Compact Error-Resilient Computational DNA Tiling Assemblies*

John H. Reif, Sudheer Sahu, and Peng Yin

Department of Computer Science, Duke University, Box 90129,
Durham, NC 27708-0129, USA
{reif, sudheer, py}@cs.duke.edu

Abstract. The self-assembly process for bottom-up construction of nanostructures is of key importance to the emerging scientific discipline Nanoscience. However, self-assembly at the molecular scale is prone to a quite high rate of error. Such high error rate is a major barrier to large-scale experimental implementation of DNA tiling. The goals of this paper are to develop theoretical methods for compact error-resilient self-assembly and to analyze these methods by stochastic analysis and computer simulation. Prior work by Winfree provided an innovative approach to decrease tiling self-assembly errors without decreasing the intrinsic error rate ϵ of assembling a single tile. However, his technique resulted in a final structure that is four times the size of the original one. This paper describes various *compact* error-resilient tiling methods that *do not increase the size of the tiling assembly*. These methods apply to assembly of boolean arrays which perform input sensitive computations (among other computations). Our 2-way (3-way) overlay redundancy construction drops the error rate from ϵ to approximately ϵ^2 (ϵ^3), without increasing the size of the assembly. These results were further validated using stochastic analysis and computer simulation.

1 Introduction

Self-assembly is a process in which simple objects associate into large (and complex) structures. The self-assembly of DNA tiles can be used both as a powerful computational mechanism [4, 6, 10, 11, 14] and as a bottom-up nanofabrication technique [8]. Periodic 2D DNA lattices have been successfully constructed with a variety of DNA tiles, for example, double-crossover (DX) DNA tiles [13], rhombus tiles [5], triple-crossover (TX) tiles [3], and 4x4 tiles [15]. Two dimensional algorithmic self-assembly, in contrast, is comparatively resistant to experimental demonstration, partially due to the large number of errors in the assembled structure.

How to decrease such errors? There are primarily two kinds of approaches. The first one is to decrease the intrinsic error rate ϵ by optimizing the physical environment in which a fixed tile set assembles [11], by improving the design of the tile itself using new molecular mechanism [2], or by using novel materials. The second approach is to design new tile sets that can reduce the total number of errors in the final structure even

* Extended abstract. For full paper, see [7]

C. Ferretti, G. Mauri and C. Zandron (Eds.): DNA10, LNCS 3384, pp. 293–307, 2005.

with the same intrinsic error rate. A seminal work in this direction is the proofreading tile set constructed by Winfree [12].

One desirable improvement on Winfree's construction (which results in an assembled structure with 4x size of the original one) is to make the design more compact. Here we report construction schemes that achieve performance similar as or better than Winfree's tile set without scaling up the assembled structure. We will describe our work primarily in the context of self-assembling Sierpinsky triangles and binary counters, but note that the design principle can be applied to a more general setting. The basic idea of our construction is to overlay redundant computations and hence force consistency in the scheme. The idea of using redundancy to enhance reliability of a system constructed from unreliable individual components goes back to von Neumann [9].

The rest of the paper is organized as follows. In Sect. 2, we introduce the assembly problem. In Sect. 3, we describe a scheme that decreases the error rate from ϵ to $6\epsilon^2$. In Sect. 4, this scheme is further improved to $30\epsilon^3$ using a three-way overlay redundancy technique. Kinetic analysis is performed in Sect. 5 to show that the assembly speed is not much decreased. Sect. 6 gives empirical study using computer simulation. We conclude with discussions about future work in Sect. 7.

2 Assembly with No Error Corrections

2.1 Assembly Problems

A general assembly problem considered in this paper is the assembly of *a Boolean array*. A Boolean array assembly is an $N \times M$ array, where the elements of each row are indexed over $\{0, \ldots, N-1\}$ from right to left and the elements of each column are indexed over $\{0, \ldots, M-1\}$ from bottom to top. The bottom row and right most column both have some given values. Let $V(i,j)$ be the value of the i-th (from the right) bit on the j-th row (from the bottom) displayed at position (i,j) and communicated to the position $(i, j+1)$. Let $U(i,j)$ be a Boolean value communicated to the position $(i+1, j)$. For $i = 1, \ldots, N-1$ and $j = 1, \ldots, M-1$, we have $V(i,j) = U(i-1,j)\ OP_1\ V(i,j-1)$ and $U(i,j) = U(i-1,j)\ OP_2\ V(i,j-1)$, where OP_1 and OP_2 are two Boolean functions, each with two Boolean arguments and one Boolean output. See Figure 1 for an illustration.

Two Boolean arrays of particular interest are the *Sierpinsky Triangle* [1] and the *Binary counter*. The Sierpinsky Triangle is an $N \times N$ Boolean binary array, where the bottom row and right most column all have 1s; its OP_1 and OP_2 operators are both XOR. Recall that XOR is exclusive OR, a binary operator that outputs bit 1 if the two input bits are different and 0 otherwise. For an illustration, see Figure 2.

