Staged Self-Assembly and Polyomino Context-Free Grammars*

Andrew Winslow

Department of Computer Science, Tufts University,
Medford, MA 02155, USA,
awinslow@cs.tufts.edu

Abstract. Previous work by Demaine et al. (2012) developed a strong connection between smallest context-free grammars and staged self-assembly systems for one-dimensional strings and assemblies. We extend this work to two-dimensional polyominoes and assemblies, comparing staged self-assembly systems to a natural generalization of context-free grammars we call *polyomino context-free grammars (PCFGs)*.
We achieve nearly optimal bounds on the largest ratios of the smallest PCFG and staged self-assembly system for a given polyomino with n cells. For the ratio of PCFGs *over* assembly systems, we show that the smallest PCFG can be an $\Omega(n/\log^3 n)$-factor larger than the smallest staged assembly system, even when restricted to square polyominoes. For the ratio of assembly systems over PCFGs, we show that the smallest staged assembly system is never more than a $O(\log n)$-factor larger than the smallest PCFG and is sometimes an $\Omega(\log n/\log\log n)$-factor larger.

Keywords: aTAM, biocomputing, combinatorial optimization, formal languages, hierarchical self-assembly, tile assembly, 2HAM

1 Introduction

In the mid-1990s, the Ph.D. thesis of Erik Winfree [14] introduced a theoretical model of self-assembling nanoparticles. In this model, which he called the *abstract tile assembly model (aTAM)*, square particles called *tiles* attach edgewise to each other if their edges share a common *glue* and the bond strength is sufficient to overcome the kinetic energy or *temperature* of the system. The products of these systems are *assemblies*: aggregates of tiles forming via crystal-like growth starting at a *seed tile*. Surprisingly, these tile systems have been shown to be computationally universal [14,5], self-simulating [8,9], and capable of optimally encoding arbitrary shapes [12,1,13].

In parallel with work on the aTAM, a number of variations on the model have been proposed and investigated. One well-studied variant called the *hierarchical* [4] or *two-handed assembly model (2HAM)* [6] eliminates the seed tile and allows tiles and assemblies to attach in arbitrary order. This model was shown

* A full version of this paper can be found at http://arxiv.org/abs/1304.7038

to be capable of (theoretically) faster assembly of squares [4] and simulation of aTAM systems [2], including capturing the seed-originated growth dynamics. A generalization of the 2HAM model proposed by Demaine et al. [6] is the *staged assembly model*, which allows the assemblies produced by one system to be used as reagents (in place of tiles) for another system, yielding systems divided into sequential assembly *stages*. They showed that such sequential assembly systems can replace the role of glues in encoding complex assemblies, allowing the construction of arbitrary shapes efficiently while only using a constant number of glue types, a result impossible in the aTAM or 2HAM.

To understand the power of the staged assembly model, Demaine et al. [7] studied the problem of finding the smallest system producing a one-dimensional assembly with a given sequence of labels on its tiles, called a *label string*. They proved that for systems with a constant number of glue types, this problem is equivalent to the well-studied problem of finding the smallest context-free grammar whose language is the given label string, also called the *smallest grammar problem* (see [11,3]). For systems with unlimited glue types, they proved that the maximum ratio of the size of the smallest context-free grammar deriving a string s of length n *over* the size of the smallest system producing an assembly with label string s, called the *separation*, is $\Omega(\sqrt{n/\log n})$ and $O((n/\log n)^{2/3})$.

In this paper we consider the two-dimensional version of this problem: finding the smallest staged assembly system producing an assembly with a given *label polyomino*. For systems with constant glue types and no cooperative bonding, we prove that the separation of grammars *over* these systems is $\Omega(n/(\log\log n)^2)$ for polyominoes with n cells (Sect. 6.1), and $\Omega(n/\log^3 n)$ when restricted to rectangular (Sect. 6.2) or square (Sect. 6.3) polyominoes with a constant number of labels. Adding the restriction that each step of the assembly process produces a single product, we achieve $\Omega(n/\log^3 n)$ separation for general polyominoes with a single label (Sect. 6.1).

We also consider the separation of staged assembly systems *over* grammars, i.e. the maximum ratio of the size of the smallest system over the size of the smallest grammar for polyominoes with n cells. For this measure we achieve bounds of $\Omega(\log n/\log\log n)$ (Sect. 4) and, constructively, $O(\log n)$ (Sect. 5). For all of these results, we use a simple definition of context-free grammars on polyominoes that generalizes the context-free grammars of [7].

When taken together, these results give a nearly complete picture of how smallest context-free grammars and staged assembly systems compare. For some polyominoes, staged assembly systems are exponentially smaller than context-free grammars ($O(\log n)$ vs. $\Omega(n/\log^3 n)$). On the other hand, given a polyomino and grammar deriving it, one can construct a staged assembly system that is a (nearly optimal) $O(\log n)$-factor larger and produces an assembly with a label polyomino replicating the polyomino. A summary of these results is the following: "Up to logarithmic factors, SASs and SSASs are never worse than PCFGs and sometimes are far better".

2 Staged Self-Assembly

An instance of the staged tile assembly model is called a *staged assembly system* or *system*, abbreviated *SAS*. A SAS $\mathcal{S} = (T, G, \tau, M, B)$ is specified by five parts: a *tile set* T of square *tiles*, a *glue function* $G : \Sigma(G)^2 \to \{0, 1, \ldots, \tau\}$, a *temperature* $\tau \in \mathbb{N}$, a directed acyclic *mix graph* $M = (V, E)$, and a *start bin function* $B : V_L \to T$ from the *leaf vertices* $V_L \subseteq V$ of M with no incoming edges.

Each tile $t \in T$ is specified by a 5-tuple (l, g_n, g_e, g_s, g_w) consisting of a label l taken from an alphabet $\Sigma(T)$ (denoted $l(t)$) and a set of four non-negative integers in $\Sigma(G) = \{0, 1, \ldots, k\}$ specifying the *glues* on the sides of t with normal vectors $\langle 0, 1\rangle$ (north), $\langle 1, 0\rangle$ (east), $\langle 0, -1\rangle$ (south), and $\langle -1, 0\rangle$ (west), respectively, denoted $g_{\boldsymbol{u}}(t)$. In this work we only consider glue functions with the constraints that if $G(g_i, g_j) > 0$ then $g_i = g_j$, and $G(0, 0) = 0$.

A *configuration* is a partial function $C : \mathbb{Z}^2 \to T$ mapping locations on the integer lattice to tiles. Any two locations $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ in the domain of C (denoted $\mathrm{dom}(C)$) are *adjacent* if $||p_2 - p_1|| = 1$ and the *bond strength* between any pair of tiles $C(p_1)$ and $C(p_2)$ at adjacent locations is $G(g_{p_2-p_1}(C(p_1)), g_{p_1-p_2}(C(p_2)))$.

