University of Texas Rio Grande Valley
ScholarWorks @ UTRGV

Theses and Dissertations - UTB/UTPA

5-2013

Nascent nanocomputers: DNA self-assembly in O(1) stages

Michael C. Barnes
University of Texas-Pan American

Follow this and additional works at: https://scholarworks.utrgv.edu/leg_etd

 Part of the Computer Sciences Commons

Recommended Citation

Barnes, Michael C., "Nascent nanocomputers: DNA self-assembly in O(1) stages" (2013). *Theses and Dissertations - UTB/UTPA*. 842.
https://scholarworks.utrgv.edu/leg_etd/842

This Thesis is brought to you for free and open access by ScholarWorks @ UTRGV. It has been accepted for inclusion in Theses and Dissertations - UTB/UTPA by an authorized administrator of ScholarWorks @ UTRGV. For more information, please contact justin.white@utrgv.edu, william.flores01@utrgv.edu.

NASCENT NANOCOMPUTERS:

DNA SELF-ASSEMBLY

IN O(1) STAGES

A Thesis

by

MICHAEL C. BARNES

Submitted to the Graduate School of the
University of Texas-Pan American
In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2013

Major Subject: Computer Science

NASCENT NANOCOMPUTERS:

DNA SELF-ASSEMBLY

IN O(1) STAGES

A Thesis
by
MICHAEL C. BARNES

COMMITTEE MEMBERS

Dr. Robert Schweller
Chair of Committee

Dr. Richard Fowler
Committee Member

Dr. Andres Figueroa Lozano
Committee Member

Dr. Laura Grabowski
Committee Member

May 2013

Copyright 2013 Michael C. Barnes
All Rights Reserved

ABSTRACT

Barnes, Michael C., Nascent Nanocomputers: DNA Self-Assembly in O(1) Stages. Master of Science (MS), May, 2013, 39 pp., 3 tables, 12 illustrations, references, 36 titles.

DNA self-assembly offers a potential for nanoscale microcircuits and computers. To make that potential possible requires the development of reliable and efficient tile assembly models. Efficiency is often achieved by minimizing tile complexity, as well as by evaluating the cost and reliability of the specific elements of each tile assembly model. We consider a 2D tile assembly model at temperature 1. The standard 2D tile assembly model at temperature 1 has a tile complexity of O(n) for the construction of exact, complete n x n squares. However, previous research found a staged tile assembly model achieved a tile complexity of O(1) to construct n x n squares, with O($logn$) stages. Our staged tile assembly model achieves a tile complexity of O($logn$) using only O(1) stages to construct n x n squares.

DEDICATION

This thesis is a product of the imagination, made visible through a rigorous application of the principles of science. It is, in large part, a testament of my love for my two sons, Michael Christopher "Quito" Borrego Barnes, and Roberto Jesus Borrego Barnes. Through this effort I hope to encourage them to always to let their imaginations serve as their guide.

ACKNOWLEDGMENTS

I will always be grateful to Dr. Robert Schweller, chair of my thesis committee, for all his mentoring and advice. He was instrumental in the completion of this process through his infinite patience. Specifically, I recall with gratitude his insistence that I participate in the 2010 Computer Science Student Research Day at UTPA, work which formed the core research for this thesis. Many thanks also go to my thesis committee members: Dr. Richard Fowler, Dr. Andres Figueroa Lozano, and Dr. Laura Grabowski. Their coursework, each in different ways, contributed to my growth and development, specifically in the area of design and analysis of algorithms. Further, their advice, input, and comments on my thesis helped to ensure the quality of my intellectual work. In addition, Dr. Richard Fowler's guidance has been especially crucial, given my non-science undergraduate background.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER I

INTRODUCTION

DNA Self-Assembly

DNA self-assembly is an exciting field that combines state of the art research in biological sciences with heavy-hitting algorithms designed by some of the top computational theorists in the world. The goal is no less than to manipulate, control, and commandeer the building blocks of life, DNA, and to use these nanostructures to design the next generation of miniature circuits, and even DNA-sized devices capable of computation. Self-assembly generally is described by Adleman [2] as "the ubiquitous process by which simple objects autonomously assemble into intricate complexes." As Doty shows [12], this type of engineering is very "hands-off." He explains that "the right molecules are placed in solution, and the structures and devices self-assemble spontaneously according to the principles of chemical kinetics." Adleman [2] continues to suggest that "self-assembly processes will ultimately be used in circuit fabrication, nano-robotics, DNA computation, and amorphous computing." As this introduction will show, these outputs are not science fiction, but theoretically proven and therefore plausible science futures.

The Standard Tile Assembly Model

There are many good overviews of the origin of the field of DNA Self-Assembly [1, 2, 4, 13, 27, 28]. Each of these overviews begin with a description of the abstract tile concept advanced by Wang [32, 33]. Wang tiles are four-sided, non-rotatable, and each side has a *glue*

that allows it to attach to abutting tiles with a similar glue type. Glue types are defined by a *label* on each side of the tile. The Wang tile concept is removed from the abstract by Winfree [34, 36] who proved that DNA, either natural or synthetic, can be converted to an equivalent form, based on previous pioneer research by Seeman [29]. Thus, the entire field of DNA self-assembly is based on the ability to associate a theoretical concept, the Wang tile, with a real-world environment, DNA manipulation at the nanoscale, as described by Winfree.

The idea of a *temperature* is introduced by Winfree [34] to ensure that two corresponding glues will only attach if their overall *strength* is equal to or greater than the temperature of the system. The standard temperature 2 model is thus established by Winfree [34, 36]. DNA tiles are also generally described as floating on a 2D surface. Each tile assembly model consists of a finite number of tile types, assuming an infinite number of copies of each tile type [12]. All of these conditions are referred to collectively as the abstract tile assembly model (aTAM), as refined by Rothemund and Winfree [27].

Because Wang's abstract tile model was already proven to simulate any universal Turing machine, the abstract tile assembly model possesses the same computational power. Hence the premise and the promise; nanocomputers build out of DNA are plausible science future, not science fiction. Specifically, while Winfree [34] showed that the aTAM can simulate any universal Turing Machine, research by Doty, et al. [12] shows that the tile assembly model is *intrinsically universal*, by which they mean that not only can the aTAM tell us what a universal Turing Machine does, but it can actually perform the functions described.

In computer science, there is a necessary emphasis on the efficiency of a given algorithm, as encouraged by Turing [31]. Specifically, in the field of DNA self-assembly, the emphasis is on the use of tile complexity to determine the efficiency of tile-assembly models [23, 30]. Tile

complexity is generally defined as the number of unique tile types needed to complete a given construction. The number of tile types needed is typically based on a measure of *tiles* and *glues*. If we let *t* and *g* represent the total *tiles* and *glues* in a system, respectively, then optimal solutions involve a *t* and *g* count that follow a logarithmic trend, or at least O(*logn*). Anything more complex than this becomes difficult to recreate in a laboratory setting. A complexity of O(1), which refers to a constant number of elements, *c* is the best possible outcome for any measure of computational complexity. A common construction emphasized here is the *n* x *n* square, which is described in one paper as the "canonical" construction [21] for the field. So the typical goal is to pursue a tile type complexity of as close to O(1) as possible, while achieving an exact *n* x *n* construction.

Generally, most researchers emphasize temperature 2 models, and several papers address the limitations of temperature 1 constructions [14, 27]. Specifically, temperature 1 tile assembly models were shown to require at least as many tile types as the diameter of the assembled shape [24]. However, research that emphasizes the limitations of temperature 1 systems tends to assume that there are no significant modifications to the aTAM. Research that allowed for reasonable modifications to temperature 1 tile-assembly models showed results that were consistent with temperature 2 constructions. These modifications include the introduction of a negative glue [25], the use of a 3D tile assembly model [9], and most crucially for our paper, the introduced of a staged tile assembly model sTAM [10, 11, 19]. Specifically, the work of Schweller within the Computer Science department at the University of Texas – Pan American emphasizes attention to potential variations in temperature 1 models with efficient tile complexities [9, 10, 25].

It is also important to consider the discussion by Doty [15] who asserts that *speed*, which in the case of the aTAM is associated with efficient use of distinct tile types, is not the only consideration in determining an effective algorithm. *Correctness* is also critical. For this reason, there has been considerable emphasis, not just on efficient constructions using the aTAM and other models, but also on error-resistant approaches, referred to as *robustness* [20]. Significant research has been conducted to improve the robustness of tile assembly models [21, 26, 35]. There is also some argument [9] that temperature 1 systems may have advantages when it comes to robustness, over the standard temperature 2 model (aTAM).

To achieve robustness in tile assembly models, researchers introduce assumptions to define the boundaries that separate reality from the theoretical realm. Some of these assumptions are applied so frequently that they become standard operating procedures. One of these examples include the assumption that mismatches are not possible. A *mismatch* would mean that two tiles whose glues do not match adhere accidentally, or incidentally, by being pressed into one another after legitimate bonds take effect, and create flaws in the final structure. This is common with large repetitions in a laboratory setting, and compares to the existence of genetic defects in real life. Another example is that of *partial construction*. Most researchers assume that every piece of their model will assembly completely, and that a given structure will not progress to a later stage as a partial construction at any point. While partial constructions are a very real possibility in lab settings, most computational theorists assume that over a large enough sample, partial constructions can simply be discarded, or "washed away," leaving behind the complete terminal structure the system seeks to construct. Models that can account for these errors without relying on the assumptions above would be described as significantly more error-resistant, and pursuit of these tile assembly models is a worthy direction for future research.

