Random Number Selection in Self-Assembly

David Doty*, Jack H. Lutz**, Matthew J. Patitz***, Scott M. Summers†, and Damien Woods‡

Abstract. We investigate methods for exploiting nondeterminism inherent within the Tile Assembly Model in order to generate uniform random numbers. Namely, given an integer range $\{0, \ldots, n-1\}$, we exhibit methods for randomly selecting a number within that range. We present three constructions exhibiting a trade-off between space requirements and closeness to uniformity.
The first selector selects a random number with probability $\Theta(\frac{1}{n})$ using $O(\log^2 n)$ tiles. The second selector takes a user-specified parameter that guarantees the probabilities are arbitrarily close to uniform, at the cost of additional space. The third selector selects a random number with probability exactly $\frac{1}{n}$, and uses no more space than the first selector with high probability, but uses potentially unbounded space.

* Department of Computer Science, Iowa State University, Ames, IA 50011, USA. ddoty@iastate.edu

** Department of Computer Science, Iowa State University, Ames, IA 50011, USA. lutz@cs.iastate.edu. This research was supported in part by National Science Foundation Grants 0652569 and 0728806 and by the Spanish Ministry of Education and Science (MEC) and the European Regional Development Fund (ERDF) under project TIN2005-08832-C03-02.

*** Department of Computer Science, Iowa State University, Ames, IA 50011, USA. mpatitz@cs.iastate.edu.

† Department of Computer Science, Iowa State University, Ames, IA 50011, USA. summers@cs.iastate.edu. This author's research was supported in part by NSF-IGERT Training Project in Computational Molecular Biology Grant number DGE-0504304.

‡ Department of Computer Science & Artificial Intelligence, University of Seville, Spain. dwoods@us.es. Supported by Junta de Andalucía grant TIC-581.

1 Introduction

The development of DNA tile self-assembly has moved nanotechnology closer to the goal of engineering useful systems that assemble themselves from molecular components. Since Seeman's pioneering work in the 1980s [17], many laboratory experiments have shown that DNA tiles can be designed to spontaneously assemble with one another into desired structures [14]. As physical and mathematical error-suppression techniques improve [3, 6, 11, 18, 20, 23], this molecular programming of matter will become practical at ever larger scales.

The Tile Assembly Model, developed by Winfree [16, 22], is a mathematical model of DNA tile self-assembly that enables us to explore the potentialities and limitations of this kind of molecular programming. The model deliberately oversimplifies the physical realities of self-assembly, but Winfree proved that it is Turing universal [22]. This implies that self-assembly can be algorithmically directed. The simplicity of the Tile Assembly Model allows us to focus on the interactions between computation and geometry that are characteristic of algorithmic self-assembly [1, 2, 4, 9, 10, 12, 13, 15, 16, 19].

One prominent feature of self-assembly is its inherent nondeterminism, and the Tile Assembly Model represents this well. At any given time, there may be many locations where a tile might attach itself to the growing assembly, and, even at a single location, there may be more than one type of tile that can attach itself. When we are designing a tile assembly system that is supposed to result in a specified terminal assembly, regardless of the particular sequence in which tiles attach themselves, this nondeterminism is a conceptual difficulty that we overcome with tools like local determinism [19] and modularity [9]. In other situations, such as when we are using randomized algorithms to reduce the number of tile types required for a self-assembly [5, 7] or when we are simulating a system that is itself nondeterministic, the Tile Assembly Model's nondeterminism is a programming resource.

This paper concerns the problem of using the nondeterminism inherent in self-assembly to implement a nondeterministic choice among an arbitrary number of options. That is, we consider the problem of designing a tile assembly system that, given a positive integer n, chooses an integer $r \in \{0, \ldots, n-1\}$.

As stated so far, this problem is not interesting. We simply count down from $n-1$ to 0, tossing a "coin" (implemented by having either one of two tiles attach at a suitable location) after each decrement to decide whether to stop or keep counting down. This solves the *logical* problem of nondeterministic choice, but, if n is large and the "coins" are fair and independent, it is nearly certain that r will be much larger than 0. This is not satisfactory if we want to use the nondeterministic choice for a randomized algorithm or a useful simulation.

So our actual problem treats nondeterminism probabilistically. We assume that, at each time and location in self-assembly, all the tile types than *can* attach at that location are *equally likely* to do so, and that the "choices" made at different locations are independent. We then seek to design a tile assembly system that, given a positive integer n (represented in binary as a seed assembly), chooses an integer $r \in \{0, \ldots, n-1\}$ in such a way that the outcomes

$r = 0, r = 1, \ldots, r = n - 1$ are all equally likely, or nearly so. (We are *not* constructing pseudorandom generators in the sense of complexity theory or cryptography. Pseudorandom generators expand a short, truly random "seed" into a longer pseudorandom string. Our *random number selectors* use at least as much randomness as they produce.)