The binary counter is an $N \times 2^N$ Boolean binary array. In a binary counter, the bottom row has all 0s and the j-th row (from the bottom) is the binary representation of counter value j, for $j = 0, \ldots, 2^N - 1$. Note that the i-th bit is i-th from the right – this is in accordance with the usual left to right binary notation of lowest precision bits to highest precision bits. $V(i,j)$ represents the value of the i-th (from the right) counter bit on the j-th row (from the bottom), and $U(i,j)$ is the value of the carry bit from the counter bit at position (i,j). In the binary counter, we have $V(0,j) = V(0,j-1)\ XOR\ 1$; $V(i,j) = U(i-1,j)\ XOR\ V(i,j-1)$ for $i = 1, \ldots, N-1$;

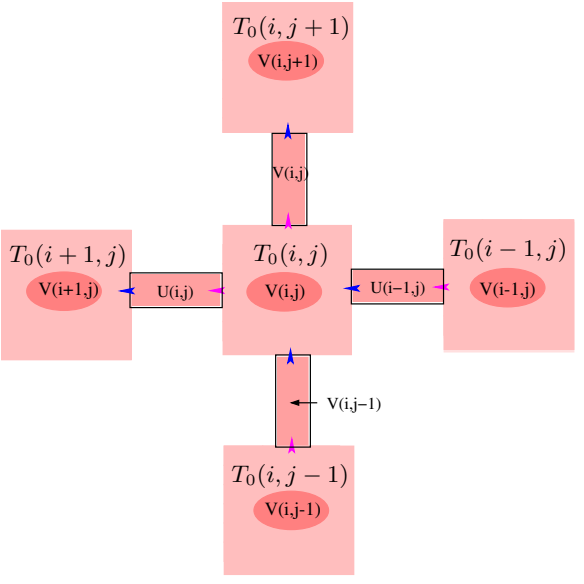

Fig. 1. Tile $T_0(i, j)$ takes input $U(i-1, j)$ and $V(i, j-1)$; determines $V(i, j) = U(i-1, j)\ OP_1\ V(i, j-1)$ and $U(i, j) = U(i-1, j)\ OP_2\ V(i, j-1)$; displays $V(i, j)$

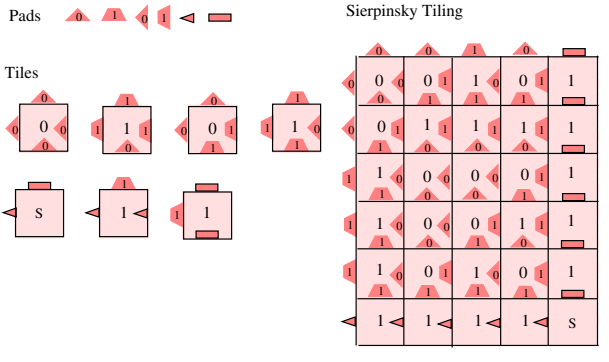

Fig. 2. Sierpinsky triangle tiling assembly

$U(i, j) = U(i-1, j)\ AND\ V(i, j-1)$. Hence OP_1 is the XOR operation and OP_2 is the AND operation.

We observe that OP_1 is XOR both for the Sierpinsky Triangle and for the binary counter and we will thus assume in our error-resilient constructions that OP_1 is XOR. Each assembly will be constructed with the 4×4 DNA tiles described in [15]. A 4×4 tile allows one pad per side (which can communicate a small constant number of bits). Furthermore, we will assume that a "frame" is assembled first for each binary array, consisting of a "bottom row" with N horizontally aligned tiles and a "right border" linear assembly with M vertically aligned tiles.

2.2 Assemblies with No Error Corrections

We first describe the naive assembly scheme without error correction. *Note that such assembly requires only* 4 *tile types in addition to* 3 *frame tiles, but results in rather small scale error-free assemblies (with the actual size contingent on the probability of single pad mismatch between adjacent tiles).* We call this scheme version 0 assembly and denote the tiles as $T_0(i,j)$.

The simplest way to construct such an assembly is to make each side of each tile a binary valued pad. Since the values of the left and top pads depend on the values of the right and bottom pads, the tile type depends on only 2 binary pads, and hence only $2^2 = 4$ tile types are required in addition to the 3 tiles for assembling the initial frame.

The bottom, right, top, and left pads of tile $T_0(i,j)$ represent the values of $V(i,j-1)$ (as communicated from the tile below $T_0(i,j-1)$), $U(i-1,j)$ (as communicated from the tile on its right $T_0(i-1,j)$), $V(i,j)$ (as computed by $V(i,j-1)\ OP_1\ U(i-1,j)$), and $U(i,j)$ (as computed by $V(i,j-1)\ OP_2\ U(i-1,j)$), respectively. A determined value $V(i,j) = 1$ can be displayed by the tile $T_0(i,j)$ using, for example, an extruding stem loop of single strand DNA.

2.3 Errors in Assemblies

All this is theoretically correct, but it has not taken into account the error rate of the assembly of individual DNA tiles. A critical issue in 2D tiling assemblies is the pad mismatch rate, which determines the size of the error-free assembly. Let ϵ be the probability of a single pad mismatch between adjacent assembling DNA tiles, and assume that the likelihood of a pad mismatch error is independent for distinct pads as long as they do not involve the binding of the same two tiles. As such, a pad mismatch rate of $\epsilon = 5\%$ would imply an error-free assembly with an expected size of only 20 tiles, which is disappointingly small. Thus, a key challenge in experimentally demonstrating large scale algorithmic assemblies is to construct error-resilient tiles. Winfree's construction is an exciting step towards this goal [12]. However, to reduce the error rate to ϵ^2 (resp. ϵ^3), his construction replaces each tile with a group of $2 \times 2 = 4$ (resp. $3 \times 3 = 9$) tiles and hence increases the size of the tiling assembly by a factor of 4 (resp. 9). Our construction described below, in contrast, reduces the tiling error rate without scaling up the size of the final assembly. This would be an attractive feature in the attempt to obtain assemblies with large computational capacity. We call our construction *compact error resilient assemblies* and describe them below in detail.