The Staged Tile Assembly Model

In this paper, we try to maximize the potential for DNA self-assembly at *temperature 1*. *Temperature 2* models have shown an amazing diversity of powerful solutions to complex problems that have no proven solutions at temperature 1 [14]. However, the cost of creating and running a temperature 1 model in a laboratory setting is significantly less, and performance guarantees of O(*logn*) or better at temperature 1 are worth exploring as alternatives to more complex temperature 2 approaches. Specifically, we extend the recent success of the *staged assembly model* [9] (sTAM) to create a modified staged tile self-assembly model that uses a constant number of stages, or O(1), to construct *n* x *n* squares, using an upper-bound of O(*logn*) tile-types and distinct glues.

The original staged tile assembly model (sTAM) focused on a constant number of tile types, or a tile set with complexity O(1). The standard tile assembly model, as described in the introduction, depends upon a set of DNA tiles that are engineered, then introduced into a solution, in which they perform interactions based on chemistry [13], and after a certain period of time, self-assemble into their terminal shapes. The staged assembly model adds a layer of complexity to the stage at which the DNA tiles are introduced into a solution. Instead of a single beaker or *bin* into which all the DNA tiles are mixed at once, there are some number of bins which hold some number of DNA tile types, which can be introduced into new bins and mixed based on a particular sequence, like mixing bowls in a baking recipe. Each step in the sequence is called a *stage*. So staged self-assembly refers to the introduction of tile types into a solution based on a particular sequence. The use of stages is a way to simulate some temperature 2 operations, including the ability to design an assembly system in sequence, and control the order of interactions between different tiletypes.

CHAPTER II

A MODIFIED STAGED TILE ASSEMBLY MODEL

To approach the same tile complexity needed to construct exact shapes, such as an n x n square, with a temperature 2 model, using instead a temperature 1 tile assembly model, requires that we modify the standard tile assembly model. Without any additional elements, the performance of a temperature 1 tile assembly model cannot improve upon the established lower bound for tile complexity. In this chapter we describe the specific changes we make in our modified staged tile assembly model.

While prior research into the staged assembly model proved that a constant number of tile types can be used to create exact shapes [10], for example the n x n square, with a stage complexity of O($logn$), we are considering an alternative case, where we use constant stages, or a stage complexity of O(1) and a tile type complexity of O($logn$). All of these alternate paths offer the practical biological researcher a range of options to consider based on what is determined to be the right mix of efficiency in an actual laboratory setting. If a staged tile assembly model is considered to be prohibitively *expensive* at a stage complexity of O($logn$), but tile type complexity is less expensive, then a staged tile assembly model with a stage complexity of O(1) may be preferred. Additionally, Becker [4] discusses the idea of time complexity, which is generally assumed to be O(n) for most models. Since a stage complexity of O($logn$) may significantly increase the time complexity, an O(1) stage complexity may be beneficial still.

Consistent with the standard tile assembly model, our staged tile assembly model is nondeterministic, and we will assume an infinite number of copies of each tile type are available. Our tile types are all defined as Wang non-rotatable tiles. Each stage will terminate when a sufficiently large number of each of the terminal constructions for that stage have been created. At the end of each stage, unused constructions, tiles, and glues will be washed away, leaving only the terminal constructions accessible to new tile types introduced in the next stage of the sequence. We also assume that glue mismatches are not permitted.

O(1) Bin Complexity

While research has been done that looks at the effects of different levels of bin complexity [19], for our modified staged tile assembly model we consider a constant number of bins, or a bin complexity of O(1). Specifically, for the purposes of our tile assembly model we assume a single mixing bin, and will no longer refer to bins in this particular work.

Implementing A Binary Counter

The use of a binary counter is essential to the overall construction of our n x n square. This is because a binary counter needs only $logn$ bits to represent a string of length n. And so we can construct our n x n square out of $logn$ x n size rectangles, and maintain a tile complexity of O($logn$). Our binary supertiles described above will affix in our first stage to create long *strings* of supertiles of length 1 x h where h represents the total number of bits in the binary counter, where $h \leq logn$. We will then allow those strings to assemble into a rectangle of size h x m, where $m \leq n$. Because a binary counter terminates at a power of 2, to arrive at a length of precisely n, we must accumulate a series of rectangles whose total length is n. The length of each rectangle in the series, and the total number of rectangles is determined by the binary representation of the number n. In our figures and example, we consider an arbitrary number 45

set to n. In this case, the binary representation is 101101. For every 1 digit in the l^{th} place of the binary sequence, we will construct a rectangle of length m, such that $m = 2^l$.

Introducing Geometry

A second modification that makes it possible to implement a staged tile assembly model with a stage complexity of O(1) is the use of novel geometry to enforce exact constructions. Recent research has confirmed the power of geometry in temperature 1 constructions, and in this case, the use of a simple geometry is essential to the performance of a *supertile* binary counter. A supertile is a construction involving multiple tiles that once combined function as a de facto single tile type in future interactions. Essentially, as we construct n x n squares in this staged tile assembly model, we will be using supertiles whose total length can be defined as 1 unit, where n units have a total length n. In other words, it is possible to create supertiles of an arbitrary size so that to achieve an n x n square we use n x n total supertiles.

The binary counter supertile consists of four tiles bonded so that the bottom two tiles represent a binary digit, 1 or 0. The first, bottom left, or SW digit represents the actual binary digit for the supertile as part of a binary sequence. The second, bottom right, or SE digit represents the next, or adjacent binary digit for the supertile that should attach to the right. The top two tiles are used to enforce the proper bonding with the next supertile in the binary sequence. In our supertiles there will generally be a single glue affixed to the North side of the top left, or NW tile, as shown in figure 1, responsible for affixing a single bit in the binary sequence to its next bit.

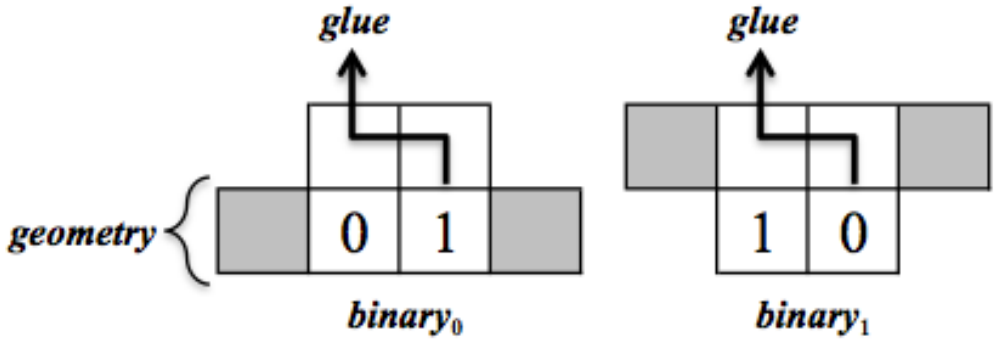

Figure 1: Binary Supertiles

The geometry of each supertile is determined by the binary digits of that specific supertile. If the digit on the SW tile, or bottom left tile, is a 1, then we affix an extra geometry, or restrictive tile on the bottom left side, or the West side of the SW tile. If the digit on the SW tile is a 0, then we affix a restrictive tile on the top left side, or the West side of the NW tile. This geometry is reversed on the East side, with a restrictive tile affixed to the bottom right side, or East side of the SE tile, if the SE tile carries a 0 digit. A restrictive tile is affixed to the top right side, or East side of the NE tile, if the SE tile carries a 1 digit. All of this is shown in figure 1. The term 'restrictive tile' is a reminder that the purpose of the extra geometry is to restrict growth to the left or right of the tile except in the case of a binary digit match. A supertile that calls for a 0 on the right side, must pair with a matching 0 supertile, for example.

By introducing stages, geometry, and applying a binary counter in our temperature 1 tile assembly model, we ensure a powerful enough tileset to construct n x n squares consistently with

a tiletype complexity of O(*logn*) and O(1) stages. However, before we explore our specific tileset, we need to consider the obstacles our construction must overcome.

Enforcing Error Free Constructions

We must ensure that whenever we expect two supertiles or two supertile strings to adhere adjacent to one another, no other supertile or supertile string can also adhere, thus disrupting the formation of the *n* x *n* square. This often results in slight increases in tiletype and stage complexity to prevent these types of mismatch.

Complete Coverage

It is important that in constructing *n* x *n* squares we ensure that our construction is complete, by which we mean that there are no gaps or spaces at any point in our square. This requires at times extra filler tiles, as well as extra stages needed to ensure that these filler tiles interact predictably with the rest of the tile assembly model.

Infinite Strings

Because we are implementing a tile assembly model at temperature 1, it is important to carefully consider how each tile will bond with the adjacent tiles, to ensure that infinite strings do not form and complicate or disrupt construction of the *n* x *n* square. As one example, the binary strings, which represent a single bit construction in our *h* x *n* rectangles, will eventually be joined together into a single rectangle using *top glues*, a small supertile with East and West glues that attaches to the top of each binary string. If we released top glues as complete supertiles into our solution, then before they even had a chance to attach to our binary strings, they might bond with one another, since there is no need to consider the overall strength of glue bonds in a temperature 1 system. These bonded top glues might form infinite strings, or long unending lines of top glues that would disrupt and ruin our planned *n* x *n* square. To account for infinite

strings in this one example, we introduce only half of our supertile in one stage, so it bonds with the binary strings, but not yet binding the binary strings to each other. Only after spare top glues are washed away does the second part of the supertile attach to the binary strings, forcing them to self-assemble into rectangles without creating disruptive infinite strings.

