Hamming Codes in the Context of DNA Self Assembly

Tiyu Wang[*]
Stanford University

Aditya Mittal[†]
Stanford University

Kathryn Hymes[‡]
Stanford University

ABSTRACT

We construct and simulate a tiling system capable of detecting errors in a given (L, D) Hamming Code represented as a 'seed' column where L is the length of the code, and D is the number of data bits. Our tiling system is a novel system which has O(1) tiles built upon modifications of the existing binary counter tile system constructed by Goel et al. Using just 32 tiles this system self assembles into a $log(L-D) \times L$ rectangle at temperature 2 with only single strength glues. The correctness of each parity check from the Hamming Code is shown as one of two glues on the northern edge of the system. The robustness of the system is demonstrated via XGrow simulations.

Introduction

In our work we develop a Wang Tile Model for using Hamming Codes in the context of DNA Self Assembly. We see this as a promising potential mechanism for error correction and reduction in combinatorial tile systems. Our system has several advantages such as modularity, making it compatible with other models and blocks we build in the future that can be attached as inputs; it has a constant number of tile types so the tile types don't grow linearly with the size of the system; and is built upon the nice mathematical formulation of Hamming Codes which gives it more potential for future development. After formulating the system, we also simulate it in XGrow to demonstrate its robustness. Our work gives rise to some open problems we discuss at the end of the paper.

To create our system we begin with a seed column and a seed row. The seed column encodes the hamming code to be checked and the seed row is used to start off the system. The seed row is the same as that of the seed row of the binary counter with double strength glues replaced by single strength glues. In solution we construct the system at temperature 2. The replacement of the double strength glues by single strength glues ensures that the system will stop growing after exactly L layers, allowing for Hamming Codes that are not necessarily powers of 2.

Then we replace each of the tiles in the eight tile binary counter system with 4 tiles capable of transmitting parity information toward the next tile, and discriminating between taking into account or skipping a data bit depending on the parity bit being checked. The 32 tiles of the system constructed in this manner are demonstrated in Figure 1.

Our idea in developing this 32 tile system is to exploit the correspondence between the parity bit positions in the hamming code and the columns of the binary counter. This idea helps us avoid keeping

Tiyu Wang, Aditya Mittal, and Kathryn Hymes are M.S. students in Institute of Computational and Mathematical Engineering at Stanford University. The model for error detection using Hamming Codes presented in this paper was constructed for the CME352 class project offered by Ashish Goel during Winter 2008.
Contact: tiyuwang@stanford.edu [*] admittal@stanford.edu [†] kehymes@stanford.edu [‡]

track of parity and data bit positions in the hamming code separately. We replace each tile in the binary counter with 4 tiles as follows:

Replace the right glue with 2 types of glues, z_0 and z_1 that will transmit the parity of the tile on the right. The lower glue will transmit the parity information from the tile below to the tile above so the information passed along sequentially to the top of the system. We do this by adding the parity for the right and bottom glues and making the sum mod-2 the parity of the top glue. This is done for all the variations of the 1-tiles. In the case where the label of tile is one of the four variations of 0, then the top are assigned the same parity as the bottom tiles as these tiles are designated to ignore the parity of the bit they are encoding. The left tile will just replicate the parity portion of the right tile as this will transmit the Hamming code over to the next column to allow for the next parity check. This system will stop growing once all the parity checks are done after all the columns have been filled.

This system was simulated in XGrow and allowed no mismatches. At first we discovered that having double bonds in the binary counter base of our system gave rise to mismatches, but after the single bond modification mentioned earlier the problem was resolved, since an incorrect tile can never attach. With a double bond a tile can start growing a new row in the binary counter that does not necessarily match up with the right glue of the corresponding Hamming code bit.

