Compact Error Resilient Computational DNA Tilings

John H. Reif[1], Sudheer Sahu[1], and Peng Yin[1]

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, algorithmic 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 algorithmic DNA tilings. The goals of this paper are to develop theoretical methods for compact error-resilient algorithmic DNA tilings and to analyze these methods by thermodynamic analysis and computer simulation. Prior work by Winfree provided an innovative approach to decrease tiling self-assembly mismatch errors without decreasing the intrinsic error rate ϵ of assembling a single tile. However, his technique resulted in a final structure larger than the original one (four times larger for decreasing the error to ϵ^2, nine times for to ϵ^3). In this paper, we describe various *compact* error-resilient tiling methods that *do not increase the size of the tiling assembly*. These methods apply to the assembly of Boolean arrays which perform input sensitive computations (among other computations). Our 2-way (3-way) overlay redundancy construction decreases the error rate from ϵ to approximately ϵ^2 (ϵ^3), without increasing the size of the assembly. As in Winfree's constructions, the number of distinct tile types required is also increased in our error-resilient tiling constructions. These results were further validated using 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 [8, 13, 21, 24, 27] and as a bottom-up nanofabrication technique [18]. Periodic 2D DNA lattices have been successfully constructed with a variety of DNA tiles, for example, double-crossover (DX) DNA tiles [26], rhombus tiles [12], triple-crossover (TX) tiles [7], "4x4" tiles [30], triangle tiles [9], and hexagonal tiles [3]. Aperiodic barcode DNA lattices have also been experimentally constructed [29]. In addition to forming extended lattices, DNA tiles can also form tubes [10, 15].

Self-assembly of DNA tiles can be used to carry out computation, by encoding data and computational rules in the sticky ends of tiles [23]. Such self-assembly of DNA tiles is known as *algorithmic self-assembly* or *computational-tilings*. Researchers have experimentally demonstrated a one-dimensional algorithmic self-assembly of triple-crossover DNA molecules (TX tiles), which performs a 4-step cumulative XOR computation [11]. A one-dimensional "string" tiling assembly was also experimentally constructed that computes an XOR table in parallel [28]. Recently, two dimensional algorithmically self-assembled DNA crystals were constructed that demonstrate the pattern of Sierpinski triangles [16] and the pattern of a binary counter [1]. However, these two dimensional algorithmic crystals suffer quite high error rates. Reducing such errors is thus a key challenge in algorithmic DNA tiling self-assembly.

How to decrease such errors? There are primarily two approaches. The first one is to decrease the intrinsic error rate ϵ by optimizing the physical environment in which a fixed tile set assembles [27], by improving the design of the tile set using new molecular mechanism [4, 6], 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 with the same intrinsic error rate [5, 17, 25]. Three kinds of errors have been studied in the direction of the second approach, namely, the mismatch error, the facet error, and the nucleation error. Here in this paper, we are interested in the study of the mismatch error. The mismatch error is first studied by Winfree [25]. Winfree designs a novel proof reading tile set, which decreases mismatch errors without decreasing the intrinsic error rate ϵ. However, his technique results in a final structure that is larger than the original one (four times larger for decreasing the error to ϵ^2, nine times for ϵ^3).

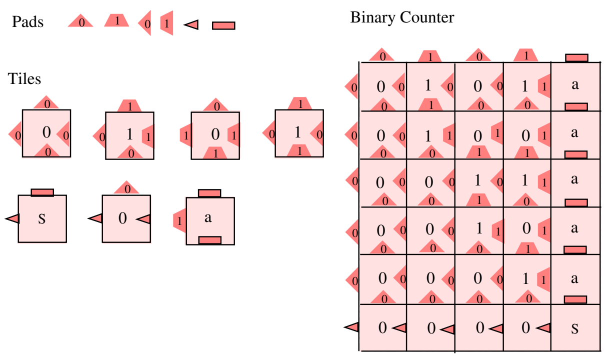
One natural improvement to Winfree's construction is to make the design more compact. Here we report construction schemes that achieve performance comparable to Winfree's proofreading tile set without scaling up the assembled structure. We will describe our work primarily in the context of self-assembling 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 (in similar spirit as in [25]). The idea of using redundancy to enhance the reliability of a system constructed from unreliable individual components goes back to von Neumann [19].

The rest of the paper is organized as follows. In Section 2, we introduce the algorithmic assembly problem by reviewing Winfree's abstract Tile Assembly Model (aTAM) and kinetic Tile Assembly Model (kTAM) [25]. In Section 3, we describe our scheme that decreases the error rate from ϵ to $3\epsilon^2$. In Section 4, this scheme is further improved to $15\epsilon^3$ using a three-way overlay redundancy technique. Two concrete constructions are given in Section 5 and empirical study with computer simulation of our tile sets is conducted. We conclude with discussions about future work in Section 6.

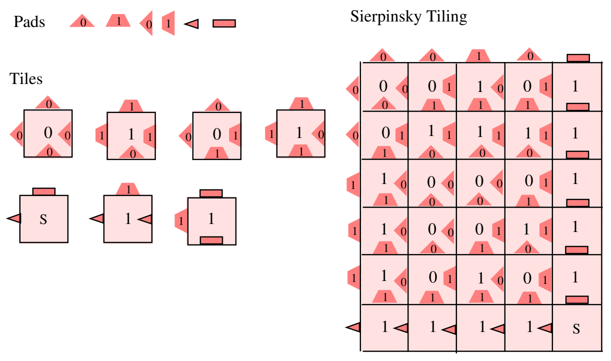
2 Algorithmic Assembly Problems

2.1 Algorithmic Assembly in Abstract Tile Assembly Model

The growth process of a tiling assembly is elegantly captured by an abstract Tile Assembly Model (aTAM) proposed by Winfree [14], which builds on the tiling model initially proposed by Wang in 1960 [20]. In this model, each of the four sides of a tile has a glue (also called *pad*) and each glue has a type and a positive integral strength. Assembly occurs by the accretion of tiles iteratively to an existing assembly, starting with a special *seed* tile. A tile can be "glued" to a position in an existing assembly if the tile can fit in the position such that each pair of adjacent pads of the tile and the



(a)



(b)

Fig. 1. (a) Binary counter tiling assembly. (b) Sierpinsky triangle tiling assembly. In both (a) and (b), the pads and the tile set are shown on the left and the corresponding assembled structures are shown on the right. The pads of strength 2 have black borders while the strength 1 pads are border-less. The first row of tiles on the left are four internal tiles (computational tiles); the second row are three frame tiles, one of which is a special seed tile (labeled with S)

assembly have the same glue type and the total strength of the these glues is greater than or equal to the *temperature*, a system parameter.

