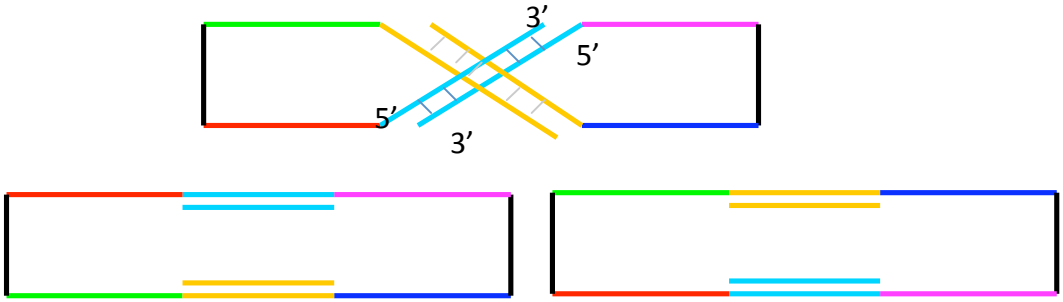


Figure 18. (A) The sticky-ends may remain crossed, or (B) one of the two tiles may flip or reflect, causing the strands to uncross. The two uncrossed schematics in B represent the two possible outcomes dependent upon whether tile one or tile two flips. The colors flanking the crisscross in A help decipher which tile flips in B.

The concept that one of the interacting tiles flips or reflects to unwind the DNA strands, called corrugation, leads to a nearly planar DNA lattice (Park, 2005; Figure 19). Figure 11 helps to visualize the reflection of individual 4x4 DNA cross tiles within a lattice, where only one tile was used and the two different colors indicate reflected versus un-reflected tiles. Now we revisit our original instance of the complementary bounded tiling problem given in Figure 12 and

proposed solution given in Figure 13. Before, when neither tile one nor tile two were reflected, the interacting sticky-ends needed to cross in order to bind correctly (Figure 14). If tile two is reflected (Figure 20), then the DNA strands do not cross (Figure 21).

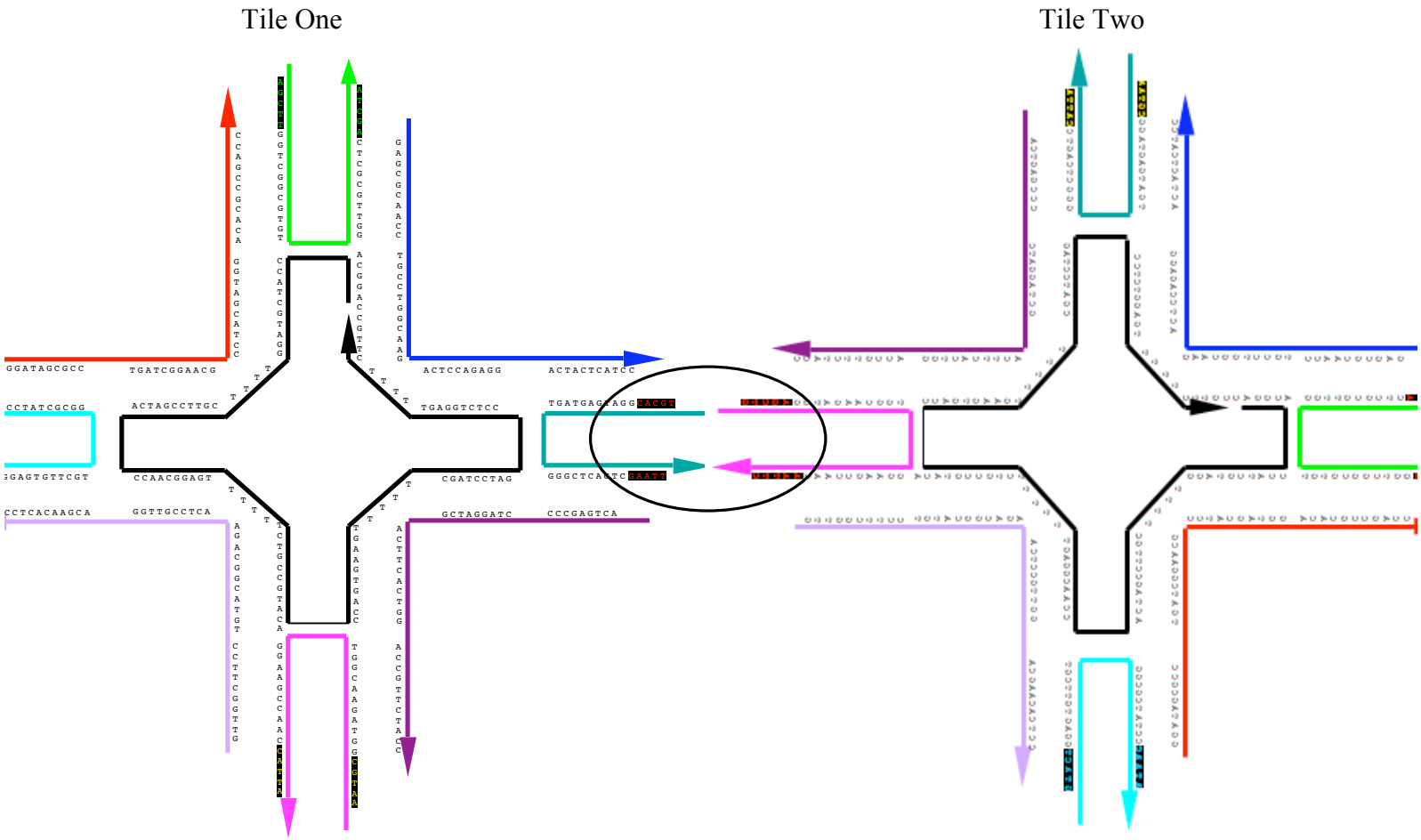


Figure 19. Tile two is not only rotated, but also reflected. The lettering labeling tile two is reflected as well.

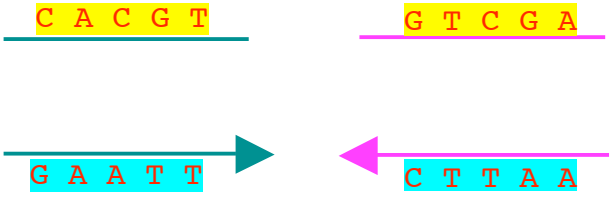


Figure 20. Sticky-ends in the region highlighted in Figure 19. Since tile two was reflected, the DNA strands no longer cross.

Understanding corrugation is important to understanding the 4x4 DNA cross tile sequence design. I designed tiles using the concept of corrugation, where at least one tile would presumably flip. Before I could begin testing the use of 4x4 DNA cross tiles in the complementary bounded tiling problem, I needed to model and understand the possible biological outcomes of different instances of the complementary bounded tiling problem. The following chapter explores a simulation of the complementary bounded tiling problem and the possible outcomes.

Chapter VI

Modeling The Complementary Bounded Tiling Problem

Before solving the complementary bounded tiling problem using 4x4 DNA cross tiles, it is important to understand the problem and how specific instances of the problem may assemble using DNA. When individual tiles are engineered to encode a specific instance of the complementary bounded tiling problem and mixed together in a test tube, random Brownian motion facilitates tile interactions and assembly into larger DNA lattices. The randomness of tile movement leads to partial and variable lattice formation. In other words, for a given instance of the problem, resulting lattices may consist of any number of tiles. Tiles are mixed in equal stoichiometric ratios, but tile concentrations affect the probability of tile incorporation. For instance, at high tile concentrations the probability that only two tiles will bind is low (if the given instance can assemble beyond two tiles). There is also no biological constraint ensuring that the tiles assemble in a prescribed structure such as a rectangle or square. Therefore modeling the complementary bounded tiling problem with DNA leads to variable DNA lattice structures not only in tile number, but also in structure and pattern. In order to answer the yes-or-no question posed by the complementary bounded tiling problem, I needed to understand and predict the possible outcomes of intermediate and variable DNA lattices associated with particular instances. To more accurately understand lattice formation within the test tube, Linda and I simulated specific instances of the complementary bounded tiling problem. Using MATLAB, we wrote two computer programs that simulate tile assembly using two different algorithms. I also constructed a Graphical User Interface (GUI) to make running the programs user-friendly.

The GUI (Figure 1) allows the user to specify:

1. The x-dimension and y-dimension of the bounded region.
2. The given tile set, specifying the number of tiles as well as the half-images on the edges of each tile. Each half-image is represented by a positive or negative integer. For example, 1 and -1 represent complementary half-images, and 2 and -2 represent another set of complementary half-images.
3. Whether or not reflections are permitted.
4. The intervals at which the step-by-step assembly of the tiling problem is displayed. For example, the user may choose to see the progression of a given instance one tile at a time or three tiles at a time.
5. Which algorithm to run, individually or a comparison of the two. Both algorithms (described in detail below) attempt to fill the specified $x \, x \, y$ bounded region with the given tile set. If the user compares the two algorithms, the choice of run number for the comparison is necessary.

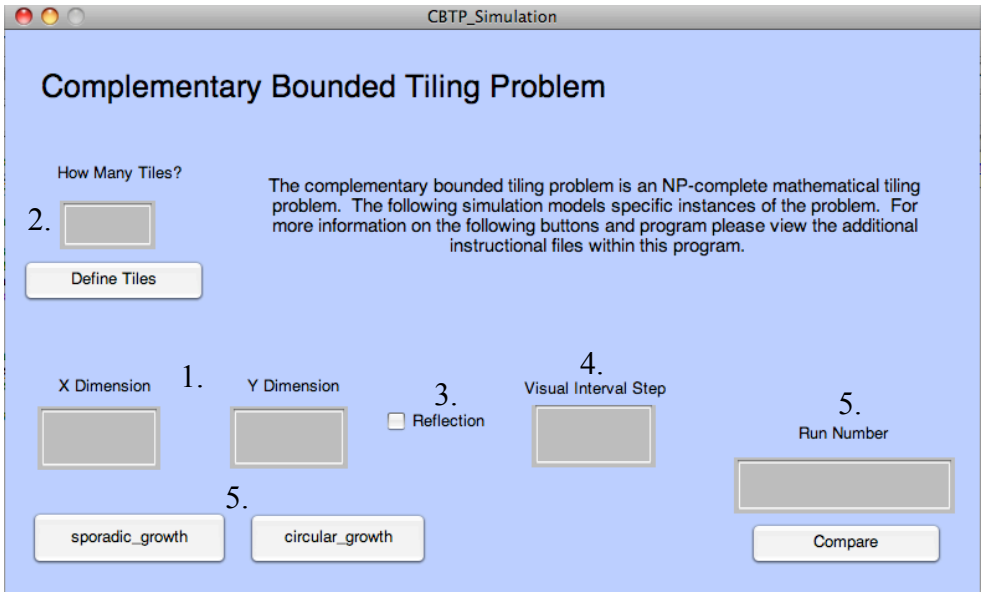


Figure 1. The GUI for the MATLAB simulation.

Algorithm One: circular_growth

Circular_growth randomly checks every edge of one tile for complementary matches before checking any other tile's edges for potential matches. Therefore, if every edge on one tile can be matched to four other tiles, then the growth of the lattice is circular, hence the name circular_growth. A random tile, x, is chosen from the given set of tiles and placed with a random rotation in the middle of the given bounded region. A random edge of tile x is chosen and a second random tile that fulfills the complementary edge matching requirements is chosen and placed next to this edge of tile x. Circular_growth continues until all four edges of tile x have been checked. This may mean that all four edges have matches or that none of the edges have matches within the given tile set. Once all four edges of tile x have been checked, a random tile, y, is chosen from among those matched with x, and the above steps are repeated. Therefore, the algorithm works on one tile at a time, attempting to match all edges of that tile with other given tiles. The algorithm terminates for two possible reasons:

1. If every unmatched edge does not have a cognate matching tile within the given tile set, then the lattice assembly cannot grow any further and the program terminates.
2. If the specified bounded region is filled with tiles that match at all touching edges. If a tile flanks an edge of the bounded region only the potential touching edges of that tile will be checked. In other words, if a tile exists within the corner of the bounded region, only two of its edges will be checked for matches.

Assembly patterns using circular_growth depend on given instances of the complementary bounded tiling problem. Different assembly patterns can be seen in Figures 3, 6B, and 7B.

Algorithm Two: sporadic_growth

In the sporadic_growth algorithm, the edge checked at each step is random. In other words, the program does not check each edge around one tile first and then move on to another tile. Instead, any open edge can be checked at any time in the progression of the program. Hence, there is a greater potential for sporadic growth, not circular growth, around one tile. Sporadic_growth represents a more realistic growth progression for a tiling problem within a test tube, because Brownian motion of 4x4 DNA cross tiles would not require all four edges of one tile be bound before other tile edges could be bound. Instead, the random movement of 4x4 DNA cross tiles would more likely lead to sporadic growth patterns. For sporadic_growth a random tile, x, in a random rotation is chosen from the given set of tiles and placed in the middle of the bounded region. Once placed, a random edge of tile x is chosen and a tile match is sought within the tile set given. If a matching tile is found, the algorithm continues but the algorithm does not need to check each edge of tile x. Instead, the sporadic_growth algorithm will randomly choose any open edge and look for a matching tile. Termination conditions are the same as for the circular_growth algorithm. Assembly patterns for sporadic_growth depend upon particular instances of the complementary bounded tiling problem and examples can be seen in Figures 4, 6C, and 7C.

Example One: Understanding Output and Comparing Algorithms

To illustrate the output of the simulation, I will walk through an example using a particular instance of the complementary bounded tiling problem. Given three tiles (Figure 2A) and their reflected versions (Figure 2B), can such tiles assemble to fill a 5x5 bounded region such that complementary images (or, in this case as represented in the MATLAB program, closely related shades of colors) match at all touching edges?

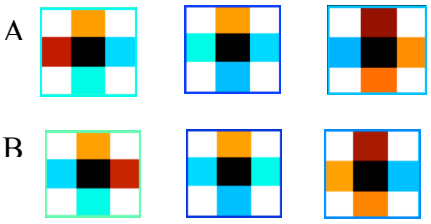


Figure 2. A six tile instance of the complementary bounded tiling problem, including three tiles (A) and their reflections (B).

Running circular_growth once will give one possible assembly of the above tiles. Figure 3 depicts three different assembly progressions (A-C) from running circular_growth three separate times. I ran each assembly progression with an interval of five, so each successive vertical image has five newly placed tiles. The vertical lane on the left-hand side of each panel represents the general tile placed in a given position. The different shades of blue are used in accordance with the blue shades outlining the given tiles in Figure 2. The right-hand side of the panel reveals information about each tile's particular rotation and shows how the close shades (or complementary shades) of colors match. The existence of two panels allows users to easily determine where specific tiles are placed and in what rotation.