We present three solutions to this problem. Our first random number selector has 324 tile types and uses only $O(\log^2 n)$ space to select the number r, but the probabilities are only $\Theta(\text{uniform})$, in the sense that the probability that r is selected is between $\frac{1}{2n}$ and $\frac{2}{n}$.

Our second random number selector has 821 tile types and takes as input both n and a user-supplied precision parameter t. The probability that it selects r is then between $\frac{1}{n} - \frac{1}{t}$ and $\frac{1}{n} + \frac{1}{t}$. This is very nearly uniform if $t \gg n$, but this added uniformity comes at a price, namely that the selector uses $O(t^2)$ space.

Our third selector has 100 tile types and selects r with probability *exactly* $\frac{1}{n}$. With high probability it uses only as much space as our first selector, but there is *no* absolute bound on the space that it takes. (This selector is a self-assembly implementation of von Neumann's *rejection method* [8, 21].)

We expect each of these random number selectors to be useful in some self-assemblies. Taken together, our selectors suggest that there is a tradeoff between how uniform a random number selector is and how much space it takes. We conjecture that this tradeoff is real, and not merely an artifact of our constructions. Proving or disproving this conjecture is an open problem.

2 The Tile Assembly Model

This section provides a very brief overview of the TAM. See [10, 15, 16, 22] for other developments of the model. Our notation is that of [10]. We work in the 2-dimensional discrete space $\mathbb{Z}^2$. We write U_2 for the set of all *unit vectors*, i.e., vectors of length 1 in $\mathbb{Z}^2$.

Intuitively, a tile type t is a unit square that can be translated, but not rotated, having a well-defined "side $\boldsymbol{u}$" for each $\boldsymbol{u} \in U_2$. Each side $\boldsymbol{u}$ of t has a "glue" of "color" $\text{col}_t(\boldsymbol{u})$ - a string over some fixed alphabet Σ - and "strength" $\text{str}_t(\boldsymbol{u})$ - a natural number - specified by its type t. Two tiles t and t' that are placed at the points $\boldsymbol{a}$ and $\boldsymbol{a} + \boldsymbol{u}$ respectively, *bind* with *strength* $\text{str}_t(\boldsymbol{u})$ if and only if $(\text{col}_t(\boldsymbol{u}), \text{str}_t(\boldsymbol{u})) = (\text{col}_{t'}(-\boldsymbol{u}), \text{str}_{t'}(-\boldsymbol{u}))$.

Given a set T of tile types, an *assembly* is a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$. An assembly is τ-*stable*, where $\tau \in \mathbb{N}$, if it cannot be broken up into smaller assemblies without breaking bonds whose strengths sum to at least τ.

Self-assembly begins with a *seed assembly* σ and proceeds asynchronously and nondeterministically, with tiles adsorbing one at a time to the existing assembly in any manner that preserves stability at all times. A *tile assembly system* (*TAS*) is an ordered triple $\mathcal{T} = (T, \sigma, \tau)$, where T is a finite set of tile types, σ is a seed assembly with finite domain, and τ is the temperature. Note that for all TAS's in this paper, $\tau = 2$. An *assembly sequence* in a TAS $\mathcal{T} = (T, \sigma, \tau)$ is a (possibly

infinite) sequence $\boldsymbol{\alpha} = (\alpha_i \mid 0 \leq i < k)$ of assemblies in which $\alpha_0 = \sigma$ and each α_{i+1} is obtained from α_i by the "τ-stable" addition of a single tile.

Proofs of correctness of all of the constructions in this paper are omitted from the current version.

3 A Θ(uniform) Selector

The first selector that we implement chooses a number at random between $s \in \mathbb{N}$ ("start") and $e \in \mathbb{Z}^+$ ("end"), where $s < e$ and both are encoded in binary in the seed as shown in the bottom row of Figure 3.

The construction is shown in Figure 3. The tile set implements the random binary search algorithm SELECTOR. It is written such that each numbered line corresponds exactly to a row in Figure 3. The identifier ¢, wherever it appears, indicates the result of a fair coin flip, with result H or T. Each evaluation of ¢ is independent of the others.

```
SELECTOR({s, ..., e})
1  s' ← s + 1
2  while s' < e
3        do sum ← s + e
4           mid ← sum >> 1          ▷ divide sum by 2 by bit shifting
5           if ¢ = H
              then s ← mid + 1      ▷ choose right subinterval
              else  e ← mid         ▷ choose left subinterval
6           s' ← s + 1
7  if ¢ = H then return s else return e
```