Background

Tiling Systems

The process of self assembly is ubiquitous in nature by the spontaneous formation of molecular structures from the reactions of smaller components: atoms assemble into molecules, which can further aggregate and react to form crystals, larger biomolecules and organisms. DNA is a prominent molecular building block with a variety of physical properties that lends itself well to use in artificial self assembly. The important thing about using DNA is its combinatorial nature. This allows us to create a combinatorial model in theory and then try to implement it in practice. It also provides us with a means to develop a model for computation. The model we use in order to exploit the combinatorial nature of DNA is known as the Wang Tile Model after its inventor Hao Wang in 1961. The concept of tiling is an ancient concept which has been used in Mosaics, flooring, and many geometrical constructions. In his model Wang used tiles to construct a formal system in order to prove results about whether a plane could be filled with a certain set of tiles. “Winfree et al. have demonstrated the feasibility of creating molecular "tiles" made from DNA (deoxyribonucleic acid) that can act as Wang tiles. Mittal et al. have shown that these tiles can also be composed of peptide nucleic acid (PNA), a stable artificial mimic of DNA.” [9].

Given that Wang Tiles compose the basic elements for assembly we can start with a seed strand of DNA corresponding to a seed tile and build a tile system on it. The tiles attach by means of individual glues of a particular strength represented on each side of the tile. These glue connections must correlate with a specified system temperature in order to assure binding of a tile. For our model we will assume this temperature to be 2 and will discuss its implication in the Simulation section. From this seed tile, tile

attachments continue until a final stable tile assembly is formed. Natural DNA self assembly utilizes explicit mechanisms for the prevention and correction of errors. Developing efficient and robust error correction methods is even more critical for artificial self assembly with the goal of constructing large systems with fine precision.

Error Correction

Binding errors naturally occur in the process of assembly. Single glue attachments occur frequently in solution but are generally not stable due to temperature constraints. Under certain circumstances, mismatched tiles do bind and become incorporated into the overall tile structure. Erroneous tile attachment at a point of potential stable binding is known as a *growth error*, while premature binding of a tile at an inactive site of the assembly is referred to as *nucleation error*. In order to construct molecular models capable of computation, precision and reliability of assembly is of paramount importance. For this reason, active effort has been taken to accurately identify and correct erroneous binding.

Existing error correction and prevention methods can largely be characterized by two classes of approach. Identifying natural mechanisms of DNA self assembly and investigating their potential use in algorithmic setting comprises one class. Examples of leveraging a biological process in an artificial system include strand invasion and the use restriction and ligation enzymes.

A second approach takes inspiration from coding theory and implements a more combinatorial design in error correction methods. Coding theory offers a myriad of error correction schemes that have been formulated for use in information transmission. An obvious goal would be to port these methods for error detection and correction to tile systems and evaluate their effectiveness. One simple technique is data repetition. Winfree and Bekbolatov modeled this approach in the context of tile systems with the introduction of tile proof reading. In their formulation, an original n x m assembly is mapped to a larger, redundant system when each individual tile is replaced with a k x k block. Proofreading tiles have been shown to reliably correct growth errors but not nucleation errors.

Another error detection method is that invented by Goel and Chen known as Snake tiles. With their formulation, each tile in the original system corresponds to four tiles in an error-correcting system. All internal glues are chosen to be unique to a specific 4-tile block. The advantages of this system become apparent with an example of binding error. When a first error tile attaches with a weak binding strength at a site that is not currently active for growth, a second tile may attach thereby securing the first tile. At this point, even leveraging previous error detection methods, the error introduced is now secured in the assembly and tiles will incorrectly propagate in the original system. With the snake tile model, no other tiles can attach without additional nucleation errors. This prevents incorrect tile propagation even after k+1 nucleation or growth errors.

Hamming Codes

Suppose one wants to detect whether there is a one bit error in a string of 0's and 1's. A commonly known mechanism to do this is to add an additional bit which describes the **parity** of the string, i.e. whether the sum of the elements of the string is even or odd. Hamming codes are an extension of this

idea of parity to add more bits to a string to detect and possibly even correct errors that may be in a string of 0's and 1's. Parity bits, Two-out of Five Codes, and Repetition are examples of error detection mechanisms predating Hamming Codes.