A. B. C.

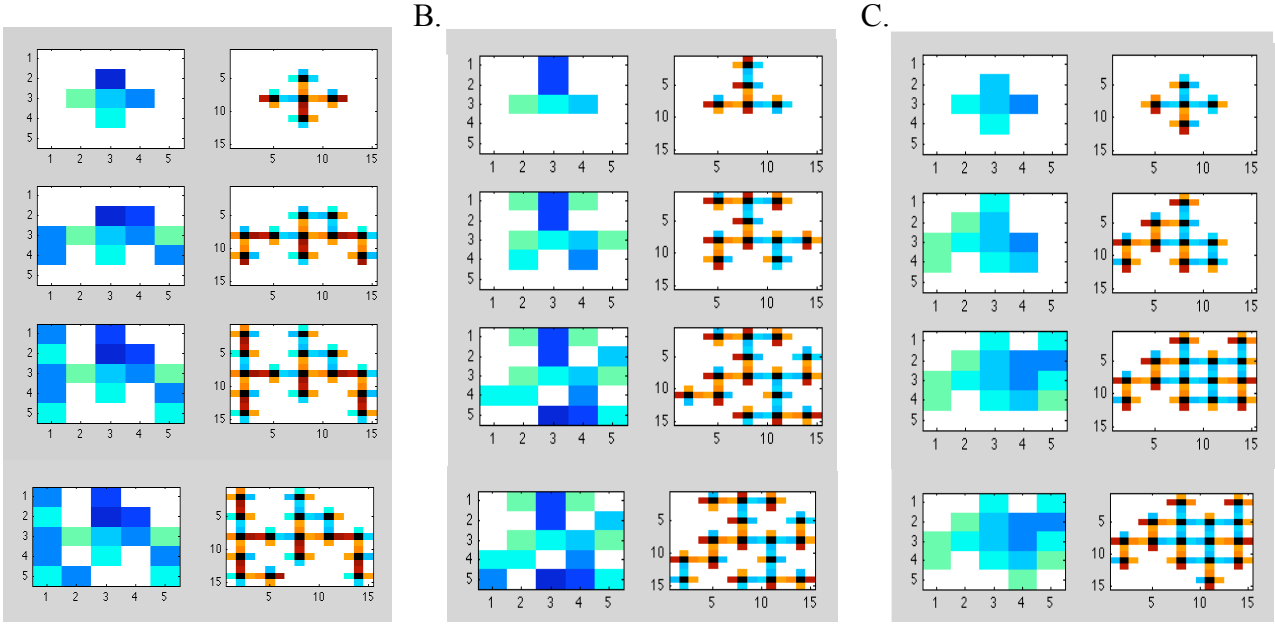


Figure 3. Different assembly progressions (A-C) of the given instance of the complementary bounded tiling problem given in Figure 2 using the circular_growth algorithm.

Using the sporadic_growth algorithm to evaluate tiling patterns of the instance given in Figure 2 shows different tiling progressions (Figure 4). The only variation from Figure 3 to Figure 4 is the algorithm used.

A. B. C.

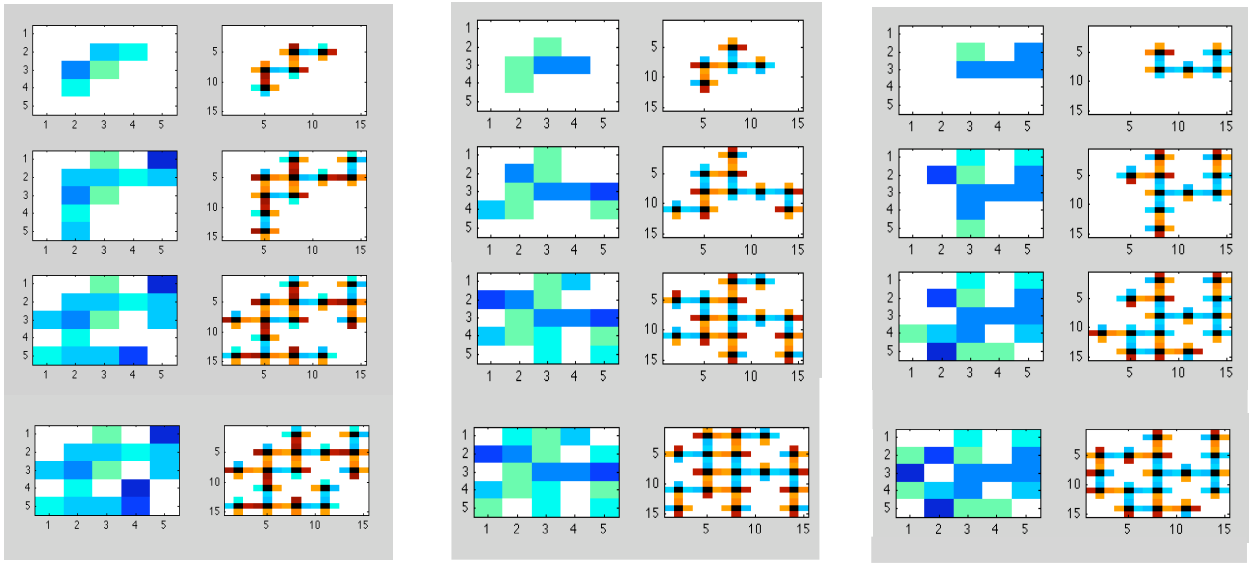


Figure 4. Different assembly progressions (A-C) of the given instance of the complementary bounded tiling problem in Figure 2 using the sporadic_growth algorithm.

The algorithm differences are clearly illustrated by the tile assembly progressions in Figure 3 (circular_growth) and Figure 4 (sporadic_growth). For example, two of the three tile assembly patterns (Figure 3A and 3C) for circular_growth completely fill in every edge surrounding the initial tile, the center tile. This progression is characteristic of the algorithm itself, but it also dependent on the instance of the problem given in Figure 2. Juxtapose these two initial tiling patterns with the initial tiling patterns in Figure 4. In Figure 4A-C the initial

tiling patterns are more sporadic, randomly choosing edges to check out of all of the placed tiles and not concentrating on the first tile. For example, in Figure 3C and 4B the initial center tile is the same. Hence we know that all four edges of this tile can be matched to other given tiles as seen in Figure 3C. Yet, the sporadic_growth algorithm does not initially fill all four edges of this tile (Figure 4B). Although, in later progressions all four edges of this initial tile are filled, they are not filled immediately, indicating the growth progression of the algorithm.

The three different assembly progressions in Figure 3 and 4 show possible lattice structure intermediates. The last vertical frame in each panel represents a final structure. The 5x5 region is not full in any of these tiling progressions because the program terminated. The program terminated because there were no tiles within the given tile set that would fit, such that all touching edges match, in the open spaces remaining. It is important to note a limitation of both algorithms: these six tiling progressions do not give a definitive answer to the given instance of the complementary bounded tiling problem in Figure 2. The complementary bounded tiling problem asks a "yes"-or-"no" question. Although the answer appears to be "no" given the final assembly patterns in Figure 3 and 4, it is impossible to know unless all possible tiling options were investigated. The two programs do not attempt to generate every possible tiling, because as noted in Chapter II, an algorithm to generate all tiling patterns runs in exponential time and is therefore inefficient to implement.

One way to provide stronger evidence that a complete tiling which fills the bounded region may not exist is to run the simulation thousands of times and study the number of tiles in the final tiling pattern. If the bounded region is full, then the final number of assembled tiles will be $x \cdot y$, or 25 in the instance shown in Figure 2. The "Compare" option generates a histogram indicating the number of final tiles for each run. For the tiling instance given in Figure 2, the

two algorithms were run 1,000 times and the results were compared (Figure 5). In the 1,000 tiling progressions, none of the final patterns completely filled the bounded region because there were no tiling progressions with 25 tiles (Figure 5), suggesting that the given tile set could not tile the bounded region.

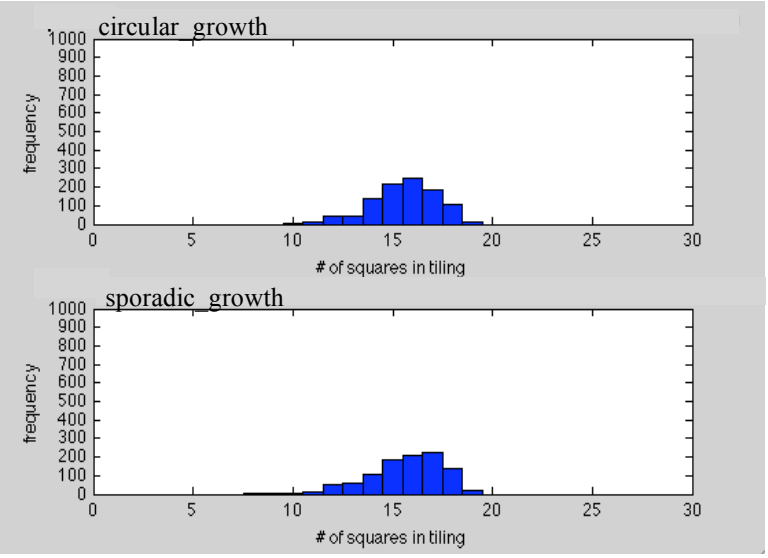


Figure 5. Histogram comparing algorithms indicating the distribution after 1,000 runs of the number of tiles within the final tiling pattern. Recall that for this instance, 25 tiles were needed to fill the region.

Example Two: Exploring Interesting Problems and Understanding Corrugation Likelihood

Understanding the algorithms and their output allows the exploration of more interesting instances of the complementary bounded tiling. For example, are there certain instances that will always tile the region? Will some only grow vertically or horizontally? Do some instances always produce the same tiling pattern? Will the addition of reflections drastically change tiling patterns? These algorithms are predicators of tiling behavior and have the potential to shed light

on many interesting tiling problems. The following figures explore different instances of the complementary bounded tiling problem, which answer some of the questions above.

Given two given tiles and no reflections (Figure 6A), can these tiles assemble to fill a 4x4 bounded region? In this instance, the tiles appear to only assemble into a square, filling a 2x2 bounded region. Assembly using circular_growth (Figure 6B) and sporadic_growth (Figure 6C) along with histograms using 1,000 runs (Figure 6D) show that a square 2x2 structure frequently forms and four tiles are in the final tiling pattern.

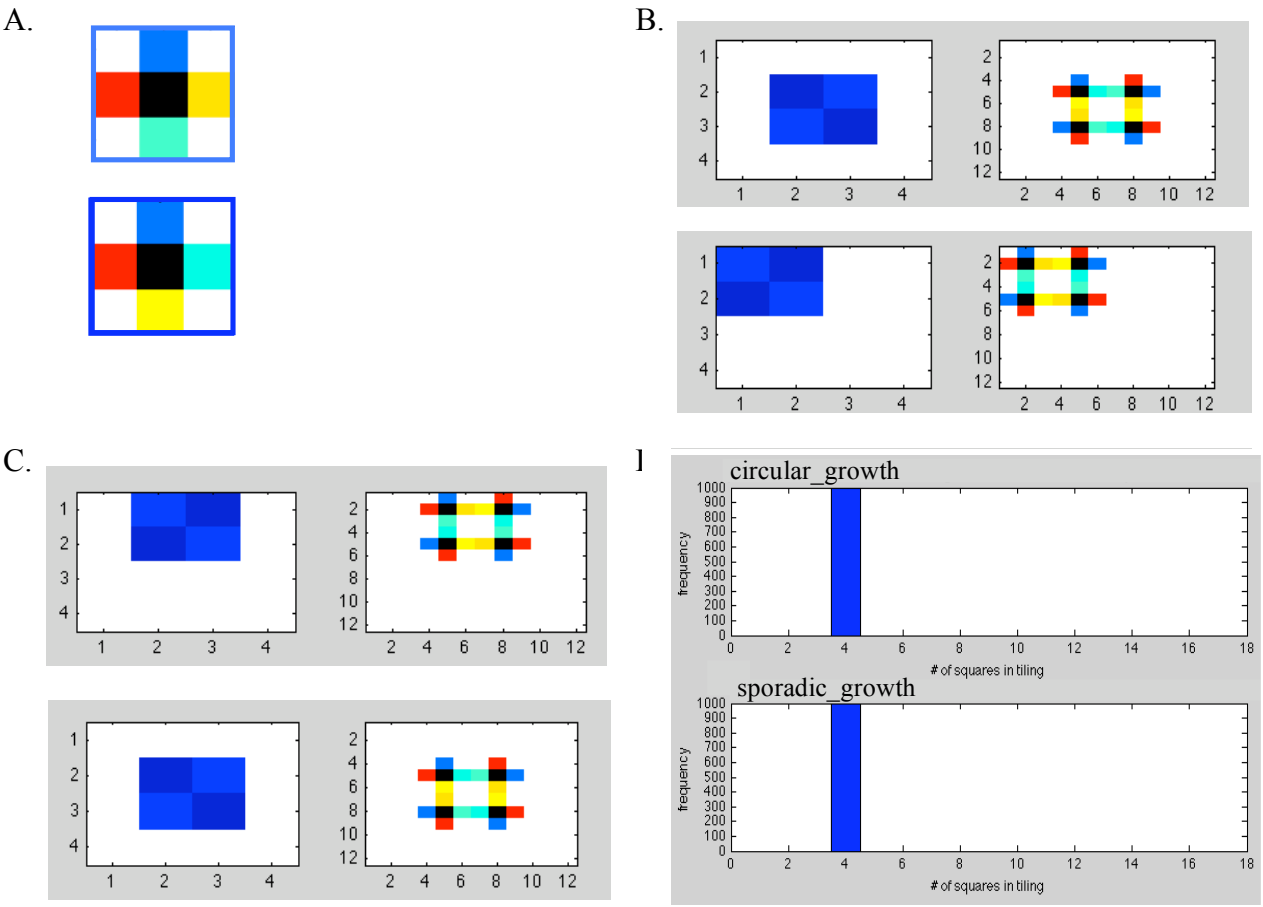
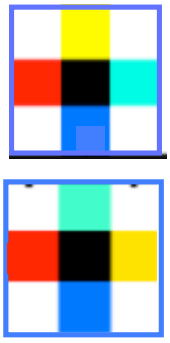


Figure 6. (A) Two tiles without reflections, using (B) circular_growth and (C) sporadic_growth form 2x2 squares. (D) Histograms for both algorithms indicate that every final tiling pattern out of 1,000 has exactly four tiles in it. Although the histograms do not suggest that these four tile

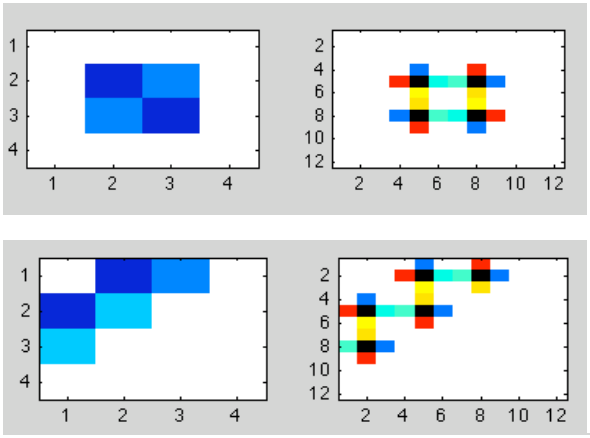
structures are squares, by cross-referencing the data with many tiling behaviors in B and C, it appears that these two tiles only form 2x2 squares.

Allowing reflections (Figure 7A) drastically alters the tiling behavior (Figure 7B-D). Although 2x2 squares still form (Figure 7B), a highly likely ladder structure can form (Figure 7C). The histograms (Figure 7D) illustrate that such 2x2 squares are not the only tiling pattern. Within the histograms, the tiling patterns consisting of four tiles could be four tiles in a 2x2 square or four tiles in a ladder. Regardless, the values within the histograms indicate that other structures form (*i.e.* larger ladders; Figure 7C). With the addition of two reflected tiles, the tiling behavior of the given complementary bounded tiling problem changes and the diversity of different structures increases.

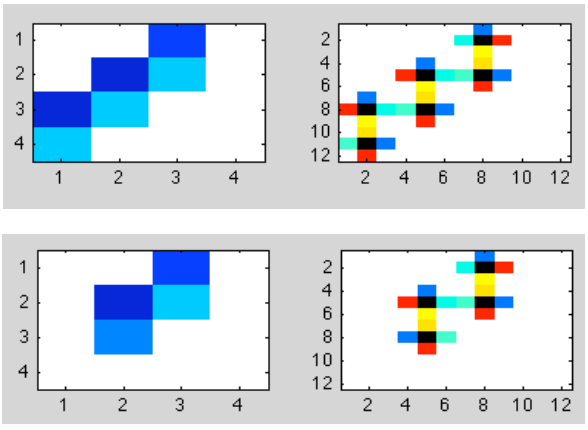
A.