A configuration is a *τ-stable assembly* or an *assembly at temperature τ* if $\mathrm{dom}(C)$ is connected on the lattice and, for any partition of $\mathrm{dom}(C)$ into two subconfigurations C_1, C_2, the sum of the bond strengths between tiles at pairs of locations $p_1 \in \mathrm{dom}(C_1)$, $p_2 \in \mathrm{dom}(C_2)$ is at least τ. Any pair of configurations C_1, C_2 are equivalent if there exists a vector $\boldsymbol{v} = \langle x, y\rangle$ such that $\mathrm{dom}(C_1) = \{p + \boldsymbol{v} \mid p \in \mathrm{dom}(C_2)\}$ and $C_1(p) = C_2(p + \boldsymbol{v})$ for all $p \in \mathrm{dom}(C_1)$. The *size* of an assembly A is $|\mathrm{dom}(A)|$, the the number of lattice locations that map to tiles in T. Two τ-stable assemblies A_1, A_2 are said to *assemble* into a *superassembly* A_3 if there exists a translation vector $\boldsymbol{v} = \langle x, y\rangle$ such that $\mathrm{dom}(A_1) \cap \{p + \boldsymbol{v} \mid p \in A_2\} = \emptyset$ and A_3 defined by the partial functions A_1 and A_2' with $A_2'(p) = A_2(p + \boldsymbol{v})$ is a τ-stable assembly.

Each vertex of the mix graph M describes a *two-handed assembly process*. This process starts with a set of τ-stable *input assemblies* I. The set of *assembled assemblies* Q is defined recursively as $I \subseteq Q$, and for any pair of assemblies $A_1, A_2 \in Q$ with superassembly A_3, $A_3 \in Q$. Finally, the set of *products* $P \subseteq Q$ is the set of assemblies A such that for any assembly A', no superassembly of A and A' is in Q.

The mix graph $M = (V, E)$ of $\mathcal{S}$ defines a set of two-handed assembly processes (called *mixings*) for the non-leaf vertices of M (called *bins*). The input assemblies of the mixing at vertex v is the union of all products of mixings at vertices v' with $(v', v) \in E$. The start bin function B defines the lone single-tile product of each mixings at a leaf bin. The system $\mathcal{S}$ is said to *produce* an assembly A if some mixing of $\mathcal{S}$ has a single product, A. We define the size of $\mathcal{S}$, denoted $|\mathcal{S}|$, to be $|E|$, the number of edges in M. Note that since each tile type appears in a distinct leaf bin of M (defined by the start bin function B), the number of edges of M is at least $|T|$ and so $|\mathcal{S}| \geq |T|$. If every mixing in a system $\mathcal{S}$ has a single product, then $\mathcal{S}$ is a *singular self-assembly system (SSAS)*.

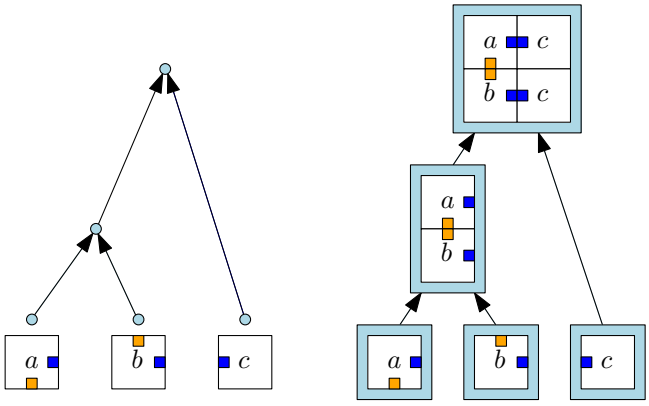


Fig. 1. A self-assembly system (SAS) consisting of a mix graph and tile types (left), and the assemblies produced by carrying out the algorithmic process of staged self-assembly (right).

The results of Section 6.4 use the notion of a self-assembly system $\mathcal{S}'$ *simulating* a system $\mathcal{S}$ by carrying out the same sequence of mixings and producing a set of scaled assemblies. Formally, we say a system $\mathcal{S}' = (T', G', \tau, M', B')$ *simulates* a system $\mathcal{S} = (T, G, \tau, M, B)$ at *scale* b if there exist two functions f, g with the following properties:

(1) The function $f : (\Sigma(T') \cup \{\varnothing\})^{b^2} \to \Sigma(T) \cup \{\varnothing\}$ maps the labels of $b \times b$ regions of tiles (called *blocks*) to a label of a tile in T. The empty label $\varnothing$ denotes no tile.
(2) The function $g : S' \to V$ maps a subset S' of the vertices of the mix graph M' to vertices of the mix graph M such that g is an isomorphism between the subgraph induced by S' in M' and the graph M.
(3) Let $P(v)$ be the set of products of the bin corresponding to vertex v in a mix graph. Then for each vertex $v \in M$ with $v' = g^{-1}(v)$, $P(v) = \{f(p) \mid p \in P(v')\}$.

The self-assembly systems constructed in Sections 5 and 6 produce only *mismatch-free assemblies*: assemblies in which every pair of incident sides of two tiles in the assembly have the same glue. A system is defined to be *mismatch-free* if every product of the system is mismatch-free.

3 Polyomino Context-Free Grammars

Here we describe polyominoes, a generalization of strings, and polyomino context-free grammars, a generalization of deterministic context-free grammars. These objects replace the strings and restricted context-free grammars (RCFGs) of Demaine et al. [7].

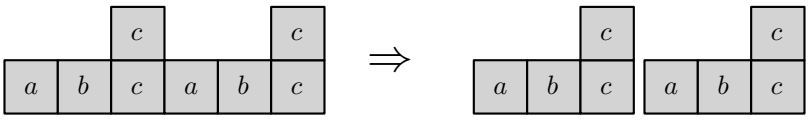
A *labeled polyomino* or *polyomino* $P = (S, L)$ is defined by a connected set of points S on the square lattice (called *cells*) containing $(0, 0)$ and a label function $L : S \to \Sigma(P)$ mapping each cell of P to a *label* contained in an alphabet $\Sigma(P)$. The *size* of P is the number of cells P contains and is denoted $|P|$. The label of the cell at lattice point (x, y) is denoted $L((x, y))$ and we define $P(x, y) = L((x, y))$ for notational convenience. We refer to the *label* or *color* of a cell interchangeably.