While there are other considerations and obstacles to overcome in developing a specific tileset, these two concerns often require careful consideration of the exact order and design of each tiletype, and also each stage.

CHAPTER III

RESULTS

In this chapter we describe in detail how precisely we construct exact, complete n x n rectangles, beginning with the construction of key $logn$ x n rectangles, while maintaining O($logn$) tile complexity, without using more than a constant number of stages, O(1), as shown in table 1.

Tile Assembly Model	Stage Complexity	Tile Complexity
Original *sTAM*	O($logn$)	O(1)
Modified *sTAM*	O(1)	O($logn$)

Table 1: Our Result

Most of the chapter will focus on the specific tile types needed to achieve this result, and will provide detailed instructions on how these tile types interact and assemble from supertiles into squares. While simplicity is key to elegant and efficient algorithms, the complexity of a complete tile set reflects the challenge in overcoming obstacles to guarantee error-free constructions. Special attention will be paid to how the design of a tile type helps ensure error-free construction of our $logn$ x n rectangles, and our final n x n squares.

Constructing *logn* x *n* Rectangles

To achieve our results efficiently, we construct our rectangles using a binary counter. To implement the binary counter in our model, we create binary counter supertiles, as our primary tile type within our overall tile set. A supertile is a construction composed of individual tiles that operates like a tile to facilitate the construction of still more complex structures.

Binary Counter Supertiles

The purpose of our binary counter supertile is to create binary strings that we will use to construct various rectangles of size *logm* x *m*, such that the sum of the lengths of these rectangles is equal to *n*, allowing us to create a combined rectangle of size *logn* x *n*. Our binary counter works through nondeterministic self assembly by allowing each bit that attaches to determine the outcome of the individual binary string. Given sufficient time, statistically we should see sufficient copies of each individual binary string, such that entire binary counters can form. The binary counter works generally by either 'carrying' a value, represented by the '10' binary supertile, or not-carrying a value, represented by the three other binary supertile types, as shown in figure 2. When the value is carried up to the maximum binary string height, the binary counter terminates. There will be exactly $4(logn - 2) + 2$ or $4logn - 6$ tiletypes, because the first level, l_0 will have only two tiletypes, a single carry binary supertile c_0 and a single non-carry binary supertile nc_0. Additionally, the top level l_h will have only stop supertiles, which are introduced below, since growth of the binary strings will be restricted to at most a height h, where $h \leq logn$.

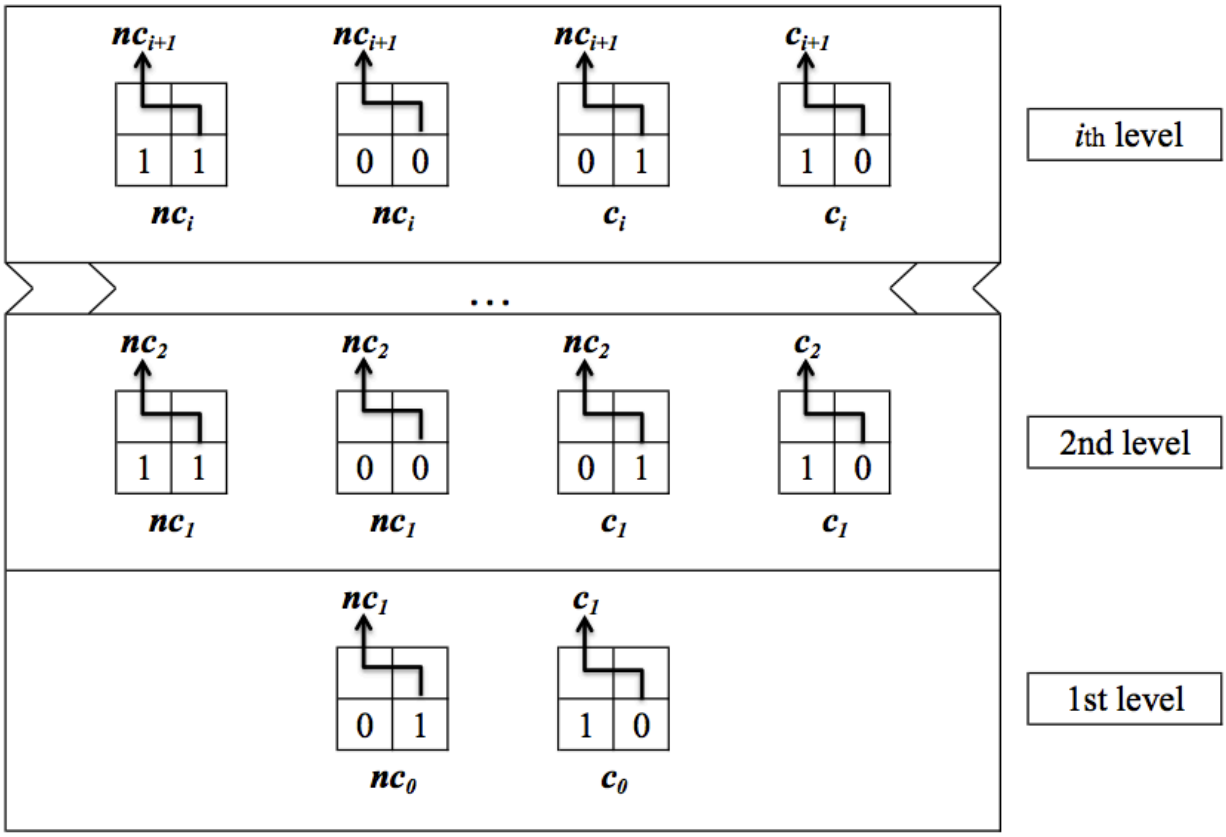

Figure 2: Binary Counter

Stop Supertiles

Our stop supertiles are designated S_i, where i represents each level that requires a set of stop supertiles. The purpose of our stop supertiles is to restrict the growth of our binary strings so that we create a size $logm$ x m rectangle for each m_i, a 2^l length rectangle representing a 1 bit in the binary sequence of n. The levels that require a stop supertile set are shown in figure 3 for the example case $n = 45$.

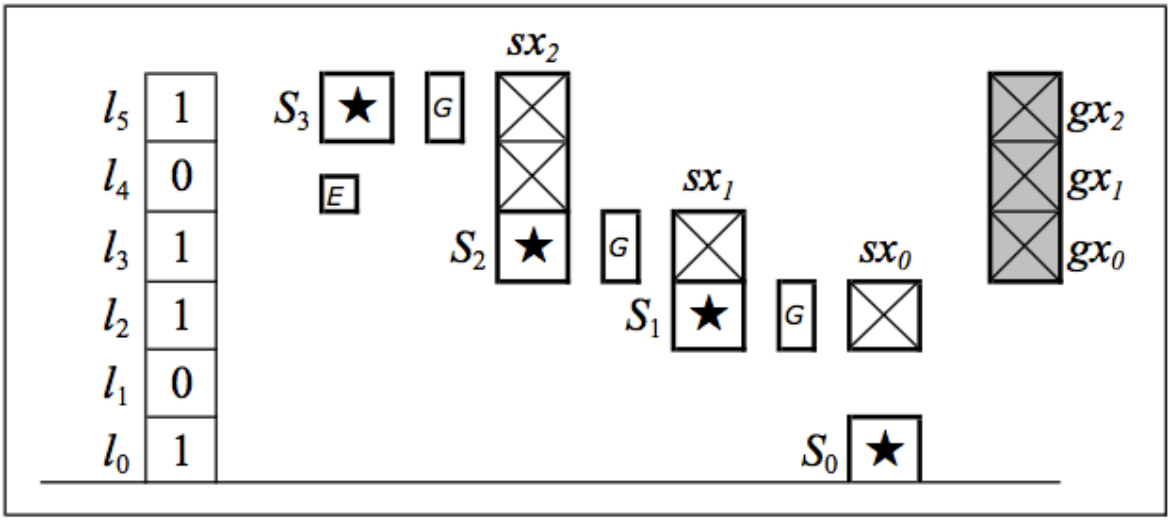

Figure 3: Tiletypes For n = 45

The total stop supertiles needed to construct our rectangles and square have a tiletype complexity of no more than O(*logn*), and therefore these tiles allow us to maintain an overall tiletype complexity of no more than O(*logn*). Generally, a stop supertile set resembles a set of binary supertiles, including equivalent geometry and bit markings, as shown in figure 4.

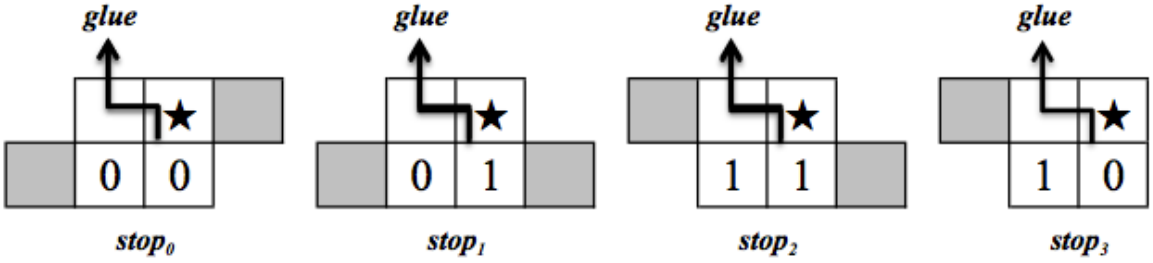

Figure 4: Stop Supertiles

However, growth beyond the stop supertile set is restricted to extension supertiles, introduced below. Additionally, the 'carry' stop supertile has no East-face geometry, due to the fact that a special glue supertile will be used to connect this stop supertile to the adjacent specific extension supertile. To be exact, each m_i has a single 'carry' stop supertile that will connect to the corresponding m_{i-1} specific extension supertile.