B.



C.



D.

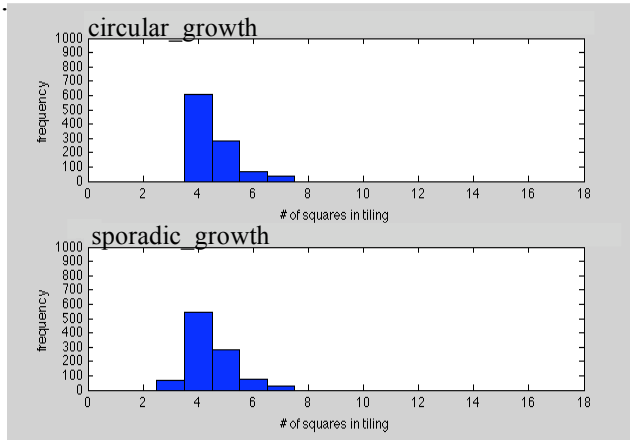


Figure 7. (A) Reflected versions of the tiles in Figure 6A, defining a tile set of four tiles. Illustration of two different tiling patterns for circular_growth (B) and sporadic_growth (C) with the given tile set. Note the occurrence of a ladder-like structure in C. The 1,000 run histograms in D reveal that not all final tiling patterns have four tiles. Hence, allowing reflections increases the final tiling pattern or structure diversity.

Understanding potential tiling behaviors can shed light on interesting tiling problems as shown above in Figures 6 and 7. By adding reflections, the tiling pattern changes, and evaluating such changes is a powerful tool when modeling the complementary bounded tiling problem. For example, part of my experimental design was to study the prevalence of corrugation within larger DNA lattices (recall corrugation from Chapter IV). If corrugation of connected tiles occurs, then DNA strands uncross, reflecting one of the tiles. If corrugation does not occur, then the DNA strands remain crossed and none of the tiles are reflected. By controlling the use of reflected tiles within both algorithms, I generated possible tiling structures for specific instances of the complementary bounded tiling problem assuming that corrugation did or did not occur. By simulating the above instance of the complementary bounded tiling problem (Figure 6 and 7), I was able to predict that the existence of corrugation (reflected tiles) will produce a ladder formation. Implementing the above instance of the complementary bounded tiling problem (Figure 6 and 7) using 4x4 DNA cross tiles allows me to test likelihood of corrugation by observing prevalence of ladder and 2x2 square structures. The study of corrugation probability is just one example of the evaluative power of both algorithms that would assist me in understanding biological implementations of 4x4 DNA cross tiles.

Using 4x4 DNA cross tiles I would need to use an atomic force microscope (AFM) to image different tile assemblies for a given instance of the complementary bounded tiling problem. I would need to scan the image for possible structures that fill the bounded region specified in the problem. Therefore, understanding the distribution of possible tiling outcomes and intermediates would help me evaluate an AFM image for potential biological solutions to the problem. Using the two algorithms above, I engineered a biological instance of the complementary bounded tiling problem in which the addition of different tiles drastically changes tiling patterns, and built and tested this instances with 4x4 DNA cross tiles.

Scramble Square Versus Complementary Bounded Tiling Problem

As mentioned in Chapter II, our original goal was to solve the Scramble Square problem using 4x4 DNA cross tiles, but various biological restrictions led Linda and I to define and solve our own NP-complete problem, the complementary bounded tiling problem. Recall in the Scramble Square problem that every given tile must be used exactly once. As seen above in both algorithms and in biological assembly, there is no control over tile incorporation in lattice structures. Tiles bind randomly based on concentration and Brownian movements. Scanning AFM images of lattices to ensure that every tile existed exactly once within a given solution adds another variable and increases the complexity of biological implementation. Instead, in the complementary bounded tiling problem, every tile can be used an unlimited number of tiles and not all tiles need to be used. The complementary bounded tiling problem resembles more closely the biological assembly of 4x4 DNA cross tiles and therefore we redefined our problem based upon biological implementation.

Chapter VI

Encoding and Assembly of 4x4 DNA Cross Tiles

The goal of my thesis research was to use DNA computation to solve the NP-complete complementary bounded tiling problem. My approach utilized structural DNA nanotechnology (4x4 DNA cross tiles) to encode instances of the complementary bounded tiling problem. The engineered DNA tiles could assemble to form solutions to such instances via Watson-Crick base pairing of overhanging sticky-ends. For this chapter, Linda and I designed the tile sequences and tested individual tile assembly. I tested multi-tile assembly, visited Duke University and ultimately designed a new instance to encode and solve. Initially, we choose a particular instance of the complementary bounded tiling problem to encode within the structural 4x4 DNA cross tiles. Two factors motivated the problem choice: 1) simplicity and 2) corrugation likelihood. In order to demonstrate that the methods worked, we began with a simple instance of the complementary bounded tiling problem. Within this same instance, we also decided to test the occurrence of corrugation to understand larger lattice formation. The tile set we engineered with 4x4 DNA cross tiles is shown in Figure 1.

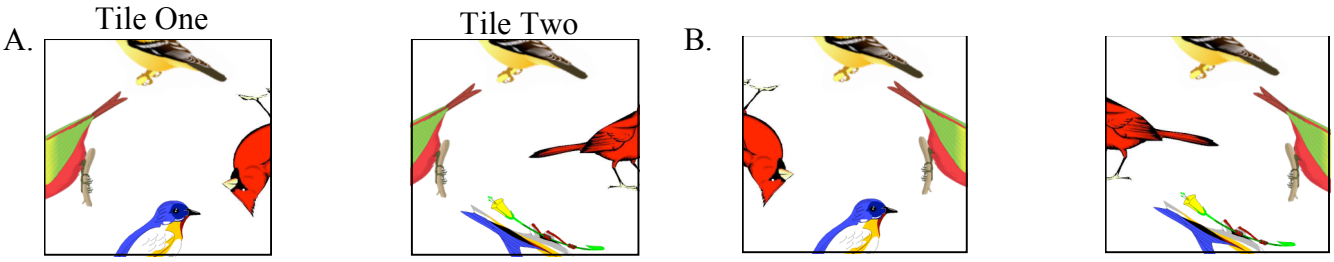


Figure 1. The tile set for the particular instance of the complementary bounded tiling problem we encoded into 4x4 DNA cross tiles. The two tiles (A) and their reflections (B) we chose.

Recall from Figures 6 and 7 in Chapter V that the instance given without reflections (without corrugation) formed a 2x2 square tiling and with reflections (with corrugation) formed a ladder or a 2x2 square. We engineered a slightly different instance shown in Figure 1. In this instance, a ladder forms (Figure 2) without reflections (no corrugation), whereas if every other tile is flipped as expected with corrugation, only a 2x2 square forms (Figure 3).

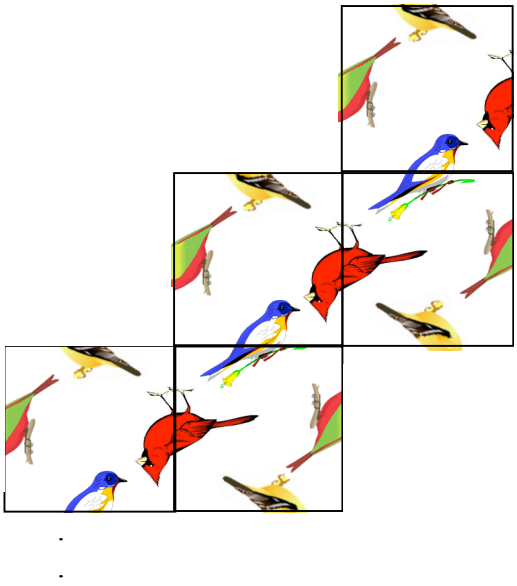


Figure 2. Expected structure when corrugation does not occur using the tiling set given in Figure 1. The ladder in A can grow infinitely in both directions.

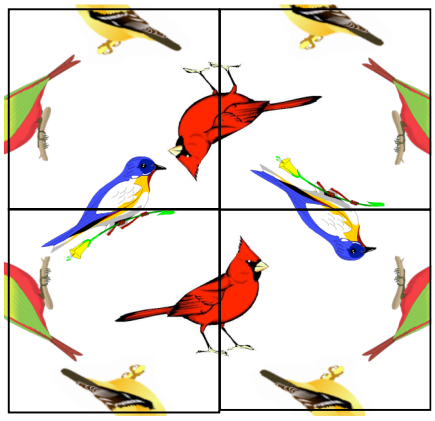


Figure 3. Expected structure, using the tiling set in Figure 1 when corrugation does occur and every other tile flips.

We hypothesized that corrugation occurred relatively frequently compared to tile interactions without corrugation (see Chapter IV). We designed the tiles in Figure 1 so a 2x2

square tiling structure would only be possible if corrugation occurred. Recall that in the example from Chapter V, a ladder formed with corrugation. Within a ladder structure, some tiles may be connected with corrugation and some may be connected without corrugation. Therefore, the new tile design in Figure 1 of this chapter ensures that we could accurately observe the frequency of corrugation.

Step One: DNA Sequence Design

Once we selected a certain tile set for experimentation, we began to construct the 4x4 DNA tiles. In order to construct each 4x4 tile, we needed a set of 9 DNA strands with complementary sequences at all of the necessary points (*i.e.* part of the sequence for shell 1 must be complementary to part of the sequence on arm 1; Figure 4).

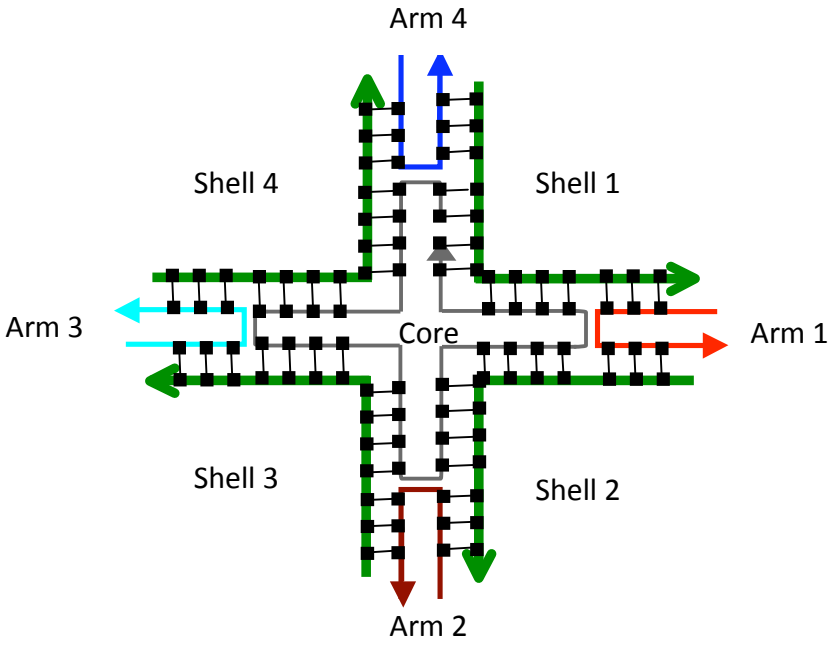


Figure 4. Diagram of DNA cross tile where thin grey lines indicate where DNA sequences are complementary. For example, the core strand must be complementary to the inner sections of each shell strand.

In addition to controlling structure assembly through sequence design, we controlled step-wise assembly by studying melting temperatures of the 9 different DNA strands. The melting temperature (T_m) of a double-stranded DNA molecule is the temperature at which half of the DNA strands are double-stranded and half are single-stranded ("Melting Temperature of

DNA," Biology Online Dictionary, 2005). The melting temperature for a DNA molecule is dependent on the sequence of nucleotides and its length. The melting temperature can also be defined as the temperature at which half of the single-stranded, complementary pieces of DNA anneal to form double-stranded DNA. Studying melting temperature is the same as studying annealing temperature. We were able to control step-wise assembly of the physical DNA by changing the temperature of the DNA mixture. Designing distinct melting temperatures for each strand interaction was important, because unique melting temperatures allowed us to assemble one strand at a time.

In addition to studying melting temperatures, we also studied the Gibbs free energy (ΔG). Gibbs free energy is used to quantify the spontaneity of a reaction. If a reaction is spontaneous, it will proceed without an outside energy source. In the study of DNA annealing, Gibbs free energy quantifies the spontaneous reaction of two DNA strands annealing via Watson-Crick base pairing. A negative Gibbs free energy value suggests that a DNA strand and its complement will anneal spontaneously. Larger negative Gibbs free energy values indicate a higher likelihood of annealing. Therefore, minimizing the Gibbs free energy of all 9 DNA strands increases the likelihood that the 4x4 DNA cross tile will assemble.

Before we could begin to analyze different DNA sequences and their thermodynamic properties, we needed to design the interchangeable sticky-ends, the nucleotide overhangs that function as the complementary images in the complementary bounded tiling problem. Thermodynamic properties of DNA sequences (T_m and ΔG) depend on DNA sequence. The DNA sequence of the four arm strands will vary based on sticky-end sequence. We had to choose sticky-end sequences before designing and analyzing all 9 DNA sequences.

For the problem instance we chose (Figure 1), we needed sticky-ends to represent six different half-images. Since each half-image is analogous to two non-compatible sticky-ends, we needed twelve, 5-nucleotide sticky-end sequences. In choosing the sequences for the 5-nucleotide sticky-ends, we utilized results from research that analyzed effectiveness and stability of certain sticky-ends in the 4x4 DNA cross tile (Dwyer *et al.*, 2006; Figure 5).

Seq.	Comp.	Seq.	Comp.	Seq.	Comp.
CGTGC	GCACG	CCTCG	CGAGG	TATGT	ACATA
CAAGC	GCTTG	ACGAC	GTCGT	TGTAT	ATACA
ACGTC	GACGT	CAGAC	GTCTG	TTAGA	TCTAA
ACAGC	GCTGT	ACTGC	GCAGT	TTACT	AGTAA
TGCAG	CTGCA	TGCTG	CAGCA	TAAGA	TCTTA
CTGTG	CACAG	TGCAC	GTGCA	AATAG	CTATT
AGCTC	GAGCT	AGAGC	GCTCT	AATTC	GAATT
CATGG	CCATG	CTTGG	CCAAG	ATACT	AGTAT
CAATC	GATTG	CATTC	GAATG	TAACT	AGTTA
AATGC	GCATT	ATTGC	GCAAT	TTAGT	ACTAA
AACGT	ACGTT	CTAAC	GTTAG	TACTT	AAGTA
CTTAC	GTAAG	TTACG	CGTAA	TAAGT	ACTTA
TAACG	CGTTA	CATTG	CAATG	TAGAT	ATCTA
ATGAC	GTCAT	ATGCT	AGCAT	TAGAC	GTCTA
TCATG	CATGA	TTGCT	AGCAA	TTCAT	ATGAA
TTGAG	CTCAA	AAGCT	AGCTT	ATTCT	AGAAT
TGCTT	AAGCA	ACTGT	ACAGT	TCAAT	ATTGA
TCACA	TGTGA	AGTAC	GTACT	TTAAC	GTTAA
TACGT	ACGTA	TGTAG	CTACA	AATCT	AGATT
TACTG	CAGTA	AAGTG	CACTT	TGATT	AATCA
TCAAC	GTTGA	TCTGA	TCAGA	TCATT	AATGA
TCTAG	CTAGA	TAGCT	AGCTA	TATGA	TCATA
AATGT	ACATT	ATGTT	AACAT	TTAAG	CTTAA
ATTGT	ACAAT	TTCAA	TTGAA	TTACA	TGTAA
GTTAT	ATAAC	AATAC	GTATT	TTGTA	TACAA
TAATG	CATTA	CAATA	TATTG	TTGAT	ATCAA
TTATG	CATAA	AATTG	CAATT		

Figure 5. The best 5-nucleotide sticky-ends for 4x4 DNA cross tile assembly (Dwyer *et al.*, 2006).