Define a *polyomino context-free grammar (PCFG)* to be a quadruple $G = (\Sigma, \Gamma, S, \Delta)$. The set Σ is a set of *terminal symbols* and the set Γ is a set of *non-terminal symbols*. The symbol $S \in \Gamma$ is a special *start symbol*. Finally, the set Δ consists of *production rules*, each of the form $N \to (R_1, (x_1, y_1)) \dots (R_j, (x_j, y_j))$ where $N \in \Gamma$ and is the left-hand side symbol of only this rule, $R_i \in N \cup T$, and each (x_i, y_i) is a pair of integers. The *size* of G is defined to be the total number of symbols on the right-hand sides of the rules of Δ.

A polyomino P can be derived by starting with S, the start symbol of G, and repeatedly replacing a non-terminal symbol with a set of non-terminal and terminal symbols. The set of valid replacements is Δ, the production rules of G, where a non-terminal symbol N with lower-leftmost cell at (x, y) can be replaced with a set of symbols R_1 at $(x+x_1, y+y_1)$, R_2 at $(x+x_2, y+y_2)$, …, R_j at $(x+x_j, y+y_j)$ if there exists a rule $N \to (R_1, (x_1, y_1))(R_2, (x_2, y_2)) \dots (R_j, (x_j, y_j))$. Additionally, the set of terminal symbol cells derivable starting with S must be connected and pairwise disjoint.

The polyomino P derived by the start symbol of a grammar G is called the *language of* G, denoted $L(G)$, and G is said to *derive* P. In the remainder of the paper we assume that each production rule has at most two right-hand side symbols (equivalent to binary normal form for 1D CFGs), as any PCFG can be converted to this form with only a factor-2 increase in size. Such a conversion is done by iteratively replacing two right-hand side symbols R_i, $R_{i'}$ with a new non-terminal symbol Q, and adding a new rule replacing Q with R_i and $R_{i'}$.

Intuitively, a polyomino context-free grammar is a recursive decomposition of a polyomino into smaller polyominoes. Because each non-terminal symbol is the left-hand side symbol of at most one rule, each non-terminal corresponds to a subpolyomino of the derived polyomino. Then each production rule is a decomposition of a subpolyomino into smaller subpolyominoes (see Figure 2).



$$N \to (R, (0, 0))(R, (3, 0))$$

Fig. 2. Each production rule in a PCFG generating a single shape is a decomposition of the left-hand side non-terminal symbol's polyomino into the right-hand side symbols' polyominoes.

In this interpretation, the smallest grammar deriving a given polyomino is equivalent to a decomposition using the fewest *distinct* subpolyominoes in the decomposition. As for the smallest CFG for a given string, the smallest PCFG for a given polyomino is deterministic and finding such a grammar is NP-hard. Moreover, even approximating the smallest grammar is NP-hard [3], and achieving optimal approximation algorithms remains open [10].

In Section 5 we construct self-assembly systems that produce assemblies whose label polyominoes are scaled versions of other polyominoes, with some amount of "fuzz" in each scaled cell. A polyomino $P' = (S', L')$ is said to be a (c, d)-*fuzzy replica* of a polyomino $P = (S, L)$ if there exists a vector $\langle x_t, y_t \rangle$ with the following properties:

1. For each block of cells $\mathcal{S}'_{(i,j)} = \{(x, y) \mid x_t + di \leq x < x_t + d(i+1), y_t + dj \leq y < y_t + d(j+1)\}$ (called a *supercell*), $\mathcal{S}'_{(i,j)} \cap S' \neq \varnothing$ if and only if $(i, j) \subseteq S$.
2. For each supercell $\mathcal{S}'_{(i,j)}$ containing a cell of P', the subset of *label cells* $\{(x, y) \mid x_t + di + (d-c)/2 \leq x < x_t + d(i+1) + (d-c)/2, y_t + dj + (d-c)/2 \leq y < y_t + d(j+1) + (d-c)/2\}$ consists of c^2 cells of P', with all cells having identical label, called the *label of the supercell* and denoted $\mathcal{L}_{(i,j)}$.
3. For each supercell $\mathcal{S}'_{(i,j)}$, any cell that is not a label cell of $\mathcal{S}'_{(i,j)}$ has a common *fuzz label* in L'.
4. For each supercell $\mathcal{S}'_{(i,j)}$, the label of the supercell $\mathcal{L}'_{(i,j)} = P(i, j)$.

Properties (1) and (2) define how sets of cells in P' replicate individual cells in P, and the labels of these sets of cells and individual cells. Property (3) restricts the region of each supercell not in the label region to contain only cells with a common fuzz label. Property (4) requires that each supercell's label matches the label of the corresponding cell in P.

4 SAS over PCFG Separation Lower Bound

This result uses a set of shapes we call *n-staggers*, an example is seen in Figure 3. The shapes consist of $\log n$ bars of dimensions $n/\log n \times 1$ stacked vertically atop each other, with each bar horizontally offset from the bar below it by some amount in the range $-(n/\log n - 1), \ldots, n/\log n - 1$. We use the shorthand that $\log n = \lfloor \log n \rfloor$ for conciseness. Every sequence of $\log n - 1$ integers, each in the range $[-(n/\log n - 1), n/\log n - 1]$, encodes a unique staggler and by the pidgeonhole principle, some n-staggler requires $\log((2n/\log n - 1)^{\log n - 1}) = \Omega(\log^2 n)$ bits to specify.

Lemma 1. *Any n-staggler can be derived by a PCFG of size* $O(\log n)$.

Proof. A set of $O(\log n)$ production rules deriving a bar (of size $\Theta(n/\log n) \times 1$) can be constructed by repeatedly doubling the length of the bar, using an additional $\log n$ rules to form the bar's exact length. The result of these production rules is a single non-terminal B deriving a complete bar.

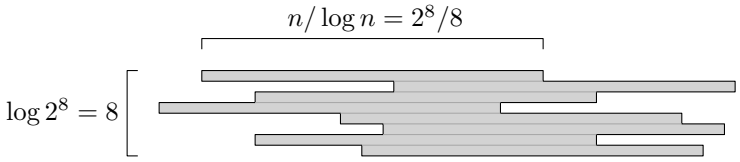


Fig. 3. The 2^8-staggler specified by the sequence $-18, 13, 9, -17, -4, 12, -10$.

Using the non-terminal B, a stack of k bars can be described using a production rule $N \to (B, (x_1, 0))(B, (x_2, 1)) \dots (B, (x_k, k-1))$, where the x-coordinates $x_1, x_2, \dots, x_k$ encode the offsets of each bar relative to the bar below it. An equivalent set of $k-1$ production rules in binary normal form can be produced by creating a distinct non-terminal for T_i each stack of the first i bars, and a production rule $T_i \to (T_{i-1}, (0,0))(B, (x_i, i))$ encoding the offset of the topmost bar relative to the stack of bars beneath it.