3 Error-Resilient Assembly Using Two-Way Overlay Redundancy

3.1 Construction

To achieve the goals stated in previous section, we propose the following error resilient tiling scheme. *Our Error-Resilient Assembly I (using two-way overlay redundancy) uses only* 8 *computational tile types plus the* 4 *frame tile types. This drops the probability of*

assembly error to $6\epsilon^2$*, which is* 1.5% *for* $\epsilon = 5\%$*, potentially allowing for error-free assemblies of expected size in the hundreds of tiles.*

The construction is depicted in Figure 3. Tiles in this construction are denoted as T_1 tiles (for version 1). Each pad of each tile encodes a pair of bits. The basic idea of this Error-Resilient assembly is the *two-way overlay redundancy*: each tile $T_1(i,j)$ computes the outputs for its own position (i,j) and also for its right neighbor's position $(i-1,j)$; the redundant computation results obtained by $T_1(i,j)$ and its right neighbor $T_1(i-1,j)$ are compared via an additional *error checking portion* on $T_1(i,j)$'s right pad (which is the same as $T_1(i-1,j)$'s left pad). Tile $T_1(i,j)$'s right neighbor $T_1(i-1,j)$ is not likely to bind to $T_1(i,j)$ if these pad values are not consistent. Hence if only one of $T_1(i,j)$ and $T_1(i-1,j)$ is in error (incorrectly placed), the kinetics of the assembly may allow the incorrectly placed tile to be ejected from the assembly.

The four pads of $T_1(i,j)$ are constructed as follows (Figure 3).

- The right and left portions of the bottom pad represent the value of $V(i-1,j-1)$ and $V(i,j-1)$ respectively as communicated from the tile $T_1(i,j-1)$.
- The top portion of the right pad represents the value of $U(i-2,j)$ as communicated from the tile $T_1(i-1,j)$. The bottom portion of the right pad represents the value of $V(i-1,j)$ as determined by the tile $T_1(i,j)$. Note that the value $V(i-1,j)$ is also redundantly determined by $T_1(i-1,j)$ and hence the bottom portion performs comparison of the two values and is referred to as *error checking portion*, and labeled with checked background in Figure 3.

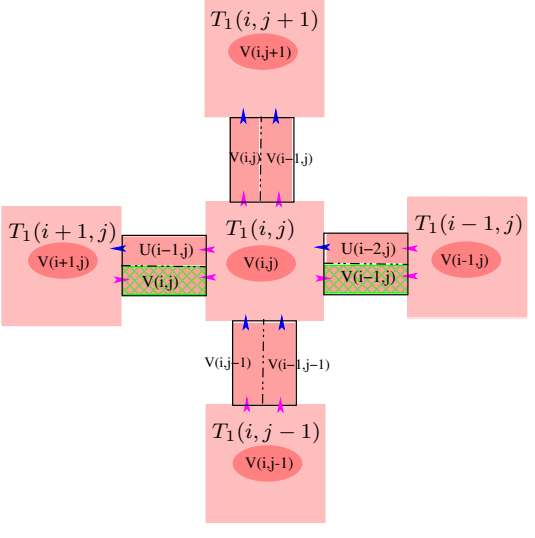

Fig. 3. Construction of compact error-resilient assembly version I. Each pad has two portions. A portion encoding an input (resp. output) value is indicated with a dark blue (resp. light pink) colored arrow head. The error checking portion is depicted as a checked rectangle. Tile $T_1(i,j)$ takes inputs $U(i-2,j)$, $V(i-1,j-1)$, and $V(i,j-1)$; determines $V(i-1,j) = U(i-2,j)\ OP_1\ V(i-1,j-1)$, $U(i-1,j) = U(i-2,j)\ OP_2\ V(i-1,j-1)$, and $V(i,j) = U(i-1,j)\ OP_1\ V(i,j-1)$; displays $V(i,j)$

- The top and bottom portions of the left pad represent the values of $U(i-1,j)$ and $V(i,j)$ respectively, as determined by the tile $T_1(i,j)$. Again, the bottom portion is the error checking portion.
- The right and left portions of the top pad represent the values of $V(i-1,j)$ and $V(i,j)$ respectively, as determined by tile $T_1(i,j)$.

The above tile design allows the values $V(i-1,j-1)$ and $V(i,j-1)$ to be communicated to tile $T_1(i,j)$ from the tile $T_1(i,j-1)$ just below $T_1(i,j)$. The value $U(i-2,j)$ is communicated to tile $T_1(i,j)$ from its immediate right neighbour $T_1(i-1,j)$. The values $V(i-1,j)$ and $U(i-1,j)$ are determined by tile $T_1(i,j)$ from $V(i-1,j-1)$ and $U(i-2,j)$: $V(i-1,j) = U(i-2,j)\ OP_1\ V(i-1,j-1)$ and $U(i-1,j) = U(i-2,j)\ OP_2\ V(i-1,j-1)$. The value $V(i,j)$ is determined from $V(i,j-1)$ and $U(i-1,j)$: $V(i,j) = U(i-1,j)\ OP_1\ V(i,j-1)$. If the determined value $V(i,j) = 1$, then it is displayed by the tile $T_1(i,j)$.

In this construction, each pad encodes two bits. However, since the values of the left pad, the top pad, and the bottom portion ($V(i-1,j)$) of the right pad each depend only on the values of the top portion ($U(i-2,j)$) of the right pad and the bottom pads, the tile type depends on only 3 input binary bits. Hence only $2^3 = 8$ tile types are required. In addition, 4 tiles are required to assemble the frame, as described in Sect. 6.

We emphasize that though a pad has two portions, it should be treated as a whole unit. A value change in one portion of a pad changes the pad to a completely new pad. If the pad is implemented as a single strand DNA, this means that the sequence of the single strand DNA will be a complete new sequence. One potential confusion to be avoided is mistakenly considering two pads encoding, say 00 and 01, as having the 0 portions identical or, in the context of single strand DNA, as having half of the DNA sequences identical. To emphasize the unity of a pad, we put a box around each pad in Figure 3.

3.2 Error Analysis

Recall that ϵ is the probability of a single pad mismatch between two adjacent DNA tiles. We further assume that the likelihood of a pad mismatch error is independent for distinct pads as long as they do not involve the binding of the same two tiles and that OP_1 is the function XOR.