As a concrete example, we describe a binary counter constructed by Winfree [14] in Figure 1 (a). Here, the temperature of the system is set to 2. Two adjacent pads (glues) on neighboring tiles can be glued to each other if they are of the same type. The assembly starts with the seed tile S at the lower right corner and proceeds to the left and to the top by the accreation of individual tiles. First, the reverse L shaped frame, composed of the frame tiles is assembled. Note that the glue strength between two neighbouring frame tiles is 2, which is greater than or equal to the temperature, and hence the assembly of the frame tiles can carry through. Next, the internal tiles are assembled. Since the glue strength of a pad on an internal tile is 1, the assembly of an internal tile requires *cooperative* support from two other already assembled tiles. More specifically, after the

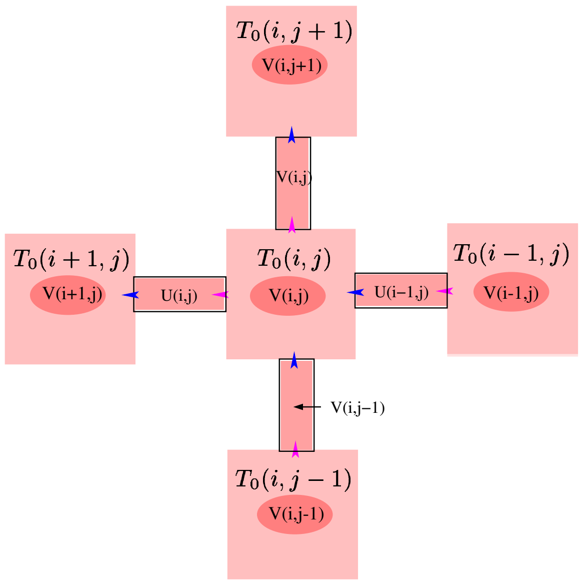


Fig. 2. Tile $T_0(i, j)$ takes input $U(i-1, j)$ and $V(i, j-1)$; determines $V(i, j) = U(i-1, j)\ \mathtt{OP}_1\ V(i, j-1)$ and $U(i, j) = U(i-1, j)\ \mathtt{OP}_2\ V(i, j-1)$; displays $V(i, j)$

assembly of the frame, the frame tile a and frame tile 0 immediately neighbouring the seed S tile cooperatively form a binding site for an internal 1 tile that has label 1 on its left side and label 0 on its bottom side. And this 1 tile can attach itself at this site. This in turn produces further growing sites for 0 internal tiles on top of and to the left of this just assembled 1 tile. Thus the growth can go on inductively by the accretion of appropriate individual tiles. It is straight forward to verify that the accretion of the tiles forms a binary counter with each row representing a binary number. As another concrete example, the tile set in Figure 1 (b) forms a Sierpinsky triangle [2]. Though the above two examples appear simple, it has been proven that algorithmic assembly of tiles holds universal computing power by simulating a one dimensional cellular automaton [22].

Note that each internal tile performs two computations: the right pad and bottom pad of each pad serve as two input bits; the left pad represents an output bit as the result of binary `AND` of the two input bits; the upper pad represents the result of the binary `XOR` operation of the two input bits (Recall that `XOR` is exclusive `OR` , a binary operator that outputs bit 1 if the two input bits are different and 0 otherwise) .

By modifying the internal computational tiles and let the left pad represent an output bit as the result of binary `XOR` of the two input bits, we obtain a set of tiles that can self-assemble into a Sierpinsky triangle [2] (Figure 1 (b)).

The above two assemblies serve as illustrating examples for the general algorithmic assembly problem considered in this paper, 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 $V(i,j)=U(i-1,j)\ \mathtt{OP}_1\ V(i,j-1)$ and $U(i,j)=U(i-1,j)\ \mathtt{OP}_2\ V(i,j-1)$, where $\mathtt{OP}_1$ and $\mathtt{OP}_2$ are two Boolean functions, each with two Boolean arguments and one Boolean output. See Figure 2 for an illustration.

The binary counter shown in Figure 1 (a) 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 $V(0,j)=V(0,j-1)\ \mathtt{XOR}\ 1$; $V(i,j)=U(i-1,j)\ \mathtt{XOR}\ V(i,j-1)$ for $i=1,\ldots,N-1$; $U(i,j)=U(i-1,j)\ \mathtt{AND}\ V(i,j-1)$. Hence $\mathtt{OP}_1$ is the $\mathtt{XOR}$ operation and $\mathtt{OP}_2$ is the $\mathtt{AND}$ operation. The Sierpinsky triangle shown in Figure 1 (b) is an $N \times N$ Boolean binary array, where the bottom row and right most column all have 1s; its $\mathtt{OP}_1$ and $\mathtt{OP}_2$ operators are both $\mathtt{XOR}$.

To construct a Boolean array assembly, we make each side of each tile, denoted $T_0(i,j)$, a binary valued pad. 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 $V(i,j-1)\ \mathtt{OP}_1\ U(i-1,j)$), and $U(i,j)$ (as computed by $V(i,j-1)\ \mathtt{OP}_2\ U(i-1,j)$), respectively. In the practical context of DNA tiling assemblies, 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. 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).

2.2 Thermodynamic Error Analysis in Kinetic Tile Assembly Model

Experimental construction of Boolean array assemblies has shown that such algorithmic assemblies are error prone. In particular, the experimental construction of Sierpinsky triangles suffers a pad mismatch rate ϵ of 1% to 10% [16]. To analyze the error rate, Winfree further extended the above aTAM model to a kinetic Tile Assembly Model (kTAM), which includes rates both for tiles to associate to (forward rate) and to dissociate from (reverse rate) growing assemblies [25]. We reproduce Winfree's kTAM model below for completeness.

Winfree's kTAM model computes the forward and reverse rates as thermodynamic parameters. The forward rate is determined solely by the concentration of tiles, but not the type of the tiles. When the concentration of the tiles is fixed, the absolute forward rate is given by

$$r_f = k_f[\text{monomer tile}] = k_f e^{-G_{mc}},$$

where $G_{mc} = -\ln[\text{monomer}]/M$ is a unitless free energy that measures the monomer, *i.e.* tile, concentration in the system.

In contrast, the reverse reaction rate depends inversely exponentially on the number of base pair bonds that must be broken for the tile to dissociate from the assembly. It is given by

$$r_{r,b} = k_{r,b} = k_f e^{-bG_{se}},$$

where $G_{se} = \Delta G/RT$ is unitless free energy corresponding to the dissociation of a single sticky end, and b is the number of such sticky ends.