In total, $O(\log n)$ rules are used to create B, the non-terminal deriving a bar, and $O(\log n)$ are used to create the stack of bars, one per bar. So the n-staggler can be constructed using a PCFG of size $O(\log n)$.

Lemma 2. *For every n, there exists an n-staggler P such that any SAS or SSAS producing an assembly with label polyomino P has size $\Omega(\log^2 n/\log\log n)$.*

Proof. The proof is information-theoretic. Recall that more than half of all n-stagglers require $\Omega(\log^2 n)$ bits to specify. Now consider the number of bits contained in a SAS $\mathcal{S}$. Recall that $|\mathcal{S}|$ is the number of edges in the mix graph of $\mathcal{S}$. Any SAS can be encoded naively using $O(|\mathcal{S}|\log|\mathcal{S}|)$ bits to specify the mix graph, $O(|T|\log|T|)$ bits to specify the tile set, and $O(|\mathcal{S}|\log|T|)$ bits to specify the tile type at each leaf node of the mix graph. Because the number of tile types cannot exceed the size of the mix graph, $|T| \leq |\mathcal{S}|$. So the total number of bits needed to specify $\mathcal{S}$ (and thus the number of bits of information contained in $\mathcal{S}$) is $O(|\mathcal{S}|\log|\mathcal{S}| + |T|\log|T| + |\mathcal{S}|\log|\mathcal{S}|) = O(|\mathcal{S}|\log|\mathcal{S}|)$. So some n-staggler requires a SAS $\mathcal{S}$ such that $O(|\mathcal{S}|\log|\mathcal{S}|) = \Omega(\log^2 n)$ and thus $|\mathcal{S}| = \Omega(\log^2 n/\log\log n)$.

Theorem 1. *The separation of SASs and SSASs over PCFGs is $\Omega(\log n/\log\log n)$.*

Proof. By the previous two lemmas, more than half of all n-stagglers require SASs and SSASs of size $\Omega(\log^2 n/\log\log n)$ and all n-stagglers have PCFGs of size $O(\log n)$. So the separation is $\Omega(\log n/\log\log n)$.

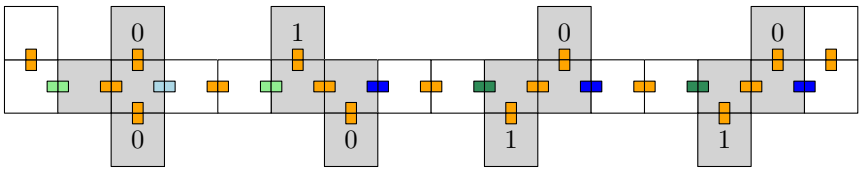
We also note that scaling the n-staggler by a c-factor produces a shape which is derivable by a CFG of size $O(\log n + \log c)$. That is, the result still holds for n-stagglers scaled by any amount polynomial in n. For instance, the $O(n)$-factor of the construction of Theorem 2.

At first it may not be clear how PCFGs achieve smaller encodings. After all, each rule in a PCFG G or mixing in SAS $\mathcal{S}$ specifies either a set of right-hand side symbols or set of input bins to use and so has up to $O(\log|G|)$ or $O(\log|\mathcal{S}|)$

bits of information. The key is the coordinate describing the location of each right-hand side symbol. These offsets have up to $O(\log n)$ bits of information and in the case that G is small, say $O(\log n)$, each rule has a number of bits *linear* in the size of the PCFG!

5 SAS over PCFG Separation Upper Bound

Next we show that the separation lower bound of the last section is nearly large as possible by giving an algorithm for converting any PCFG G into a SSAS $\mathcal{S}$ with system size $O(|G| \log n)$ such that $\mathcal{S}$ produces an assembly that is a fuzzy replica of the polyomino derived by G. Before describing the full construction, we present approaches for efficiently constructing general binary counters and for simulating glues using geometry.



Increment 0011_b by 1, yielding 0100_b.

Fig. 4. A binary counter row constructed using single-bit constant-sized assemblies. Dark blue and green glues indicate 1-valued carry bits, light blue and green glues indicate 0-valued carry bits.

The *binary counter row assemblies* used here are a generalization of those by Demaine et al. [6] consisting of constant-sized bit assemblies, and an example is seen in Figure 4. Our construction achieves $O(\log n)$ construction of arbitrary ranges of rows and increment values, in contrast to the contruction of [6] that only produces row sets of the form $0, 1, \ldots, 2^{2^m} - 1$ that increment by 1. To do so, we show how to construct two special cases from which the generalization follows easily.

Lemma 3. *Let i, j, n be integers such that $0 \leq i \leq j < n$. There exists a SSAS of size $O(\log n)$ with a set of bins that, when mixed, assemble a set of $j - i + 1$ binary counter rows with values $i, i + 1, \ldots, j$ incremented by 1.*

Lemma 4. *Let k, n be integers such that $0 \leq k \leq n$ and $n = 2^m$. There exists a SSAS of size $O(\log n)$ with a set of bins that, when mixed, assemble a set of 2^m binary counter rows with values $0, 1, \ldots, 2^m - 1$ incremented by k.*

Lemma 5. *Let i, j, k, n be integers such that $0 \leq i \leq j < n$ and $0 \leq k \leq n$. There exists a SSAS of size $O(\log n)$ with a set of bins that, when mixed, assemble a set of $j - i + 1$ binary counter rows with values $i, i + 1, \ldots, j$ incremented by k.*

Theorem 8 of Demaine et al. [6] describes how to reduce the number of glues used in a system by replacing each tile with a large *macrotile* assembly, and encoding the tile's glues via unique geometry on the macrotile's sides. We prove a similar result for labeled tiles, used for proving Theorems 2, 3, and 7.

Lemma 6. *Any mismatch-free $\tau = 1$ SAS (or SSAS) $\mathcal{S} = (T, G, \tau, M)$ can be simulated by a SAS (or SSAS) $\mathcal{S}'$ at $\tau = 1$ with $O(1)$ glues, system size $O(\Sigma(T)|T| + |\mathcal{S}|)$, and $O(\log |G|)$ scale.*

Armed with these tools, we are ready to convert PCFGs into SSASs. Recall that in Section 4 we showed that in the worst case, converting a PCFG into a SSAS (or SAS) *must* incur an $\Omega(\log n / \log \log n)$-factor increase in system size. Here we achieve a $O(\log n)$-factor increase.