As we develop the field of DNA computing, it is a necessity to determine errors, in order to build robust computing systems. While there are some existing models for error detection and correction built on the Wang tile model such as the proof reading tiles and snake tiles described previously, these methods have high growth rates and are not easily extended to novel systems. Given this, we decided to use the well understood mathematical formulation of Hamming Codes in order to detect errors. Unlike the other known methods, we have not yet shown how to correct or reduce errors, only error detection given a Hamming Code. This is not to say that we will not be able to correct errors, as Hamming Codes have an established mathematical formulation that is also useful for error correction. Investigating this will be a future area of work. Before this, the next step will be to develop ways to generate the Hamming Code that acts as input to our error detection TAK model. First let us describe the construction of Hamming Codes.

Hamming Codes are linear error detection and correction codes that were first prescribed by Richard Hamming. In order to construct a Hamming code from a string of 0's and 1's we begin by assigning parity bits to every position that is a power of 2. All other positions correspond to data bits. The way the value of the parity bits is determined is by letting the first parity bit encode the parity of every other bit, the second parity bit encode the parity of every other two bits, the fourth parity bit encode the parity of every other fourth bits and so on. The goal of a Hamming Code then is to create a set of parity bits that overlap such that a single-bit error (the bit is logically flipped in value) in a data bit *or* a parity bit can be detected and corrected.

Bit Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Transmitted Bit	P1 (2^0)	P2(2^1)	D1	P3(2^2)	D2	D3	D4
P1	Include	No	Include	No	Include	No	Include
P2	No	Include	Include	No	No	Include	Include
P4	No	No	No	Include	Include	Include	Include

The table above demonstrates the Hamming (7,4) Code, meaning that four bit data is encoded in 7 bits. This code allows us to detect and correct all errors with Hamming distance 1 and detect all errors with Hamming distance 2 because the parity beat provide the appropriate overlap. A *Hamming Distance* is the bitwise distance between any two words of equal length and has all the properties of a metric. For example, the Hamming Distance between 010101 and 101110 is 5. The first tile system we built correctly computed the Hamming Distance between two words. We then built a tile system that given a Hamming Code determines whether it is a feasible code or there are errors which grew linearly in O(n). We finally improved upon our construction using the workings of a Binary counter with some modifications such as changing the double strength glues to single strength glues and adding it into our model to achieve a tile system that could do the same in constant number of tiles. Finally we simulated

our model in the XGrow Simulator and got it to work perfectly. In the next section we describe our tile model, and then we describe our simulations.

The TAK Tile Model

We have called our tile model TAK tile model. The TAK model assumes that it is given a "seed column" of tiles as input which represents the purported Hamming Code for a certain data string. We build a tile system capable of detecting whether the given Hamming code is feasible. To do this, we check the correctness of every parity bit in the code against the data bits that it encodes. In the context of self-assembling tile systems, this intuitively entails building a column of tiles next to the seed column for each parity bit. Each column has as its output a "0" or a "1" represented as "north" glues of the top tile in the column. For example, the Hamming (7,4) code is checked by a 3x7 rectangular block of tiles that has 3 outputs of 0's and 1's that tells us whether each of the parity bits (p1,p2 and p4) is correct. As it turns out, this is the correct intuition and is the structure for our error detection model.

A naïve approach for building such a tile system entails having a unique set of tiles for each column. This approach requires $O(\log(L) \times L)$ tiles in solution rendering it infeasible. We now demonstrate a feasible tile system that requires $O(1)$ tiles.

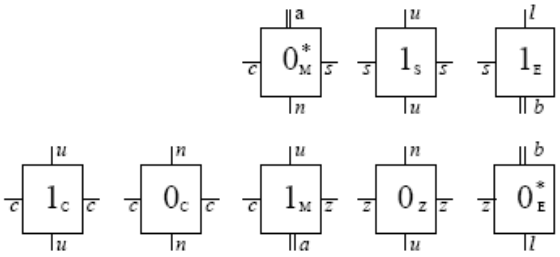
The key insight that motivates this system is the observation that each successive parity bit of the Hamming code checks bits that correspond to the distribution of 1's and 0's in the corresponding columns of the binary counter. For example, the first parity bit of the code encodes every other bit in the Hamming code while the second parity bit encodes (starting at the parity bit) and then skips two bits at once. Since we decided to use the binary counter tiles to build our structure, we also do not have to keep track of which bits are parity bits and which bits are data bits, and therefore, do not need special glues to encode this information, as is the case with the naïve approach.