There will be at most $4logn - 5$ stop supertiles, which is the case when $n = 2^i - 1$, where i is any nonnegative integer. There will be at least 4 stop supertiles, which is the case when $n = 2^i$. In the binary representation of n, every bit set to 1 represents a level of our binary counter that will require its own rectangle, some l_i which will require 4 stop supertiles. The exceptions are for the first bit, which will require only a single stop supertile, since this rectangle is a 1 x 1 construction, and in the second bit, which will require only two stop supertiles, since this represents a 1 x 2 construction. For our example $n = 45$, the binary representation of 101101 means there will be four stop supertiles created for levels l_2, l_3, and l_5, and one stop supertile created for level l_0, for a total of 13 stop supertiles.

Extension Supertiles

The purpose of the extension supertiles is to ensure that after all the rectangles of size *logm x m* are created, they extend to a constant height of h, or about *logn*, so that our final rectangle is a consistent size *h x n*, or *logn x n*. There are two types of extension supertiles.

Specific Extension Supertiles. Specific extension supertiles, designated sx_i, are used to connect a length m_i rectangle to a length m_{i+1} rectangle, by extending the height of the first rectangle from $logm_i$ to $logm_{i+1}$, as shown in figure 3. These extension supertiles must be specific to each m_i rectangle to ensure that the rectangles attach in a predictable order to form perfect *logn x n* rectangles in our final construction. Additionally, for every specific extension

supertile type, there is a twin supertile type that attaches to the right-most, or East-most binary supertile string, as shown in figure 5. This binary supertile string represents the string of all 1 bits, and can be considered the 'carry' binary supertile string. While the addition of an entire twin set of extension supertiles seems like an added layer of complexity, it is essential to distinguishing the right-most or East side of each rectangle m_i and is essential to completing our final *logn x n* rectangle. There will be at most $2logn - 1$ specific extension supertiles, where $n = 2^i - 1$, but in the case where $n = 2^i$ there will be no extension supertiles at all, because there will only be one rectangle in the series.

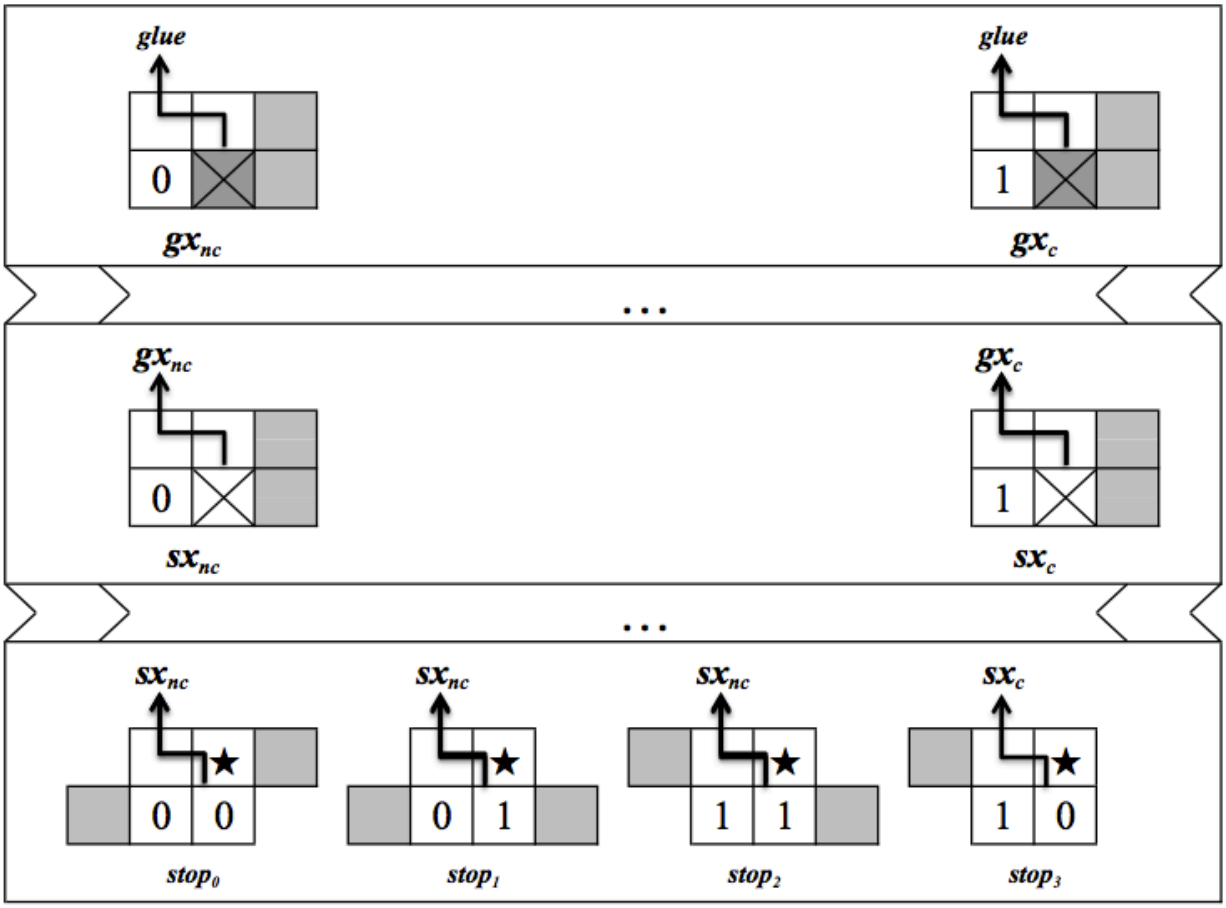

Figure 5: Stop and Extension Supertile Sets

Generic Extension Supertiles. Generic extension supertiles, designated gx_i, are used to extend a height $logm_i$ rectangle that has already been extended to height $logm_{i+1}$ by specific extension supertiles, to a final string height of about $logn$, or $logm_{\text{h}}$. These generic extension supertiles are the same for each rectangle m in our series. Also, like the specific extension supertile, there is a twin generic extension supertile that attaches to the right-most or East-most binary supertile string in each rectangle m_i, the 'carry' string. There will be at most $2logn - 2$ generic extension supertiles, where $n = 2^i - 1$, but in the case where $n = 2^i$ there will be no extension supertiles at all, because there will only be one rectangle in the series. In total, therefore, the maximum extension supertiles will be $4logn - 5$ and the minimum 0.

Both specific extension supertiles and generic extension supertiles, as described above, contain similar geometry. To both prevent binary string mismatches, and also to ensure a complete rectangle and square, there is a geometry unique to extension supertiles, which consists of an all East-face geometry, and no geometry tiles on the West-face of each supertile, as shown in figure 5. These supertiles, when added to the binary and stop supertiles will create our complete binary strings, shown in figure 6, which will ultimately connect to one another through complimentary geometry as well as through matching glue types.

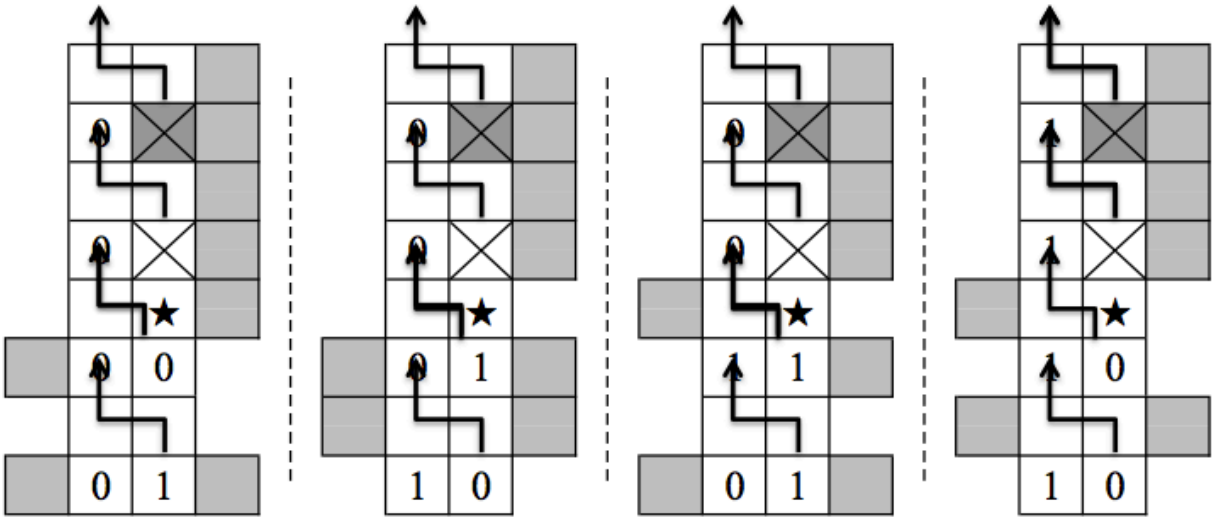

Figure 6: Complete Binary Supertile Strings

Top Glue Supertile

To bind our binary supertile strings into rectangles, we will attach a pair of top glue supertile types to the North side, or top, of each level l_h supertile, as shown in figure 7.