Our intention is that the individual tiling assembly error rate (and hence the propagation of these errors to further tile assemblies) is substantially decreased, due to cooperative assembly of neighboring tiles, which redundantly compute the $V(-,-)$ and $U(-,-)$ values at their positions and at their right neighbours.

Without loss of generality, we consider only the cases where the pad binding error occurs on either the bottom pad or the right pad of a tile $T_1(i,j)$. Otherwise, if the pad binding error occurs on the left (resp. top) pad of tile $T_1(i,j)$, then use the same below argument for tile $T_1(i+1,j)$ (resp. $T_1(i,j+1)$). We define the *neighborhood* of tile $T_1(i,j)$ to be the set of 8 distinct tiles $\{\ T_1(i',j') : |i'-i| < 2,\ |j'-j| < 2\ \} \setminus \{\ T_1(i,j)\ \}$ with coordinates that differ from (i,j) by at most 1. A neighborhood tile $T_1(i',j')$ is *dependent* on $T_1(i,j)$ if both its coordinates are equal to or greater than those of $T_1(i,j)$; otherwise $T_1(i',j')$ is *independent* of $T_1(i,j)$. Note that a neighborhood tile $T_1(i',j')$ is dependent on $T_1(i,j)$ if and only if the values $V(i',j')$ and

$U(i', j')$ are determined at least partially from $V(i, j)$ or $U(i, j)$. More specifically, the neighborhood tiles dependent on $T_1(i, j)$ are $T_1(i + 1, j + 1)$, $T_1(i + 1, j)$, and $T_1(i, j + 1)$. The neighborhood tiles independent of $T_1(i, j)$ are $T_1(i + 1, j - 1)$, $T_1(i, j - 1)$, $T_1(i - 1, j + 1)$, $T_1(i - 1, j)$, and $T_1(i - 1, j - 1)$.

Lemma 1. *Suppose that the neighborhood tiles independent of tile $T_1(i, j)$ have correctly computed $V(-, -)$ and $U(-, -)$. If there is a single pad mismatch between tile $T_1(i, j)$ and another tile just below $T_1(i, j)$ or to its immediate right, then there is at least one further pad mismatch in the neighborhood of tile $T_1(i, j)$. Furthermore, given the location of the initial mismatch, the location of the further pad mismatch can be determined among at most three possible pad locations.*

Proof. Suppose that a pad binding error occurs on the bottom pad or the right pad of tile $T_1(i, j)$ but no further pad mismatch occurs between two neighborhood tiles which are independent of $T_1(i, j)$. We now consider a case analysis of possible pad mismatches.

(1) First consider the case where the pad binding error occurs on the bottom pad of tile $T_1(i, j)$. Recall that the right and left portions of the bottom pad represent the values of $V(i - 1, j - 1)$ and $V(i, j - 1)$ respectively as communicated from tile $T_1(i, j - 1)$. Observe that neighborhood tiles $T_1(i, j - 1)$, $T_1(i - 1, j - 1)$, and $T_1(i - 1, j)$ are all independent of $T_1(i, j)$ and so all *correctly* compute $V(-, -)$ and $U(-, -)$ according to the assumption of the lemma.

(1.1) Consider the case where the pad binding error is due to the *incorrect* value of the right portion $V(i - 1, j - 1)$ of the bottom pad of tile $T_1(i, j)$ as shown in Figure 4. Note that the left portion $V(i, j - 1)$ of the bottom pad of tile $T_1(i, j)$ may also

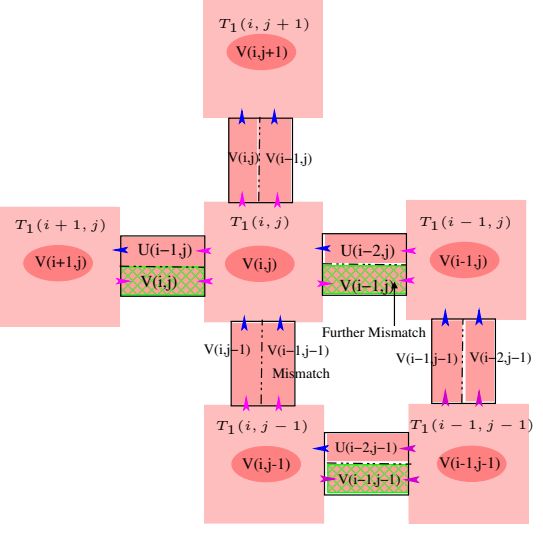

Fig. 4. Case 1.1 in the proof of Lemma 1: error in right portion $V(i - 1, j - 1)$ of the bottom pad of tile $T_1(i, j)$ causes a further mismatch on the right pad of tile $T_1(i, j)$

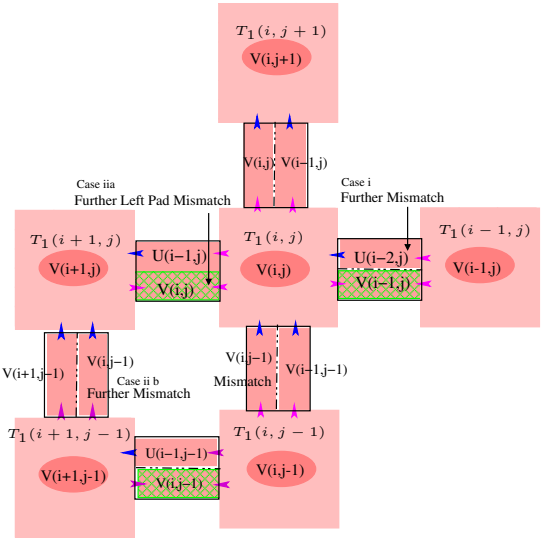

Fig. 5. Case 1.2 in the proof of Lemma 1: a further mismatch is caused by an error in the $V(i, j-1)$ portion of the bottom pad of tile $T_1(i, j)$

