Nat Comput (2016) 15:115–128
DOI 10.1007/s11047-015-9512-7

Fast arithmetic in algorithmic self-assembly

Alexandra Keenan[1] · **Robert Schweller**[1] · **Michael Sherman**[1] · **Xingsi Zhong**[1]

Published online: 17 November 2015

Abstract In this paper we consider the time complexity of adding two n-bit numbers together within the tile self-assembly model. The (abstract) tile assembly model is a mathematical model of self-assembly in which system components are square tiles with different glue types assigned to tile edges. Assembly is driven by the attachment of singleton tiles to a growing seed assembly when the net force of glue attraction for a tile exceeds some fixed threshold. Within this frame work, we examine the time complexity of computing the sum of two n-bit numbers, where the input numbers are encoded in an initial seed assembly, and the output sum is encoded in the final, terminal assembly of the system. We show that this problem, along with multiplication, has a worst case lower bound of $\Omega(\sqrt{n})$ in 2D assembly, and $\Omega(\sqrt[3]{n})$ in 3D assembly. We further design algorithms for both 2D and 3D that meet this bound with worst case run times of $O(\sqrt{n})$ and $O(\sqrt[3]{n})$ respectively, which beats the previous best known upper bound of $O(n)$. Finally, we consider average case complexity of addition over uniformly distributed n-bit strings and show how we can achieve $O(\log n)$ average case time with a simultaneous $O(\sqrt{n})$ worst case run time in 2D. As additional evidence for the speed of our algorithms, we implement our algorithms, along with the simpler $O(n)$ time algorithm, into a probabilistic run-time simulator and compare the timing results.

Keywords Algorithmic self-assembly · Computational geometry · Adder · Tile assembly model

✉ Alexandra Keenan
abkeenan@utpa.edu

Robert Schweller
rtschweller@utpa.edu

Michael Sherman
mjsherman@utpa.edu

Xingsi Zhong
zhong@utpa.edu

[1] University of Texas - Pan American, Edinburg, TX, USA

1 Introduction

Self-assembly is the process by which systems of simple objects autonomously organize themselves through local interactions into larger, more complex objects. Self-assembly processes are abundant in nature and serve as the basis for biological growth and replication. Understanding how to design and efficiently program molecular self-assembly systems to harness this power promises to be fundamental for the future of nanotechnology. One particular direction of interest is the design of molecular computing systems for efficient solution of fundamental computational problems. In this paper we study the complexity of computing arithmetic primitives within a well studied model of algorithmic self-assembly, the abstract tile assembly model.

The abstract tile assembly model (aTAM) models system monomers with four sided Wang tiles with glue types assigned to each edge. Assembly proceeds by tiles attaching, one by one, to a growing initial seed assembly whenever the net glue strength of attachment exceeds some fixed temperature threshold (Fig. 2). The aTAM has been shown to be capable of universal computation (Winfree 1998), and has promising computational potential with a DNA implementation (Mao et al. 2000). Research leveraging this computational power has lead to efficient assembly of complex geometric shapes and patterns with a number of recent results in FOCS, SODA, and ICALP

(Meunier et al. 2014; Abel et al. 2010; Bryans et al. 2011; Cook et al. 2011; Doty et al. 2012, 2010; Schweller and Sherman 2013; Fu et al. (2012; Chandran et al. 2009; Doty 2010; Kao et al. 2008; Demaine et al. 2013, 2014). This universality also allows the model to serve directly as a model for computation in which an input bit string is encoded into an initial assembly. The process of self-assembly and the final produced terminal assembly represent the computation of a function on the given input. Given this framework, it is natural to ask how fast a given function can be computed in this model. Tile assembly systems can be designed to take advantage of massive parallelism when multiple tiles attach at distinct positions in parallel, opening the possibility for faster algorithms than what can be achieved in more traditional computational models. On the other hand, tile assembly algorithms must use up geometric space to perform computation, and must pay substantial time costs when communicating information between two physically distant bits. This creates a host of challenges unique to this physically motivated computational model that warrant careful study.