It has been shown that when G_{mc} is a little smaller than $2G_{se}$, the algorithmic self-assembly under temperature 2 proceeds with optimal error rate. Intuitively, when $G_{mc} \approx 2G_{se}$, the assembly occurs near melting temperature of the system. Under such conditions, the self-assembly can achieve equilibrium, and the probability of observing a particular assembly $\mathcal{A}$ is given by

$$\Pr(\mathcal{A}) = \frac{1}{Z} e^{-G(\mathcal{A})} \text{ with } Z = \sum_{\mathcal{A}'} e^{-G(\mathcal{A}')},$$

where $G(\mathcal{A}) = nG_{mc} - iG_{se}$ is the free energy of the assembly, n is the number of tiles in the assembly, i is the number of mismatches in the assembly, and Z is the partition function. As such, an n-assembly with Δi more mismatches will occur $e^{\Delta i G_{se}}$ less likely.

Now let $\mathcal{A}_i$ be the collection of assemblies with i mismatches in the assembly and let k_i be the number of the distinct types of $\mathcal{A}_i$ assemblies. In particular, $\mathcal{A}_0$ is the unique correct assembly and $k_0 = 1$. In addition, $\mathcal{A}_1$ represents the assemblies with exactly one mismatch. Since there are altogether $2n$ bonds in an n assembly, $k_1 = 2n$. Then we have

$$\begin{aligned}
\Pr(\mathcal{A}_0) &= \frac{[\mathcal{A}_0]}{\sum_{i=0}^{n} [\mathcal{A}_i]} && (1)\\
&= \frac{1}{\sum_{i=0}^{n} [\mathcal{A}_i]/[\mathcal{A}_0]} && (2)\\
&= \frac{1}{1 + k_1 e^{-G_{se}} + k_2 e^{-2G_{se}} + k_3 e^{-3G_{se}} + ... + k_n e^{-nG_{se}}} && (3)\\
&\approx \frac{1}{1 + 2ne^{-G_{se}}} && (4)\\
&\approx 1 - 2ne^{-G_{se}}. && (5)
\end{aligned}$$

On the other hand, since it takes n error-less steps to assembly A_0, we have

$$\Pr(\mathcal{A}_0) = ((1-\epsilon)^2)^n \approx 1 - 2n\epsilon, \qquad (6)$$

where ϵ is pad mismatch rate. Comparing equations 5 and 6, we have $\epsilon = e^{-G_{se}}$.

Under the equilibrium conditions, Winfree further showed that the net growth rate of the assembly is given by

$$r_0 = r_f - r_r \approx \beta e^{-G_m c} \approx \beta e^{-2G_s e} = \beta \epsilon^2,$$

where β is a constant reflecting the small difference between G_{mc} and $2G_{se}$. Now, based on the above formula, a straightforward method to reduce the error rate ϵ is to reduce the growth rate r_0. However, since r_0 depends quadratically on ϵ, a small decrease in error rate may entail dramatic decrease in the growth rate.

3 Error-Resilient Assembly Using Two-Way Overlay Redundancy

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 pair of 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 [25]. 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 constructions *compact error resilient assemblies* and describe them below in detail.

3.1 Construction

To achieve the goals stated above, 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* 4 *frame tile types. This drops the probability of assembly error to* $3\epsilon^2$*, 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 to achieve error resiliency is to use *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

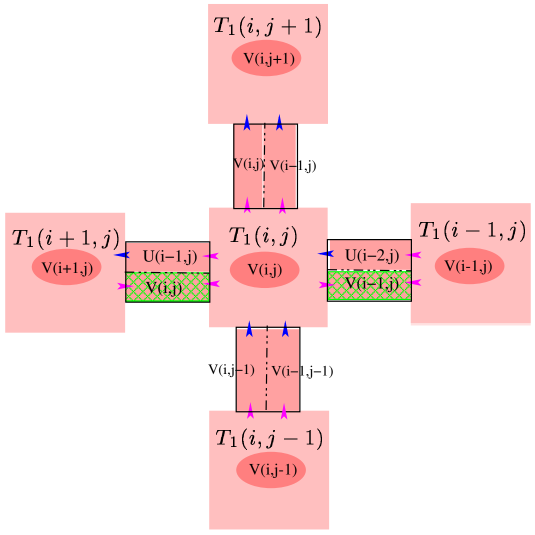


Fig. 3. Construction of compact error-resilient assembly version I. Each pad has two portions. A portion encoding an input (resp. output) value is indicated with a dark blue (resp. light pink) colored arrow head. The error checking portion is depicted as a checked rectangle. Tile $T_1(i,j)$ takes inputs $U(i-2,j)$, $V(i-1,j-1)$, and $V(i,j-1)$; determines $V(i-1,j) = U(i-2,j)\ \texttt{OP}_1\ V(i-1,j-1)$, $U(i-1,j) = U(i-2,j)\ \texttt{OP}_2\ V(i-1,j-1)$, and $V(i,j) = U(i-1,j)\ \texttt{OP}_1\ V(i,j-1)$; displays $V(i,j)$

redundantly determined by $T_1(i-1,j)$ and hence this bottom portion performs comparison of the two values and is referred to as *error checking portion*, and labeled with checked background in Figure 3.

- 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 $T_1(i-1,j)$. These three values, $V(i-1,j-1)$, $V(i,j-1)$, and $U(i-2,j)$, can be viewed as input bits to tile $T_1(i,j)$, and the other portions of the pads as outputs. 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 $U(i-2,j)$: $V(i-1,j) = U(i-2,j)\ \texttt{OP}_1\ V(i-1,j-1)$ and $U(i-1,j) = U(i-2,j)\ \texttt{OP}_2\ V(i-1,j-1)$. The value $V(i,j)$ is determined from $V(i,j-1)$ and $U(i-1,j)$: $V(i,j) = U(i-1,j)\ \texttt{OP}_1\ V(i,j-1)$. The determined value $V(i,j) = 1$ 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, namely, $V(i-1,j-1)$, $V(i,j-1)$, and $U(i-2,j)$. Hence only $2^3 = 8$ tile types are required. In addition, 4 tiles are required to assemble the frame, as described in Sect. 5.

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 $\mathtt{OP}_1$ is the function $\mathtt{XOR}$.

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.