be *incorrect*. In case (i), $T_1(i, j)$ has an *incorrect* value for the $U(i-2, j)$ portion of its right pad and hence there is a further pad mismatch on the right pad of $T_1(i, j)$. In case (ii), $T_1(i, j)$ has a *correct* value for the $U(i-2, j)$ portion of its right pad. Since $T_1(i, j)$ uses the formula $V(i-1, j) = U(i-2, j)\ OP_1\ V(i-1, j-1)$ to compute $V(i-1, j)$ and OP_1 is assumed to be the XOR function, it will determine an *incorrect* value for $V(i-1, j)$, which is distinct from the *correct* value of $V(i-1, j)$ determined by its (independent) right neighbor tile $T_1(i-1, j)$. This again implies a further pad mismatch on the right pad of tile $T_1(i, j)$.

(1.2) Next consider the case in Figure 5 where the pad binding error is due to the wrong value of the left portion $V(i, j-1)$ of the bottom pad of tile $T_1(i, j)$. However, there is a *correct* match in the right portion $V(i-1, j-1)$ of the bottom pad of tile $T_1(i, j)$. In case (i), $T_1(i, j)$ has an *incorrect* value for the top portion $U(i-2, j)$ of its right pad, then there will be a mismatch on the right pad of $T_1(i, j)$. In case (ii), $T_1(i, j)$ has a *correct* value for the top portion $U(i-2, j)$ of its right pad, then it will further determine a *correct* value for $U(i-1, j)$, since $U(i-1, j) = U(i-2, j)\ OP_2\ V(i-1, j-1)$ and both $U(i-2, j)$ and $V(i-1, j-1)$ have correct values. Since $V(i, j) = U(i-1, j)\ OP_1\ V(i, j-1)$, $U(i-1, j)$ is correct and $V(i, j-1)$ is incorrect, $T_1(i, j)$ will determine an *incorrect* value for $V(i, j)$.

Note that the neighborhood tiles $T_1(i-1, j-1)$, $T_1(i, j-1)$, and $T_1(i+1, j-1)$ are independent of $T_1(i, j)$ and so both *correctly* compute $V(-, -)$ and $U(-, -)$. However, $T_1(i, j)$'s immediate left neighbour $T_1(i+1, j)$ is dependent both on the *incorrect* value communicated by the pad of $T_1(i, j)$ and the *correct* values communicated by the pad of $T_1(i+1, j-1)$. So in case (ii) there must be a further pad mismatch at tile $T_1(i+1, j)$ as argued below. In case (iia) there is pad mismatch on the right pad of $T_1(i+1, j)$ either due to a mismatch on the portion of $U(i-1, j)$ or on the portion of $V(i, j)$. Otherwise,

in case (iib) there is no mismatch on either the $U(i-1,j)$ or the $V(i,j)$ portion of the pad between $T_1(i,j)$ and $T_1(i+1,j)$. This implies that $V(i,j)$ is *incorrectly* computed by $T_1(i+1,j)$ (since $T_1(i,j)$ has incorrectly computed $V(i,j)$), but $T_1(i+1,j)$ has a correct value of $U(i-1,j)$. However, $V(i,j) = U(i-1,j)\ OP_1\ V(i,j-1)$ and OP_1 is XOR, this implies that the right portion $V(i,j-1)$ of the bottom pad of $T_1(i+1,j)$ has an *incorrect* value, and hence there is a mismatch between $T_1(i+1,j)$ and $T_1(i+1,j-1)$.

(2) Next consider the case where the pad binding error occurs on the right pad of tile $T_1(i,j)$, but there is no error on the bottom pad of $T_1(i,j)$. We first note that the value of the top portion $U(i-2,j)$ of the right pad of $T_1(i,j)$ must have an *incorrect* value. Assume the opposite case where $U(i-2,j)$ is correct. But the $V(i-1,j-1)$ portion of $T_1(i,j)$'s bottom pad must also have a correct value (no mismatch on the bottom pad), this results in a further correct value for the $V(i-1,j)$ portion of $T_1(i,j)$'s right pad. Thus both $U(i-2,j)$ and $V(i-1,j)$ portions of $T_1(i,j)$'s right pad are correct and there must be no mismatch on the right pad. A contradiction. Therefore, $U(i-2,j)$ must have an *incorrect* value, and hence we only need to consider this case.

(2.1) Now consider the case where the pad binding error is due to the *incorrect* value of the top portion $U(i-2,j)$ of the right pad of tile $T_1(i,j)$ as shown in Figure 6. We note that $T_1(i,j)$ will compute an *incorrect* value for the right portion $V(i-1,j)$ of its top pad, according to the formula $V(i-1,j) = U(i-2,j)\ OP_1\ V(i-1,j-1)$. Note that $T_1(i,j+1)$ is dependent on $T_1(i,j)$. In case (i), tile $T_1(i,j+1)$ has a *correct* value of $V(i-1,j)$. There must be a pad mismatch on $V(i-1,j)$ between $T_1(i,j+1)$ and $T_1(i,j)$, since the value of $V(i-1,j)$ determined by $T_1(i,j)$ is incorrect. In case (ii),