In this paper we consider the time complexity of adding two n-bit numbers within the abstract tile assembly model. We show that this problem, along with multiplication, has a worst-case lower bound of $\Omega(\sqrt{n})$ time in 2D and $\Omega(\sqrt[3]{n})$ time in 3D. These lower bounds are derived by a reduction from a simple problem we term the *communication* problem in which two distant bits must compute the AND function between themselves. This general reduction technique can likely be applied to a number of problems and yields key insights into how one might design a sub-linear time solution to such problems. We in turn show how these lower bounds, in the case of 2D and 3D addition, are matched by corresponding worst case $O(\sqrt{n})$ and $O(\sqrt[3]{n})$ run time algorithms, respectively, which improves upon the previous best known result of $O(n)$ (Brun 2007). We then consider the average case complexity of addition given two uniformly generated random n-bit numbers and construct a $O(\log n)$ average case time algorithm that achieves simultaneous worst case run time $O(\sqrt{n})$ in 2D. To the best of our knowledge this is the first tile assembly algorithm proposed for efficient average case adding. Our results are summarized in Table 1. Also, tile self-assembly software simulations were conducted to visualize the diverse approaches to fast arithmetic presented in this paper, as well as to compare them to previous work. The adder tile constructions described in Sects. 4, 3 and 5, and the previous best known algorithm from Brun (2007) were simulated using the two timing models described in Sect. 2.4. These results can be seen in the graphs of Fig. 1.

2 Definitions

2.1 Basic notation

Let $\mathbb{N}_n$ denote the set $\{1, \ldots, n\}$ and let $\mathbb{Z}_n$ denote the set $\{0, \ldots, n-1\}$. Consider two points $p, q \in \mathbb{Z}^d$, $p = (p_1, \ldots p_d)$, $q = (q_1, \ldots, q_d)$. Define the *maximum norm* to be $\Delta_{p,q} \triangleq \max_{1 \le i \le d}\{|p_i - q_i|\}$ (Fig. 2).

2.2 Abstract tile assembly model

Tiles Consider some alphabet of symbols Π called the *glue types*. A tile is a finite edge polygon (polyhedron in the case of a 3D generalization) with some finite subset of border points each assigned some glue type from Π. Further, each glue type $g \in \Pi$ has some non-negative integer strength $str(g)$. For each tile t we also associate a finite string *label* (typically "0", or "1", or the empty label in this paper), denoted by label(t), which allows the classification of tiles by their labels. In this paper we consider a special class of tiles that are unit squares (or unit cubes in 3D) of the same orientation with at most one glue type per face, with each glue being placed exactly in the center of the tile's face. We denote the *location* of a tile to be the point at the center of the square or cube tile. In this paper we focus on tiles at integer locations.

Assemblies An assembly is a finite set of tiles whose interiors do not overlap. Further, to simplify formalization in this paper, we require the center of each tile in an assembly to be an integer coordinate (or integer triplet in 3D). If each tile in A is a translation of some tile in a set of tiles T, we say that A is an assembly over tile set T. For a given assembly Υ, define the *bond graph* G_Υ to be the weighted graph in which each element of Υ is a vertex, and the weight of an edge between two tiles is the strength of the overlapping matching glue points between the two tiles.

Table 1 Summary of results

	Time complexity	Average case
Addition(2D)	$\Theta(\sqrt{n})$ (Theorems 2, 4, 6)	$O(\log n)$ (Theorems 5, 6)
Addition(3D)	$\Theta(\sqrt[3]{n})$ (Theorems 2, 7)	$O(\log n)$ (Theorem 7)
Previous Best(2D)	$O(n)$ Brun (2007)	–
Multiplication(d-D)	$\Omega(\sqrt[d]{n})$ (Theorem 3)	–