From the list of thermodynamically stable 5-nucleotide sticky-ends shown in Figure 5, we chose sticky-ends that could be cut with restriction enzymes. If we were able to selectively disconnect (cut) 4x4 DNA cross tiles within a lattice, then we could test and study lattice formation. For example, in the above instance with corrugation (Figure 2B), if a restriction enzyme disconnected the red bird's beak and tail, the 2x2 structure would remain. Observing that the 2x2 structure existed after adding a restriction enzyme to cut the red bird in half would indicate that at least one tile was connected to two other tiles. I also chose sticky-ends with restriction enzymes sites to determine if annealed tiles could be separated by restriction enzymes. To my knowledge there is no report on the use of restriction enzymes in 4x4 DNA cross tiles. Using sticky-end sequences with restriction enzymes sites led to one necessary change in the DNA sequences of all four shell and arm strands. Most restriction enzymes cut sites are 6-nucleotides in length, but the sticky-ends within the 4x4 DNA cross tile are only 5-nucleotides long.

Therefore, the last nucleotide in the arm strand was altered to accommodate the extra nucleotide needed for restriction enzyme site recognition. Most restriction enzyme sites begin with a G and therefore we had to ensure that the last nucleotide on every arm strand (before the 5-nucletotide sticky-end that was a 6-nucleotide long restriction enzyme site) was a G. Due to this change, we also had to ensure that the complementary base on the shell strand was a C. Hence, I designed all necessary shell strands to begin with a C and all arm strands to ends with a G before each set of 6-nucleotide long restriction enzyme sites.

After choosing the sticky-ends, we began DNA sequence design and analysis by studying sequences that other researchers implemented to produce 4x4 DNA cross tiles (Li, 2008; Yan *et al*., 2003). These DNA sequences were generated using a thermodynamic modeling program known as SEQUIN. SEQUIN was written by Ned Seeman, the inventor of structural DNA nanotechnology, as a sequence design program specifically for 4x4 DNA cross tile assembly (Seeman, 1990). We used a similar program called TileDesigner to design and analyze DNA sequences. TileDesigner is a more recent program for the study of 4x4 cross tile DNA sequences (Limura *et al*. 2007). We used TileDesigner to generate DNA sequences and study previously used DNA sequences for the formation of 4x4 DNA cross tiles. TileDesigner computes melting temperature (T_m) and Gibbs free energy (ΔG) of each DNA strand in the context of the 4x4 DNA cross tile structure. We compared data for two different sets of DNA sequences: a set of 9 DNA sequences generated by TileDesigner and DNA sequences previously utilized in other 4x4 DNA cross tiles (Table 2). Both sets of arm strands utilize our sticky-end sequences for the two tile types in Figure 1A.

A.

Strand	Length	5'→3'	T_M (°C)	ΔG
Core	96	CAATTCTGTAGAGACTTTTTGCCATTGTG GCCCTATGTCTTTTGGCTGGGGCGCAGCA CGTTCTTTTGCCAACTGATCCCATCCGTG TTTTAGATG	46	-7.7
Shell 1	40	CGACACTCTGCATACGGTCCAATTGCATG TGACGGATGGG	42	-4.6
Shell 2	40	CACCTTAGTAATCAGTTGGCGAACGTGCT GCGTGTAGTCG	40	-2.2
Shell 3	40	CATCTGATAGCGCCCCAGCCGACATAGGG CCCGTAGGGGG	46	-2.9
Shell 4	40	CAATAGGGCGCACAATGGCAGTCTCTACA GCAAATTTTGG	38	-1.2
Tile 1, Arm 1	30	GTGCACCCATCCGTCTACTAAGGTGAATTC	36	-2.5
Tile 1, Arm 2	30	AAGCTCGACTACACGCTATCAGATGTAGCT	42	-2.3
Tile 1, Arm 3	30	AATGCCCCCCTACGGCGCCCTATTGCATTA	34	-1.6
Tile 1, Arm 4	30	GTACTCCAAAATTTGCAGAGTGTCGCAATA	37	-1.6
Tile 2, Arm 1	30	TGCACCCCATCCGTCTACTAAGGTGGAATT	36	-1.9
Tile 2, Arm 2	30	AGCTTCGACTACACGCTATCAGATGAGCTA	42	-0.7
Tile 2, Arm 3	30	AATGCCCCCCTACGGCGCCCTATTGCATTA	34	-1.6
Tile 2, Arm 4	30	GTACTCCAAAATTTGCAGAGTGTCGCAATA	37	-1.6

B.

Strand	Length	5'→3'	T_M (°C)	ΔG
Core	100	AGGCACCATCGTAGGTTTTCGTTCCGATCACC AACGGAGTTTTTTCTGCCGTACACCAGTGAAG TTTTTCGATCCTAGCACCTCTGGAGTTTTTCT TGCC	45	-5.5
Shell 1	42	CAGCGCAACCTGCCTGGCAAGACTCCAGAGGA CTACTCATGG	44	-3.1
Shell 2	42	CACTGAGCCCTGCTAGGATCGACTTCACTGGA CCGTTCTAGG	43	-1.8
Shell 3	42	CTTGGCTTCCTGTACGGCAGAGCTCCGTTGGA CGAACACTGG	44	-3.6
Shell 4	42	CCATAGCGCCTGATCGGAACGCCTACGATGGA CACGCCGAGG	46	-3.2
Tile 1, Arm 1	31	GTGCACCATGAGTAGTGGGCTCAGTGAATTC	39	-3.5
Tile 1, Arm 2	31	AAGCTCCTAGAACGGTGGAAGCCAAGTAGCT	40	-1.8
Tile 1, Arm 3	31	AATGCCCAGTGTTCGTGGCGCTATGGCATTA	41	-3.3
Tile 1, Arm 4	31	GTACTCCTCGGCGTGTGGTTGCGCTGCAATA	40	-3.1
Tile 2, Arm 1	31	TGCACCCATGAGTAGTGGGCTCAGTGGAATT	39	-3.8
Tile 2, Arm 2	31	AGCTTCCTAGAACGGTGGAAGCCAAGAGCTA	40	-5.1
Tile 2, Arm 3	31	AATGCCCAGTGTTCGTGGCGCTATGGCATTA	41	-3.3
Tile 2, Arm 4	31	GTACTCCTCGGCGTGTGGTTGCGCTGCAATA	40	-3.1

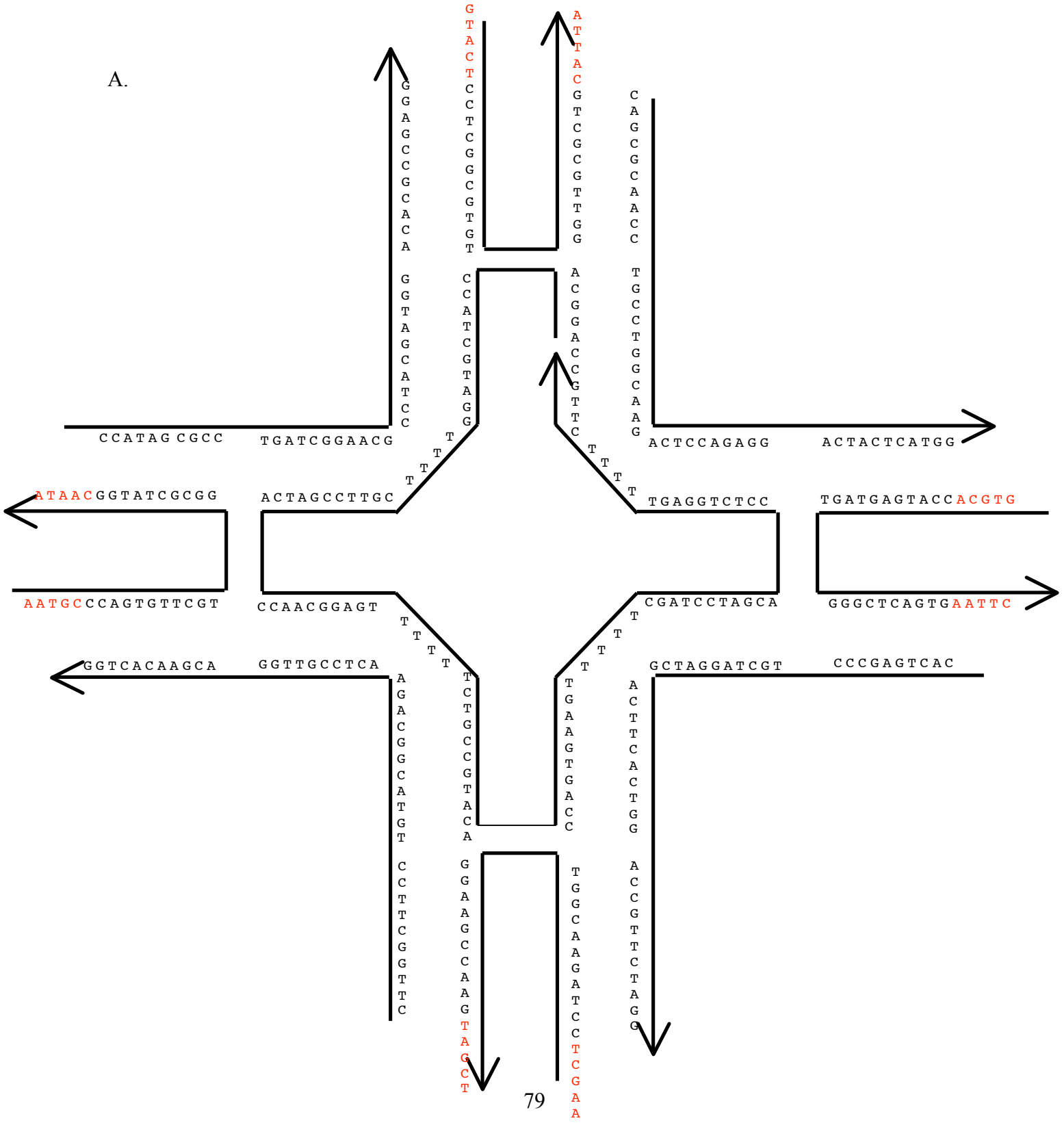
Table 2. Tables representing thermodynamic sequence information calculated by TileDesigner. (A) DNA sequences generated by TileDesigner and (B) DNA sequences previously used in the construction of 4x4 DNA cross tiles. Our sticky-ends sequences are highlighted in red.

Based on data generated from TileDesigner, we chose to use the DNA sequences in Table 2B from previously studied 4x4 DNA cross tiles. We chose this set of DNA sequences for three main reasons:

1) The melting temperatures for the DNA sequences in Table 2B present a better step-wise assembly progression. All four shell strands and core strands would assemble first between 46°C and 43°C before the addition of arm strands (41-39°C) for the DNA sequences in Table 2B. The DNA sequences in Table 2A would assemble in a much more random manner. For instance, some arm strands would bind to the shell strands before the shell strand would bind to the core strand. The DNA sequences in Table 2B are more likely to produce a 4x4 DNA cross tile shape because of the step-wise addition of smaller strands.
2) The average ΔG for all DNA strands in Table 2B is more negative, which means annealing is more favorable. Although the ΔG of the core strand in Figure 6A is highly negative (-7.7), the ΔG of other DNA strands is barely negative (*i.e.* -0.7).
3) The twelve sticky-end sequences that we chose had limited sequence similarity and limited sequence complementarity within the other DNA sequences in Table 2B. We checked for 5-nucleotide, 4-nucelotide and 3-nucleotide recurrence of the sticky-end sequences and their complements within all other DNA sequences and found that the sequences in Table 2B had less sequence similarity when compared to sequences in Table

2A. Minimizing sequence similarity and complementarity decreases the opportunity for unintended structures to form and therefore increases the specificity of DNA sequences to assemble into a prescribed shape.

A.



B.

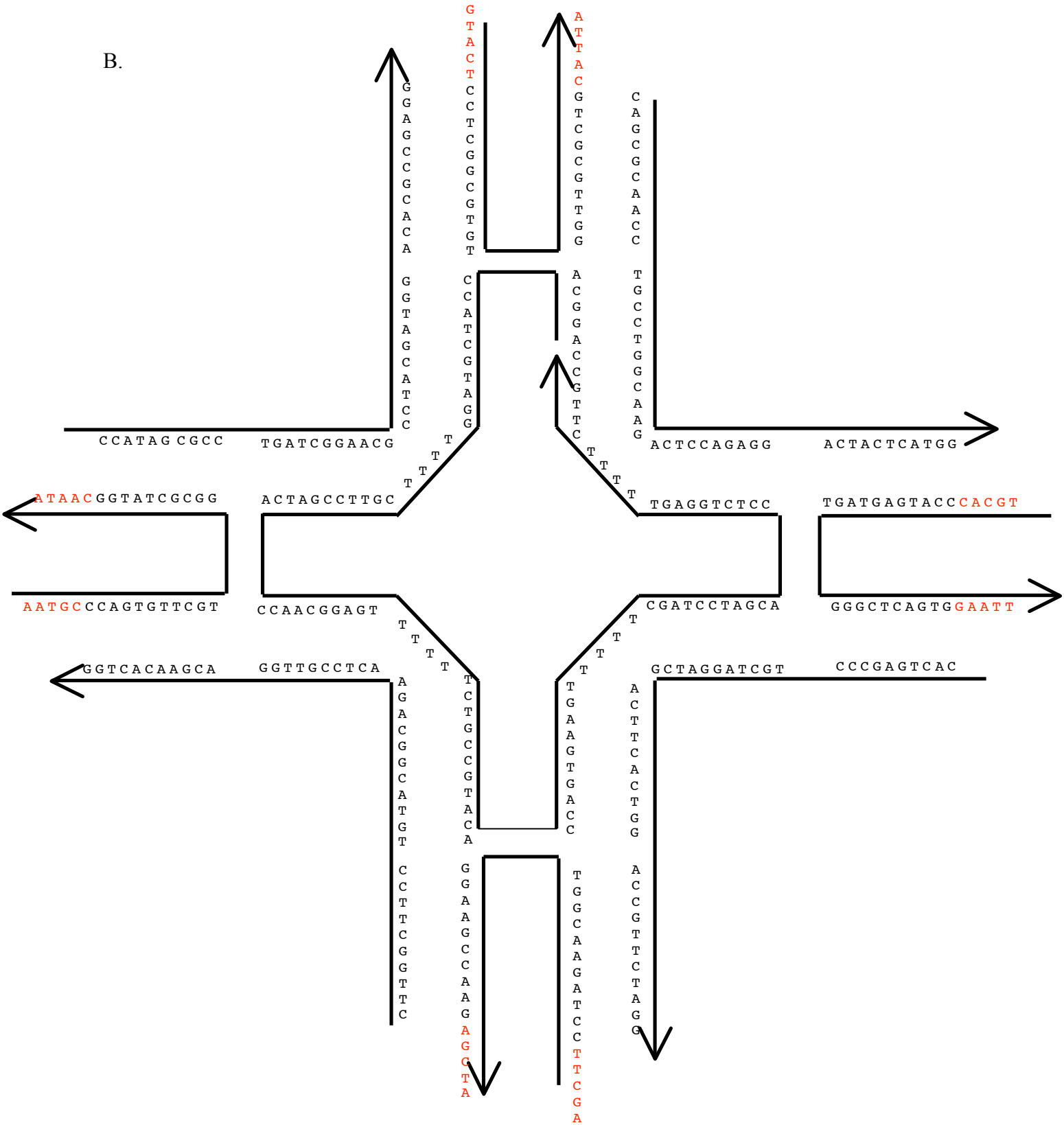

GTACTCCTCGGCGTGT
ATTACGTCGCGTTGG
GGAGCCGCACA
GGTAGCATCCC
CAGCGCAACACC
TGCCTGGCAAG
CCATAG CGCC
TGATCGGAACG
CCATCGTAGG
ACGGACCGTTC
ACTCCAGAGG
ACTACTCATGG
ATAACGGTATCGCGG
ACTAGCCTTGC
TGAGGTCTCC
TGATGAGTACCCACGT
AATGCCCAGTGTTCGT
CCAACGGAGT
CGATCCTAGCA
GGGCTCAGTGGAATT
GGTCACAAGCA
GGTTGCCTCA
TCTGCCGTACA
TGAAGTGACC
GCTAGGATCGT
CCCGAGTCAC
AGACGGCATGT
CCTTCGGTTC
ACTTCACTGG
ACCGTTCTAG
GGAAGCCAAGAGTA
TGGCAAGATCCTTCGA

Figure 6. Schematic of 4x4 DNA cross tile with DNA sequences for both tile one (A) and tile two (B) as given in Figure 1.