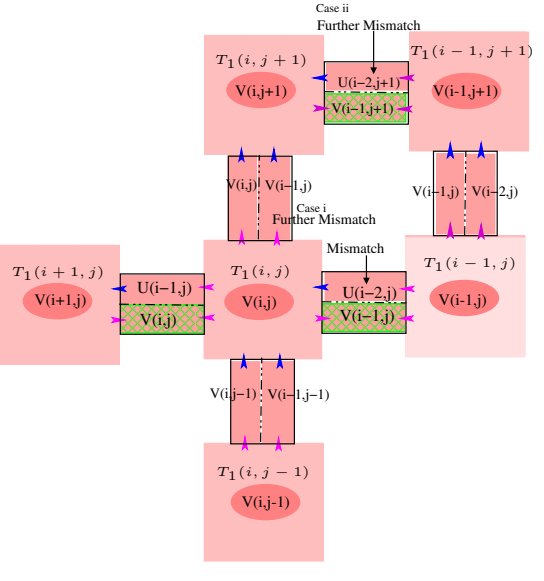

Fig. 6. Case 2.1 in the proof of Lemma 1: a further mismatch is caused by an error in the$U(i-2,j)$ portion of the right pad of tile $T_1(i,j)$

tile $T_1(i, j+1)$ has an *incorrect* value of $V(i-1, j)$, using similar argument as in case 1.1, we can show that there must be a pad mismatch on the $U(i-2, j+1)$ portion of $T_1(i, j+1)$'s right pad.

Hence we conclude that in each case, there is a further pad mismatch between a pair of adjacent tiles in the neighborhood of tile $T_1(i, j)$. Furthermore, we have shown in each case that given the location of the initial mismatch, the location of the further pad mismatch can be determined among at most three possible pad locations. □

Recall that we have let ϵ be the probability of a single pad mismatch between adjacent assembling tiles. This implies that $1-\epsilon$ is the probability of no single pad mismatch between a given pair of adjacent tiles. So the probability that there is no pad mismatch between tile $T_1(i, j)$ and another tile just below or to its immediate right is $(1-\epsilon)^2$. Hence the probability that there is a pad mismatch between tile $T_1(i, j)$ and another tile just below or to its immediate right is $1-(1-\epsilon)^2 = 2\epsilon-\epsilon^2$, which is at most 2ϵ. But by Lemma 1, if there is a pad mismatch between tile $T_1(i, j)$ and another tile just below or to its immediate right, then there is a further pad mismatch between a pair of adjacent tiles in the immediate neighborhood of tile $T_1(i, j)$, and the location of the further pad mismatch can be determined among at most three possible pad locations. The probability that there is such a further pad mismatch between tiles at most three possible pad locations is at most $1-(1-\epsilon)^3$, which is at most 3ϵ. This implies that with probability at most $(3\epsilon)(2\epsilon) = 6\epsilon^2$, there is both (i) a pad mismatch between tile $T_1(i, j)$ and another tile just below or to its immediate right; and (ii) furthermore, there is also a further pad mismatch between tiles in the immediate neighborhood of tile $T_1(i, j)$ as considered in the case analysis in the proof of Lemma 1. Hence we have shown:

Theorem 1. *Suppose that the neighborhood tiles independent of tile $T_1(i, j)$ have correctly computed $V(-,-)$ and $U(-,-)$. Then the assembly error probability for tile $T_1(i, j)$ is at most $6\epsilon^2$, where ϵ is the probability of a single pad mismatch.*

4 Error-Resilient Assembly Using Three-Way Overlay Redundancy

4.1 Construction

We next extend the design of our scheme to a 3-way overlay scheme. The Error-Resilient Assembly II (using 3-way overlay redundancy) uses 16 computational tile types and 5 frame tile types. One mismatch on a tile forces two more mismatches in its neighborhood. This property further lowers the assembly error.

The basic construction is shown in Figure 7. In this construction, each pad encodes a tuple of 3 bits and hence is an 8-valued pad. The basic idea of this error-resilient assembly is to have each tile $T_2(i, j)$ compute error checking values for positions $(i-1, j)$, $(i, j-1)$, $(i+1, j)$, and $(i, j+1)$, which are compared with corresponding error checking values computed by $T_2(i, j)$'s four neighbors. The neighbors are unlikely to bind with $T_2(i, j)$ if such error checking values are inconsistent, and the kinetics of the assembly will allow these tiles to dissociate from each other, as in version 1 (2-way overlay redundancy). However, instead of introducing just one additional mismatch in $T_2(i, j)$'s

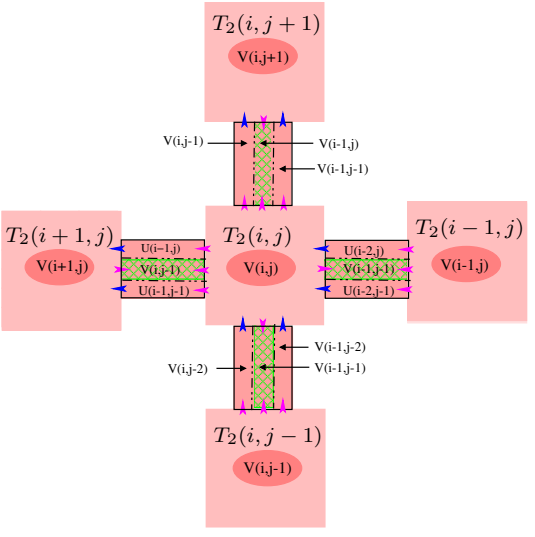

Fig. 7. Tile T_2 takes inputs $U(i-2,j)$, $U(i-2,j-1)$, $V(i-1,j-2)$, and $V(i,j-2)$; determines $V(i-1,j-1) = U(i-2,j-1)\ OP_1\ V(i-1,j-2)$, $U(i-1,j-1) = U(i-2,j-1)\ OP_2\ V(i-1,j-2)$, $V(i,j-1) = U(i-1,j-1)\ OP_1\ V(i,j-2)$, $U(i-1,j) = U(i-2,j)\ OP_2\ V(i-1,j-1)$, $V(i,j) = U(i-1,j)\ OP_1\ V(i,j-1)$, and $V(i-1,j) = U(i-2,j)\ OP_1\ V(i-1,j-1)$; displays $V(i,j)$

neighborhood, the 3-way overlay redundancy (version 2) forces two mismatches, and hence we have a further lowered error rate.