Figure 3 depicts a selector that randomly chooses an integer between s and e, such that, letting $n = |\{s, \ldots, e\}|$, we have $\frac{1}{2} \cdot \frac{1}{n} \leq \Pr(a) \leq 2 \cdot \frac{1}{n}$ for each $a \in \{s, \ldots, e\}$. The bottom row encodes s and e in binary. Note that the horizontal arrows indicate direction of growth. Each gray row is a comparison row whose result indicates whether the binary search is complete. The rows between each adjacent pair of gray rows implement the main loop of the random binary search. The interval $\{s, \ldots, e\}$ is subdivided into two subintervals, with the left subinterval always chosen to be one larger if $\{s, \ldots, e\}$ has an odd number of elements. One of the subintervals is randomly chosen to become the new interval, by replacing either s or e with a value near their midpoint. When $\{s, \ldots, e\}$ has either 1 or 2 elements, the loop terminates, and one of s or e is randomly chosen.

In the technical appendix, we show that $n \in \mathbb{N}$ and $r \in \{1, \ldots, n\}$,

$$\frac{1}{2n} \leq \frac{1}{2^{\lfloor \log_2 n \rfloor + 1}} \leq \Pr[\text{SELECTOR}(1, n) = r] \leq \frac{1}{2^{\lfloor \log_2 n \rfloor}} < \frac{2}{n}.$$

The asymmetry of the interval sizes is what prevents the tile set from achieving perfect uniformity. Intuitively, if the left subinterval has length l and the right has length $l+1$, then we should pick the right with probability $\frac{l+1}{2l+1}$, as opposed

to $\frac{1}{2}$ as our algorithm does. It may seem at first glance that by randomly selecting which interval is larger, we could smooth out the probabilities and approach a uniform distribution in the limit as the interval size approaches infinity.

However, this does not work. Roughly speaking, selecting r from $\{0, \ldots, n-1\}$ using this method, we can express $\Pr[r] = \prod_{i=1}^{\approx \log n} p_i$, where each p_i represents the probability that r is in the subinterval picked during the i^{th} stage. If n is a power of 2, then each p_i will be $\frac{1}{2}$, and the selector (both that just described and the selector constructed earlier in this section) will select r with probability exactly $\frac{1}{n}$. If n is not a power of 2, then *at most* one of the p_i's will be unequal to $\frac{1}{2}$; this will occur precisely at the stage when the interval length is odd and r is the middle element, which can happen at most once in the course of the algorithm. In the case of $r = 1$ or $r = n - 2$, when this occurs, the interval will have length 3, so p_i will be equal to $\frac{1}{2}$, when in fact it ought to be $\frac{2}{3}$ to achieve uniformity. Therefore, no matter how large n is, $r = 1$ will be selected with probability a constant factor away from uniform.

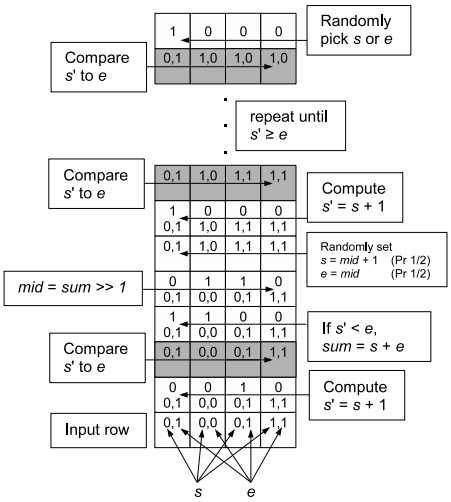

Fig. 1: Θ(uniform) selector

In the next section we introduce a random selector that allows the user to *control* the desired closeness to uniformity as a parameter, trading off space for error in approximating uniformity.

4 A Controllable-Error Selector

Unfortunately, the bounds of Theorem 1 are tight, in the sense that for certain n and $r \in \{0, \ldots, n-1\}$, $\Pr[r]$ can be nearly twice or half of $\frac{1}{n}$.

In this section, we implement the self-assembly version of a random number generator whose deviation from uniform can be controlled by a user-specified parameter. More precisely, given an upper bound $n \in \mathbb{N}$, and a precision parameter $t \in \mathbb{N}$, the tile set generates a random number $r \in \{0, \ldots n-1\}$ with probability approaching $\frac{1}{n}$ as $t \to \infty$. We first outline the algorithm.

The following algorithm is one of the more intuitive methods of generating a random number from an arbitrary range, using flips of an unbiased coin, although the number is not generated with perfect uniformity.