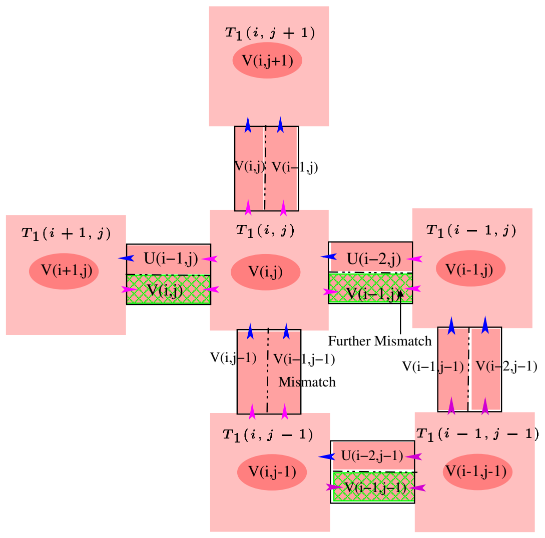


Fig. 4. Case 1.1 in the proof of Lemma 1: error in right portion $V(i-1, j-1)$ of the bottom pad of tile $T_1(i, j)$ causes a further mismatch on the right pad of tile $T_1(i, j)$

(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 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 $V(i-1, j) = U(i-2, j)\ \mathtt{OP}_1\ V(i-1, j-1)$ to compute $V(i-1, j)$ and $\mathtt{OP}_1$ is assumed to be the $\mathtt{XOR}$ 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 $U(i-1, j) = U(i-$

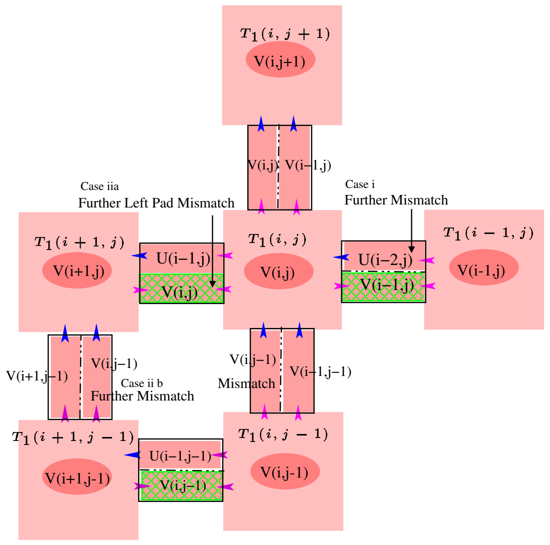


Fig. 5. Case 1.2 in the proof of Lemma 1: a further mismatch is caused by an error in the $V(i, j-1)$ portion of the bottom pad of tile $T_1(i, j)$

$2, j)$ $\mathtt{OP}_2$ $V(i-1, j-1)$ and both $U(i-2, j)$ and $V(i-1, j-1)$ have correct values. Since $V(i, j) = U(i-1, j)$ $\mathtt{OP}_1$ $V(i, j-1)$, $U(i-1, j)$ 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, $V(i, j) = U(i-1, j)$ $\mathtt{OP}_1$ $V(i, j-1)$ and $\mathtt{OP}_1$ is $\mathtt{XOR}$, 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

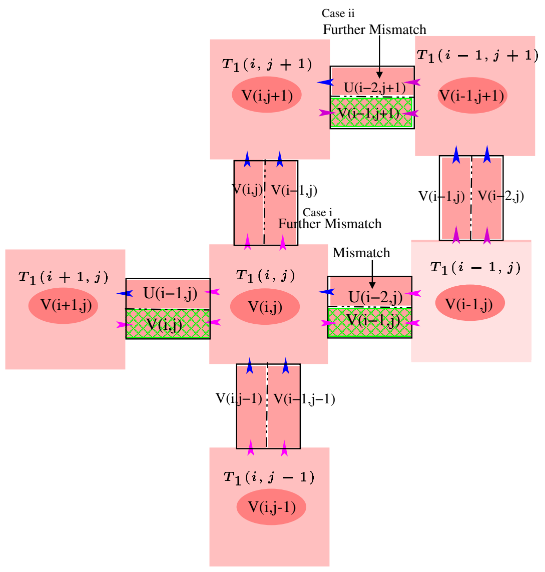


Fig. 6. Case 2.1 in the proof of Lemma 1: a further mismatch is caused by an error in the $U(i-2, j)$ portion of the right pad of tile $T_1(i, j)$

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 $V(i-1, j) = U(i-2, j)\ \texttt{OP}_1\ V(i-1, j-1)$. 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), 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.

Using the analytical methodology described in Sect. 2.2, we next calculate the error rate ϵ_1 in our two-way overlay construction. The key observation here is that the number of assemblies with one mismatch is $k_1 = 0$. In addition, since one pad mismatch is linked with one of three possible further mismatches, we have $k_2 = 2n * 3 = 6n$. This gives us

$$\begin{aligned}
\Pr(\mathcal{A}_0) &= \frac{1}{1 + k_1 e^{-G_{se}} + k_2 e^{-2G_{se}} + k_3 e^{-3G_{se}} + ... + k_n e^{-nG_{se}}} && (7)\\
&\approx \frac{1}{1 + k_2 e^{-2G_{se}}} && (8)\\
&= \frac{1}{1 + 6ne^{-2G_{se}}} && (9)\\
&\approx 1 - 6ne^{-2G_{se}}, && (10)
\end{aligned}$$

where $\mathcal{A}_0$ is the unique error-less assembly.

Again, we also have

$$\Pr(\mathcal{A}_0) = ((1 - \epsilon_1)^2)^n \approx 1 - 2n\epsilon_1. \tag{11}$$

Putting together equations 10 and 11, we have $\epsilon_1 = 3e^{-2G_{se}} = 3\epsilon^2$. Thus we have shown,

Theorem 1. *The error rate ϵ_1 for assemblies constructed from version 1 error resilient tiles is $3\epsilon^2$, where ϵ is the error rate for the corresponding assembly system with no error correction.*

Note that the growth rate $r_1 \approx \beta e^{-G_{mc}} \approx \beta e^{-2G_{se}} = \frac{\beta}{3}\epsilon_1$. Hence the growth rate depends linearly on the error rate. Recall that, in contrast, in the system with no error correction, the growth rate $r_0 \approx \beta\epsilon^2$. As such, compared with the system with no error correction, decreasing error rate in our version 1 error resilient system results in a much less decrease in the speed of assembly.