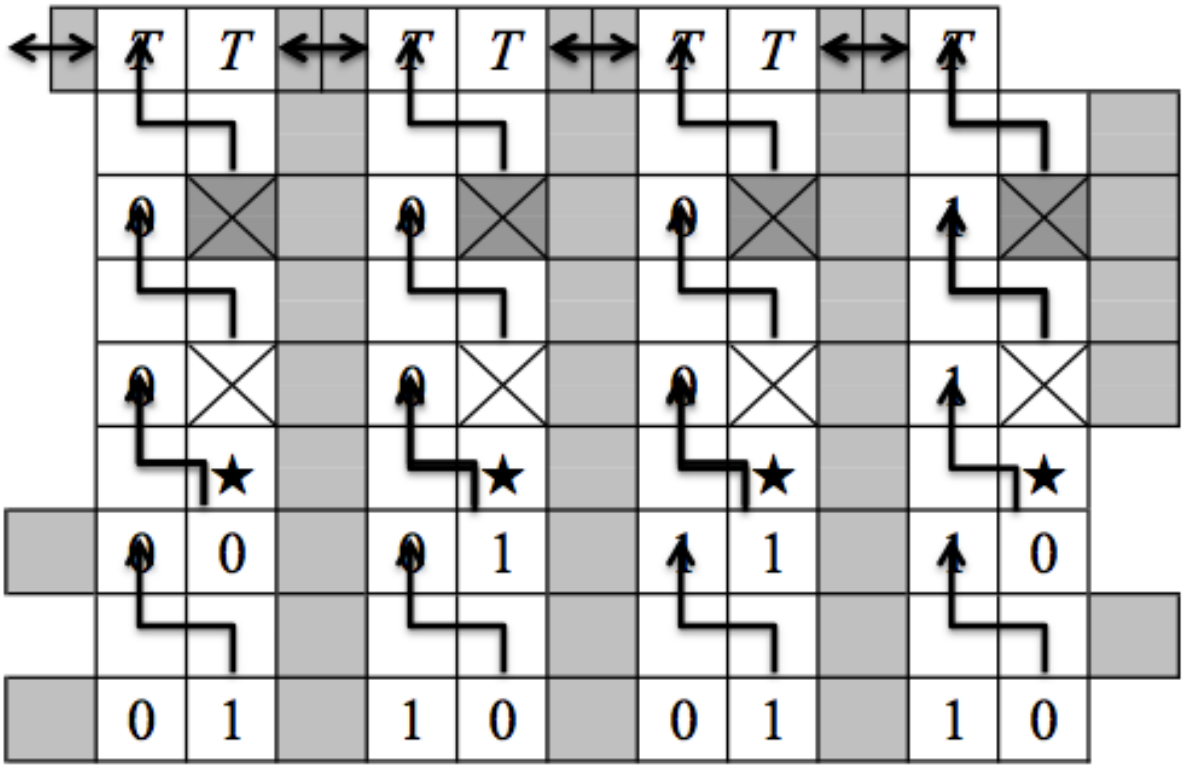

Figure 7: Assembled Supertile Strings

Supertiles at this level all have two North side glues, one meant to attach to a top left (West) glue supertile, designated T_W, and one to a top right (East) glue supertile, designated T_E. The reason for two different glue supertile types, is that if we release them both at once, they might bond immediately into infinite strings, and float around or otherwise destroy our planned n x n squares. Instead, by allowing either type to bond first to each binary supertile string, and then rinsing away the remaining glue supertiles, we avoid this problem.

Additionally, a third top glue supertile type (East-most), designated T_{E+}, will be used specifically to assist in connecting all m_i rectangles together into a single *logn x n* rectangle. This top glue supertile is a right (East) supertile type. It is designed solely to attach to two types of level l_h supertiles, the far-right, or East-most stop supertile for m_h, and the East-most extension supertile for each m_i, where $i < h$, whether specific or generic for each rectangle. This third top

glue supertile type will be released in the second to last stage. It is the absence of the third top glue supertile type prior to this stage that is as helpful as its eventual presence. That is because with only two top glue supertile types, rectangles might adhere out of order. In reality, this third top glue supertile type is not even essential to binding adjacent m_i rectangles. It is really just essential for creating a complete *logn x n* rectangle, and its glue properties can only help strengthen the overall rectangle. There are exactly 3 top glue supertile types, shown in figure 8.

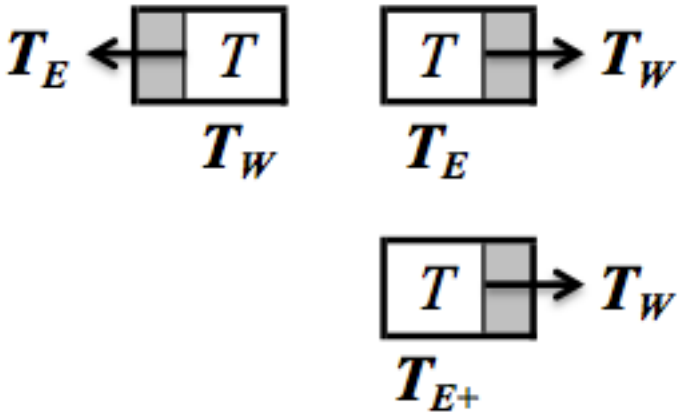

Figure 8: Top Glue Supertiles

East Geometry Filler Tile

The East geometry filler tile, designated *E*, is used to guarantee that our final *n* x *n* square is complete. The East geometry filler tile binds to the unused 'teeth' or restrictive geometry on the binary supertile string located on the East, or far-right side of each *m* rectangle, prior to the binding of an m_i rectangle to rectangle m_{i+1}. East geometry filler tiles are only needed for levels *l* where there are no stop supertiles. In other words, in every level which is represented by a 0 bit, as shown in figure 3. This is for the specific case where a binary supertile is adjacent to an extension tile, and therefore there is a gap between the two, as shown in figure 9. There will be

at most *logn* – 1 East geometry filler tiletypes, in the case where $n = 2^i$. In the specific case where $n = 2^i - 1$, there will be no geometry filler tiles, because there are no 0 bit levels.

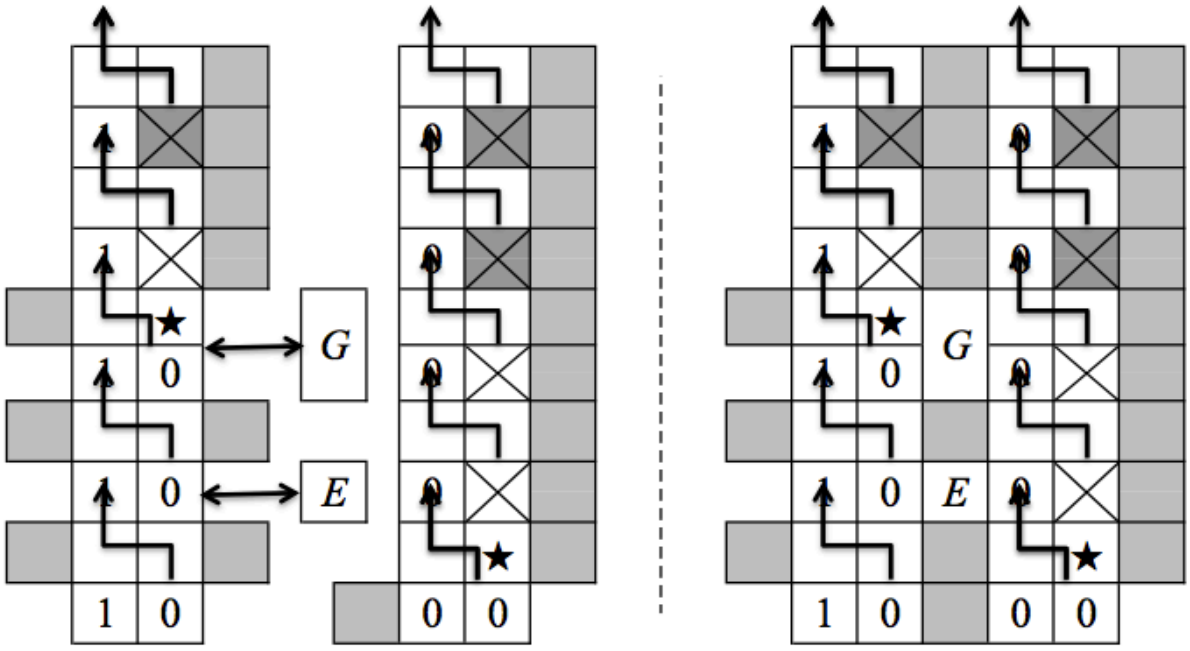

Figure 9: East and Glue Tiles

Glue Supertile

The glue supertile, designated *G*, is designed to connect every rectangle m_i to the subsequent rectangle m_{i+1}. Specifically, the glue supertile binds the far-right or East side stop supertile of rectangle m_{i+1} to the far-left or West side specific extension supertile of rectangle m_i, as shown in figure 9. There is no restrictive geometry between these two supertiles, and so the glue supertile serves as both the restrictive geometry, preventing same-size rectangles from binding in the next stage, and also a binding supertile, adhering corresponding binary supertile strings from adjacent rectangles. There is exactly one glue supertile type for each combination of m_i and m_{i+1}, which is important because otherwise two mismatched rectangles might attach

adjacent to each other. There are at most *logn* – 1 glue supertile types, in the case where $n = 2^i - 1$, and in the case where $n = 2^i$ there will be no glue supertiles.