MSB stands for most significant bit, and LSB stands for least significant bit.

```
CONTROLLABLE-SELECTOR(n,t)
1  Uniformly at random choose m ∈ {0, ..., t − 1}
2  return MOD(m, n)

MOD(m,n)
   ▷ compute m mod n in binary
1   Line up MSBs of m and n by shifting n to the left
    sm ← the integer represented by the ⌊log n⌋ + 1 MSBs of m ("small m")
2   while n's LSB lies to the left of m's LSB
        do if sm ≥ n
3             then sm ← sm − n
              concatenate the bit of m to the right of n's LSB to the end of sm
4             shift n one bit to the right relative to sm and m
5  if sm ≥ n
6     then sm ← sm − n
7  return sm
```

As before, we have generally numbered the lines corresponding to rows in the tile assembly. However, in this case, lines 2 and 4 are implemented in the same row, since it is possible to shift n to the right by one bit and simultaneously compare it to sm. This occurs on all comparisons except for the first check of the loop condition, immediately after n has been shifted *left* to line it up with the MSB of m. Also, line 3 represents the row that subtracts $sm - n$ if $sm \geq n$, or simply copies sm otherwise, although the copying does not appear as an explicit line in the pseudocode. Finally, line 1 is actually implemented as a series of rows as shown in Figure 3, but we express it as a single instruction in the pseudocode.

Our construction takes as inputs two positive integers, n and t, which we encode in a seed row. The construction first grows a number of rows upward, each having a width one greater than the previous, until a row having a width of exactly $4t - 1$ is formed. Finally, the top most row assembles in which, at each location, one of exactly two tile types is randomly selected to bind. This effectively generates a random number m in the set $\left\{0, \ldots 2^{4t-1} - 1\right\}$ with a uniform distribution. Then, the subsequent rows to attach implement step 2 in CONTROLLABLE-SELECTOR The resulting modulus is encoded in the top most row of the construction and is the number r.

We now present a more detailed discussion of this construction.

First Stage: In the initial stage, our goal is to take as input n and t, and (uniformly) produce a random number, say m, in the range $\left\{0, \ldots 2^{4t-1} - 1\right\}$. We will then feed the numbers m and n to the second stage of the construction. Our input is encoded in a seed row whose length k is the greater of $\lfloor \log n \rfloor + 1$ and $\lfloor \log t \rfloor + 1$ The seed row is the bottom most row in Figure 2.

1. On top of the seed row, we use a (zig-zagging) Rothemund-Winfree binary subtractor [16] that counts down from t while "blindly" passing the value of n up through the subsequent rows. This gives us a k by $2t + 1$ rectangle.

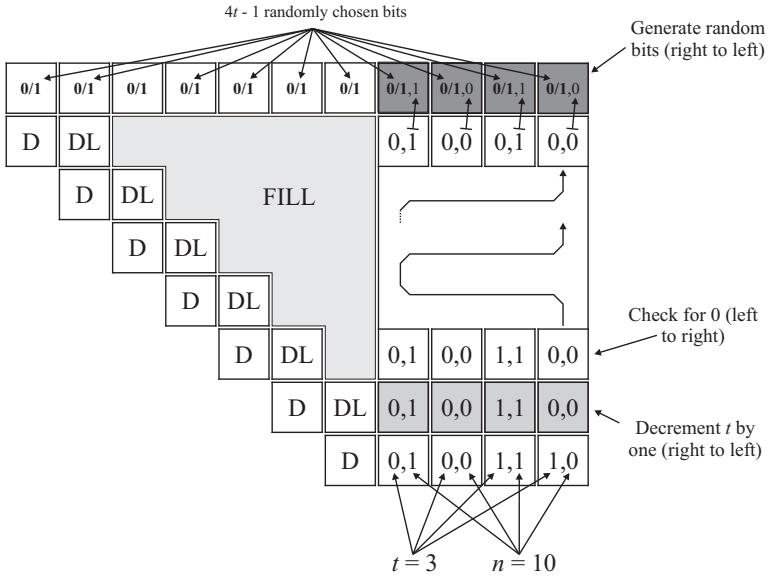

Fig. 2: The first stage of the construction. The "direction" of self-assembly for each row is either right-to-left or left-to-right (as noted in parentheses).

2. We use filler tiles (labeled 'D', 'DL' and 'FILL' in Figure 2) to self-assemble a right-triangle to the left of the rectangle that was built in the previous step.
3. (line 1 of the pseudocode for CONTROLLABLE-SELECTOR) The top most row of the construction is $4t-1$ tiles long and self-assembles from right to left such that, for each of the $4t-1$ locations in the row, there are two possible tile types that can attach - one tile representing a 0 and the other a 1. The choice of each tile that attaches in this row can be made independently of its predecessor (i.e., the tile immediately to its right). This results in the generation of a random number m in the range $\{0, \ldots 2^{4t-1}-1\}$ with uniform distribution.

The top of the last row of the initial stage of the construction encodes (as output) the number n (dark grey tiles in Figure 2) in the right most bits(n) tiles along with the number m (here we allow leading zeros).