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

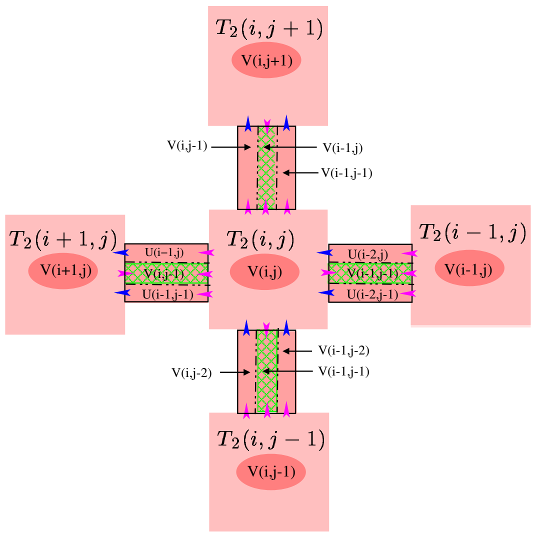


Fig. 7. Tile T_2 takes inputs $U(i-2, j)$, $U(i-2, j-1)$, $V(i-1, j-2)$ and $V(i, j-2)$; determines $V(i-1, j-1) = U(i-2, j-1)\ \mathtt{OP}_1\ V(i-1, j-2)$, $U(i-1, j-1) = U(i-2, j-1)\ \mathtt{OP}_2\ V(i-1, j-2)$, $V(i, j-1) = U(i-1, j-1)\ \mathtt{OP}_1\ V(i, j-2)$, $U(i-1, j) = U(i-2, j)\ \mathtt{OP}_2\ V(i-1, j-1)$, $V(i, j) = U(i-1, j)\ \mathtt{OP}_1\ V(i, j-1)$ and $V(i-1, j) = U(i-2, j)\ \mathtt{OP}_1\ V(i-1, j-1)$; displays $V(i, j)$

4.2 Error Analysis

For error analysis, in addition to the assumptions made in Sect. 3.2, we require that $\mathtt{OP}_2$ 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. One example instance of $\mathtt{OP}_2$ is given in Table 4.2.

Table 1. An instance of $\mathtt{OP}_2$. This binary operation can detect the incorrect value of input 1, regardless of the correctness of input 2

Input 1	Input 2	Output
0	0	0
1	0	1
0	1	0
1	1	1

The middle portions of all the four pads (top, right, left, bottom) are computed as described in the caption of Figure 7 and serve as the part to redundantly compute and compare the outputs of two neighboring tiles as shown in the figure.

Without loss of generality, we again consider only the cases where the pad binding error occurs on either the bottom pad or right pad of a tile $T_2(i, j)$. Otherwise, if the pad binding error occurs on the left pad of tile $T_2(i, j)$, then use the same below argument for tile $T_2(i + 1, j)$; likewise if the pad binding error occurs on the top pad of tile $T_2(i, j)$, use the same below argument for tile $T_2(i, j + 1)$.

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.*

Proof. Suppose a pad binding error occurs on a bottom pad or right pad of tile $T_2(i, j)$ but no further pad mismatch occurs between two neighborhood tiles which are independent of $T_2(i, j)$. We now consider a case analysis of possible pad mismatches.

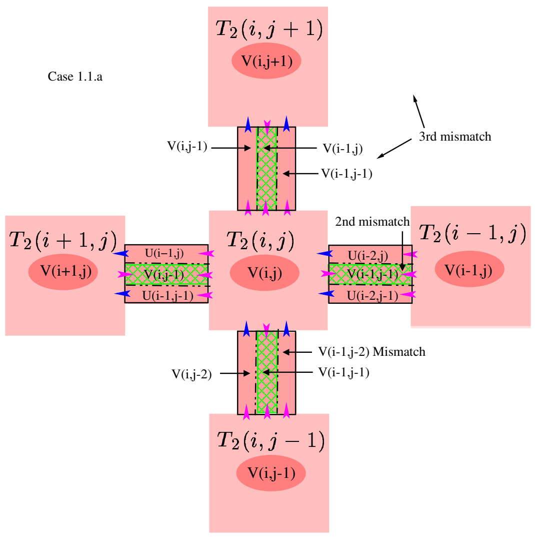


Fig. 8. Case 1.1.a in the proof of Lemma 2

(1) First consider the case where the pad binding error occurs on the $V(i - 1, j - 2)$ portion of the bottom pad of tile $T_2(i, j)$.

(1.1) Consider the case where the pad binding error is due to the *incorrect* value of the right portion $V(i - 1, j - 2)$ of the bottom pad of tile $T_2(i, j)$ (there may also be the *incorrect* value of the other portions of the bottom pad of tile $T_2(i, j)$). Further consider case **(1.1a)** (Figure 8) when there is no mismatch on the bottom portion $U(i - 2, j - 1)$

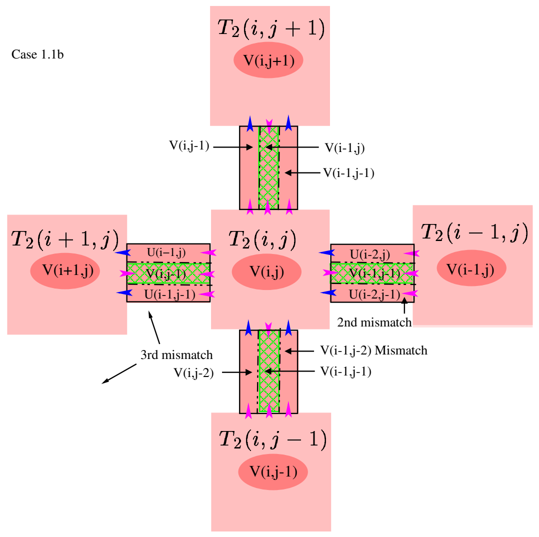


Fig. 9. Case 1.1.b in the proof of Lemma 2

of the right pad. Immediately, we have a mismatch on the portion $V(i-1, j-1)$ of the right pad of $T_2(i,j)$, since $V(i-1,j-1) = U(i-2,j-1)\ \mathtt{OP}_1\ V(i-1,j-2)$ and $\mathtt{OP}_1$ is $\mathtt{XOR}$. Furthermore, tile $T_2(i,j)$ will determine an incorrect value for the $V(i-1,j-1)$ portion of its top pad, resulting in a mismatch either on the bottom or on the right pad of $T_2(i,j+1)$. Next consider case **(1.1b)** (Figure 9) when there is a mismatch on the $U(i-2,j-1)$ portion of the right pad of $T_2(i,j)$. This will result in an incorrect value of $U(i-1,j-1)$ portion of $T_2(i,j)$'s left pad (since $\mathtt{OP}_2$ can detect the incorrect value of $U(i-2,j-1)$), leading to a further mismatch either on the right pad or on the bottom pad of $T_2(i+1,j)$.