We ordered the DNA sequences (Table 2B) from Eurofins MWG Operon. All DNA sequences were ordered wet at 75 μM, meaning that upon delivery the DNA sequences had already been suspended in liquid at an equal concentration of 75 μM. The core and four shell strands were ordered with an additional purification method, known as HPSF (High Purity Salt Free), to favor sequences of full length. The arm strands were not ordered with additional purification steps, because shorter DNA sequences are less likely to be synthesized incorrectly. As mentioned in Chapter III, error rate and uncertainty are limitations when computing with DNA. I diluted the DNA sequences to 7.5 μM and stored at -20°C.

Step Two: Assembly of Individual 4x4 DNA Cross Tiles

To assemble one 4x4 DNA cross tile, we mixed all 9 strands (1 core, 4 shell and 4 arm) together in equal stoichiometric ratios into a final concentration of 1 μM in a 30 μL solution (Li, 2008). The 30 μL solution also contained 20 mM Tris pH 7.6, 2 mM EDTA and 12.5 mM $MgCl_2$ (Li, 2008). Tris buffers pH of the solution and EDTA buffers Mg^{2+} ions. The Mg^{2+} ions neutralize the negatively charged phosphates in DNA, facilitating annealing to form the 4x4 DNA cross tile shape (Kuzuya *et al.*, 2010).

We used a thermal cycler to control the temperature of the 30 μL solution containing all 9 DNA strands. The temperature gradient, programmed into the thermal cycler, controlled the step-wise assembly of the 4x4 DNA cross tile by exposing all 9 strands to their respective melting temperatures at different time points. Initially we programmed the thermal cycler, in a program called temperature cycle 0 (TC0), to linearly decrease from 90-20°C with a step-down

of 0.1 degrees every 67 seconds between 90-47°C and between 38-20°C. The time spent between 47-38°C was lengthened to be 3 minutes and 25 seconds. The run time of the TC0 was approximately 16 hours. Temperature cycle 0 stayed at 90°C for 10 minutes and then began the step down (Figure 7). The initial 90°C exposure denatured any pre-existing secondary structure within the DNA strands.

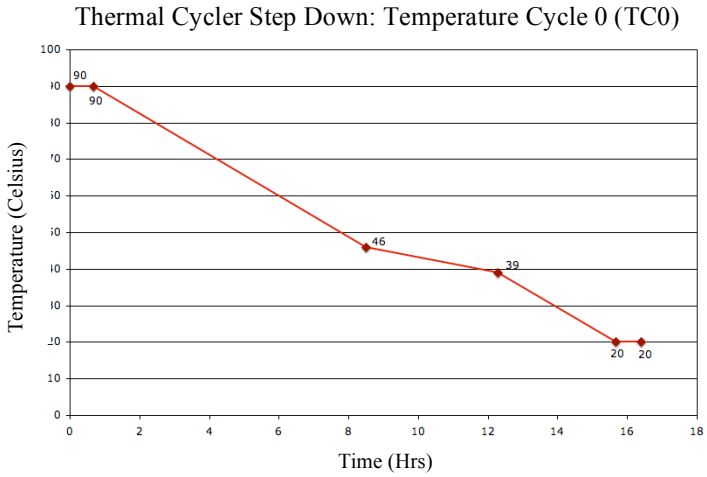


Figure 7. Graph depicting the general trend of temperature zones for the step-down in temperature cycle 0 (TC0).

As shown in Table 2B all of the annealing temperatures (46-39°C) fall within the temperature step-down of the temperature cycle 0 (TC0) step-down. As the thermal cycler reaches the melting temperature of a particular DNA strand, that strand should anneal its complementary strand. Annealing of specific DNA strands will continue in a step-wise fashion until annealing temperatures have been reached for each of the 9 strands. The cooling time between 46 and 39°C is slowed, meaning that the amount of time spent at each step-down is longer when compared to the amount of time spent at each step-down between 90 and 47°C and 38 and 20°C. Lengthening time spent at each temperature step-down within this temperature zone (46 to 39°C) facilitates more DNA strands annealing and forming structures. In order to ensure that the tiles did not degrade, we stored tiles after assembly in a 4°C refrigerator for no more than 7 days.

To determine if 4x4 DNA cross tiles properly formed during TC0, we ran the products on a 10% non-denaturing TBE polyacrylamide gel (Figure 8). The pores in polyacrylamide gels are smaller compared to agarose gels. A piece of DNA migrates on a gel based on its mass and shape. When pore sizes are smaller, shape has a greater affect on migration because larger shapes cannot move through the small pores (Stellwagen, 2009). Therefore polyacrylamide gels are best for separating DNA based on shape and size. Lane 1 in Figure 8 represents a 50 bp molecular weight marker (Zymo Research ready-to-load 50 µg /600 µL DNA marker). I added 4 µL loading dye (1X TBE, 50% glycerol, 0.2% bromophenol blue and 0.2% xylene cyanol) to the 10 µL of the original 30 µL DNA mixture before loading the DNA. I ran the gel at 100V/13mA for 1 hour and post-stained (81.5µL solution of 1X TBE and 0.184 µg/ µL ethidium bromide concentration) for 17 minutes.

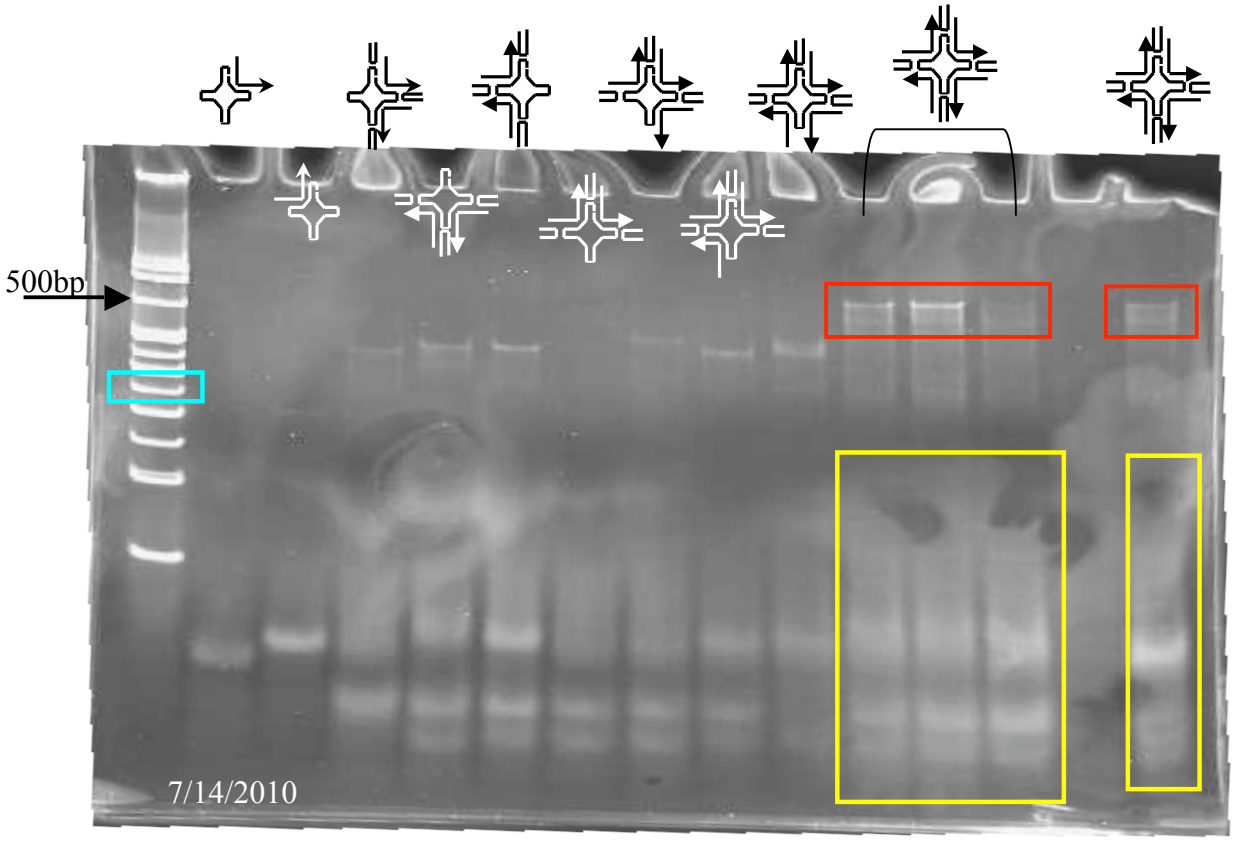
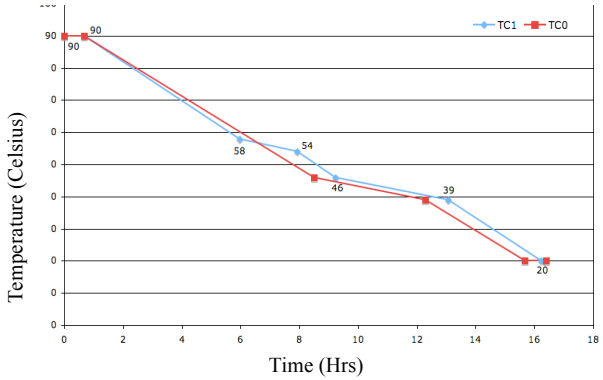


Figure 8. A 10% non-denaturing polyacrylamide gel run to analyze successful step-wise assembly of individual 4x4 DNA cross tiles. The step-wise assembly contents in each well is indicated by alternating schematics above or below a given well. Highlighted in yellow are unincorporated DNA strands, whereas complete 4x4 DNA structures are highlighted in red. The blue band in the MW marker highlights expected migration based solely on base pairings within the structure (~230bp). Lane 14 (second to last) includes all 9 DNA strands but was not exposed to TC0.

All products run on the gel were run through the TC0 thermal profile before loading, except for lane 14. The successive addition of different strands shows a successful step-wise assembly of the 4x4 DNA cross tile because the bands shift upwards (increase in size) with the addition of new strands. The bands in lanes 11-13 and 15 appear at the 500 bp mark. The actual base-pair size of the 4x4 DNA cross tile is 230 bp (boxed in blue on the MW marked in Figure 8), but because the structure is non-linear, it does not migrate as far on the gel. Figure 8 indicates that the 9 DNA strands form a complicated shape, providing evidence of successful 4x4 DNA cross tile assembly. Because smaller bands are also visible (outlined in yellow), not all of the DNA strands are annealing to form a 4x4 DNA cross tile; some strands remain unincorporated. The yield of complete 4x4 DNA cross tile structures is not 100%. In order to increase the likelihood of structure formation, we experimented with different temperature profiles for individual tile assembly (Figure 9). We lengthened certain temperature zones to increase annealing time of certain DNA strands in order to favor strand incorporation.

A.



B.

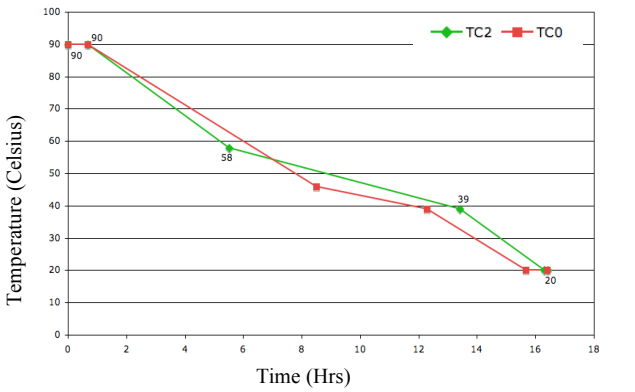


Figure 9. Graphs depicting the general trend of temperature zones for the step-down in (A) temperature cycle 1 (TC1) and (B) temperature cycle 2 (TC2).

The changes made to the thermal cycler temperature profiles shown in Figure 9 for both TC1 and TC2 were based on additional melting temperature analysis using a program called two-state melting hybridization. In two-state melting hybridization, the two DNA strands are not required to be complementary (Markham and Zuker, 2005). In other melting temperature algorithms the melting temperature is computed for a given DNA strand and its exact complement. Within the structure of the 4x4 DNA cross tile a strand is complementary to many different strands. For example, instead of analyzing the melting temperature of a shell strand with its exact complement, two-state melting hybridization can calculate the melting temperature of a shell strand with two arm strands. Melting temperature analysis of the chosen DNA sequences (Table 2B) using two-state melting hybridization givens melting temperatures as shown in Table 3.

Strand	TileDesigner T_M (°C)	Two-state T_M (°C)
Core	45	43.3
Shell 1	44	47.6
Shell 2	43	55.3
Shell 3	44	58.3
Shell 4	46	56.5

Table 3. Comparison of melting temperatures generated by TileDesigner and Two-state Melting Hybridization for the 5 main strands (core and 4 shell strands).

Table 3 indicates (as highlighted in red) that some shell strands may be annealing at higher temperatures than predicted with TileDesigner. With these data we re-designed the thermal cycler programs (Figure 9A and 9B), attempting to increase strand incorporation efficiency.

In order to further separate complete 4x4 DNA cross tile structures from unincorporated strands, we used a PCR column clean-up kit. PCR column clean-up is a technique used to separate small pieces of DNA using various buffers and a spin-column. Using PCR column clean-up would collect larger pieces of DNA, individual tiles, within the spin column and discard the unincorporated smaller strands. We tested the effects of using TC1 and TC2 when compared to TC0 and the effects of PCR column clean-up on samples in another 10% TBE non-denaturing polyacrylamide gel (Figure 10).

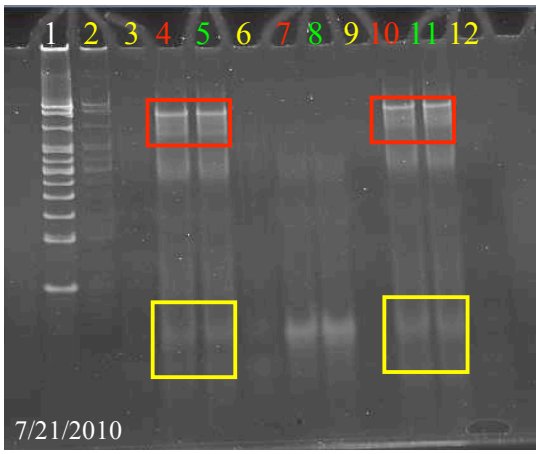


Figure 10. A 10% TBE non-denaturing polyacrylamide gel analyzing effects of different step-wise temperature programs (TC1 and TC2) as well as a procedural PCR column clean-up method on unincorporated strand concentration. Yellow lane numbers denote samples that were cleaned up prior to loading the gel. Red and green lane numbers were exposed to TC1 and TC2

respectively before loading. Lanes 4 and 5 represent a mixture of all 9 strands of tile one, whereas lanes 10 and 11 are all 9 strands for tile two. Lanes 7 and 8 represent a mixture of the core and four shell strands.