Second Stage: (Line 1 of the pseudocode for MOD) The second stage of the construction takes as input m and n (output from the previous stage), and shifts (all of the bits of) n to the left so that the MSB of both m and n line up.

1. Every other row (starting with the first row) in this stage of the construction sends a "request-a-shift" signal from left-to-right only if the MSB of both m and n do not line up. For example, the second (from the

bottom most) row in Figure 3 self-assembles left-to-right and carries the first request-a-shift signal.

2. On top of each row that carries a request-a-shift signal, a row self-assembles right-to-left in which each bit of n is shifted once to the left. The third (from the bottom most) row in Figure 3 is the first row that carries out a shift operation. For shift rows, as self-assembly proceeds right-to-left, the position of the MSB of n is compared with that of m, and if they line up, then a subsequent request-a-shift signal is not sent. It is at this point that the third, and final stage of the construction is carried out.

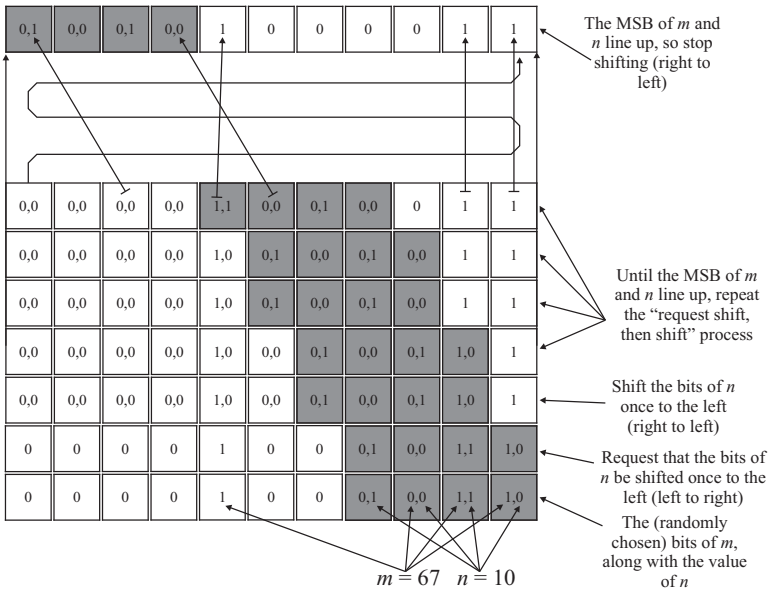

Fig. 3: The second stage of the construction. The "direction" of self-assembly for each row is either right-to-left or left-to-right (as noted in parentheses).

Third Stage: The third stage of the construction performs the task of calculating $m \mod n$ (step 2 in CONTROLLABLE-SELECTOR) and then (as a final step) self-assembles a row which represents only that value as the final output of the construction. The third stage begins with a row of tiles which encode the values of m and n such that each of their most significant bits are aligned (represented in the leftmost tile of the row). Throughout this discussion, "small m" will refer to the bits of m that begin with the leftmost and continue right to the location of the rightmost bit of n in that row (these positions are represented by the dark colored tiles in Figure 3). The third stage proceeds as follows.

1. (line 2 of the pseudocode for MOD) Self-assemble a row from left-to-right which performs an initial comparison of the values of "small m" and n.
2. (line 3 of the pseudocode for MOD) Self-assemble the next row from right to left which performs one of the following functions based on the comparison from the previous step.
   (a) If $n >$ "small m," send a signal requesting that (all of the bits of) n be shifted one position to the right.
   (b) Else, subtract the value of n from "small m." Note that this will result in a new value of "small m."
3. (line 4 of the pseudocode for MOD) Now self-assemble a row from left-to-right which shifts the value of n one position to the right (shifting 0 bits in from the left) and keeping the current value of "small m" in the same position. Note that this row also performs a comparison against the newly shifted value of n and the current value of "small m." The latter will now extend one bit further to the right than it previously did.
4. (lines 2 and 4 of the pseudocode for MOD) Continue the self-assembly of rows which perform the previous two steps in a loop until the value of n has shifted far enough to the right so that its LSB is aligned with that of m.
5. (line 6 of the pseudocode for MOD) Only if the last comparison resulted in $n <$ "small m," self-assemble rows to do one more subtraction of n from "small m," and a final comparison (as done above).
6. (line 7 of the pseudocode for MOD) The remaining value of "small m" now represents the result of $m \mod n$. The final step in the third stage of the construction self-assembles row from right-to-left which represents only that remainder, and thus the final output, $r \in \{0, \ldots, n-1\}$ of the construction.