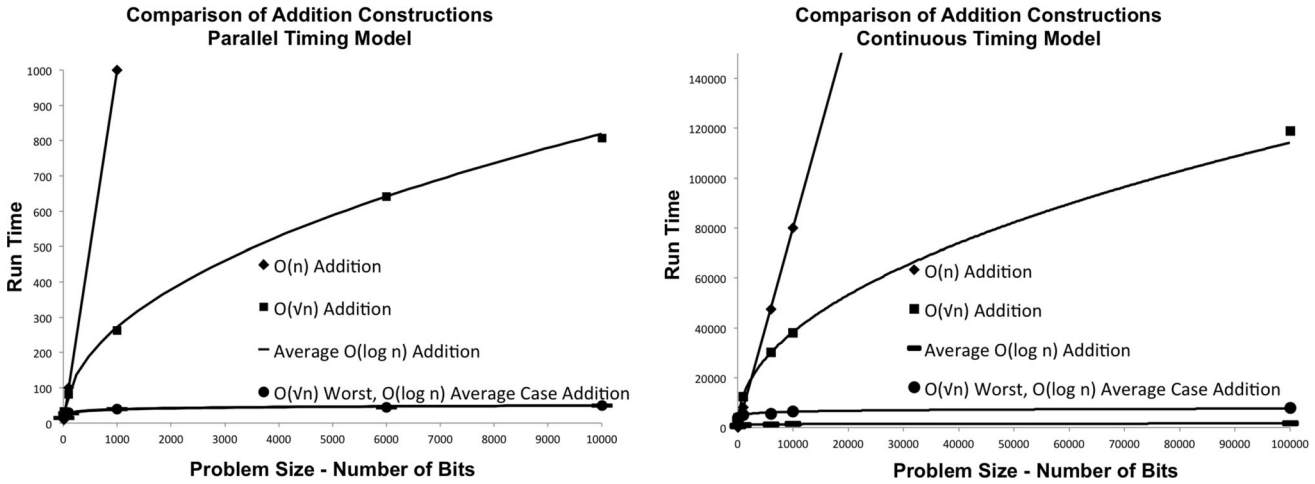

Fig. 1 Adder timing model simulations

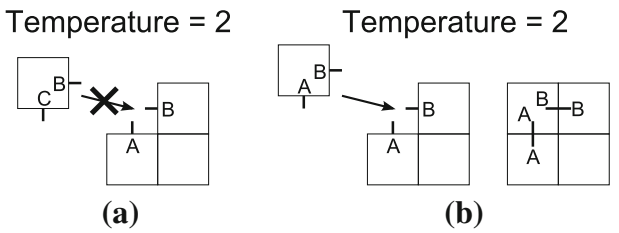

Fig. 2 Cooperative tile binding in the aTAM. **a** Incorrect binding. **b** Correct binding

Note that only overlapping glues that are the same type contribute a non-zero weight, whereas overlapping, non-equal glues always contribute zero weight to the bond graph. The property that only equal glue types interact with each other is referred to as the *diagonal glue function* property and is perhaps more feasible than more general glue functions for experimental implementation (see Cheng et al. (2005) for the theoretical impact of relaxing this constraint). An assembly Υ is said to be τ-*stable* for an integer τ if the min-cut of G_Υ is at least τ.

Tile attachment Given a tile t, an integer τ, and a τ-stable assembly A, we say that t may attach to A at temperature τ to form A' if there exists a translation t' of t such that $A' = A \bigcup \{t'\}$, and A' is τ-stable. For a tile set T we use notation $A \rightarrow_{T,\tau} A'$ to denote that there exists some $t \in T$ that may attach to A to form A' at temperature τ. When T and τ are implied, we simply say $A \rightarrow A'$. Further, we say that $A \rightarrow^* A'$ if either $A = A'$, or there exists a finite sequence of assemblies $\langle A_1 \ldots A_k \rangle$ such that $A \rightarrow A_1 \rightarrow \cdots \rightarrow A_k \rightarrow A'$.

Tile systems A tile system $\Gamma = (T, S, \tau)$ is an ordered triplet consisting of a set of tiles T referred to as the system's *tile set*, a τ-stable assembly S referred to as the system's *seed* assembly, and a positive integer τ referred to as the system's *temperature*. A tile system $\Gamma = (T, S, \tau)$ has an associated set of *producible* assemblies, $\texttt{PROD}_\Gamma$, which define what assemblies can grow from the initial seed S by any sequence of temperature τ tile attachments from T. Formally, $S \in \texttt{PROD}_\Gamma$ is a base case producible assembly. Further, for every $A \in \texttt{PROD}_\Gamma$, if $A \rightarrow_{T,\tau} A'$, then $A' \in \texttt{PROD}_\Gamma$. That is, assembly S is producible, and for every producible assembly A, if A can grow into A', then A' is also producible. We further denote a producible assembly A to be *terminal* if A has no attachable tile from T at temperature τ. We say a system $\Gamma = (T, S, \tau)$ *uniquely produces* an assembly A if all producible assemblies can grow into A through some sequence of tile attachments. More formally, Γ *uniquely produces* an assembly $A \in \texttt{PROD}_\Gamma$ if for every $A' \in \texttt{PROD}_\Gamma$ it is the case that $A' \rightarrow^* A$. Systems that uniquely produce one assembly are said to be *deterministic*. In this paper, we focus exclusively on deterministic systems, and our general goal will be to design systems whose uniquely produced assembly specifies the solution to a computational problem. For recent consideration of non-determinism in tile self-assembly see Chandran et al. (2009), Bryans et al. (2011), Cook et al. (2011), Kao et al. (2008) and Doty (2010).