The cleaned-up MW marker (lane 2) served as a control verifying that the clean-up procedure was successful at eliminating smaller DNA pieces. In lanes 3, 6, 9 and 12 no DNA was left, indicating that the clean-up procedure filtered the entire DNA sample and therefore was not useful. TC1 and TC2 appear to produce similar amounts of tiles and unincorporated strands. Although a sample from TC0 was not loaded on this gel, comparison of relative band intensity between unincorporated strands (boxed in yellow) and complete 4x4 DNA cross tiles (boxed in red) indicates a decrease in unincorporated strands. TC1 and TC2 successfully increased strand incorporation as is evident by the decrease in unincorporated DNA strands. I decided to proceed using the temperature cycle 1 (TC1) program for individual tile assembly, acknowledging that some unincorporated strands still existed, but recognizing that PCR purification was not helpful.

Step Three: Multi-Tile Assembly

Once we had evidence suggesting that individual 4x4 DNA cross tiles assembled properly (Figures 8 and 10), we began to mix together the two tiles (see Figure 1A) to solve one instance of complementary bounded tiling problem. In a microfuge tube, we mixed equal amounts of tile one and tile two, keeping all solutions on ice to maintain 0°C and prevent tile degradation. We conducted four experiments testing different environments for multi-tile assembly:

1) Thermal cycler consistently at 25°C for 5 hours.
2) Thermal cycler on a linear decrease from 25 to 4°C for 5 hours.

3) Water bath at 42°C over 7 hours to room temperature.

4) Water bath experiment increasing $MgCl_2$ to 13.75 mM from 12.5 mM. Increasing the Mg^{2+} ion levels enhances the ability of the negatively charged DNA tiles to interact.

After these procedures, we maintained all mixtures at 0-4°C until we added loading dye and ran them on a 4-20% TBE non-denaturing polyacrylamide gel for 1 hour and 40 minutes at 4°C. We ran the gel in a 4°C refrigerator because the melting temperatures of the sticky-end tile interactions are below room temperature as indicated by two-state melting hybridization. We ran at 100V/10mA and post-stained for 15 minutes. Figure 12 indicates structures too large to enter the gel (highlighted in yellow), such as multi-tile lattice structures.

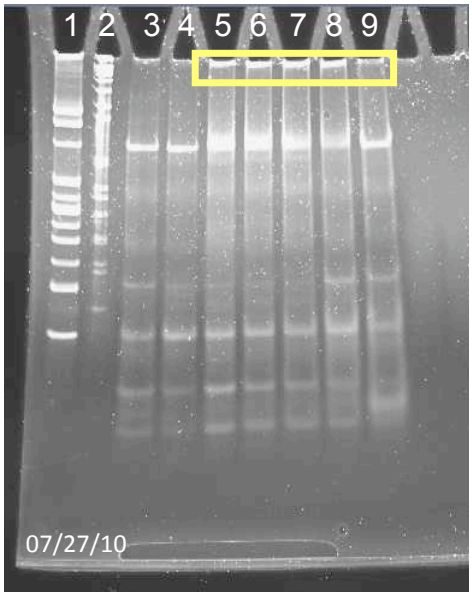


Figure 12. A 4-20% non-denaturing TBE polyacrylamide gel indicating possible multi-tile assemblies (lanes 5-9) that are too large to enter the gel (highlighted in yellow). Lane 1 and 2 are MW markers, 50bp and 1kb respectively. Lanes 3 and 4 are tile one and tile two respectively, both assembled using TC1. Lanes 5-8 represent the different methods, in the same order as above, for mixing tile one and tile two together. Lane 9 is a mixture of tile one with only the arms of tile two.

In Figure 12 all tile one and tile two mixtures (lanes 5-8) appear to indicate a tiling structure that is too large to enter the gel regardless of the mixing method used. The different methods for multi-tile assembly as described above (lane 5: method 1, lane 6: method 2, lane 7: method 3,

lane 8: method 4) do not appear to effect multi-tile formation as the same amount of DNA did not enter the gel.

With the evidence above, Dr. Chris Dwyer, associate professor in the Department of Electrical & Computer Engineering and the Department of Computer Science at Duke University allowed me to come and work with his PhD. student, Viresh Thusu, to image possible multi-tile DNA lattice formations using their AFM (atomic force microscope). Before I traveled to Duke University, on September 30, 2010, I re-ran the multi-tile lattice formations evidenced in Figure 12. Unfortunately, I did not see the same results (Figure 13).

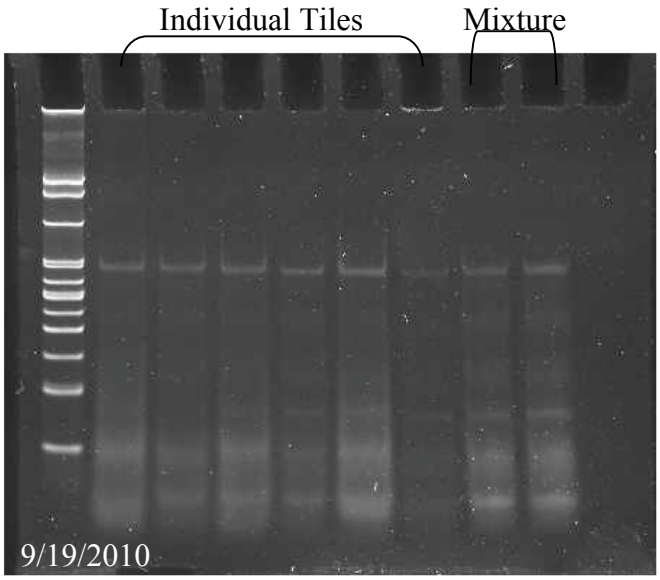


Figure 13. A 5% non-denaturing polyacrylamide gel. Lane 1 is the 50 bp MW marker. Lanes 2-6 are individual 4x4 DNA cross tiles and lanes 7 and 8 are mixtures of tile one and tile two, annealing at 25°C consistently for 5 hours and then 4°C overnight. No evidence of structures too large to entire the gel exists.

Further testing of the above samples indicated DNA degradation. Between the time the original gel (Figure 12) was run on 7/27/2010 and the new verification gels were run (Figure 13) on 9/19/2010, the stock DNA samples had significantly degraded (Figure 14). DNA, especially

short single-stranded molecules of DNA, can degrade over time a short amount of time if stored improperly or at a low concentration (as later noted by Duke graduate student Viresh Thusu). The DNA sequences used to assemble the 4x4 DNA cross tile are single-stranded, many of them are short, and most are stored at low concentrations. Degraded DNA decreases the probability of accurate 4x4 DNA cross tile formation due to the lack of complete and accurate DNA sequences.

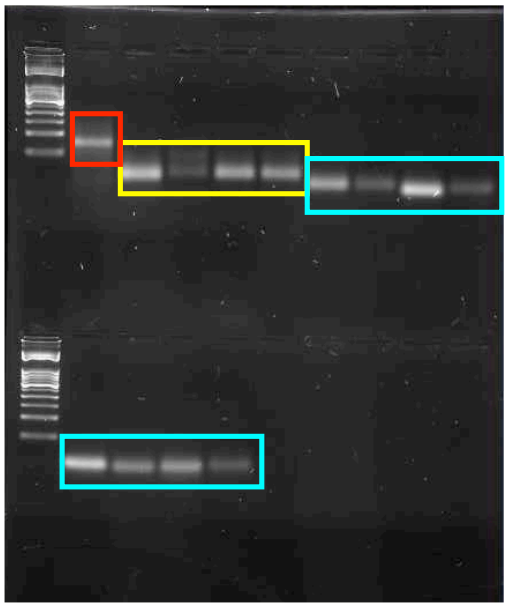


Figure 14. Double-decker 2.5% agarose gel run to determine degradation of stock DNA sequences. Equal amounts of each strand (core in red, shells in blue and arms for both tiles in yellow) were run.

In Figure 14 every band should be the same intensity because stock DNA samples were originally the same concentration and equal amounts were loaded. Also, band migration should be the same for the shells strands (highlighted in yellow) and the same for the arm strands (highlighted in blue). Not only does intensity drastically differ, but band migration also slightly differs, indicating degraded DNA. Due to the degradation of DNA, I traveled to Duke University with samples for which I did have gel evidence of multi-tile lattice formation. The images did not reveal any multi-tile lattice structures, possibly due to strand degradation. However, I did receive four tips from researchers at Duke, which I have begun to integrate.

1) Model an instance of the complementary bounded tiling problem that can form large sheets of 4x4 DNA cross tiles. The researchers at Duke indicated that smaller multi-tile lattice structures are harder to see with atomic force microscopy.
2) Store DNA aliquots at -80°C to decrease chances of degradation.
3) To facilitate multi-tile lattice formation, expose a sample of mixed tiles to room temperature for 5 hours and subsequently store at 4°C overnight.
4) Use the Nanodrop to detect changes between solutions with individual 4x4 DNA cross tiles and presumed multi-tile lattices. Increased absorption readings between the two indicates that larger multi-tile lattices formed because more double-stranded DNA exists in multi-tile lattices.

To implement suggestion #1, I designed a new tile set for a new instance of the complementary bounded tiling problem (Figure 15). The instance below does not test corrugation likelihood, but different tile combinations would lead to drastically different tiling patterns. For instance, if tile three (Figure 15) were the only tile used, a vertical/horizontal line tiling pattern would be predominant (Figure 16). Tile three mixed with tile four would form a large partially complete tiling pattern (Figure 17).

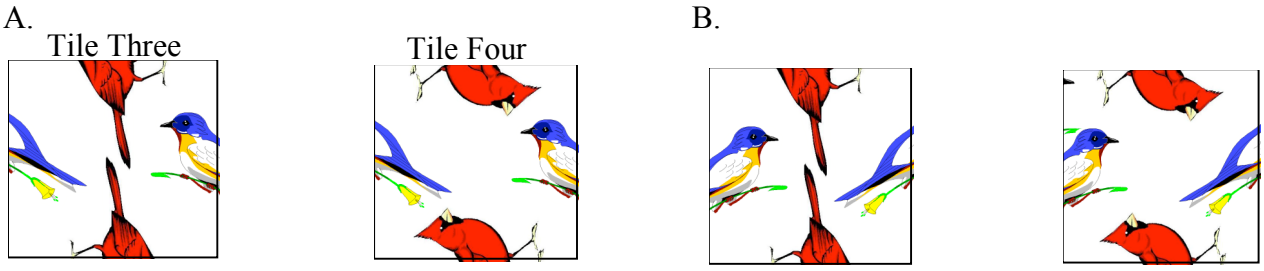


Figure 15. The tile set for the new instance of the complementary bounded tiling problem. The two tiles (A) and their reflections (B).

A. B.

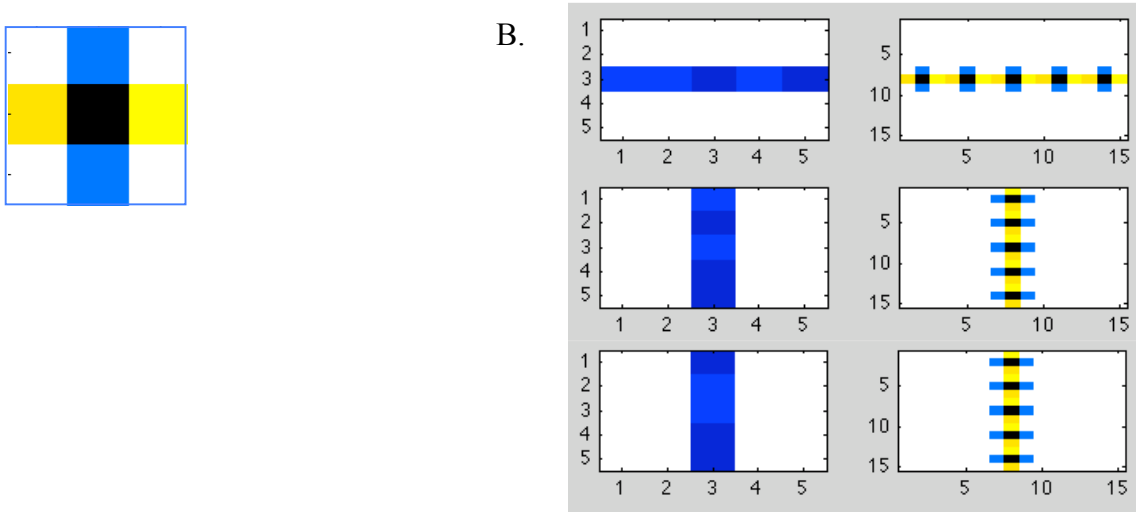


Figure 16. (A) Tile three as represented in the MATLAB simulation referenced in Chapter V. (B) Three different tiling patterns for tile three including reflections run using sporadic_growth. All three show horizontal or vertical tiling patterns