In the technical appendix, we prove that, for all $n \in \mathbb{N}$, and $r \in \{0, \ldots, n-1\}$,

$$\lim_{t \to \infty} \Pr\left[\textsc{Controllable-Selector}(n, t) = r\right] = \frac{1}{n}.$$

5 An Exactly Uniform Selector

Even though the selector of Section 4 asymptotically approaches a uniform distribution as the precision parameter t grows to infinity, for certain n it deviates slightly from uniform.

In this section we outline a construction of a random number selector with the property that it selects a random number in the range $\{0, \ldots, n-1\}$ with probability exactly $\frac{1}{n}$. The catch is that while the selector has a very low expected number of tiles that need to be used, with some small probability, an unbounded number of tiles could be required, making this selector unsuitable for applications in which space constraints must absolutely be enforced.

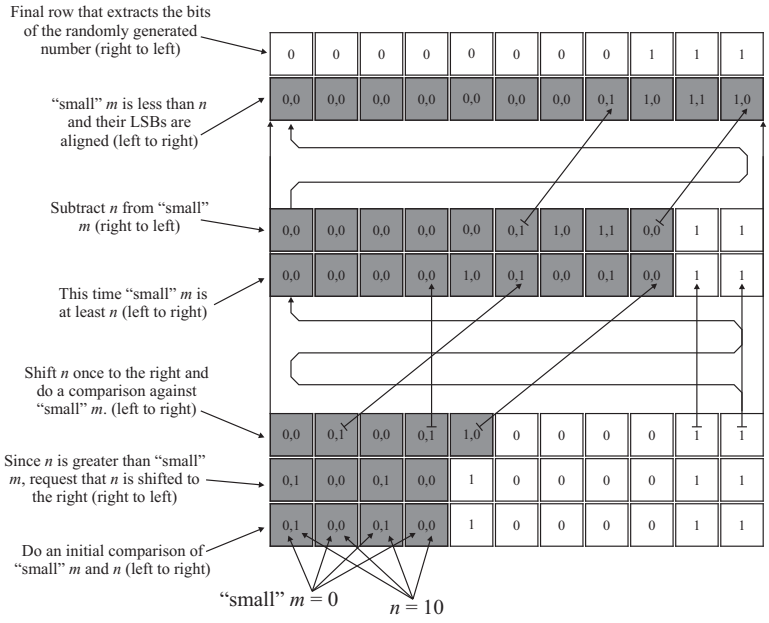

Fig. 4: The third stage of the construction The "direction" of self-assembly for each row is either right-to-left or left-to-right (as noted in parentheses).

The construction is shown in Figure 5. The tile set implements the following algorithm. The random element r is generated by selecting t bits uniformly and independently at random, where t is the length of the binary expansion of n.

UNIFORM-SELECTOR(n)

1 $r \leftarrow$ random element of $\{0, 1, \ldots, 2^{\lfloor \log n \rfloor + 1} - 1\}$
2 **while** $r \geq n$
3 **do** $r \leftarrow$ random element of $\{0, 1, \ldots, 2^{\lfloor \log n \rfloor + 1} - 1\}$
4 **return** r

Since the element of $\{0, 1, \ldots, 2^{\lfloor \log n \rfloor + 1} - 1\}$ is selected with uniform probability, then conditioned on that element being less than n, the probability each element $r \in \{0, \ldots, n-1\}$ is equal, i.e., $\frac{1}{n}$. Furthermore, since the next power of two greater than n is at most $2n$, the probability that $r \geq n$ is at most $\frac{1}{2}$, whence the expected number of iterations i is a geometric random variable with expected value at most 2, subject to the tail bound, for all $k \in \mathbb{N}, \Pr[i > k] \leq 2^{-k}$.

A selector that randomly chooses an integer between 0 and $n-1$, where each element r in the interval has probability $\frac{1}{n}$. It simply chooses r between 0 and the next power of 2 above n and outputs r if $r < n$. The selector will use a small

number of rows with high probability, but may potentially use an unbounded number of rows.

Since two rows are used per iteration, so allowing, for instance, 100 rows (plus the two for the seed and final row) ensures that sufficient room will exist to generate r with probability at least $1 - 2^{-50}$.

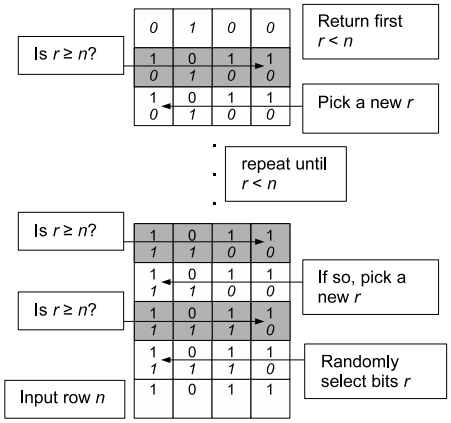

Fig. 5: The uniform selector

6 Conclusion

We have introduced variants of a random number selector in the Tile Assembly Model, powered by the randomness inherent in nondeterministic tile sets. They allow for a more algorithmic control of randomness than by hard-coding probabilities by fixed tile concentrations.