Theorem 2. *For any polyomino P with $|P| = n$ derived by a PCFG G, there exists a SSAS $\mathcal{S}$ with $|\mathcal{S}| = O(|G| \log n)$ producing an assembly with label polyomino P', where P' is a $(O(\log n), O(n))$-fuzzy replica of P.*

Proof. We combine the macrotile construction of Lemma 6, the generalized counters of Lemma 5, and a macrotile assembly invariant that together enable efficient simulation of each production rule in a PCFG by a set of $O(\log n)$ mixing steps.

Macrotiles. The macrotiles used are extended versions of the macrotiles in Lemma 6 with two modifications: a secondary, *resevoir macroglue* assembly on each side of the tile in addition to a primary *bonding macroglue*, and a thin *cage* of dimensions $\Theta(n) \times \Theta(\log n)$ surrounding each resevoir macroglue (see Figure 5).

Mixing a macrotile with a set of bins containing counter row assemblies constructed by Lemma 5 causes completed (and incomplete) counter rows to attach to the macrotile's macroglues. Because each macroglue's geometry matches the geometry of exactly one counter row, a partially completed counter row that attaches can only be completed with bit assemblies that match the macroglue's value. As a result, mixing the bin sets of Lemma 5 with an assembly consisting of macrotiles produces the same set of products as mixing a completed set of binary counter rows with the assembly.

An attached counter row effectively causes the macroglue's value to change, as it presents geometry encoding a new value and covers the macroglue's previous value. The cage is constructed to have height sufficient to accomodate up to n counter rows attached to the reservoir macroglue, but no more.

Because of the cage, no two macrotiles can attach by their bonding macroglues unless the macroglue has more than n counter rows attached. Alternatively, one can produce a thickened counter row with thickness sufficient to extend beyond the cage. We call such an assembly a *macroglue activator*, as it "activates" a bonding macroglue to being able to attach to another promoted macroglue on another macrotile. Notice that a macroglue activator will never attach to a bonding macroglue's resevoir twin, as the cage is too small to contain the activator.

An invariant. Counter rows and activators allow precise control of two properties of a macrotile: the identities of the macroglues on each side, and whether

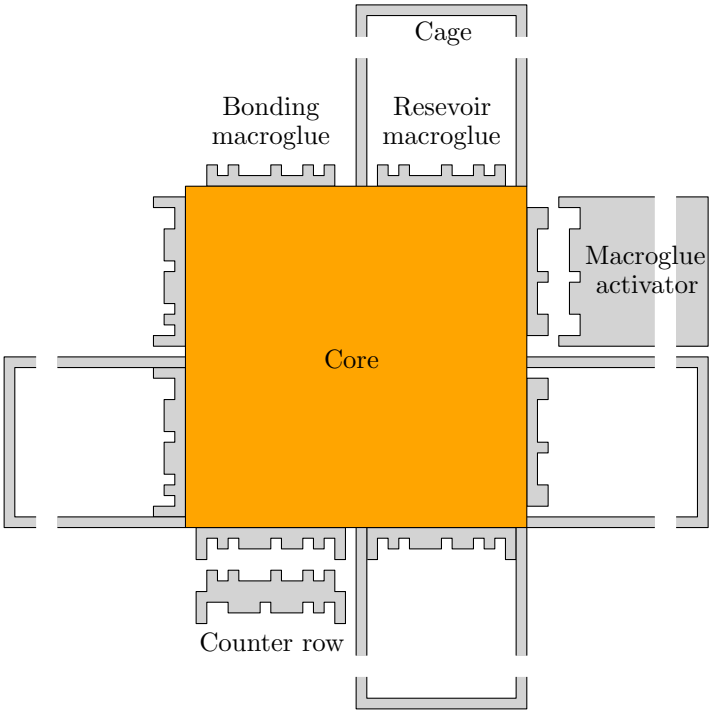


Fig. 5. A macrotile used in converting a PCFG to a SAS, and examples of value maintenance and offset preparation.

these glues are activated. In a large assembly containing many macroglues, the ability to change and activate glues allows precise encoding of how an assembly can attach to others. In the remainder of the construction we maintain the invariant that every macrotile has the same glue identity on all four sides, and any macrotile assembly consists of macrotiles that form a contiguous sequence of distinct glue values. For instance, the values $4, 5, 6, 7$ are permitted, but not $4, 5, 5, 6$ (not distinct) or $4, 5, 6, 8$ (not contiguous). These contiguous sequences are denoted $l..u$, e.g. 4..7.

Production rule simulation. Consider a PCFG with non-terminal N and production rule $N \to (R_1, (x_1, y_1))(R_2, (x_2, y_2))$ and a SSAS with two bins containing assemblies A_1, A_2 with the label polyominoes of A_1 and A_2 being fuzzy replicas of the polyominoes derived by R_1 and R_2. Also assume A_1 and A_2 are assembled from the macrotiles just described, including obeying the invariant that the glue values are identical on all sides of each macrotile and are contiguous and distinct across the assembly. Let the glue values for A_1 and A_2 be $l_1..u_1$ and $l_2..u_2$, respectively.

First, we change the glue values of A_1 and A_2 depending on the values $l_1..u_1$ and $l_2..u_2$. Without loss of generality, assume $u_1 \leq u_2$. Then if $u_1 < l_2$, we

increment all values of A_1 by $l_2 - u_1 - 1$. Otherwise $u_1 \geq l_2$ and we increment all values of A_2 by $u_1 - l_2 + 1$.

By Lemma 5, a set of row counters incrementing all glue values on a macrotile assembly can be produced using $O(\log n)$ work. In both cases the incrementation results in values that, if found on a common assembly, obey the invariants. Because each macrotile has a reservoir macroglue on each side, any bonding macroglue with an activator already attached has a reservoir macroglue that accepts the matching row counter, so each mixing has a single product and specifically no row counter products.

Next, select two cells, c_{R_1} and c_{R_2}, in the polyominoes derived by R_1 and R_2 adjacent in polyomino derived by N. Define the glue values of the two macrotiles in A_1 and A_2 forming the supercells mapped to c_{R_1} and c_{R_2} to be v_1 and v_2 and define the pair of coincident sides of c_{R_1} and c_{R_2} to be s_1 and s_2. We mix A_1 and A_2 with activators for sides s_1 and s_2 with values v_1 and v_2, respectively. Activators, because they are nearly rectangular save $O(\log n)$ cells of bit geometry, can also be produced using $O(\log n)$ work.

System scale. The PCFG P contains at most n production rules. Also, each step shifts glue identities by at most n (the number of distinct glues on the macrotile), so the largest glue identity on the final macrotile assembly is n^2. So we produce macrotiles with core assemblies of size $O(\log n) \times O(\log n)$ and cages of size $O(n)$. Assembling the core assemblies, cages, and initial macroglue assemblies of the macrotiles takes $O(|P| \log n + \log n + \log n) = O(|P| \log n)$ work, dominated by the core assembly production. Simulating each production rule of the grammar takes $O(\log n)$ work spread across a constant number of $O(\log n)$-sized sequences of mixings to produce sets of row counters and macroglue activators.