A. B.

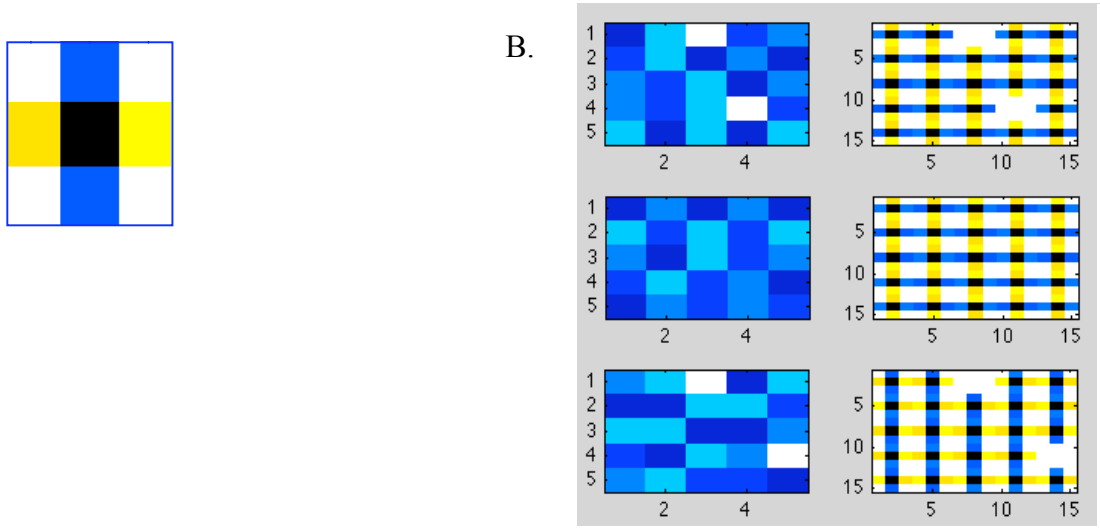
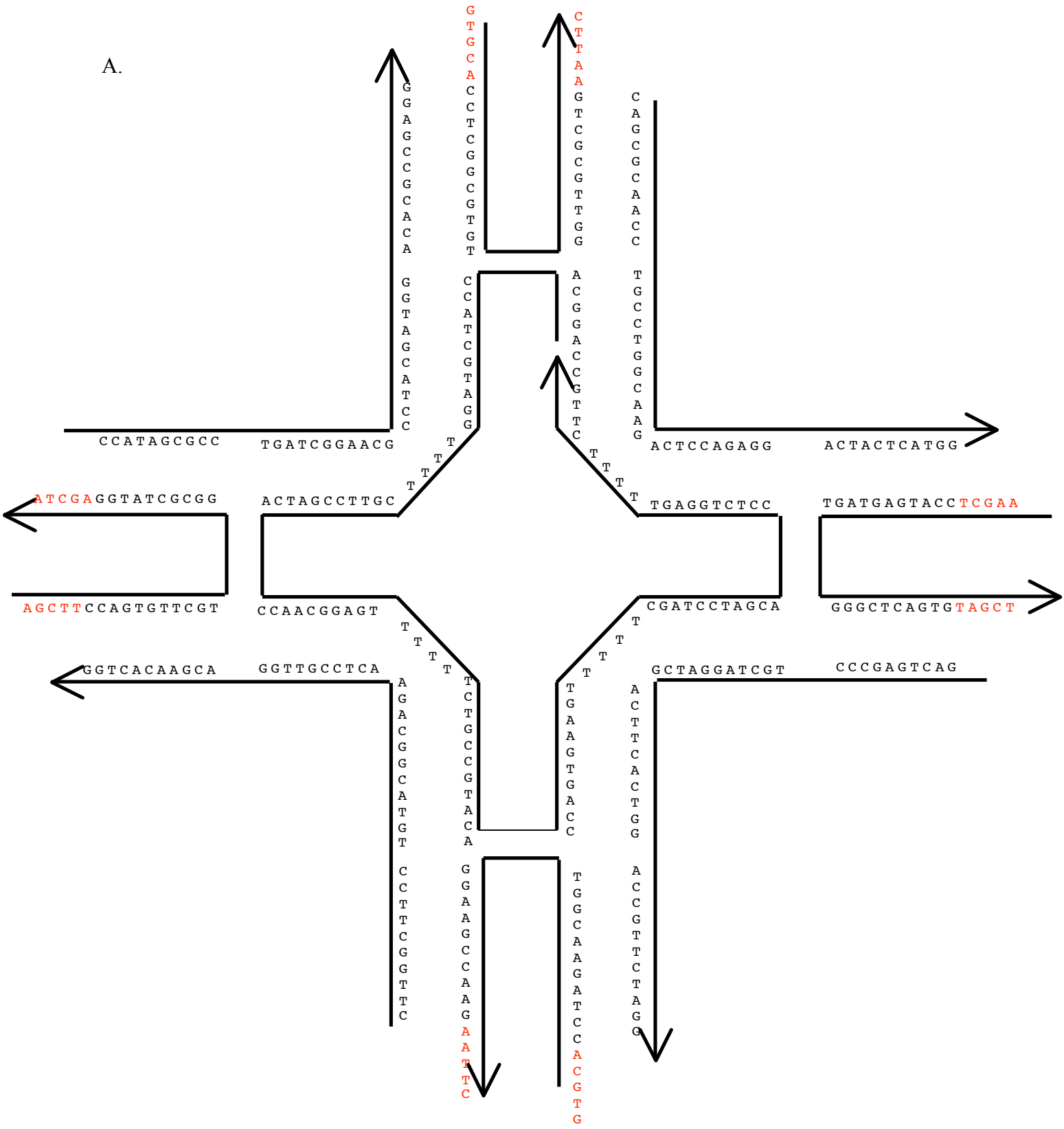


Figure 17. (A) Tile four (B) Three tiling patterns for tile three and tile four run using sporadic_growth. All three show large lattice formations compared to tile three in Figure 16.

Equipped with a new instance of the complementary bounded tiling problem, I ordered new DNA strands from Eurofins MWG Operons. I utilized the same sequences, interchanging only the sticky-end components to align with the new instance (Figure 18)

A.


GTGCACCTCCGGCGTGT
CTTAAGTCGCGTTGG
GGAGCCGCACA
GGTAGCATCC
CAGCGCAACAC
TGCCTGGCAAG
CCATCGTAGG
ACGGACCGTTC
CCATAGCGCC
TGATCGGAACG
ACTCCAGAGG
ACTACTCATGG
TTTTT
TTTTT
ATCGAGGTATCGCGG
ACTAGCCTTGC
TGAGGTCTCC
TGATGAGTACCTCGAA
AGCTTCCAGTGTTCGT
CCAACGGAGT
CGATCCTAGCA
GGGCTCAGTGTAGCT
TTTTT
TTTTT
GGTCACAAGCA
GGTTGCCTCA
GCTAGGATCGT
CCCGAGTCAG
TCTGCCGTACA
TGAAGTGACC
AGACGGCATGT
ACTTCACTGG
CCTTCGGTTC
GGAAGCCAAGAATTC
TGGCAAGATCCACGTG
ACCGTTCTAG

B.

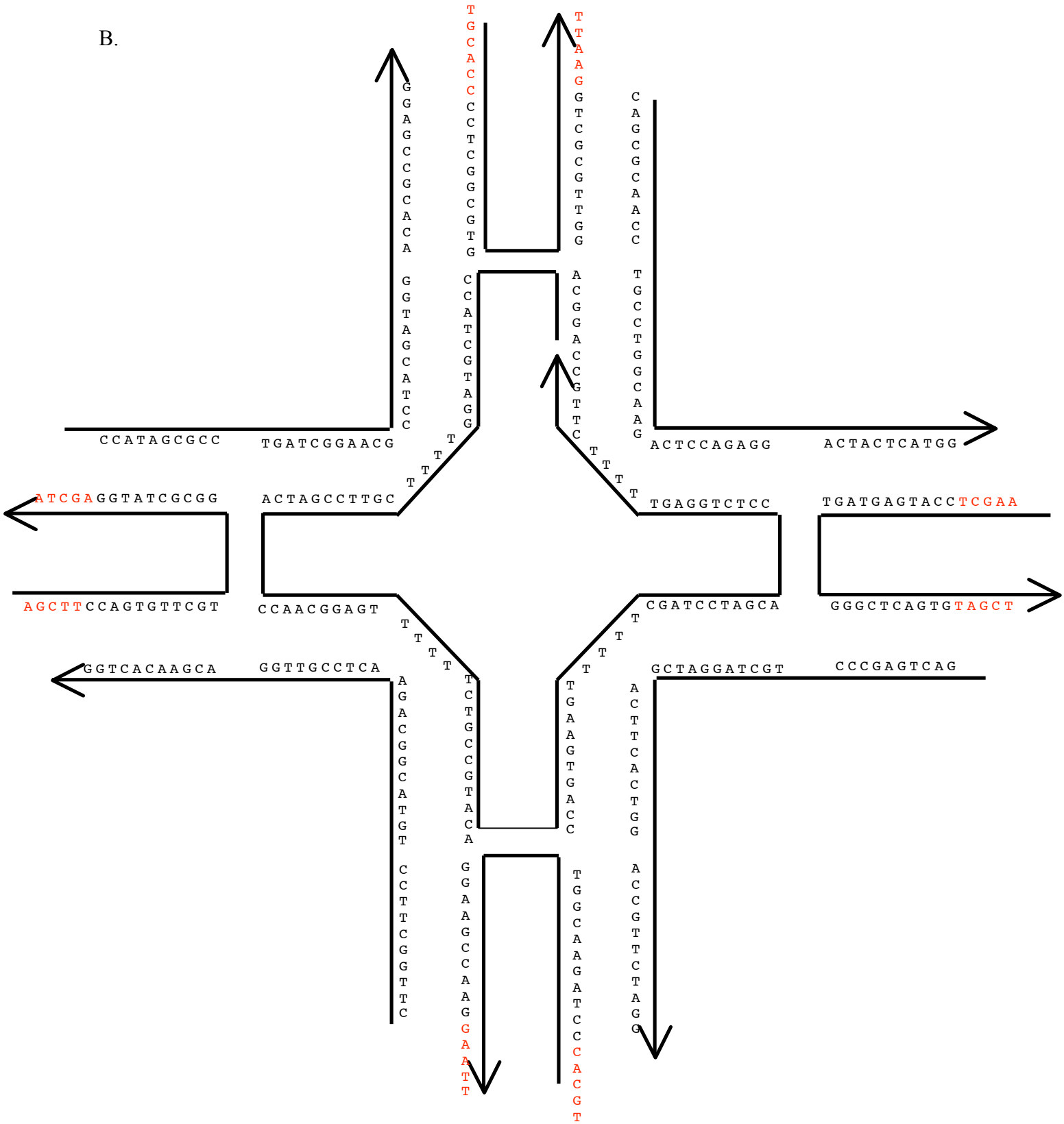

GAGCCGCACA
GGTAGCATCC
CCATAGCGCC
TGATCGGAACG
TGCACCCTCGGCGTG
CCATCGTAGG
TTTTT
TTAAGGTCGCGTTGG
ACGGACCGTTC
TTTTT
CAGCGCAACAC
TGCCTGGCAAG
ACTCCAGAGG
ACTACTCATGG
ATCGAGGTATCGCGG
AGCTTCCAGTGTTCGT
ACTAGCCTTGC
CCAACGGAGT
TTTTT
TGAGGTCTCC
CGATCCTAGCA
TTTTT
TGATGAGTACCTCGAA
GGGCTCAGTGTAGCT
GGTCACAAGCA
GGTTGCCTCA
AGACGGGCATGT
CCTTCCGGTTC
TTTTT
TCTGCCGTACA
GGAAGCCAAGGAATT
TTTTT
TGAAGTGACC
TGGCAAGATCCCACGT
GCTAGGATCGT
CCCGAGTCAG
ACTTCACTGG
ACCGTTCTAGG

Figure 18. Schematic of 4x4 DNA cross tile with DNA sequences for both tile three (A) and tile four (B) as given in Figure 15.

I ordered DNA samples dry, so that I could control dilutions. Once I received the DNA sequences, I diluted twenty-five aliquots of each strand to 1.67 μM and stored in the -80°C freezer to slow degradation. I began experimentation to assemble the new 4x4 DNA cross tiles and mix such tiles together. I had similar success in individual tile assemble, which can be seen in the lanes containing only individual tiles in Figure 19. With similar evidence suggesting successful individual tile assembly, I began multi-tile assembly. In addition to running mixtures of tile three and tile four using PAGE (polyacrylamide gel electrophoresis; Figure 19), I also gathered data on absorbance readings for samples before and after multi-tile assembly (Table 4). Both methods to determine multi-tile assembly indicated that I was unsuccessful in forming multi-tile lattices.

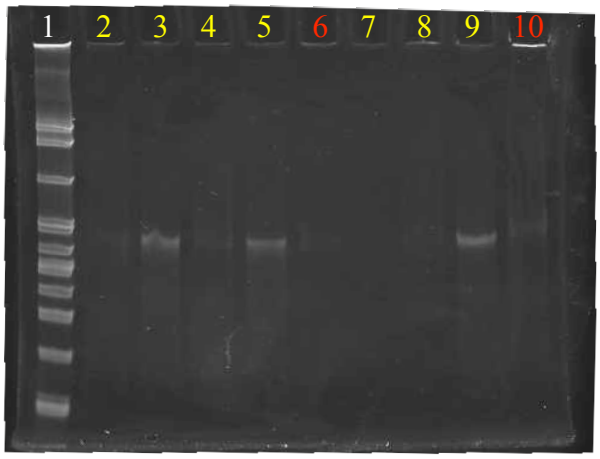


Figure 19. A 5% non-denaturing polyacrylamide gel. Lane 1 is the same 50 bp MW marker (note that this gel was run longer than the others and the 50bp marker is off of the gel so the

marker begins at 100bp). The lanes highlighted in yellow are individual tiles, whereas the lanes highlighted in red are mixtures between tile three and four. I loaded no DNA into lane 7.

The 5% non-denaturing TBE gel pictured in Figure 19 tests two identical mixture samples of tile three and tile four after annealing procedures (exposure to room temperature for 5 hours and 4°C overnight; lanes 6 and 10 in red). In lane 10 there appears to be a structured that will not enter the gel, but I should have seen the same in lane 6, which does not appear. Also, analysis of absorbance readings indicates no statistical difference (Table 4). For the same samples run on the gel in Figure 20, I recorded the respective absorbance readings using Nanodrop.

	Average Nanodrop Reading Before Multi-tile Assembly	**Average Nanodrop Reading After Multi-tile Assembly**	**Difference**	**p-value**
Mixture in Lane 6 (Tile 3 and Tile 4)	0.235	0.326	0.091	0.1336
Mixture in Lane 10 (Tile 3 and Tile 4)	0.346	0.359	0.013	0.355

Table 4. Table indicating for both mixtures (lanes 6 and 10) the average absorbance reading both before and after multi-tile assembly. The respective p-values indicate a lack of statistical difference between both the before and after readings for each mixture.

Out of three sample points for the mixture between tile three and tile four run in lanes 6 and 10 in Figure 19 respectively, the average absorbance before multi-tile lattice formation was 0.235 and 0.346. Average absorbance readings after overnight storage in 4°C refrigerator for lanes 6 and 10 respectively were 0.359 and 0.326 (Table 4). This difference is not statistically significant as observed by the p-values, although this is probably due to the few number of replicates. During

my visit at Duke, Viresh indicated that the increase we should observe from Nanodrop readings between individual tiles and tile lattices is drastic, whereas the above p-values indicate that there is no statistical difference between the two readings. The lack of increase in absorbance readings suggested that larger multi-tile lattices were not forming. These two sources of data indicated that I had not assembled multi-tile DNA lattices and therefore had not assembled potential solutions to the complementary bounded tiling problem.

Analysis of my results indicated two main reasons why I was unable to form multi-tile lattice formation. 1) Individual tile structures may not be forming as expected, preventing larger structures from forming. 2) The sticky-ends may not produce a stable thermodynamic interaction to maintain multi-tile lattice formation. Most research conducted with 4x4 DNA cross tiles uses only two tile types with the knowledge that both tile types are identical and will occur in a regular repeating pattern. The increased randomness encoded within instances of the complementary bounded tiling problem might prevent stable and large structure formation. Although I have yet to show that multi-tiles can assemble to solve the complementary bounded tiling problem, the methodology still appears to be viable for DNA computation. I have various suggestions for further research numbered below.

1) Conduct melting curve analysis of individual tiles by measuring absorbance change upon thermal denaturation. Previous research indicates a unique melting curve of the 4x4 DNA cross tiles (Li, 2008). Cross-referencing my analysis with previous research would help verify individual tile assembly.
2) Use temperature-dependent FRET (fluorescence resonance energy transfer) spectroscopy to analyze multi-tile lattice formation. Recently, researchers successfully demonstrated unique curves using temperature-dependent FRET for multi-tile 4x4 DNA cross tile

lattices. Researchers indicated that studying absorbance changes for multi-tile assemblies presents unique challenges because "the amplitude of the absorbance change for the dissociation of sticky ends (usually only 5–10 nucleotides long) is overshadowed by the much larger absorbance change resulting from the dissociation of the core of the DNA tile" (Nangreave *et al*., 2009). FRET analysis measures the proximity of fluorescence probes attached to the sticky-ends of various tiles allowing Nangreave *et al.* to more accurately quantify absorbance changes between individual tiles and multi-tile lattices. The research findings of Nangreave *et al.* indicate that the lack of statistical difference between absorbance readings seen in Table 4 may not accurately suggest failed multi-tile assembly. Therefore, conducting temperature-dependent FRET analysis would more concretely verify potential multi-tile assembly.

3) Re-analyze melting temperature studies across a wider range of predictive programs to ensure thermal profiles maximize the specific annealing patterns of the 4x4 DNA cross tile. Include the original SEQUIN program for strand analysis.
4) Reassess problems of DNA degradation, buffer and Mg^{2+} concentrations, including experimentation with TBE versus TAE.
5) Select sticky-ends that previous researchers have determined to be the most thermodynamically stable, disregarding the initial selection for sticky-ends with restriction enzyme sites. If tile formation does not occur using new sticky-ends, design two 4x4 DNA cross tiles that other researchers have shown can successfully assemble. Use these replicas of previously tested 4x4 DNA cross tiles to test my protocols and methodology.