The third selector is not entirely unlike the procedure used to generate random integers from random bits used in many standard libraries; for instance, the method `Random.nextInt` in the Java standard library. Why then have we bothered to describe the first two selectors, which are provably different from uniform? In a programming language, a "ZPP" algorithm that almost certainly takes a small amount of time, but may rarely use more time, is not a problem except perhaps in critical real-time systems. However, in a tile assembly system, consisting of possibly billions of tiles, the presence of even a single region that uses too much space will destroy the correctness of the entire assembly. In such situations, it may be preferable to guarantee that space bounds are adhered to at the cost of a slight deviation from uniformity.

References

1. L. Adleman, Q. Cheng, A. Goel, and M.-D. Huang. Running time and program size for self-assembled squares. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of Computing*, pages 740–748, New York, NY, USA, 2001. ACM.
2. F. Becker, I. Rapaport, and E. Rémila. Self-assembling classes of shapes with a minimum number of tiles, and in optimal time. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 45–56, 2006.
3. H.-L. Chen and A. Goel. Error free self-assembly with error prone tiles. In *Proceedings of the 10th International Meeting on DNA Based Computers*, 2004.
4. E. D. Demaine, M. L. Demaine, S. P. Fekete, M. Ishaque, E. Rafalin, and D. L. S. Robert Schweller. Staged self-assembly: Nanomanufacture of arbitrary shapes with o(1) glues. *Natural Computing*. to appear. Preliminary version appeared in Proceedings of the 13th International Meeting on DNA Computing (DNA13), Memphis, Tennessee, June 4-8, 2007, pp. 46-55.

5. D. Doty. Randomized self-assembly for exact shapes. Technical Report 0901.1849, Computing Research Repository, 2009.
6. K. Fujibayashi, D. Y. Zhang, E. Winfree, and S. Murata. Error suppression mechanisms for dna tile self-assembly and their simulation. *Natural Computing*. to appear.
7. M.-Y. Kao and R. T. Schweller. Randomized self-assembly for approximate shapes. In L. Aceto, I. Damgrd, L. A. Goldberg, M. M. Halldrsson, A. Inglfsdttir, and I. Walukiewicz, editors, *International Colloqium on Automata, Languages, and Programming (ICALP)*, volume 5125 of *Lecture Notes in Computer Science*, pages 370–384. Springer, 2008.
8. D. E. Knuth. *The Art of Computer Programming, volume 2: Seminumerical Algorithms*. Addison-Wesley, 1997.
9. J. I. Lathrop, J. H. Lutz, M. J. Patitz, and S. M. Summers. Computability and complexity in self-assembly. In *Proceedings of The Fourth Conference on Computability in Europe (Athens, Greece, June 15-20, 2008)*, 2008.
10. J. I. Lathrop, J. H. Lutz, and S. M. Summers. Strict self-assembly of discrete Sierpinski triangles. *Theoretical Computer Science*, 410:384–405, 2009.
11. U. Majumder, T. H. LaBean, and J. H. Reif. Activatable tiles for compact error-resilient directional assembly. In *13th International Meeting on DNA Computing (DNA 13), Memphis, Tennessee, June 4-8, 2007.*, 2007.
12. M. J. Patitz and S. M. Summers. Self-assembly of decidable sets. In *Proceedings of The Seventh International Conference on Unconventional Computation (Vienna, Austria, August 25-28, 2008)*, 2008.
13. M. J. Patitz and S. M. Summers. Self-assembly of discrete self-similar fractals (extended abstract). In *Proceedings of The Fourteenth International Meeting on DNA Computing (Prague, Czech Republic, June 2-6, 2008). To appear.*, 2008.
14. P. W. Rothemund, N. Papadakis, and E. Winfree. Algorithmic self-assembly of dna sierpinski triangles. *PLoS Biology*, 2(12), 2004.
15. P. W. K. Rothemund. *Theory and Experiments in Algorithmic Self-Assembly*. PhD thesis, University of Southern California, December 2001.
16. P. W. K. Rothemund and E. Winfree. The program-size complexity of self-assembled squares (extended abstract). In *STOC*, pages 459–468, 2000.
17. N. Seeman. Nucleic-acid junctions and lattices. *Journal of Theoretical Biology*, 99:237–247, 1982.
18. D. Soloveichik and E. Winfree. Complexity of compact proofreading for self-assembled patterns. In *The eleventh International Meeting on DNA Computing*, 2005.
19. D. Soloveichik and E. Winfree. Complexity of self-assembled shapes. In *SIAM Journal on Computing 36*, pages 1544–1569, 2007.
20. T. L. Urmi Majumder, Sudheer Sahu and J. H. Reif. Design and simulation of self-repairing DNA lattices. In *DNA Computing: DNA12*, volume 4287 of *Lecture Notes in Computer Science*. Springer-Verlag, 2006.
21. J. von Neumann. Various techniques for use in connection with random digits. In *von Neumann's Collected Works*, volume 5, pages 768–770. Pergamon, 1963.
22. E. Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.
23. E. Winfree and R. Bekbolatov. Proofreading tile sets: Error correction for algorithmic self-assembly. In J. Chen and J. H. Reif, editors, *DNA*, volume 2943 of *Lecture Notes in Computer Science*, pages 126–144. Springer, 2003.