4.2 Error Analysis

For error analysis, in addition to the assumptions made in Sect. 3.2, we require that OP_2 can detect incorrect value of input 1 regardless of the correctness of input 2. This property seems essential to guarantee two further mismatches in a tile's neighborhood when there is an initial mismatch on one of the tile's four pads.

Using a similar but more involved analysis as in Lemma 1 and Theorem 1, we can show

Lemma 2. *Suppose that the neighborhood tiles independent of tile $T_2(i,j)$ have correctly computed $V(-,-)$ and $U(-,-)$. If there is a single pad mismatch between tile $T_2(i,j)$ and another tile just below or to its immediate right, then there are at least two further pad mismatches between pairs of adjacent tiles in the immediate neighborhood of tile $T_2(i,j)$. Furthermore, given the location of the initial mismatch, the location of the second mismatch can be determined among at most three locations in the neighborhood of $T_2(i,j)$; given the location of the initial and the second mismatches, the location of the third mismatch can be determined among at most five locations.*

Theorem 2. *Suppose that the neighborhood tiles independent of tile $T_2(i,j)$ have correctly computed $V(-,-)$ and $U(-,-)$. Then the assembly error probability for tile*

$T_2(i,j)$ is at most $2\epsilon \times 3\epsilon \times 5\epsilon = 30\epsilon^3$, where ϵ is the probability of a single pad mismatch.

For detailed analysis, see [7]. It is also easy to see that this schemes requires $2^4 = 16$ computational tile types and 5 frame tile types.

5 Kinetic Analysis

Our kinetic analysis is based upon the analysis done by Winfree [12]. Two parameters, G_{se} and G_{mc}, are defined in [12]. G_{mc} measures the entropic cost of fixing the location of a monomer unit and G_{se} measures the free energy cost of breaking a single sticky-end bond. A non-rigorous condition for good self-assembly is given as $G_{mc} \approx 2G_{se}$ and the growth rate of assembly r is approximately $\alpha e^{-G_{mc}}$.

For the construction with no error-correction, the equilibrium error rate δ_0 for an assembly is approximately $k_0 e^{-G_{se}}$, which yields an assembly rate $r_0 \approx \alpha e^{-G_{mc}} \approx \alpha e^{-2G_{se}} \approx (\alpha/k_0^2)\delta_0^2$ [12]. For our version 1 error-resilient construction, it can be shown that $\delta_1 \approx k_1 e^{-2G_{se}}$, which further yields $r_1 \approx \alpha e^{-G_{mc}} \approx \alpha e^{-2G_{se}} \approx (\alpha/k_1)\delta_1$. For our version 2 error resilient scheme, it can be shown that the error rate is approximately $\delta_2 \approx k_2 e^{-3G_{se}}$, which yields $r_2 \approx \alpha e^{-G_{mc}} \approx \alpha e^{-2G_{se}} \approx (\alpha/k_2)\delta_2^{(2/3)}$. k_0, k_1 and k_2 are constants. See [7] for details. The above analysis shows that while the error rates δ_1 and δ_2 are significantly reduced in our error resilient assemblies, the aggregation speeds r_1 and r_2 stay approximately the same as r_0.

6 Computer Simulation

We first give below the construction of a Sierpinsky Triangle using our error resilient assembly version 1, and then perform empirical study of the error rates using computer simulation of assembly of the Sierpinsky Triangle and compare the results with that of Winfree's [12].

Figure 8 illustrates the construction of a Sierpinsky triangle, using 8 computational tiles and 4 frame tiles. We would like to again emphasize that although we give the construction of the tiles in previous sections with each pad having two or three distinct portions, a mismatch on any portion of a pad results in a *total* mismatch of the whole pad instead of a partial mismatch of only that portion. Hence, in Figure 8, we use a distinct label for each pad, emphasizing the wholeness of the pad.

For the simulation study, we used the Xgrow simulator by Winfree [12] and simulated the assembly of Sierpinsky triangles for the following cases:

- assembly without any error correction,
- assembly using Winfree's 2×2 proofreading tile set,
- assembly using Winfree's 3×3 proofreading tile set,
- assembly using our error resilient scheme version 1, T_1 (construction in Figure 8),
- assembly using our error resilient scheme version 2, T_2 (construction not shown).

We performed simulations of the assembly process of a target aggregate of 512×512 tiles. A variable N is defined as the number of tiles assembled without any permanent

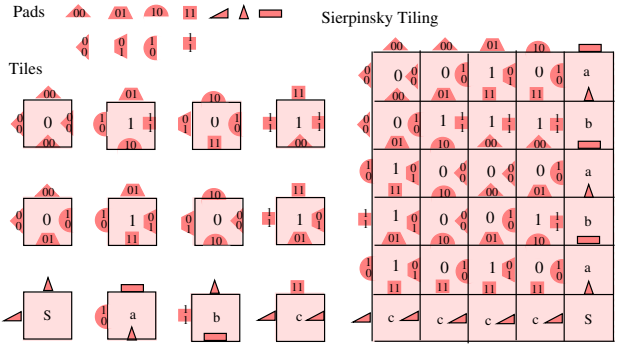

Fig. 8. The construction of a Sierpinsky Triangle using error resilient assemblies version 1. The pads and the tile set are shown on the left and the assembled Sierpinsky Triangle is shown on the right. The pads of strength 2 have black borders while the strength 1 pads are border-less. The seed tile is labeled with S. Tiles a, b, and c are the other frame tiles

error in the assembly in 50% cases. The variations in the value of N are measured as we increase value of the probability of a single mismatch in pads (ϵ) by changing the values of G_{mc} and G_{se}, where G_{mc} and G_{se} are the free energies [12]. We used the fact mentioned by Winfree [12] that $\epsilon \approx 2e^{-G_{se}}$ and for a good assembly we need to have $G_{mc} \approx 2G_{se}$.