Example: The 3 bit binary counter starting at 1:

Bit Number	1	2	3	4	5	6	7
P1	1	0	1	0	1	0	1
P2	0	1	1	0	0	1	1
P4	0	0	0	1	1	1	1

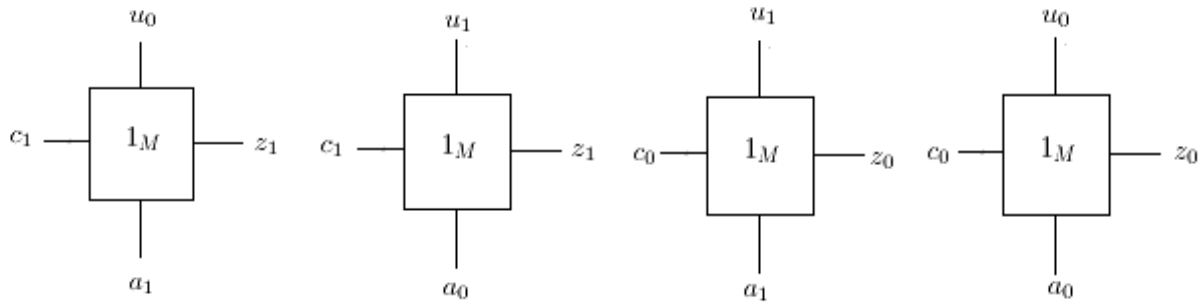
If we line up the Hamming(7,4) code that we need to check against the binary counter, then if we check the 1 positions of the code for the corresponding parity bit then if the parity of the "subcode" is 0 then we know that the parity check is correct as one of two situations occur: The parity bit and the checked code are both 1 or the parity and checked code are both 0. If the parity of the "subcode" is 1 then the parity bit is not the same as the parity of the checked code and hence is an error.

We now give the details of our 32 tile system for error detection in any (L,D) Hamming Code. We base this system on modifications of the binary counter tiling system constructed by Goel et al. in *Optimal Self-Assembly of Counters at Temperature Two.* Their system contains 8 tiles as follows:

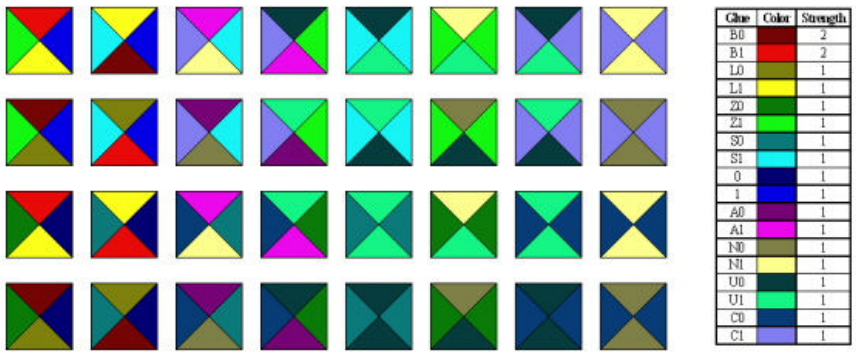


[3]

We expand each of the 8 tiles in the binary counter system to 4 separate tiles as the following example on the 1_M tile illustrates:



We replace the z glue on the right with 2 types of z glues, z_0 and z_1 that will transmit the parity of the tile on the right. The lower glue will transmit the parity information about the tile immediately below to the tile above so the information will be transmitted sequentially to the top of the module. This is done by adding the parity for the right and bottom glues and making the sum mod-2 the parity portion of the top letter. This is done for all the variations of the 1-tiles. In the case where the central label of tile is one of the variations of 0, then the top tiles will be designed to replicate the parity of the bottom tiles as these tiles are designated to ignore the parity of the code they are encoding. The left tile will just replicate the parity portion of the right tile as this will transmit the Hamming code to the next column to allow for the next parity check. Hence we just need to provide a seed column of tiles that represent the Hamming code along with a seed on the bottom that has a row of "starter" tiles attached to the left that will have the appropriate 0 tiles on top to start off the parity check. The following graphic illustrates all the 32 tiles in our system encoded as Wang tiles along with a key for identifying bond types with the corresponding color scheme:



Glue	Color	Strength
B0		2
B1		2
L0		1
L1		1
Z0		1
Z1		1
S0		1
S1		1
0		1
1		1
A0		1
A1		1
N0		1
N1		1
U0		1
U1		1
C0		1
C1		1

Figure 1

Figure 2 illustrates the application of our system to the 1011001 Hamming code, on the right we have the seed column with light blue representing 1's and dark blue representing 0's. The outputs are the top glues of the 3 leftmost tiles on the top row. With the yellow glue representing an error and the navy blue glue representing a correct parity check. So for 1011001 p_1 is incorrect while p_2 and p_4 are correct parity bits.

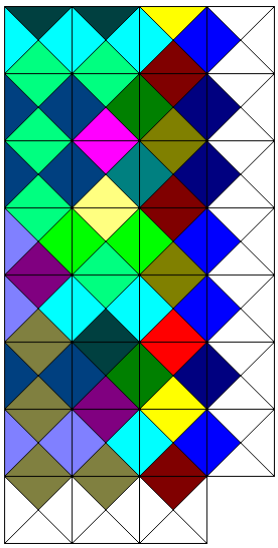


Figure 2

Simulation

A model would not be a useful error detection/correction system if it was error prone itself. Therefore, we present the simulation results of our model that demonstrate its robustness. The 32 tiles of our system in addition to the seed row and column are represented in XGrow where the seed row and column are designed using a single seed that filled the empty square in the bottom left corner in Figure 2. Figure 3 below shows the simulation with the seed tile in green.

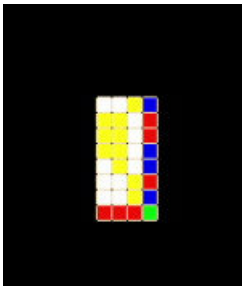


Figure 3

The top row in Figure 3 of the simulation has 2 white blocks and one yellow block showing a 1 bit error in the Hamming(7,4) Code which is represented in the rightmost column. The Blue tiles represent 1's and the Red Tiles represent 0's. Through simulation we discover that we can get perfect robustness by altering the binary counter inside our structure by replacing all double bonds with single bonds whereas

without this we had 2 mismatches per run on average and an extra layer of growth on top of the system. We checked that our system works with multiple hamming codes and it works correctly with 0 mismatches every time showing its robustness. Figure 4 shows that a sequence of 12 zeros results in all white tiles in the top rows representing that it is a valid hamming code. Figure 5 shows that the (12,8) hamming code 011100101010 is completely correct.

http://www.ee.unb.ca/cgi-bin/tervo/hamming.pl is a Hamming Code Simulator which we used to continue checking more codes, and our simulations show that the model works with 0 mismatches all the time and is robust.