2.3 Problem description

We now formalize what we mean for a tile self-assembly system to compute a function. To do this we present the concept of a *tile assembly computer* (TAC) which consists of a tile set and temperature parameter, along with input and output *templates*. The input template serves as a seed structure with a sequence of *wildcard positions* for which tiles of label "0" and "1" may be placed to construct an initial seed assembly. An output template is a sequence of points denoting locations for which the TAC, when grown

from a filled in template, will place tiles with "0" and "1" labels that denote the output bit string. A TAC then is said to compute a function f if for every seed assembly derived by plugging in a bitstring b, the terminal assembly of the system with tile set T and temperature τ will be such that the value of $f(b)$ is encoded in the sequence of tiles placed according to the locations of the output template. We now develop the formal definition of the TAC concept. We note that the formality in the input template is of substantial importance. Simpler definitions which map seeds to input bit strings, and terminal assemblies to output bitstrings, are problematic in that they allow for the possibility of encoding the computation of function f in the seed structure. Even something as innocuous sounding as allowing more than a single type of "0" or "1" tile as an input bit has the subtle issue of allowing pre-computing of f.[1]

Input template Consider a tile set T containing exactly one tile t_0 with label "0", and one tile t_1 with label "1". An n-bit input template over tile set T is an ordered pair $U = (R, B(i))$, where R is an assembly over $T - \{t_0, t_1\}$, $B : \mathbb{N}_n \rightarrow \mathbb{Z}^2$, and $B(i)$ is not the position of any tile in R for every i from 1 to n. The sequence of n coordinates denoted by B conceptually denotes "wildcard" tile positions for which copies of t_0 and t_1 will be filled in for every instance of the template. For notation we define assembly U_b over T, for bit string $b = b_1, \ldots b_n$, to be the assembly consisting of assembly R unioned with a set of n tiles t^i for i from 1 to n, where t^i is equal a translation of tile $t_{b(i)}$ to position $B(i)$. That is, U_b is the assembly R with each position $B(i)$ tiled with either t_0 or t_1 according to the value of $B(i)$.

Output template A k-bit output template is simply a sequence of k coordinates denoted by function $C : \mathbb{N}_k \rightarrow \mathbb{Z}^2$. For an output template V, an assembly A over T is said to represent binary string $c = c_1, \ldots, c_k$ over template V if the tile at position $C(i)$ in A has label c_i for all i from 1 to k. Note that output template solutions are much looser than input templates in that there may be multiple tiles with labels "1" and "0", and there are no restrictions on the assembly outside of the k specified wildcard positions. The strictness for the input template stems from the fact that the input must "look the same" in all ways except for the explicit input bit patterns. If this were not the case, it would likely be possible to encode the solution to the computational problem into the input template, resulting is a trivial solution.

Function computing problem A *tile assembly computer* (TAC) is an ordered quadruple $\mathfrak{I} = (T, U, V, \tau)$ where T is a tile set, U is an n-bit input template, and V is a k-bit output template. A TAC is said to compute function $f : \mathbb{Z}_2^n \rightarrow \mathbb{Z}_2^k$ if for any $b \in \mathbb{Z}_2^n$ and $c \in \mathbb{Z}_2^k$ such that $f(b) = c$, then the tile system $\Gamma_{\mathfrak{I},b} = (T, U_b, \tau)$ uniquely assembles an assembly A which represents c over template V. For a TAC $\mathfrak{I}$ that computes the function $f : \mathbb{Z}_2^{2n} \rightarrow \mathbb{Z}_2^{n+1}$ where $f(r_1 \ldots r_{2n}) = r_1 \ldots r_n + r_{n+1} \ldots r_{2n}$, we say that $\mathfrak{I}$ is an n-bit *adder* TAC with inputs $a = r_1 \ldots r_n$ and $b = r_{n+1} \ldots r_{2n}$. An n-bit *multiplier* TAC is defined similarly.