Figure 9 shows the variation in N with $\log_e \epsilon$. From the figure it can be seen that the performance of our version 1 (T_1) construction is comparable to Winfree's 2×2 proofreading tile set construction, while our version 2 (T_2) performs comparably to Winfree's 3×3 proofreading tile set construction.

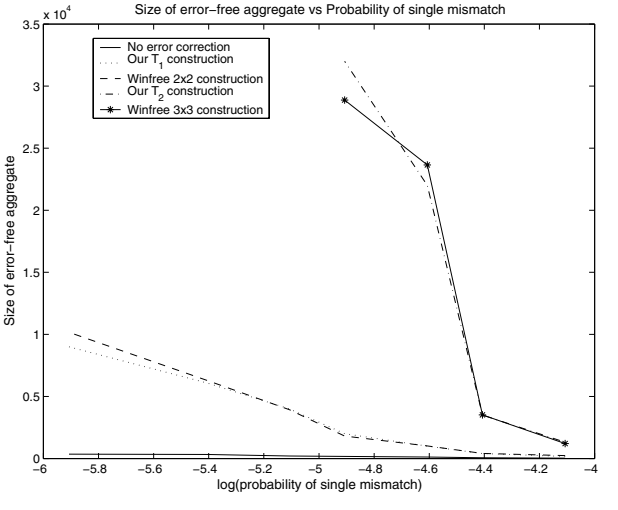

Fig. 9. A graph showing the variation of N *v.s.* increasing value of error (probability of single mismatch)ϵ

7 Discussion

In the proof of this paper, we require OP_1 to be XOR, for concreteness. However, note that our constructions apply to more general boolean arrays in which OP_1 is an *input sensitive operator*, i.e. the output changes with the change of exactly one input.

Note that OP_1 and OP_2 are both the function XOR for the example assemblies for the Sierpinsky Triangle but this is not true for the assembly for a binary counter of N bits, since OP_2 is the logical AND in that example. It is an open question whether our above error-resilient constructions can be further simplified in the case of special computations, such as the Sierpinsky Triangle, where the OP_1 and OP_2 are the same function such as XOR.

Another open question is to extend the construction into a more general construction such that the error probability can be decreased to ϵ^k for any given k, or alternatively, prove an upper bound for k.

Acknowledgements

We would like to thank Erik Winfree for the simulation software Xgrow. We are also grateful to Thomas H. LaBean and Hao Yan for helpful discussions. This work was supported by NSF under ITR Grant EIA-0086015 and ITR Grant 0326157, by NSF under QuBIC Grant EIA-0218376 and QuBIC Grant EIA-0218359, by NSF under EMT Grant CCF-0432038 and EMT Grant CCF-0432047, by DARPA/AFSOR under Contract F30602-01-2-0561, and by RGC under Grant HKBU2107/04E.

References

1. B. A. Bondarenko. *Generalized Pascal Triangles and Pyramids, Their Fractals, Graphs and Applications*. The Fibonacci Association, 1993. Translated from Russian and edited by R. C. Bollinger.
2. H. L. Chen, Q. Cheng, A. Goel, M. D. Huang, and P. M. de Espanes. Invadable self-assembly: Combining robustness with efficiency. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2004.
3. T. H. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J. H. Reif, and N. C. Seeman. The construction, analysis, ligation and self-assembly of DNA triple crossover complexes. *J. Am. Chem. Soc.*, 122:1848–1860, 2000.
4. M. G. Lagoudakis and T. H. LaBean. 2-D DNA self-assembly for satisfiability. In *DNA Based Computers V*, volume 54 of *DIMACS*, pages 141–154. American Mathematical Society, 2000.
5. C. Mao, W. Sun, and N. C. Seeman. Designed two-dimensional DNA holliday junction arrays visualized by atomic force microscopy. *J. Am. Chem. Soc.*, 121:5437–5443, 1999.
6. J. H. Reif. Local parallel biomolecular computation. In H. Rubin and D. H. Wood, editors, *DNA-Based Computers 3*, volume 48 of *DIMACS*, pages 217–254. American Mathematical Society, 1999.
7. J. H. Reif, S. Sahu, and P. Yin. Compact error-resilient computational DNA tiling assemblies. Technical Report CS-2004-08, Duke University, Computer Science Department, 2004.
8. N. C. Seeman. DNA in a material world. *Nature*, 421:427–431, 2003.

9. J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Autonomous Studies*, pages 43–98, 1956.
10. E. Winfree. On the computational power of DNA annealing and ligation. In R. J. Lipton and E. B. Baum, editors, *DNA Based Computers 1*, volume 27 of *DIMACS*, pages 199–221. American Mathematical Society, 1996.
11. E. Winfree. Simulation of computing by self-assembly. Technical Report 1988.22, Caltech, 1998.
12. E. Winfree and R. Bekbolatov. Proofreading tile sets: logical error correction for algorithmic self-assembly. In *DNA Based Computers 9*, volume 2943 of *Lecture Notes in Computer Science*, pages 126–144, 2004.
13. E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394:539–544, 1998.
14. E. Winfree, X. Yang, and N. C. Seeman. Universal computation via self-assembly of DNA: Some theory and experiments. In L. F. Landweber and E. B. Baum, editors, *DNA Based Computers II*, volume 44 of *DIMACS*, pages 191–213. American Mathematical Society, 1999.
15. H. Yan, S. H. Park, G. Finkelstein, J. H. Reif, and T. H. LaBean. DNA-templated self-assembly of protein arrays and highly conductive nanowires. *Science*, 301:1882–1884, 2003.