Final Geometry Filler Tiles

If all previous tiletypes are applied, we will have a *logn* x *n* size rectangle, with only a few 'teeth' or restrictive geometry, on either side of the rectangle, like a jagged edge. To smooth this edge on either side requires adding small geometry filler tiles, on the left-most binary supertile string, or rather the bit sequence representing 0, and also on the right-most binary supertile string, or the bit sequence representing m_1, located at the *n*th position in our *logn* x *n* size rectangle.

If we are ending our construction with a *logn* x *n* size rectangle, we can apply a final pair of Final geometry filler tiles, one East and one West, designated by F_E and F_W. There would only need to be exactly two tiletypes. As will be seen in the next chapter, the final geometry filler tile becomes a supertile as part of our plan to construct an *n* x *n* square.

Stages Needed To Complete *logn* x *n* Rectangles

To create a complete *logn* x *n* rectangle requires a total of 6 stages, which achieves a stage complexity of O(1), while maintaining a tile complexity of O(*logn*), as shown in table 2. In the first stage we mix all binary, stop, and extension supertiles, which form into binary strings. In the second and third stages we introduce West and East top glue supertiles, respectively, which allows our binary strings to form into *logn* x *m* rectangles. In our fourth stage, we introduce the East geometry filler tiles, which prepare for stage five, the addition of glue supertiles, which bind our *logn* x *m* rectangle types into a single *logn* x *n* length rectangle. To guarantee a complete rectangle, in stage six we introduce East-most top glues, and our Final geometry filler tiles. The only stage that will change when we transition into the creation of our

final construction, *n* x *n* squares is stage six, when Final geometry filler tiles become supertiles that allow our *logn* x *n* rectangles to bind in parallel to a base rectangle, described below.

Stage and Tile Complexity for *logn* x *n* Rectangles

Stage	Tile types introduced	Tile Complexity
1	binary supertiles, stop supertiles, extension supertiles	O(*logn*)
2	West top glue supertiles, designated T_W	O(1)
3	East top glue supertiles, designated T_E	O(1)
4	East geometry filler tiles, designated *E*	O(1)
5	Glue supertiles, designated *G*	O(*logn*)
6	East-most top glues, Final geometry filler tiles	O(1)
Stage Complexity: O(1)		Tile Complexity: O(*logn*)

Table 2: Stage and Tile Complexity for logn x n Rectangles

Constructing *n* x *n* Squares

Once we establish that we can create a *logn* x *n* rectangle with tile complexity O(*logn*) and stage complexity of O(1), we can approach the final construction of a complete *n* x *n* square. If we lay our *logn* height rectangles in parallel stacks, we can reach a near *n* x *n* shape using exactly *n* / *logn* rectangles. However, without some means of binding these parallel rectangles in position, we cannot create an actual *n* x *n* rectangle. The challenge then is to create an *n* length line with some type of binding point at intervals of *logn*, and to create this line without using more than O(1) additional stages, and no more than O(*logn*) tile types. To do this, we actually construct a separate type of *logn* x *n* rectangle that consists of binary supertiles, like our previous

logn x *n* rectangle, but instead of creating binary supertile strings of size 1 x *logn*, we create binary supertile blocks of size *logn* x *logn*.

Constructing *logn* x *n* Rectangles with *logn* x *logn* Supertile Blocks

To construct a base rectangle of size *logn* x *n*, using *logn* x *logn* supertile blocks, we begin with a binary supertile set needed to create a rectangle of length 2^h, where *h* represents the largest possible bit in the binary sequence with *logn* bits. In our example situation where $n = 45$, $h = 5$, and $2^h = 32$. Since each binary supertile string will actually be of length *logn* then our rectangle if fully constructed would be of length $2^h logn$, or in the case of 45, (32)5 or 160. Because the length of this new rectangle exceeds *n*, we can create a special terminal string that will attach and prevent the full construction of the binary sequence.

In the previous construction, we attached multiple, complete rectangles into a single rectangle. Here, we cut short a rectangle to fit our purpose. In this specific case, we will stop our construction after a number of supertile blocks equal to $n / logn$, which in this case of $n = 45$ would be 9. However, because the terminal string itself is a supertile, it is important to consider that one string as part of our length, so in effect we set $n - 1$ as our rectangle length. In this specific case we will then only have 8 *logn* x *logn* size supertiles in our base rectangle, because $n - 1 / logn = 8$. This also corresponds to the number of parallel rectangles that will ultimately attach to form our final *n* x *n* square. In total we will have one base rectangle and 8 parallel rectangles attached to the base in our complete *n* x *n* square, for the example case of $n = 45$.

Binary Supertiles. First, we consider our binary supertiles, which is the same exact tile set used in the previous construction. In this case, we do not need stop supertiles, or extension supertiles, because we will have only one partial rectangle, instead of a series of connected rectangles. However, in addition to our binary supertile set, we will have an East extension

supertile, which simply extends each binary supertile string to the right, or East by *logn* – 1 supertiles.

East Extension Supertiles. East extension supertiles, designated ex_i, have exactly 2(*logn* – 1) tiletypes. At each position in the extension sequence, there is a 0 bit extension tiletype, and a 1 bit extension tiletype, shown in figure 10. The East extension supertiles will bind directly to the binary supertile strings, and to each other in sequence, as shown in figure 11. Entire supertile blocks will still use top glues to bind to one another.

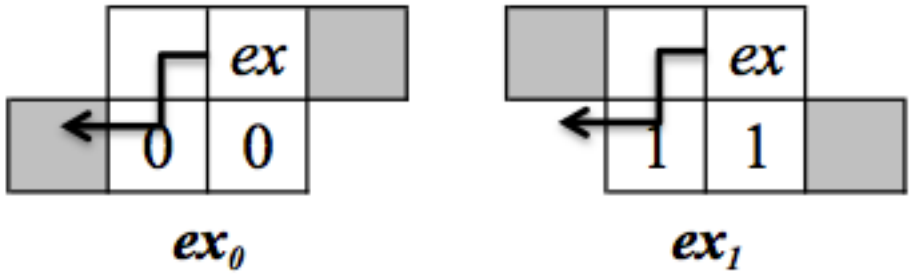

Figure 10: Extension Supertiles

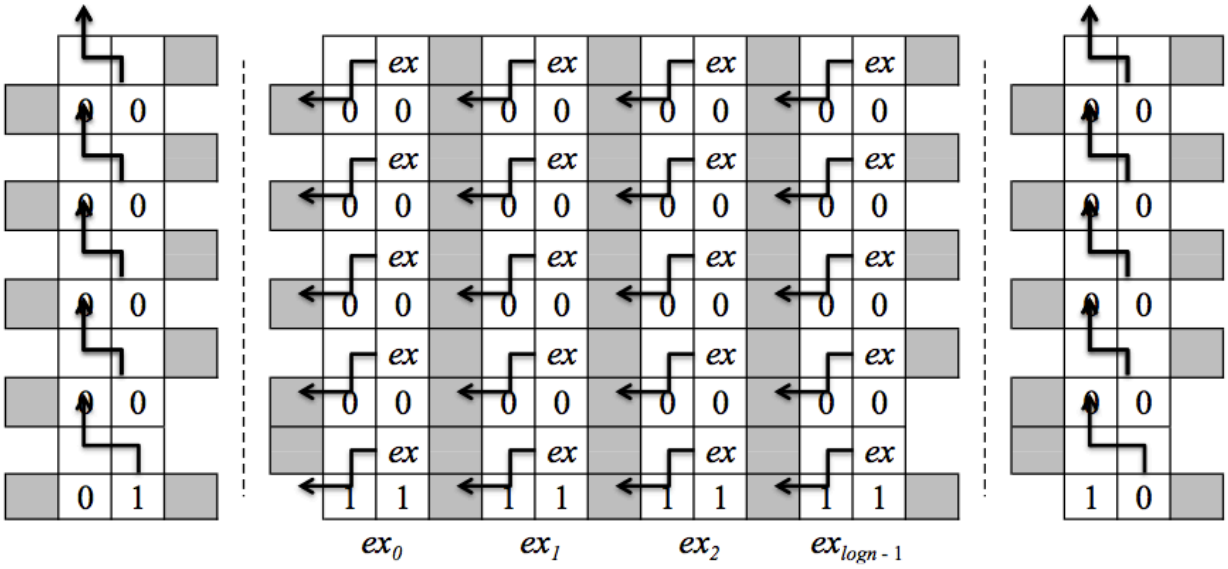

Figure 11: logn x logn Supertile Blocks

Top Glues. We will also have a set of top glues, similar to our previous construction. There will be exactly three top glue tiletypes. As before, we will release only one, this time right-side, or East top glue, allowing it to bond first, before releasing other top glues. There is a second West top glue, and a third top glue specific to the terminating supertile string. Key to the entire rectangle is the terminating supertile string, which is an engineered stop supertile string that restricts growth beyond length *n* in the rectangle.

Terminating Supertile String. To create a terminating supertile string requires exactly *logn* terminating supertile types. The terminating supertile string will be attached to the right binary supertile string because it will bond with a distinct terminal top glue, designated by T_T, released after the East top glue, so that it attaches before any other binary supertile string can possibly attach, as shown in figure 12.