Applying Lemma 6 to the construction (creating macrotiles of macrotiles) gives a constant-glue version of Theorem 2:

Theorem 3. *For any polyomino P with $|P| = n$ derived by a PCFG G, there exists a SSAS $\mathcal{S}'$ using $O(1)$ glues with $|\mathcal{S}'| = O(|G| \log n)$ producing an assembly with label polyomino P', where P' is a $(O(\log n \log \log n), O(n \log \log n))$-fuzzy replica of P.*

The results in this section and Section 4 achieve a "one-sided" correspondence between the smallest PCFG and SSAS encoding a polyomino, i.e. the smallest PCFG is approximately an *upper bound* for the smallest SSAS (or SAS). Since the separation upper bound proof (Theorem 2) is constructive, the bound also yields an algorithm for converting a PCFG into a SSAS.

6 PCFG over SAS and SSAS Separation Lower Bound

Here we develop a sequence of PCFGs over SAS and SSAS separation results, all within a polylogarithmic factor of optimal. The results also hold for polynomially scaled versions of the polyominoes, which is used to prove Theorem 7 at the end of the section. This scale invariance also surpasses the scaling of the fuzzy replicas

in Theorems 2 and 3, implying that this relaxation of the problem statement in these theorems was not unfair.

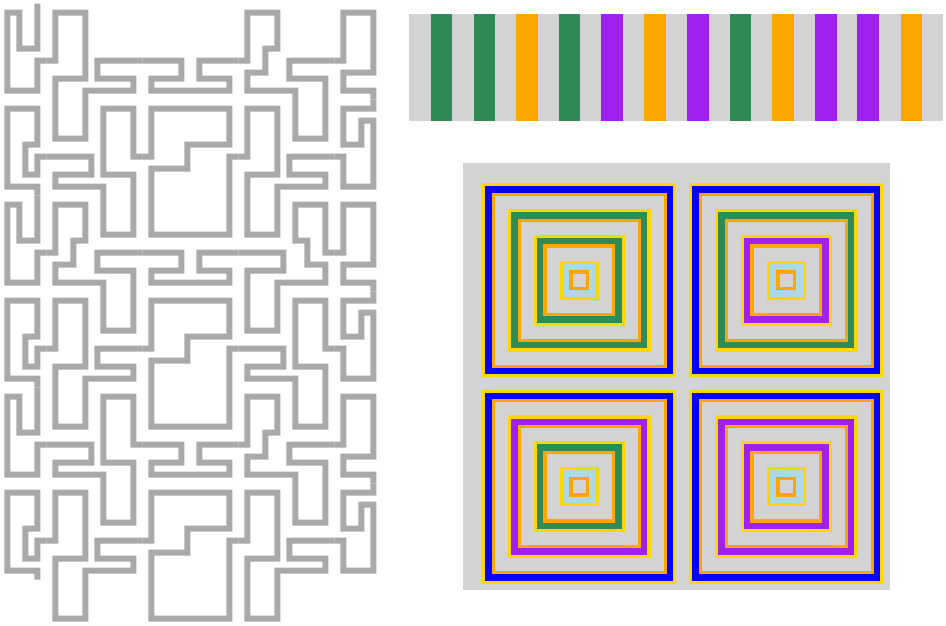


Fig. 6. Two-bit examples of the weak (left), end-to-end (upper right), and block (lower right) binary counters used to achieve separation of PCFGs over SASs and SSASs in Section 6.

6.1 General shapes

We show that the separation of PCFGs over SASs and SSASs is $\Omega(n/\log n)$ using a *weak binary counter*, seen in Figure 6. These shapes are macrotile versions of the doubly-exponential counters found in [6] with three modifications:

1. Each row is a single path of tiles, and any path through an entire row uniquely identifies the row.
2. Adjacent rows do not have adjacent pairs of tiles, i.e. they do not touch.
3. Consecutive rows attach at alternating (east, west, east, etc.) ends.

Lemma 7. *There exists a $\tau = 1$ SAS of size $O(b)$ that produces a 2^b-bit weak counter.*

Lemma 8. *For any PCFG G deriving a 2^b-bit weak counter, $|G| = \Omega(2^{2^b})$.*

Theorem 4. *The separation of PCFGs over $\tau = 1$ SASs for single-label polyominoes is $\Omega(n/(\log\log n)^2)$.*

Proof. By the previous two lemmas, there exists a SAS of size $O(b)$ producing a b-bit weak counter, and any PCFG deriving this shape has size $\Omega(2^{2^b})$. The assembly itself has size $n = \Theta(2^{2^b} b)$, as it consists of 2^{2^b} rows, each with b subassemblies of constant size. So the separation is $\Omega((n/b)/b) = \Omega(n/(\log\log n)^2)$.

Corollary 1. *The separation of PCFGs over $\tau = 1$ SSASs for single-label polyominoes is $\Omega(n/\log^2 n)$.*

6.2 Rectangles

For the weak counter construction, the lower bound in Lemma 8 depended on the poor connectivity of the weak counter polyomino. This dependancy suggests that such strong separation ratios may only be achievable for special classes of "weakly connected" or "serpentine" shapes. In this section we show separation ratios are possible for rectangular constant-label polyominoes that nearly match those achieved for single-label general polyominoes. The result is achieved using an *b-bit end-to-end binary counter* polyomino, seen in the upper right of Figure 6.

End-to-end counters. The polyominoes constructed resemble binary counters whose rows have been arranged in sequence horizontally. Each row of the counter is assembled from tall, thin macrotiles (called *bars*), each containing a *color strip* of orange, purple, or green. The color strip is coated on its east and west faces with gray geometry tiles that encode the bar's location within the counter. Each row of the counter has a sequence of green and purple *display bars* encoding a binary representation of the row's value flanked by orange *reset bars* (see Figure 6).

Each bar has dimensions $O(1) \times 3(\log_2 b + b + 2)$, sufficient for encoding two pieces of information specifying the location of the bar within the assembly. The *row bits* specify which row the bar lies in (e.g. the 7th row). The *subrow bits* specify where within the row the bar lies (e.g. the 4th bit). The subrow value starts at 0 on the east side of a reset bar, and increments through the display bars until reaching $b + 1$ on the west end of the next reset bar. Bars of all three types with row bits ranging from 0 to $2^b - 1$ are produced.