(1.2) (Figure 10) Consider the case where the pad binding error is due to the *incorrect* value of the middle portion $V(i-1,j-1)$ of the bottom pad of tile $T_2(i,j)$, but there is a *correct* match in the right portion $V(i-1,j-2)$ of tile $T_2(i,j)$ (there may also be the *incorrect* value of the left portion $V(i,j-2)$ of the bottom pad of tile $T_2(i,j)$). Since the value of $V(i-1,j-2)$ is correct and $V(i-1,j-1)$ is determined by $U(i-2,j-1)\ \mathtt{OP}_1\ V(i-1,j-2)$ and $\mathtt{OP}_1$ is $\mathtt{XOR}$, we immediately have that there must be a mismatch on the $U(i-2,j-1)$ portion of $T_2(i,j)$'s right pad, due to the incorrect value of $U(i-2,j-1)$ portion of this pad. However, since $V(i-1,j-1) = U(i-2,j-1)\ \mathtt{OP}_1\ V(i-1,j-2)$, the value of $V(i-1,j-1)$ (right portion of its top pad) computed by $T_2(i,j)$ must be incorrect, resulting in a further mismatch either on the bottom or on the right pad of $T_2(i,j+1)$.

(1.3) Consider the case where the pad binding error is due to the *incorrect* value of the left portion $V(i,j-2)$ of the bottom pad of tile $T_2(i,j)$, but there are both *correct* matches in the right portion $V(i-1,j-2)$ and middle portion $V(i-1,j-1)$ of the bottom pad of tile $T_2(i,j)$. Further consider case **(1.3a)** (Figure 11) when there is no mismatch on the $U(i-2,j-1)$ portion of the right pad. Then $T_2(i,j)$ must compute

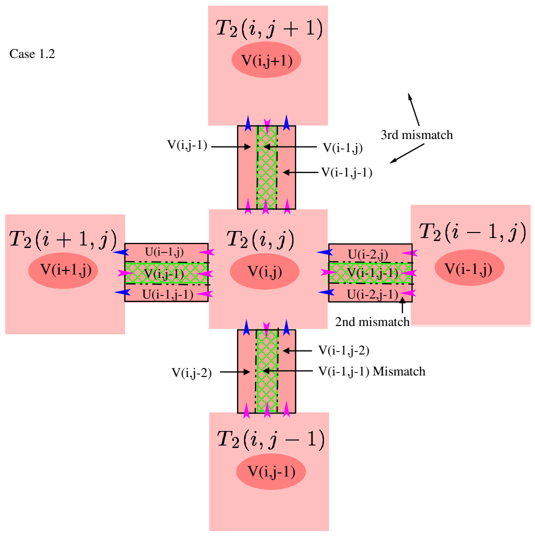


Fig. 10. Case 1.2 in the proof of Lemma 2

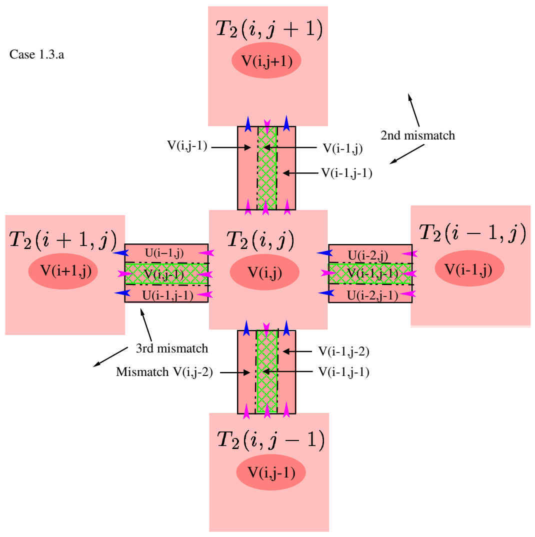


Fig. 11. Case 1.3.a in the proof of Lemma 2

a correct value for $U(i-1,j-1) = U(i-2,j-1)\ \mathtt{OP}_2\ V(i-1,j-2)$. $T_2(i,j)$ further computes both an incorrect value of $V(i,j-1)$ portion of its top pad (since $V(i,j-1) = U(i-1,j-1)\ \mathtt{OP}_1\ V(i,j-2)$) and an incorrect value for $V(i,j-1)$ portion of its left pad. The first incorrect value will result in a mismatch either on the bottom or on the right pad of $T_2(i,j+1)$. The second incorrect value will result in a mismatch either on the right or on the bottom pad of $T_2(i+1,j)$. Next consider case **(1.3b)** when there is a mismatch on the $U(i-2,j-1)$ portion of the right pad of $T_2(i,j)$. But this case cannot occur since both $V(i-1,j-1)$ and $V(i-1,j-2)$ portions of $T_2(i,j)$'s bottom pad are correct, and $V(i-1,j-1) = U(i-2,j-1)\ \mathtt{OP}_1\ V(i-1,j-2)$, where $\mathtt{OP}_1 = \mathtt{XOR}$.

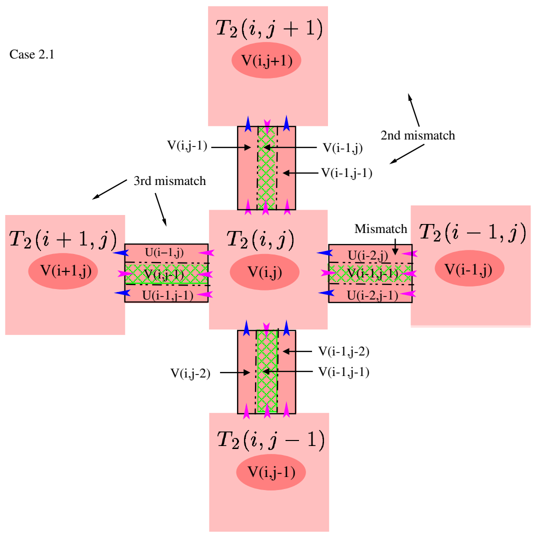


Fig. 12. Case 2.1 in the proof of Lemma 2

(2) Now consider the case where the pad binding error occurs on the right pad of $T_2(i,j)$, but there is no binding error on the bottom pad of $T_2(i,j)$.