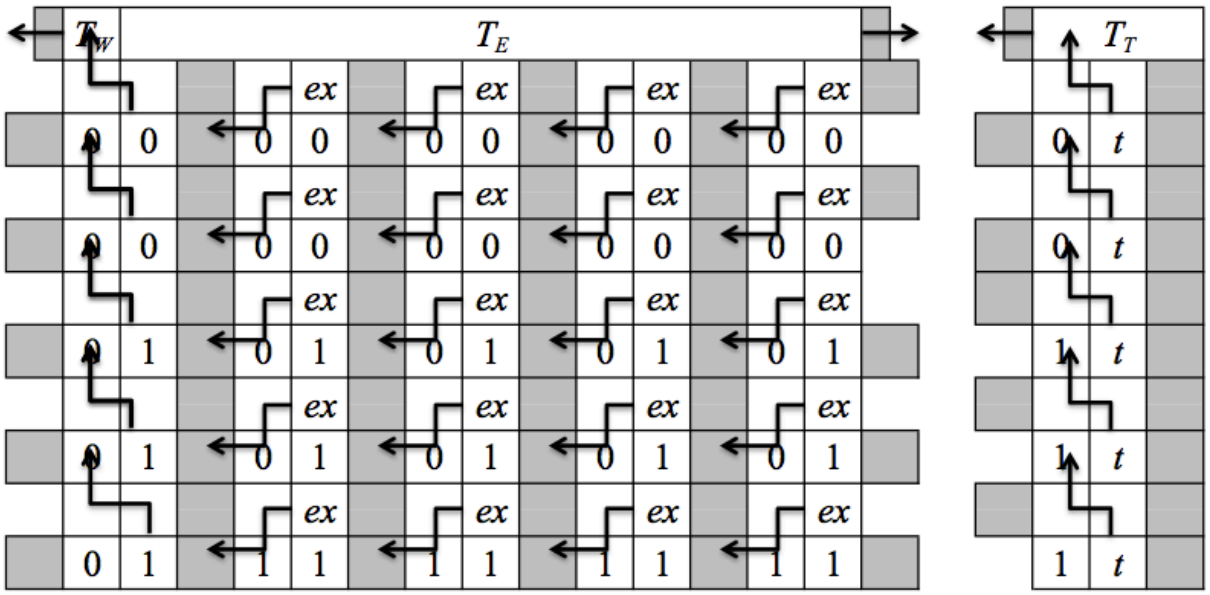

Figure 12: Assembled Supertile Block with Terminating String

When the West top glues are released, no binary supertile strings will be able to attach to the main construction past the terminating supertile string. Because there will be a remainder length

(n mod $logn$) – 1 we will also introduce a special East extension supertile to ensure our square is complete and measures n x n. The special East extension supertile can attach directly to the terminating supertile string. There will be at most $logn$ – 1 East extension supertile types. Lastly, we release final geometry filler tiles, which fill spaces between the gaps in the 'teeth' or restrictive geometry for the left-most, or West-most binary supertile block, representing the binary string for the value 0. There is exactly one type of filler tile in this construction.

Stages Needed to Complete New *logn* x *n* Rectangles. With these tiletypes, we can create a second type of *logn x n* rectangle, which will serve as the base, and binding site for our parallel construction of *logn x n* rectangles as described previously. The stages needed to create this second *logn x n* rectangle are shown in table 3. In the first stage we introduce our binary and terminating supertiles, allowing these supertile strings to construct in parallel. In the second stage, we turn our binary supertile strings into *logn* x *logn* blocks by introducing the East extension supertiles. In the third and fourth stages, we introduce East top glue supertiles, and terminal top glue supertiles respectively, allowing the binary and terminating supertile strings to attach at the string that represents the value n – 1 / $logn$. In the fifth stage, we introduce West top glue supertiles and East extension supertiles, allowing the rest of the binary strings to attach into a single rectangle, and extending that rectangle so the final measure is *logn* x *n*. Lastly, in stage six we add a final geometry filler tile which will fill in the gaps on the left-most, or West side of our rectangle. None of these stages change in the context of constructing an n x n square, except that some of the supertiles mentioned have glues which permit the earlier *logn* x *n* rectangles to attach at intervals of *logn* along our base rectangle length.

Stage and Tile Complexity for *logn* x *n* Rectangles using *logn* x *logn* Blocks		
Stage	**Tile types introduced**	**Tile Complexity**
1	binary supertiles, terminating supertiles	O(*logn*)
2	East extension supertiles, designated *ex*	O(*logn*)
3	East top glue supertiles, designated T_E	O(1)
4	terminal top glue supertiles, designated T_T	O(1)
5	West top glue supertiles, East extension supertiles	O(*logn*)
6	final geometry filler tile	O(1)
Stage Complexity: O(1)		Tile Complexity: O(*logn*)

Table 3: Complexity for Rectangles using logn x logn Blocks

Finishing the *n* x *n* Square

To finish our square of size *n* x *n* then, we need only add a few small adjustments. In our original rectangle, instead of simply having East filler tiles, we will have an entire East filler supertile, that includes the glues necessary to bind the original rectangle to the 'top' of the binary supertile string in the second rectangle, constructed out of size *logn* x *logn* supertiles. Both types of *logn* x *n* rectangles, because they will have different orientation (N, E, S, W), can be constructed in parallel stages, so long as we ensure there is no prematurely binding. Specifically, the final *n* x *n* square requires only seven total stages, because stages 1—6 can be carried out in parallel for both rectangle types. The last stage, stage seven, completes our *n* x *n* square by extending the rest of the way to *n* after considering the amount *n* mod *logn*. We can create a final filler tile that attaches in a sequence equal to *n* mod *logn* and therefore has no more than *logn* tiletypes. The maximum is *logn* – 1 and the minimum 0. In this way we use two types of

rectangles, each with stage complexity of O(1) and tile complexity of no more than O(*logn*), to construct an *n* x *n* rectangle also with stage complexity of O(1) and tile complexity of O(*logn*).

CHAPTER IV

CONCLUSIONS AND FUTURE WORK

Improvements to the Modified Staged Tile Assembly Model

First, it is important to consider areas where our modified staged tile assembly model can be improved to create more stable structures. We create a complete, exact n x n square in O(1) stages with a tile complexity of O($logn$). Is it possible to create a fully-connected square of the same dimensions without a reduction in tile complexity and stage complexity? Full connectivity means that each tile in our construction is bonded with each adjacent tile. In our current model, we simply ensure that each tile is reliably bonded to at least one adjacent tile. In a real laboratory, without additional bonding between tiles, our construction might shift and lead to unforeseen errors. Thus a fully connected model might be a significant improvement.

Second, while the modified staged tile assembly model assumes that partial constructions, or incomplete components at each stage that do not fully assembly into the desired construction, are washed away, a more robust model might ensure there were no partial constructions. While in a theoretical model, we can assume partial constructions are 'washed' away, in a real laboratory that process might both prove problematic, as well as costly in terms of effort and resources. It is actually an open-question to determine whether or not it is possible to construct a modified staged tile assembly model that is free of partial constructions, that still adheres to our tile complexity constraints of O(1) stages and O($logn$) tiletypes. However, even without a model that is free of all partial constructions, improvements could be made. The second type of $logn$ x n rectangle, constructed out of $logn$ x $logn$ blocks, terminates at a

relatively early stage, in this case at $n = 45$ for a rectangle that if fully assembled would achieve a length of $n = 160$. That means the 'partial' construction in this circumstance is larger than the terminal construction. By metaphorical comparison, that would be the equivalent of throwing away 75 percent of a chicken to create a chicken nugget. This could easily be improved by allowing the bit size of the *logn* x *n* rectangle to vary, instead of automatically setting it to *h*, which is the largest bit in the binary representation of *n*, such that $h \leq logn$. In the example of $n = 45$, we could use a 16 bit binary counter instead to assemble into a (16)*logn* length rectangle, for a total length of 80, reducing the overall construction by 50 percent, and reducing the waste, as represented by the washed away partial construction, by 67 percent.

Lastly, in order to solve a problem in creating an efficient and algorithmically inexpensive way to bind each original *logn* x *n* rectangle in parallel, we constructed an entirely different form of *logn* x *n* rectangle, constructed out of supertiles that measure *logn* x *logn*. It is possible that in its simplicity, this problem-solving rectangle might actually be more efficient than the original rectangles we created. We could easily create an *n* x *n* square using only the second type of rectangle. In searching for a solution to a problem within our model, we identified a very different model entirely, one that terminates a single binary counter, instead of appending multiple rectangles into a larger rectangle of exactly length *n*. Posing the question, which rectangle is more efficient, would involve evaluating tradeoffs between tiletype complexity—the second rectangle is less complex—versus inefficiency through increased partial constructions. To compare, the first rectangle has no 'waste' or partial constructions, except for those that simply do not have enough time to assemble into terminal shapes.

Determining Comparable Costs for Various Tile Assembly Models

In the original tile assembly model (TAM) researchers quickly advanced toward the use of temperature 2 systems, because of the improvement in tile complexity, from an original temperature 1 tile complexity of $O(n^2)$. However, results such as the original staged tile assembly model (sTAM) and our modified staged tile assembly model, can achieve constructions with much improved tile complexity, including the original result of O(1) tile complexity, using O(*logn*) stages, and our new result of O(*logn*) tile complexity with O(1) stages. Each of these results presents options for researchers to consider when determining which tile assembly model is most adaptable to real-world conditions.

Furthermore, as researchers in biological laboratories begin implementing tile assembly systems under real conditions, it is clear that tile complexity is not the only real measure of efficiency and reliability. One area of future research that is key is to find some measure of determining a comparable cost in terms of real resources between one tile assembly model and another. What is lost in terms of cost and reliability in adopting a temperature 2 tile assembly model over a temperature 1 tile assembly model? How does the tradeoff in improved tile complexity for temperature 2 systems offset these costs? Is a staged tile assembly model with O(*logn*) tile complexity cheaper to use for constructions, and potentially more reliable, than comparable temperature 2 models?