[1] This subtle issue seems to exist with some previous formulations of tile assembly computation.

2.4 Run time models

We analyze the complexity of self-assembly arithmetic under two established run time models for tile self-assembly: the *parallel* time model (Becker et al. 2006; Brun 2007) and the *continuous* time model (Adleman et al. 2001, 2002; Cheng et al. 2004; Becker et al. 2006). Informally, the parallel time model simply adds, in parallel, all singleton tiles that are attachable to a given assembly within a single time step. The continuous time model, in contrast, models the time taken for a single tile to attach as an exponentially distributed random variable. The parallelism of the continuous time model stems from the fact that if an assembly has a large number of attachable positions, then the *first* tile to attach will be an exponentially distributed random variable with rate proportional to the number of attachment sites, implying that a larger number of attachable positions will speed up the expected time for the next attachment. When not otherwise specified, we use the term *run time* to refer to parallel run time by default. The stated asymptotic worst case and average case run times for each of our algorithms hold within both run time models.

Parallel run-time For a deterministic tile system $\Gamma = (T, S, \tau)$ and assembly $A \in \mathtt{PROD}_\Gamma$, the 1-*step transition* set of assemblies for A is defined to be $STEP_{\Gamma,A} = \{B \in \mathtt{PROD}_\Gamma | A \rightarrow_{T,\tau} B\}$. For a given $A \in \mathtt{PROD}_\Gamma$, let $\mathtt{PARALLEL}_{\Gamma,A} = \bigcup_{B \in STEP_{\Gamma,A}} B$, ie, $\mathtt{PARALLEL}_{\Gamma,A}$ is the result of attaching all singleton tiles that can attach directly to A. Note that since Γ is deterministic, $\mathtt{PARALLEL}_{\Gamma,A}$ is guaranteed to not contain overlapping tiles and is therefore an assembly. For an assembly A, we say $A \rightrightarrows_\Gamma A'$ if $A' = \mathtt{PARALLEL}_{\Gamma,A}$. We define the *parallel run-time* of a deterministic tile system $\Gamma = (T, S, \tau)$ to be the non-negative integer k such that $A_1 \rightrightarrows_\Gamma A_2 \rightrightarrows_\Gamma \ldots \rightrightarrows_\Gamma A_k$ where $A_1 = S$ and $\{A_k\} = \mathtt{TERM}_\Gamma$. As notation we denote this value for tile system Γ as ρ_Γ. For any assemblies A and B in $\mathtt{PROD}_\Gamma$ such that $A_1 \rightrightarrows_\Gamma A_2 \rightrightarrows_\Gamma \ldots \rightrightarrows_\Gamma A_k$ with $A = A_1$ and $B = A_k$, we say that $A \rightrightarrows_\Gamma^k B$. Alternately, we denote B with notation $A \rightrightarrows_\Gamma^k$. For a TAC $\mathfrak{I} = (T, U, V, \tau)$ that computes function f, the run time of $\mathfrak{I}$ on input b is defined to be the parallel run-time of tile system $\Gamma_{\mathfrak{I},b} = (T, U_b, \tau)$. Worst case and average case run time are then defined in terms of the largest run time inducing b and the average run time for a uniformly generated random b.