6) With successful lattice formation, expose potential lattices to combinations of restriction enzymes. Recall that I engineered the sticky-ends to form restriction enzyme sites if multi-tile lattices formed. Exposing multi-tile lattices to restriction enzymes could produce lattices of different sizes. I would like to observe via PAGE potential changes in individual tile band intensities between mixtures that have and have not been exposed to restriction enzymes. Restriction enzymes should cut apart the sticky-ends holding the individual tiles together. Therefore, mixtures exposed to restriction enzymes should show increased individual tile band intensities when compared to mixtures not exposed to restriction enzymes.
7) Engineer annealed sticky-ends to form a binding site for a protein or probe. If two annealed tiles could bind a protein, a Western Blot analysis of DNA mixtures would indicate the presence of the protein only if the DNA tiles successfully annealed, forming the binding site. A fluorescently labeled DNA probe with greater specificity to the 10-nucloetide sequence would decrease the potential for protein partial recognition and binding.
8) Expose multi-tile mixtures to pulse-chase gel electrophoresis. Pulse-chase gel electrophoresis uses multiple currents to attract negatively charged DNA in different directions, producing lateral and vertical movement of DNA molecules through a gel. The combination of different currents pulls large pieces of DNA into the gel. This technique may be used to pull the large DNA lattices into the gel, allowing potential multi-tile visualization without using an AFM.

The eight suggestions above are listed by priority and are not exhaustive. Previous success of controlled 4x4 DNA cross tile lattice formation indicates potential for my computational design

to assemble correctly. These suggestions will help shape my next steps in solving the complementary bounded tiling problem using 4x4 DNA cross tiles.

Conclusions

My research took a novel structural approach using bionanotechnology concepts to solve the NP-complete complementary bounded tiling problem. I made contributions in the field of mathematics by contributing a succinct definition of the complementary bounded tiling problem. The proof that the complementary bounded tiling problem is NP-complete was never formally written before my thesis. I was the first to write and analyze an algorithm to prove that the complementary bounded tiling problem is in NP. The MATLAB simulation has powerful predictive possibilities when studying tiling patterns forming from different instances of the complementary bounded tiling problem. To my knowledge, no one has simulated tiling problems in such a manner. Therefore, the MATLAB program is a first-of-its-kind predictor for tiling problems such as the complementary bounded tiling problem. I would like to submit it for publication in collaboration with Dr. Heyer.

Although I faced obstacles in the biological assembly of the 4x4 DNA cross tiles, using such a structural approach still has promise. There are more obstacles and considerations to overcome, including the limitations on defining tile locations within the solution, defining a bounded region biologically, and difficulties ensuring assembly of multi-tile lattices in the first place. Although 4x4 DNA cross tiles have been previously studied, my research is the first of my knowledge to use such tiles to solve computational problems. In addition, my investigations into corrugation frequency are necessary for the field of bionanotechnology to proceed with assumptions about the interactions of DNA cross tiles. I have made various suggestions for future experimental directions in Chapter VI. Overall, the basics of my research continues to show promise as a structural approach to DNA computation but require more time to ensure accuracy and stability.

References

Aaronson, Scott. "Computational Complexity and the Anthropic Principle." 2006 Stanford Institute for Theoretical Physics. 15 Dec. 2006.

Adleman, Leonard. "Molecular Computation of Solutions to Combinatorial Problems." *Science*. 1994: 266(5187): 1021-1024.

Aldaye *et al*. "Assembling Materials with DNA as the Guide." *Science.* 321 (2008): 1795–1799.

Allender *et al.* "Complexity Classes." Algorithms and the Theory of Computation Handbook. Ed. Mikhail J. Atallah. Florida: CRC Press, 1999.

Amos, Martyn. "DNA Computing." Encyclopedia of Complexity and System Science. 2008

Barker-Plummer, David. "Turing Machines." Stanford Encyclopedia of Philosophy. California: Metaphysics Research Lab, 2010.

Brandt *et al*. "Using Backtracking to Solve the Scramble Squares Puzzle." *Journal of Computer Science* 17.4 (2002).

Brucale, Marco. "Design, Synthesis, and Characterization of DNA Supramolecular Nanostructures." Dissertation of University of Bologna Department of Biochemistry, 2006.

Brun *et al.* "Building Blocks for DNA Self-Assembly." Proceedings of the 1st Foundations of Nanoscience: Self-Assembled Architectures and Devices, FNANO, Snowbird, UT, 2004.

Carrano, Frank and Janet Prichard. Data Abstraction and Problem Solving with Java. Boston: Pearson/Addison-Wesley, 2006.

Cook, William. "Applications of the TSP." The Traveling Salesman Problem. Jan 2007. Web 10 Mar 2011 <http://www.tsp.gatech.edu/apps/index.html>.

Dasgupta *et al*. Algorithms. McGraw-Hill, 2006. 169-199 and 247-278.

Deaton, *et al*. "DNA Computing: A Review." *Fundamenta Informaticae*. 30 (1997): 23-41.

Demaine, Erik and Martin Demaine. "Jigsaw Puzzles, Edge Matching, and Polyomino Packing: Connections and Complexity." *Graphs and Combinatorics* 23 (2007): 195–208.

De Rezende, Pedro J. Class Lecture. "NP-hard Problems." Complexity of Algorithms. State University of Campinas, Brazil. 4 Dec 2009. Web 10 Mar 2011 <http://www.ic.unicamp.br/~rezende/ensino/mo417/2009s2/notas/16-nphard.pdf>.

Dwyer, *et al.* "Design Automation for DNA Self-Assembled Nanostructures." *DAC: Proc. 43rd Design Automation Conf.* 2006.

Fourman, Michael. Class Lecture. "NP-Completeness and Cook's Theorem." Logic and Computation. The University of Edinburgh, Scotland. 15 Jan 2002.

Friedman, Harvey M. "Clay Millennium Problem: P = NP." Mathematics Colloquium. Ohio State University. Columbus, 20 October 2005.

Garey, Michael and David Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. San Francisco: W.H. Freeman and Company, 1979.

Gopalkrishnan, Nikhil. "The Tiling Problem." Computational Geometry. Duke University. 2008.

Haris *et al.* "The Basics." Combinatorics and Graph Theory. New York: Springer, 2008. 5-10.

Hays *et al.* "Caution! DNA Crossing: Crystal Structures of Holliday Junctions." *Journal of Biochemistry*. 278 (2003): 49663-49666.

Li, Hanying. "Self- Assembled DNA Nanostructures: From Structural Material to Biomedical Nanodevices." Dissertation of Duke University Department of Pathology, 2008.

Johnson, Matthew. Class Lecture. "NP-completeness and the Traveling Salesman Problem." Algorithms and Complexity. Durham University, United Kingdom. 24 April 2009.

Kamvysselis, Manolis. "Universal Turing Machine." Massachusetts Institute of Technology. 5

Dec. 1999. Web 12 Feb 2011 < http://web.mit.edu/manoli/turing/www/turing.html>.

Kari, Lila. "DNA Computing: The Arrival of Biological Mathematics." *The Mathematical Intelligencer*. 1997: 19(2): 9-22.

Kingsford, Carl. Class Lecture. "SAT, Coloring, Hamiltonian Cycle, TSP." Design and Analysis of Algorithms. University of Maryland, College Park. 2009.

Kleinberg, Jon and Eva Tardos. Algorithm Design. Boston: Pearson/Addison-Wesley, 2006.

Kuzuya *et al.* "DNA origami: Fold, stick and beyond." *Nanoscale*. 2 (2010): 310-322.

Limura *et al*. "Sequence Design Suport System for 4x4 DNA Tiles." *Proceedings of the 13th International Conference on DNA Computing* (2007): 140-145.

Lin *et al.* "Designer DNA Nanoarchitectures." *Biochemistry*. 48.8 (2009): 1663-1674.

Liu *et al.* "Tensegrity: Construction of Rigid DNA Triangles with Flexible Four-Arm DNA Junctions." *Journal of the American Chemical Society*. 126 (2004): 2324-2325.

Los Angles Harbour College. "Nanoscale- Nanometers." 19 Mar 2009. Web 15 Feb 2011 <http://www.lahc.edu/classes/nanotechnology/nanoscale.html>.

Markham, Nicholas and Michael Zuker. "DINAMelt web server for nucleic acid melting prediction." *Nucleic Acids Res.* 33 (2005): 577-581.

Martin, John C. "Nondeterministic Turing Machines." Introduction to Languages and the Theory of Computation. New York : McGraw-Hill, 1997. 277-281.

"Melting Temperature of DNA." Biology Online Dictionary. 3 Oct 2005. Web 22 Mar 2011 <http://www.biology-online.org/dictionary/Melting_temperature_of_dna>.

Nangreave *et al. "*Studies of Thermal Stability of Multivalent DNA Hybridization in a Nanostructured System." *Biophysical Journal* 97.2 (2009): 563–571.

"nondeterministic polynomial time." The Free On-line Dictionary of Computing. 2010. Denis Howe. Web 10 Mar 2011 <http://dictionary.reference.com/browse/nondeterministic

polynomial time>.

Park *et al.* "Finite-Size, Fully Addressable DNA Tile Lattices Formed by Hierarchical Assembly Procedures." *Angewandte Chemie*. 118 (2006): 749-753.

Park, Sung Ha. "Self-Assembled DNA Nanostructures and DNA-Templated Silver Nanowires." Dissertation of Duke University Department of Physics, 2005.

Parker, Jack. "Computing with DNA." *EMBO reports* 4.1 (2003): 7-10.

Reddy, Himanshu. "A Seminar Report on DNA Computing." Department of Electronics and Communication Engineering. The National Institute of Technology, Calicut, India. 2010

Rothemund, Paul. "Scaffolded DNA origami: From generalized multicrossovers to polygonal networks." Nanotechnology: Science and Computation (2006): 3–21.

Sanderson, Katharine. "What to make with DNA origami." *Nature*. 464 (2010): 158-159.

Seeman, Nadrian. "Biochemistry and Structural DNA Nanotechnology: An Evolving Symbiotic Relationship." *Biochemistry*. 42.24 (2003): 7259-7269.

Seeman, Nadrian. "De Novo Design of Sequences for Nucleic Acid Structural Engineering." *Journal of Biomolecular Structure & Dynamics* 8.3 (1990): 573-581.

Seeman, Nadrian. "An Immobile Nucleic Acid Junction Constructed from Oligonucleotides." *Nature* 305 (1983): 829-831.

Seeman, Nadrian. "An overview of structural DNA nanotechnology." *Molecular Biotechnology*. 37.3 (2007): 246-257.

Seeman *et al.* "New Motifs in DNA Nanotechnology." *Nanotechnology* 9 (1998): 257-273.

Sipser, Michael. "The Class P." Introduction to the Theory of Computation. Boston: PWS Publishing Company, 2006. 234-241.

Stellwagen, Nancy. "Electrophoresis of DNA in agarose gels, polyacrylamide gels and in free

solution.” *Electrophoresis* (2009):188-95.

Terr, David. "Polynomial Time." From *MathWorld*--A Wolfram Web Resource, created by Eric W. Weisstein. 2010. Web 10 Mar 2011 <http://mathworld.wolfram.com/PolynomialTime.html>.

Van Emde Boas, P. and M. P. W. Savelsbergh. “Bounded Tiling, an alternative to Satisfiability?” *Proceedings of the 2nd Frege Memorial Conference*. Akademie Verlag, 1984, 401-407.

Vigliaturo, Cecilia. “Digital DNA.” The University of Arkansas. 2010.

Yan *et al.* “DNA-Templated Self-Assembly of Protein Arrays and Highly Conductive Nanowires.” *Science*. 301 (2003): 1882-1884.

Appendix

Basic MATLAB code for checking candidate solutions to the Scramble Square problem in P:

```
%if the solution is a square, then it is the Scramble Square problem
if (xDim==yDim)
        %calculate the total number of touching edges that must be checked
        totalSides=xDim*(xDim-1)+xDim*(xDim-1)

        %create matrix A
        A=zeros (xDim*(xDim-1)+xDim*(xDim-1), nInSolution*4)

        %test top and bottom edges of the square
        a=3;
        b=1+4*xDim;

        for i=1:xDim*(xDim-1)
                A(i,a)=1;
                A(i,b)=1;

                %redefine the edges that need to be checked
                a=a+4;
                b=b+4;
    end

        %test left and right edges of the square
        %when you reach the end of a row in the tiling solution, you must jump
        a=2;

        for j=(xDim*(xDim-1))+1:2*xDim*(xDim-1)
                jump=0;
                if rem (j,sqrt(nInSolution)-1)==0;
                        jump=4;
                end

                A(j,a)=1;
                A(j,a+6)=1;

                a=a+4+jump;
        end
```

```
%multiply the matrix A by the inverse of the given vector, V
        if (A*V'== zeros(xDim*(xDim-1)+xDim*(xDim-1),1))
                'Congratulations, you found a solution!!:-)'
            else
                'Sorry, try again!'
            end
end
```

Basic MATLAB code for checking candidate solutions to the complementary bounded tiling problem in P:

```
%Recall that the bounded region can be different shapes for the complementary bounded tiling problem

%the bounded region is a horizontal chain
if(yDim==1)
        A=zeros (nInSolution-1,nInSolution*4);
        %only need to check left and right
        a=2;
        b=8;
        for i=1:(nInSolution-1)
                A(i,a)=1;
                A(i,b)=1;
                a=a+4;
                b=b+4;
        end

        if (A*V'== zeros(nInSolution-1,1))
                'Congratulations, you found a solution!!:-)'
        else
                'Sorry, try again!'
        end
end


%the bounded region is a vertical chain
if(xDim==1)
        A=zeros (nInSolution-1,nInSolution*4);
        %only need to check top and bottom of the squares
        a=3;
        b=5;
        for i=1:(nInSolution-1)
                A(i,a)=1;
                A(i,b)=1;
                a=a+4;
                b=b+4;
        end
```

```
        if (A*V'== zeros(nInSolution-1,1))
                'Congratulations, you found a solution!!:-)'
        else
                'Sorry, try again!'
        end
end


%the bounded region is a rectangle, where the horizontal is longer
if (xDim>yDim)
        A=zeros (nInSolution+1,nInSolution*4);
        %check edges on top and bottom of the squares
        a=3;
        b=1+4*xDim;
        for i=1:(xDim*(yDim-1))
                A(i,a)=1;
                A(i,b)=1;
                a=a+4;
                b=b+4;
        end

        %right and left of the squares
        a=2;
        indicatorTwo=1;  %tells you when you need to jump
        for j=(xDim*(yDim-1))+1:nInSolution+1
                jump=0;
                if indicatorTwo==xDim-1;
                        jump=4;
                end

                A(j,a)=1;
                A(j,a+6)=1;
                a=a+4+jump;
                indicatorTwo=indicatorTwo+1;
        end

        if (A*V'== zeros(nInSolution+1,1))
                'Congratulations, you found a solution!!:-)'
        else
                ‘Sorry, try again!'
```

```
        end
end
%rectangle in the other, verticle, direction
if (yDim>xDim)
        A=zeros (nInSolution+1,nInSolution*4);
        a=3;
        b=1+4*xDim;
        %top and bottom squares
        for i=1:(yDim-1)*xDim
                A(i,a)=1;
                A(i,b)=1;
                a=a+4;
                b=b+4;
        end

        %right and left edges
        a=2;
        for j=((yDim-1)*xDim)+1: nInSolution+1
                jump=0;
                if rem (j,xDim-1)==0;
                        jump=4;
                end
                A(j,a)=1;
                A(j,a+6)=1;
                a=a+4+jump;
        end

        if (A*V'== zeros(nInSolution+1,1))
                'Congratulations, you found a solution!!:-)'
        else
                'Sorry, try again!'
        end
end


%if the solution is a square, then do the same as the Scramble Square code above
```