Encouraging Ethical Considerations for DNA Self-Assembly Research

While writers in the popular media are quick to pose questions of ethical consideration when writing about advanced science generally, it does not seem to be a practice currently en vogue or encouraged by computer science researchers in the field of DNA self-assembly. In my literature review, I came across not one single paper that addresses potential areas of ethical

concern with respect to research involving the manipulation of DNA, and specifically the power of DNA, once manipulated and configured into real-world Wang tiles, to self-assemble into nanoscale objects. Perhaps because algorithmic design and analysis is an abstract science, computer science researchers in the field of DNA self-assembly do not seem to consider in their works the potential impacts on humanity, both short-term, or long-term, from the actual construction of nanoscale circuits or, as we are seeing already, primitive nanoscale computers.

It is probable, that as the field of DNA Self-Assembly moves from a more theoretical study, lodged in the computer sciences, to a study of practical applications, lodged in the biological sciences, we will see a greater level of inquiry dedicated to determining the exact ends of the research, and whether those ends are ethically just. I know this may be unfamiliar territory for researchers, but without knowledgeable judgment of ethical questions by researchers, those questions will necessarily be left to political, or legal actors, whose familiarity with the actual science may be significantly less. In short, if we do not judge the ethical consequences of our own work, it may be judged by those who are far less informed, yet we may still bear the burden of those judgments in the form of potential research and funding restrictions. For these reasons, I fully encourage the development of research lines that expend energy evaluating the ethical considerations for DNA Self-Assembly, with vigor equal to those who study, for example, the robustness of particular DNA Self-Assembly models. While there are leaders in various subfields of DNA Self-Assembly, there has yet to emerge a leader in the area of Ethics of DNA Self-Assembly, and I hope that one day this niche is filled.

REFERENCES

[1] Leonard Adleman, Qi Cheng, Ashish Goel, and Ming-Deh Huang, *Running time and program size for self-assembled squares*, STOC '01: Proceedings of the thirty-third annual ACM Symposium on Theory of Computing (New York, NY, USA), ACM, 2001, pp. 740-748.

[2] Adleman, L.M., Cheng, Q., Goel, A. and Huang, M-D., Kempe, D., de Espanés, P.M. and Rothemund, P.W.K. *Combinatorial optimization problems in self-assembly*. In Proceedings of STOC (2002), 23–32.

[3] Gagan Aggarwal, Michael H. Goldwasser, Ming-Yang Kao, and Robert T. Schweller, *Complexities for generalized models of self-assembly*, Proceedings of ACM-SIAM Symposium on Discrete Algorithms, 2004.

[4] Florent Becker, Ivan Rapaport, and Eric Remila, *Self-Assembling classes of shapes with a minimum number of tiles, and in optimal time*, Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 2006, pp. 45- 56.

[5] Cannon, S., Demaine, E.D., Demaine, M.L., Eisenstat, S., Patitz, M.J., Schweller, R.T., Summers, S.M. and Winslow, A., *Two hands are better than one (up to constant factors),* Technical Report 1201.1650, Computing Research Repository, 2012.

[6] Chandran, H., Gopalkrishnan, N. and Reif, J.H., *The tile complexity of linear assemblies*, In SIAM Journal on Computing 41, 4 (2012) 1051–1073. Preliminarly version appeared in ICALP (2009).

[7] Ho-Lin Chen, David Doty, and Shinnosuke Seki, *Program size and temperature in self-assembly*, ISAAC 2011: Proceedings of the 22nd International Symposium on Algorithms and Computation, (Yokohama, Japan, December 5-8, 2011), LNCS 7074, pp. 445-453.

[8] Qi Cheng, and Pablo Moisset Espanes. *Resolving two open problems in the self- assembly of squares*. Technical Report 03-793, University of Southern California, June 2003.

[9] Cook, M., Fu, Y., Schweller, R.T., *Temperature 1 Self-Assembly: Deterministic Assembly in 3D and Probabilistic Assembly in 2D*, Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2011). San Francisco, California, Jan. 2011

[10] Erik D. Demaine, Martin L. Demaine, Sandor P. Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T. Schweller, and Diane L. Souvaine, Staged self-assembly: nanomanufacture of arbitrary shapes with O(1) glues, Natural Computing 7 (2008), no. 3, 347-370.

[11] Demaine, E.D., Eisenstat, S., Ishaque, M. and Winslow, A., *One-dimensional staged self-assembly*, DNA (2011), 100–114.

[12] Doty, D., Lutz, J.H., Patitz, M.J., Schweller, R.T., Summers, M. and Woods, D. *The tile assembly model is intrinsically universal*. In Proceedings of FOCS (2012), to appear. IEEE.

[13] David Doty, *Theory of algorithmic self-assembly*, CACM 2012: Communications of the ACM 55(12): 78-88, 2012.

[14] David Doty, Matthew J. Patitz, and Scott M. Summers, *Limitations of self-assembly at temperature 1*, TCS 2011: Theoretical Computer Science 412(1-2):145-158, 2011. Special issue of invited papers from Complexity of Simple Programs workshop, Cork, Ireland, 2008.

[15] David Doty, *Applications of the theory of computation to nanoscale self-assembly*, Ph.D. Thesis, Iowa State University, 2009.

[16] Fogel D.B., *What is Evolutionary Computation?*, Spectrum, IEEE Press, February 2000, pp. 26-32.

[17] Fujibayashi, K., Hariadi, R., Park, S.H., Winfree, E. and Murata, S., *Toward reliable algorithmic self-assembly of DNA tiles: A fixed-width cellular automaton pattern*, Nano Letters 8, 7 (2007), 1791–1797.

[18] Gö ö s, M. and Orponen, P., *Synthesizing minimal tile sets for patterned DNA self-assembly*, DNA (2010), 71–82.

[19] Nicolas Gutierrez, DNA Staged Self-Assembly at temparature 1, Master's Thesis, University of Texas Pan-American, May 2010.

[20] Ho-Lin Chen, Qi Cheng, Ashish Goel, Ming-Deh Huang, and Pablo Moisset de Espanes, *Invadable self-assembly: Combining robustness with efficiency*, SODA: ACM-SIAM

Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms), 2004.

[21] Ho-Lin Chen and Ashish Goel, *Error free self-assembly with error prone tiles*, Proceedings of the 10th International Meeting on DNA Based Computers, 2004.

[22] Ming-Yang Kao and Robert T. Schweller, *Reducing tile complexity for self- assembly through temperature programming*, Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006), Miami, Florida, Jan. 2006, pp. 571-580, 2007.

[23] Kolmogorov, A.N., *Three approaches to the quantitative definition of 'information'*, Problems of Information Transmission 1:1 (1965), 7.

[24] Manuch, J., Stacho, L. and Stoll, C., *Two lower bounds for self-assemblies at Temperature 1*, Journal of Computational Biology 17, 6 (2010), 841–852.

[25] Patitz, M.J., Schweller, R.T. and Summers, S.M., *Exact shapes and Turing universality at Temperature 1 with a single negative glue*, DNA (2011), 175–189.

[26] John Reif, Sudheer Sahu, and Peng Yin, *Compact error-resilient computational DNA tiling assemblies*, DNA: International Workshop on DNA-Based Computers, LNCS, 2004.

[27] Paul W. K. Rothemund and Erik Winfree, *The program-size complexity of self-assembled squares (extended abstract)*, STOC 2000: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, 2000, pp. 459–468.

[28] Paul W. K. Rothemund, *Theory and experiments in algorithmic self-assembly*, Ph.D. thesis, University of South California, December 2001.

[29] Nadrian C. Seeman, *Nucleic-acid junctions and lattices*, Journal of Theoretical Biology 99 (1982), 237-247.

[30] Soloveichik, D. and Winfree, E., *Complexity of self- assembled shapes*, SIAM Journal on Computing 36, 6 (2007), 1544–1569. Preliminary version appeared in DNA (2004).

[31] Turing, A.M., *On computable numbers, with an application to the Entscheidungsproblem*, In Proceedings of the London Mathematical Society (1936), 230–265.

[32] Hao Wang, *Proving theorems by pattern recognition – II*, The Bell System Technical Journal XL (1961), no. 1, 1–41.

[33] Hao Wang, *Dominoes and the AEA case of the decision problem*, Proceedings of the Symposium on Mathematical Theory of Automata (New York, 1962), Polytechnic Press of Polytechnic Inst. Of Brooklyn, Brooklyn, N.Y., 1963, pp. 23- 55.

[34] Erik Winfree, *Algorithmic self-assembly of DNA*, Ph.D. thesis, California Institute of Technology, June 1998.

[35] Erik Winfree and Renat Bekbolatov, *Proofreading tile sets: Error correction for algorithmic self-assembly*, DNA (Junghuei Chen and John H. Reif, eds.), Lecture Notes in Computer Science, vol. 2943, Springer, 2003, pp. 126-144.

[36] Erik Winfree, *Simulations of computing by self-assembly*, Tech. Report Caltech CSTR:1998.22, California Institute of Technology, 1998.

BIOGRAPHICAL SKETCH

Michael C. Barnes earned a Master of Science in Computer Science from the University of Texas – Pan American in May 2013. He earned a Bachelor of Arts in Political Science from Macalester College in May 2006. He taught seven years as an 8th Grade History and Math teacher at Carlos F. Truan Jr. High in Edcouch-Elsa ISD. His current mailing address is PO Box 1000, Edcouch, Texas 78538.