Continuous run-time In the *continuous* run time model the assembly process is modeled as a continuous time Markov chain and the time spent in each assembly is a random variable with an exponential distribution. The states of the Markov chain are the elements of $PROD_\Gamma$ and the transition rate from A to A' is $\frac{1}{|T|}$ if $A \to A'$ and 0 otherwise. For a given tile assembly system Γ, we use notation Γ_{MC} to denote the corresponding continuous time Markov chain. Given a deterministic tile system Γ with unique assembly U, the run time of Γ is defined to be the expected time of Γ_{MC} to transition from the seed assembly state to the unique sink state U. As notation we denote this value for tile system Γ as ς_Γ. One immediate useful fact that follows from this modeling is that for a producible assembly A of a given deterministic tile system, if $A \to A'$, then the time taken for Γ_{MC} to transition from A to the first A'' that is a superset of A' (through 1 or more tile attachments) is an exponential random variable with rate $1/|T|$. For a more general modeling in which different tile types are assigned different rate parameters see Adleman et al. (2001, 2002) and Cheng et al. (2004). Our use of rate $1/|T|$ for all tile types corresponds to the assumption that all tile types occur at equal concentration and the expected time for 1 tile (of any type) to hit a given attachable position is normalized to 1 time unit. Note that as our tile sets are constant in size in this paper, the distinction between equal or non-equal tile type concentrations does not affect asymptotic run time. For a TAC $\Im = (T, U, V, \tau)$ that computes function f, the run time of $\Im$ on input b is defined to be the continuous run-time of tile system $\Gamma_{\Im,b} = (T, U_b, \tau)$. Worst case and average case run time are then defined in terms of the largest run time inducing b and the average run time for a uniformly generated random b.

2.5 Communication problem

The Δ-communication problem is the problem of computing the function $f(b_1, b_2) = b_1 \wedge b_2$ for bits b_1 and b_2 in the 3D aTAM under the additional constraint that the input template for the solution $U = (R, B(i))$ be such that $\Delta = \max(|B(1)_x - B(2)_x|, |B(1)_y - B(2)_y|, |B(1)_z - B(2)_z|)$.

We first establish a lemma which intuitively states that for any 2 seed assemblies that differ in only a single tile position, all points of distance greater than r from the point of difference will be identically tiled (or empty) after r time steps of parallelized tile attachments:

Lemma 1 *Let $S_{p,t}$ and $S_{p,t'}$ denote two assemblies that are identical except for a single tile t versus t' at position $p = (p_x, p_y, p_z)$ in each assembly. Further, let $\Gamma = (T, S_{p,t}, \tau)$ and $\Gamma' = (T, S_{p,t'}, \tau)$ be two deterministic tile assembly systems such that $S_{p,t} \rightrightarrows^r_{\Gamma'} R$ and $S_{p,t'} \rightrightarrows^r_{\Gamma'} R'$ for non-negative integer r. Then for any point $q = (q_x, q_y, q_z)$ such that $r < \max(|p_x - q_x|, |p_y - q_y|, |p_z - q_z|)$, it must be that $R_q = R'_q$, ie, R and R' contain the same tile at point q.*

Proof We show this by induction on r. As a base case of $r = 0$, we have that $R = S_{p,t}$ and $R' = S_{q,t'}$, and therefore R and R' are identical at any point outside of point p by the definition of $S_{p,t}$ and $S_{p,t'}$.

Inductively, assume that for some integer k we have that for all points w such that $k < \Delta_{p,w} \triangleq \max(|p_x - w_x|, |p_y - w_y|, |p_z - w_z|)$, we have that $R^k_w = R'^k_w$, where $S_{p,t} \rightrightarrows^k_\Gamma R^k$, and $S_{p,t'} \rightrightarrows^k_{\Gamma'} R'^k$. Now consider some point q such that $k + 1 < \Delta_{p,q} \triangleq \max |p_x - q_x|, |p_y - q_y|, |p_z - q_z|$, along with assemblies R^{k+1} and R'^{k+1} where $S_{p,t} \rightrightarrows^{k+1} R^{k+1}$, and $S_{p,t'} \rightrightarrows^{k+1} R'^{k+1}$. Consider the direct neighbors (6 of them in 3D) of point q. For each neighbor point c, we know that $\Delta_{p,c} > k$. Therefore, by inductive hypothesis, $R^k_c = R'^k_c$ where $S_{p,t} \rightrightarrows^k_\Gamma R^k$, and $S_{p,t'} \rightrightarrows^k_{\Gamma'} R'^k$. Therefore, as attachment of a tile at a position is only dependent on the tiles in neighboring positions of the point, we know that tile R^{k+1}_q may attach to both R^k and R'^k at position q, implying that $R^{k+1}_q = R'^{k+1}_q$ as Γ and Γ' are deterministic. □