We note that since both $V(i-1,j-2)$ and $V(i-1,j-1)$ portions of the bottom pad are correct, the $U(i-2,j-1)$ and $V(i-1,j-1)$ portions of the right pad must also be correct, so we only need to consider the case **(2.1)** (Figure 12) where the binding error is due to an incorrect value of the top portion $U(i-2,j)$ of the right pad of $T_2(i,j)$, but there is no mismatch on other portions of the right pad of $T_2(i,j)$. First note an incorrect value of $U(i-2,j)$ will result in an incorrect value of the right portion $V(i-1,j-1)$ of the top pad of $T_2(i,j)$. And this will lead to a further mismatch either between $T_2(i,j)$ and $T_2(i,j+1)$ or between $T_2(i,j+1)$ and $T_2(i-1,j+1)$. Next note that $T_2(i,j)$ must compute an incorrect value for the $U(i-1,j)$ portion of its left pad, resulting in yet another mismatch either between $T_2(i,j)$ and $T_2(i+1,j)$ or between $T_2(i+1,j)$ and $T_2(i+1,j+1)$.

We have thus proven that a mismatch in the right or bottom pad of $T_2(i,j)$ results in at least two further mismatches. And given the location of the first mismatch, the location of the second mismatch can be determined among at most three locations (between $T_2(i,j)$ and $T_2(i-1,j)$, or between $T_2(i,j)$ and $T_2(i,j+1)$, or between $T_2(i,j+1)$ and $T_2(i-1,j+1)$). Furthermore, given the locations of the first two mismatches, the location of the third mismatch can be determined among at most five locations (between $T_2(i,j)$ and $T_2(i+1,j)$, between $T_2(i+1,j)$ and $T_2(i+1,j-1)$, between $T_2(i,j)$ and $T_2(i,j+1)$, between $T_2(i,j+1)$ and $T_2(i-1,j+1)$, or between $T_2(i+1,j)$ and $T_2(i+1,j+1)$).

We again calculate the error rate ϵ_2 for our versioin 2 construction using thermodynamic analysis. The key observation here is that the number of assemblies with exactly one mismatch or exactly two mismatches are 0. In addition, since one pad mismatch is linked with a second mismatch at one of three possible locations, and each of these three second mismatch is in turn linked with a third mismatch at one of five possible locations, we have $k_3 = 2n * 3 * 5 = 30n$. As such, we have

$$\begin{aligned}\Pr(\mathcal{A}_0) &= \frac{1}{1 + k_1 e^{-G_{se}} + k_2 e^{-2G_{se}} + k_3 e^{-3G_{se}} + ... + k_n e^{-nG_{se}}} && (12)\\ &\approx \frac{1}{1 + k_3 e^{-3G_{se}}} && (13)\\ &= \frac{1}{1 + 30n e^{-3G_{se}}} && (14)\\ &\approx 1 - 30n e^{-3G_{se}}, && (15)\end{aligned}$$

where $\mathcal{A}_0$ is the unique error-less assembly.

Again, we also have

$$\Pr(\mathcal{A}_0) = ((1-\epsilon_2)^2)^n \approx 1 - 2n\epsilon_2. \qquad (16)$$

Putting together equations 15 and 16, we have $\epsilon_2 = 15e^{-3G_{se}} = 15\epsilon^3$. Thus we have shown,

Theorem 2. *The error rate ϵ_2 for assemblies constructed from version 2 error resilient tiles is $15\epsilon^3$, where ϵ is the error rate for the corresponding assembly system with no error correction.*

Note that the growth rate $r_2 \approx \beta e^{-2G_{se}} \approx (1/15)^{2/3}\beta(\epsilon_2)^{2/3}$.

Note that each pad encodes a tuple of three bits, and the values of the left pad, the top pad, the middle portion of the right pad, and the middle portion of the bottom pad each depend only on the values of the top portion and the bottom portion of the right pad and the right and left portion of the bottom pad. As such, the tile type depends on only 4 binary bits, and hence only $2^4 = 16$ tile types are required in addition to the initial frames at the bottom and to the right (requiring 5 additional tiles).

5 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 those of Winfree's [25].

We show below the construction of a binary counter and a Sierpinsky triangle. For each of them we use a total of 12 tiles, including 8 counter tiles and 4 frame tiles as shown in Figure 13 and Figure 14.

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 13 and Figure 14, we use a distinct label for each pad, emphasizing the wholeness of the pad.

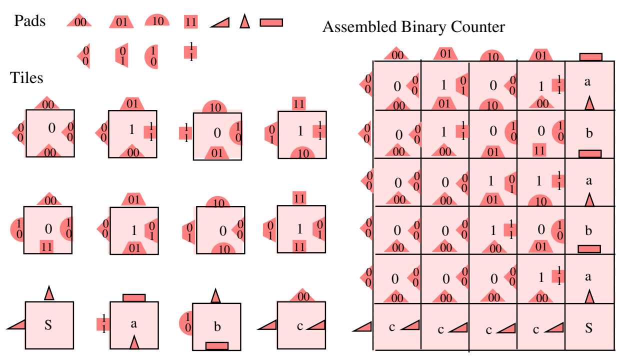


Fig. 13. The construction of a binary counter using error resilient assemblies version 1. The pads and the tile set are shown on the left and the assembled binary counter is shown on the right. The pads of strength 2 have black borders while the strength 1 pads are border-less. The seed tile is labeled with S. Tiles a, b and c are the other frame tiles

For the simulation study, we used the Xgrow simulator by Winfree [25] 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 14),
- 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 largest number of tiles assembled without any permanent error in the assembly in 50% of all test cases. The variations in the value of N are measured as we increase the 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 [25]. As suggested in [25], the experiments were performed near equilibrim, where $G_{mc} \approx 2G_{se}$, to achieve optimal error rate $\epsilon \approx 2e^{-G_{se}}$.

Figure 15 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

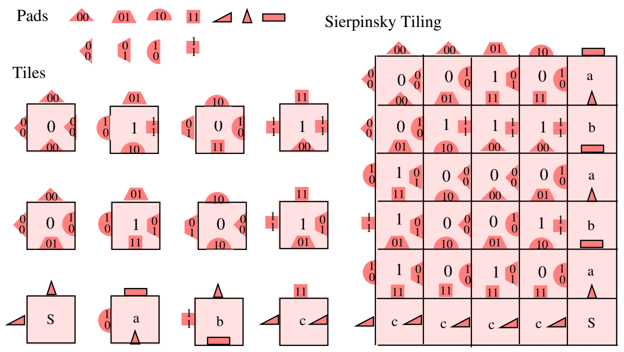


Fig. 14. The construction of a Sierpinsky triangle using error resilient assemblies version 1. The pads and the tile set are shown on the left and the assembled Sierpinsky triangle is shown on the right. The pads of strength 2 have black borders while the strength 1 pads are border-less. The seed tile is labeled with S. Tiles a, b, and c are the other frame tiles

proofreading tile set construction, while our version 2 (T_2) performs comparably to Winfree's 3×3 proofreading tile set construction.