Efficient assembly. The counter is constructed using a SAS of size $O(b)$ in two phases. First, sequences of $O(b)$ mixings are used to construct five families of bars: reset bars, 0-bit display bars resulting from a carry, 0-bit display bars without a carry, 1-bit display bars resulting from a carry, and 1-bit display bars without a carry. The mixings produce five bins, each containing *all* of the bars of a family. These five bins are then combined into a final bin where the bars attach to form the $\Theta(2^b) \times \Theta(b)$ rectangular assembly. The five families are seen in Figure 7.

As a warmup, consider the assembly of all reset bars. For these bars, the west subrow bits encode b and the east subrow bits encode 0. The row bits encode a value i on the west side, and $i + 1$ on the east side, for all i between 0 and $2^b - 1$. For each of the $\log_2 b + 1$ subrow bits, create an assembly where the west and east bits are 1 and 0 respectively, *except* for the most significant bit (bit $\log_2 b + 1$), where the west and east bits are both 0.

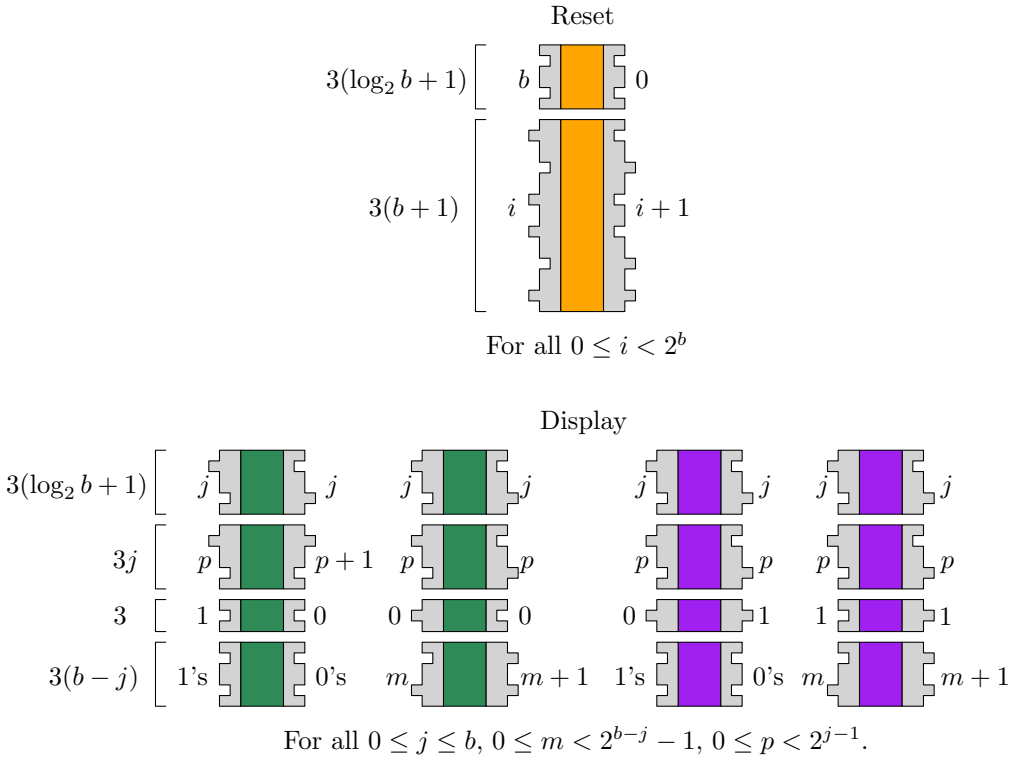


Fig. 7. The decomposition of bars used assemble a b-bit end-to-end counter.

For the row bits we use the same technique as in [6] and extended in Lemma 4: create a constant-sized set of assemblies for each bit that encode input and output value and carry bits. For bits 1 through $b-1$ (zero-indexed) create four assemblies corresponding to the four combinations of value and carry bits, for bit 0 create two assemblies corresponding to value bits (the carry bit is always 1), for bit b create three assemblies corresponding to all combinations except both value and carry bits valued 1, and for bit $b+1$ create a single assembly with both bits valued 0. Give each bit assembly a unique south and north glue encoding its location within the bar and carry bit value, and give all bit assemblies a common orange color strip. Mixing these assemblies produces all reset bars, with subrow west and east values of b and 0, and row values i and $i+1$ for all i from 0 to $2^b - 1$.

Assembling display bars. Recall that for all bars, the row value i locates the bar's row and the subrow value j locates the bar within this row. So the correct color strip for a display bar is green if the jth bit of i is 0, and purple if the jth bit of i is 1.

As previously mentioned, we produce four families of display bars, two for each value of the jth bit of i. Each subfamily is produced by mixing a subrow

assembly encoding j on both east and west ends with three component assemblies of the row value: the *least significant bits (LSB) assembly* encoding bits 1 through $j-1$ of i, the *most significant bits (MSB) assembly* encoding bits $j+1$ through b of i, and the constant-sized jth bit assembly. This decomposition is seen in the bottom half of Figure 7.

The four families correspond to the four input and carry bit values of the jth bit. These values determine what collections of subassemblies should appear in the other two components of the row value. For instance, if the input and carry bit values are both 1, then the LSB assembly must have all 1's on its west side (to set the jth carry bit to 1) and all 0's on its east side. Similarly, the MSB assembly must have some value p encoded on its west side and the value $p+1$ encoded on its east side, since the jth bit and and jth carry bit were both 1, so the $(j+1)$st carry bit is also 1.

Notice that each of the four families has b subfamilies, one for each value of j. Producing all subfamilies of each family is possible in $O(b)$ work by first recursively producing a set of b bins containing successively larger sets of MSB and LSB assemblies for the family. Then each subfamily can be produced using $O(1)$ amortized work, mixing one of b sets of LSB assembly subfamilies, one of b sets of MSB assemblies, and the jth bit assembly together.

Lemma 9. *There exists a $\tau = 1$ SAS of size $O(b)$ that produces a b-bit end-to-end counter.*

Proof. This follows from the description of the system. The five families of bars can each be produced with $O(b)$ work and the bars can be combined together in a single mixing to produce the counter. So the system has total size $O(b)$.

Lemma 10. *For any PCFG G deriving a b-bit end-to-end counter, $|G| = \Omega(2^b)$.*

Proof. Let G be a PCFG deriving a b-bit end-to-end counter. Define a *minimal row spanner* to be a non-terminal symbol N with production rule $N \to (B,(x_1,y_1))(C,(x_2,y_2))$ such that the polyomino derived by N (denoted p_N) horizontally spans the color strips of all bars in row $\mathcal{R}_i$ including the reset bar at the end of the row, while the polyominoes derived by B and C (denoted p_B and p_C) do not. Consider the bounding box D of these color strips (see Figure 8).