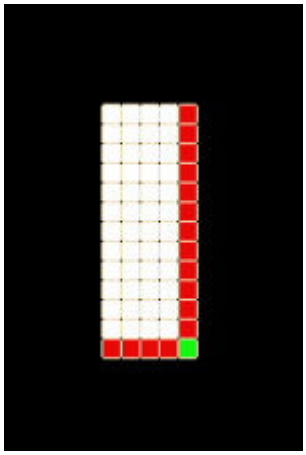


Figure 4

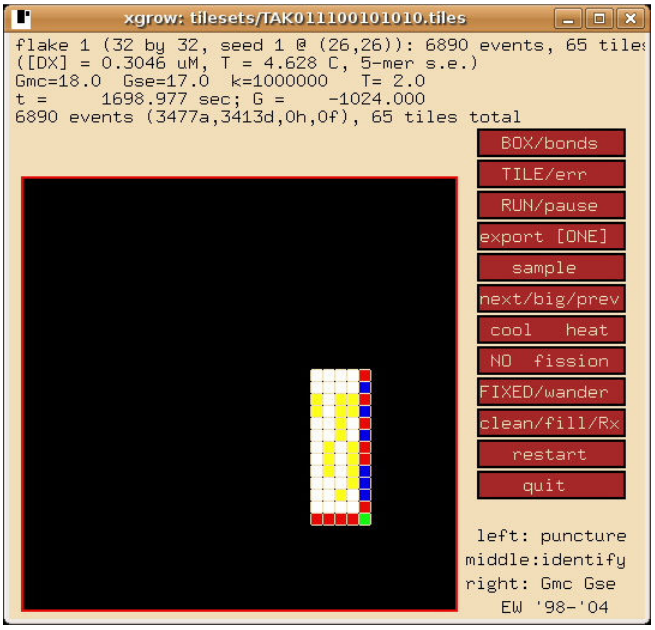


Figure 5

Future Work and Open Questions

So far our module takes a Hamming Code as input and detects p-1 bit errors where p is the number of parity bits. Hamming Codes can also be modified to have the power to correct 1 bit errors but our system does not yet do this and we hope to build a system for this in the future. We would also like to improve our system so along with error detection it also performs error correction, meaning if we input a Hamming code with one error we would get as our output the Hamming with that error corrected. Also, we need to determine how to generate tile models which result in L bit hamming codes that can be used as an input to our TAK model. Another open problem is to determine how to update the hamming code input for checking a system like a binary counter as the data bits are updated.

Acknowledgements

Special thanks to Professor Ashish Goel for teaching us about molecular algorithms in a clear and comprehensive way. Without his advice and guidance this paper would not have been possible.

References

[1] L. Adelman, *Toward a Mathematical Theory of Self Assembly*. Technical Report 00-722, Department of Computer Science, University of Southern California, 2000.

[2] H. Chen, Q. Cheng, A. Goel, M.-D. Huang, and P. Moisset de Espans. *Proofreading tile sets: Error correction for algorithmic self-assembly*. In Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 883–892, 2004.

[3] Cheng, Q. Goel, A., and Moisset, P. *Optimal Self-Assembly of Counters at Temperature Two*. University of Oklahoma, Stanford University, University of Southern California.

[4] Goel, A., *Notes from CME* 352: *Molecular Algorithms*, Winter quarter, 2008.

[5] Goel, A. et al., *Error Free Self-Assembly using Error Prone Tiles.* Stanford University, Stanford, California.

[6] Hamming Code. http://en.wikipedia.org/wiki/Hamming_code, March 2008.

[7] Lukeman, P., Seeman, N. and Mittal, A. (2002). *Hybrid PNA/DNA Nanosystems*. In 1st International Conference on Nanoscale/Molecular Mechanics (N-M2-I), Outrigger Wailea Resort, Maui, Hawaii.

[8] Tervo, Richard. *Hamming Code Tool.* University of New Brunswick, Department of Electrical and Computer Engineering, September 2002.

[9] *Wang Tile System*. http://en.wikipedia.org/wiki/Wang_tile, March 2008.

[10] E. Winfree et al., *The XGrow Simulator*. http://dna.caltech.edu/Xgrow/xgrow_www.html.

[11] E. Winfree and R. Bekbolatov. *Proofreading tile sets: Error correction for algorithmic self-assembly*. In Proceedings of the Ninth International Meeting on DNA Based Computers. Madison, Wisconsin, June 2003.

[12] Winfree, E., Liu, F., Wenzler, L.A., and Seeman, N.C. (1998). *Design and Self-Assembly of Two-Dimensional DNA Crystals*, Nature 394, 539-544.