7 Technical Appendix

7.1 Analysis of SELECTOR

Theorem 1. *For all $n \in \mathbb{N}$ and $r \in \{1, \ldots, n\}$,*

$$\frac{1}{2n} \leq \frac{1}{2^{\lfloor \log_2 n \rfloor + 1}} \leq \Pr[\textsc{Selector}(1, n) = r] \leq \frac{1}{2^{\lfloor \log_2 n \rfloor}} < \frac{2}{n}$$

Proof. Let $2^{i-1} < n \leq 2^i$, $i \in \mathbb{N}$. Immediately after k coin flips we are working within one of 2^k subintervals of $[1, n]$ that are non-overlapping and cover all of $[1, n]$. The leftmost subinterval, denoted $[1, n_k]$, has maximal size (number of elements) out of the 2^k sub-intervals, which can be shown by the following simple inductive argument. For $k = 1$, we have two sub-intervals and the leftmost is maximal since

$$\left\lfloor \frac{1+n}{2} \right\rfloor \geq n - \left\lfloor \frac{1+n}{2} \right\rfloor \tag{1}$$

If we assume that the leftmost interval $[1, n_k]$ is maximal at flip k, then by letting $n = n_k$ in the inequality (1), the leftmost interval is maximal at flip $k + 1$.

So a maximal length sequence of coin flips, over all such valid sequences, is given by a sequence of 1's (we choose the left sub-interval at each coin flip). We next show that this maximal number of coin flips is no more than $\lfloor \log_2 n \rfloor + 1$. The leftmost interval $[1, n_k]$ has size

$$n_k = \left\lfloor \frac{1 + n_{k-1}}{2} \right\rfloor = \left\lceil \frac{n_{k-1}}{2} \right\rceil \tag{2}$$

$$= \begin{cases} \frac{n_{k-1}}{2} & n_{k-1} \text{ even} \\ \frac{n_{k-1}+1}{2} & n_{k-1} \text{ odd} \end{cases} \tag{3}$$

Assuming that n_k is odd for all k (to give an upperbound on the number of flips), we have $n_{\lfloor \log n \rfloor + 1} = 1$. This gives

$$\frac{1}{2^{\lfloor \log_2 n \rfloor + 1}} \leq \Pr[\textsc{Selector}(1, n) = r]$$

Similarly, using (1), it can be shown that at flip k, the rightmost subinterval is of minimal size over all 2^k possible subintervals. A lowerbound of $\log_2 n$ on the number of flips is found when n_k is even for all k in (3), and this lowerbound is reached when $n = 2^i$, $i \in \mathbb{N}$. Thus

$$\Pr[\textsc{Selector}(1, n) = r] \leq \frac{1}{2^{\lfloor \log_2 n \rfloor}}$$

.

7.2 Analysis of Controllable-Selector

Theorem 2. *For all $n \in \mathbb{N}$, and $r \in \{0, \ldots, n-1\}$,*

$$\lim_{t \to \infty} \Pr\left[\textsc{Controllable-Selector}(n, t) = r\right] = \frac{1}{n}$$

Proof. First, fix $t \in \mathbb{N}$, and note that there are either $\left\lfloor \frac{t}{n} \right\rfloor$ or $\left\lceil \frac{t}{n} \right\rceil$ ways to choose a number congruent to r (modulo n) from the set $\{0, \ldots, t-1\}$, whence

$$\begin{aligned}
\Pr\left[\textsc{Controllable-Selector}(n, t) = r\right] &\leq \frac{\left\lceil \frac{t}{n} \right\rceil}{t} \\
&\leq \frac{\frac{t}{n} + 1}{t} \\
&= \frac{1}{n} + \frac{1}{t}.
\end{aligned}$$

Similarly,

$$\Pr\left[\textsc{Controllable-Selector}(n, t) = r\right] \geq \frac{1}{n} - \frac{1}{t}.$$

It follows, by the Squeeze theorem, that

$$\lim_{t \to \infty} \Pr\left[\textsc{Controllable-Selector}(n, t) = r\right] = \frac{1}{n}.$$