Without loss of generality, assume p_B intersects the west boundary of D but does not reach the east boundary, while p_C intersects the east boundary but does not reach the west boundary, so any location at which p_B and p_C touch must lie in D. Then any row spanned by p_N and not spanned by p_B or p_C must lie in D, since spanning it requires cells from both p_B and p_C. So p_N is a minimal row spanner for at most one row: row $\mathcal{R}_i$.

Because the sequence of green and purple display bars found in D is distinct and separated by display bars in other rows by orange reset bars, each minimal row spanner spans a unique row $\mathcal{R}_i$. Then since each non-terminal is a spanner for at most one unique row, G must have 2^b non-terminal symbols and $|G| = \Omega(2^b)$.

Theorem 5. *The separation of PCFGs over $\tau = 1$ SASs for constant-label rectangles is $\Omega(n/\log^3 n)$.*

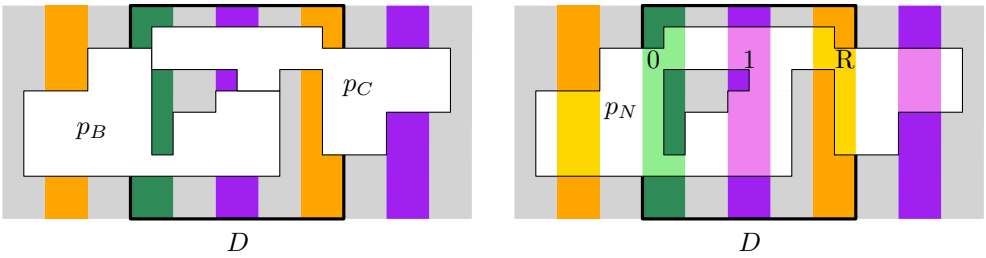


Fig. 8. A schematic of the proof that a non-terminal is a minimal row spanner for at most one unique row. (Left) Since p_B and p_C can only touch in D, their union non-terminal N must be a minimal row spanner for the row in D. (Right) The row's color strip sequence uniquely determines the row spanned by N (01_b).

Proof. By construction, a b-bit end-to-end counter has dimensions $\Theta(2^b b) \times \Theta(b)$. So $n = \Theta(2^b b^2)$ and $b = \Theta(\log n)$. Then by the previous two lemmas, the separation is $\Omega((n/b^2)/b) = \Omega(n/\log^3 n)$.

A simple replacement of orange, green, and purple color strips with distinct horizontal sequences of black/white color substrips yields the same result but using fewer distinct labels.

6.3 Squares

The rectangular polyomino of the last section has exponential aspect ratio, suggesting that this shape requires a large PCFG because it approximates a patterned one-dimensional assembly reminiscent of those in [7]. Creating a polyomino with better aspect ratio but significant separation is possible by extending the polyomino's labels vertically. For a square this approach gives a separation of PCFGs over SASs of $\Omega(\sqrt{n}/\log n)$, non-trivial but far worse than the rectangle.

Our final result achieves $\Omega(n/\log n)$ separation of PCFGs over SASs for squares using a *block binary counter* (seen in Figure 6). Each "row" of the counter is actually a set of concentric square rings called a *block*.

Lemma 11. *For even b, there exists a $\tau = 1$ SAS of size $O(b)$ that produces a b-bit block counter.*

Lemma 12. *For any PCFG G deriving a b-bit block counter, $|G| = \Omega(2^b)$.*

Theorem 6. *The separation of PCFGs over $\tau = 1$ SASs for constant-label squares is $\Omega(n/\log^3 n)$.*

6.4 Constant-glue constructions

Lemma 6 proved that any system $\mathcal{S}$ can be converted to a slightly larger system (both in system size and scale) that simulates $\mathcal{S}$. Applying this lemma to the constructions of Section 6 yields identical results for constant-glue systems:

Theorem 7. *All results in Section 6 hold for systems with $O(1)$ glues.*

Acknowledgements

This work was supported in part by National Science Foundation grant CBET-0941538. We thank Benjamin Hescott and anonymous reviewers for helpful comments and feedback that greatly improved the presentation of the paper.

References

1. L. Adleman, Q. Cheng, A. Goel, and M.-D. Huang. Running time and program size for self-assembled squares. In *Proceedings of Symposium on Theory of Computing (STOC)*, 2001.
2. S. Cannon, E. D. Demaine, M. L. Demaine, S. Eisenstat, M. J. Patitz, R. T. Schweller, S. M. Summers, and A. Winslow. Two hands are better than one (up to constant factors): Self-assembly in the 2HAM vs. aTAM. In *Proceedings of International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 20 of *LIPIcs*, pages 172–184, 2013.
3. M. Charikar, E. Lehman, A. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and a. shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.
4. H.L. Chen and D. Doty. Parallelism and time in hierarchical self-assembly. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2012.
5. M. Cook, Y. Fu, and R. Schweller. Temperature 1 self-assembly: determinstic assembly in 3D and probabilistic assembly in 2D. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2011.
6. E. D. Demaine, M. L. Demaine, S. Fekete, M. Ishaque, E. Rafalin, R. Schweller, and D. Souvaine. Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues. *Natural Computing*, 7(3):347–370, 2008.
7. E. D. Demaine, S. Eisenstat, M. Ishaque, and A. Winslow. One-dimensional staged self-assembly. *Natural Computing*, 2012.
8. D. Doty, J. H. Lutz, M. J. Patitz, R. T. Schweller, S. M. Summers, and D. Woods. Intrinsic universality in self-assembly. In *Proceedings of Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 5 of *LIPIcs*, pages 275–286, 2010.
9. D. Doty, J. H. Lutz, M. J. Patitz, R. T. Schweller, S. M. Summers, and D. Woods. The tile assembly model is intrinsically universal. In *Proceedings of Foundations of Computer Science (FOCS)*, pages 302–310, 2012.
10. A. Jeż. Approximation of grammar-based compression via recompression. Technical report, arXiv, 2013.
11. E. Lehman. *Approximation Algorithms for Grammar-Based Data Compression*. PhD thesis, MIT, 2002.
12. P. W. K. Rothemund and E. Winfree. The program-size complexity of self-assembled squares. In *Proceedings of Symposium on Theory of Computing (STOC)*, pages 459–468, 2000.
13. D. Soloveichik and E. Winfree. Complexity of self-assembled shapes. In C. Ferretti, G. Mauri, and C. Zandron, editors, *DNA 11*, volume 3384 of *LNCS*, pages 344–354. Springer Berlin Heidelberg, 2005.
14. E. Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, Caltech, 1998.