6 Discussion

We report a theoretical design that can reduce mismatch errors in algorithmic DNA tiling without increasing the size of assembled structure. We have proved the correctness of our result using theoretical analysis and computer simulation. Next, we will further test the effectiveness of our construction using wet lab experimental demonstration. The self-assembled Sierpinsky crystals [16] and binary counters [1] provide an amiable platform for experimentally evaluating the effectiveness of our proposed methods. Another candidate system is the "4x4" tile system [29], and we have obtained some preliminary results in assembling binary counter crystals using this system.

There are also some open theoretical problems in our proposed system. First, in the proof of this paper, we require $\mathtt{OP}_1$ to be $\mathtt{XOR}$, for concreteness. However, note that our constructions apply to more general Boolean arrays in which $\mathtt{OP}_1$ is an *input sensitive operator*, *i.e.* the output changes with the change of exactly one input. Second, we note that $\mathtt{OP}_1$ and $\mathtt{OP}_2$ are both the function $\mathtt{XOR}$ for the Sierpinsky triangle but this is not true for the assembly for a binary counter of N bits, since $\mathtt{OP}_2$ is the logical $\mathtt{AND}$ 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 $\mathtt{OP}_1$ and $\mathtt{OP}_2$ are the same function such as $\mathtt{XOR}$. Finally, 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, to prove an upper bound for k.

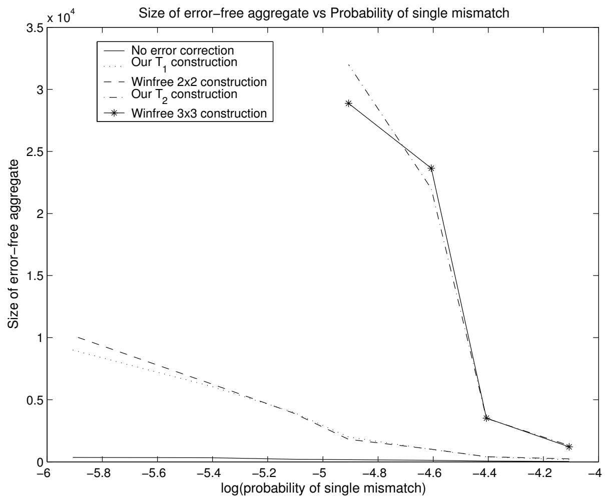


Fig. 15. A graph showing the variation of N *v.s.* increasing value of error (probability of single mismatch) ϵ

References

1. Robert Barish, Paul W. K. Rothemund, and Erik Winfree. Algorithmic self-assembly of a binary counter using DNA tiles. 2005. In preparation.
2. B. A. Bondarenko. *Generalized Pascal Triangles and Pyramids, Their Fractals, Graphs and Applications*. The Fibonacci Association, 1993. Translated from Russion and edited by R. C. Bollinger.
3. N. Chelyapov, Y. Brun, M. Gopalkrishnan, D. Reishus, B. Shaw, and L. Adleman. DNA triangles and self-assembled hexagonal tilings. *J. Am. Chem. Soc.*, 126:13924–13925, 2004.
4. H. L. Chen, Q. Cheng, A. Goel, M. D. Huang, and P. M. de Espanes. Invadable self-assembly: Combining robustness with efficiency. In *Proceedings of the 15th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 890–899, 2004.
5. H. L. Chen and A. Goel. Error free self-assembly using error prone tiles. In *DNA Based Computers 10*, pages 274–283, 2004.
6. K. Fujibayashi and S. Murata. A method for error suppression for self-assembling DNA tiles. In *DNA Based Computing 10*, pages 284–293, 2004.
7. 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.
8. 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.
9. D. Liu, M. S. Wang, Z. X. Deng, R. Walulu, and C. D. Mao. Tensegrity: Construction of rigid DNA triangles with flexible four-arm dna junctions. *J. Am. Chem. Soc.*, 126:2324–2325, 2004.
10. Dage Liu, Sung Ha Park, John H. Reif, and Thomas H. LaBean. DNA nanotubes self-assembled from triple-crossover tiles as templates for conductive nanowires. *Proc. Natl. Acad. Sci. USA*, 101:717–722, 2004.

11. C. Mao, T. H. LaBean, J. H. Reif, and N. C. Seeman. Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. *Nature*, 407:493–496, 2000.
12. 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.
13. 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.
14. P. W. K. Rothemund and E. Winfree. The program-size complexity of self-assembled squares (extended abstract). In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 459–468. ACM Press, 2000.
15. Paul W.K. Rothemund, Axel Ekani-Nkodo, Nick Papadakis, Ashish Kumar, Deborah Kuchnir Fygenson, and Erik Winfree. Design and characterization of programmable DNA nanotubes. *J. Am. Chem. Soc.*, 126:16344–16353, 2004.
16. Paul W.K. Rothemund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of DNA sierpinski triangles. *PLoS Biology 2 (12)*, 2:e424, 2004.
17. Rebecca Schulman and Erik Winfree. Programmable control of nucleation for algorithmic self-assembly. In *DNA Based Computers 10*, LNCS, 2005.
18. N. C. Seeman. DNA in a material world. *Nature*, 421:427–431, 2003.
19. J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Autonomous Studies*, pages 43–98, 1956.
20. H. Wang. Proving theorems by pattern recognition ii. *Bell Systems Technical Journal*, 40:1–41, 1961.
21. E. Winfree. Complexity of restricted and unrestricted models of molecular computation. In R. J. Lipton and E. B. Baum, editors, *DNA Based Computers 1*, volume 27 of *DIMACS*, pages 187–198. American Mathematical Society, 1996.
22. 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.
23. E. Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, 1998.
24. E. Winfree. Simulation of computing by self-assembly. Technical Report 1998.22, Caltech, 1998.
25. E. Winfree and R. Bekbolatov. Proofreading tile sets: Error correction for algorithmic self-assembly. In *DNA Based Computers 9*, volume 2943 of *LNCS*, pages 126–144, 2004.
26. E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394(6693):539–544, 1998.
27. 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.
28. H. Yan, L. Feng, T. H. LaBean, and J. H. Reif. Parallel molecular computation of pair-wise xor using DNA string tile. *J. Am. Chem. Soc.*, 125(47), 2003.
29. H. Yan, T. H. LaBean, L. Feng, and J. H. Reif. Directed nucleation assembly of DNA tile complexes for barcode patterned DNA lattices. *Proc. Natl. Acad. Sci. USA*, 100(14):8103–8108, 2003.
30. 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(5641):1882–1884, 2003.