Theorem 1 *Any solution to the Δ-communication problem has run time at least $\frac{1}{2}\Delta$.*

Proof Consider a TAC $\Im = (T, U = (R, B(i)), V(i), \tau)$ that computes the Δ-communication problem. First, note that B has domain of 1 and 2, and V has domain of just 1 (the input is 2 bits, the output is 1 bit). We now consider the value Δ_V defined to be the largest distance between the output bit position of V from either of the two input bit positions in B: Let $\Delta_V \triangleq \max(\Delta_{B(1),V(1)}, \Delta_{B(2),V(1)})$. Without loss of generality, assume $\Delta_V = \Delta_{B(1),V(1)}$. Note that $\Delta_V \geq \frac{1}{2}\Delta$.

Now consider the computation of $f(0, 1) = 0$ versus the computation of $f(1, 1) = 1$ via our TAC $\Im$. Let A^0 denote the terminal assembly of system $\Gamma_0 = (T, U_{0,1}, \tau)$ and let A^1 denote the terminal assembly of system $\Gamma_1 = (T, U_{1,1}, \tau)$. As $\Im$ computes f, we know that $A^0_{V(1)} \neq A^1_{V(1)}$. Further, from Lemma 1, we know that for any $r < \Delta_V$, we have that $W^0_{V(1)} = W^1_{V(1)}$ for any W^0 and W^1 such that $U_{0,1} \rightrightarrows^r_{\Gamma_0} W^0$ and $U_{1,1} \rightrightarrows^r_{\Gamma_1} W^1$. Let $d_\Im$ denote the run time of $\Im$. Then we know that $U_{0,1} \rightrightarrows^{d_\Im}_{\Gamma_0} A^0$, and $U_{1,1} \rightrightarrows^{d_\Im}_{\Gamma_1} A^1$ by the definition of run time. If $d_\Im < \Delta_V$, then Lemma 11 implies that that $A^0_{V(1)} = A^1_{V(1)}$, which contradicts the fact that $\Im$ compute f. Therefore, the run time $d_\Im$ is at least $\Delta_V \geq \frac{1}{2}\Delta$. □

2.6 $\Omega(\sqrt[d]{n})$ Lower bounds for addition and multiplication

We now show how to reduce instances of the communication problem to the arithmetic problems of addition and multiplication in 2D and 3D to obtain lower bounds of $\Omega(\sqrt{n})$ and $\Omega(\sqrt[3]{n})$ respectively.

We first show the following Lemma which lower bounds the distance of the farthest pair of points in a set of n points. We believe this Lemma is likely well known, or is at least the corollary of an established result and omit it's proof.

Lemma 2 *For positive integers n and d, consider* $A \subset \mathbb{Z}^d$ *and* $B \subset \mathbb{Z}^d$ *such that* $A \bigcap B = \emptyset$ *and* $|A| = |B| = n$. *There must exist points* $p \in A$ *and* $q \in B$ *such that* $\Delta_{p,q} \geq \lceil \frac{1}{2} \lceil \sqrt[d]{2n} \rceil \rceil - 1$.

Theorem 2 *Any n-bit adder TAC that has a dimension d input template for* $d = 1$, $d = 2$, *or* $d = 3$, *has a worst case run time of* $\Omega(\sqrt[d]{n})$.

Proof To show the lower bound, we will reduce the Δ-communication problem for some $\Delta = \Omega(\sqrt[d]{n})$ to the n-bit adder problem with a d-dimension template. Consider some n-bit adder TAC $\mathfrak{I} = (T, U = (F, W), V, \tau)$ such that U is a d-dimension template. The $2n$ sequence of wildcard positions W of this TAC must be contained in d-dimensional space by the definition of a d-dimension template, and therefore by Lemma 2 there must exist points $W(i)$ for $i \leq n$, and $W(n+j)$ for $j \leq n$, such that $\Delta_{W(i),W(n+j)} \geq \lceil \frac{1}{2} \lceil \sqrt[d]{2n} \rceil \rceil - 1 = \Omega(\sqrt[d]{n})$. Now consider two n-bit inputs $a = a_n \ldots a_1$ and $b = b_n \ldots b_1$ to the adder TAC $\mathfrak{I}$ such that: $a_k = 0$ for any $k > i$ and any $k < j$, and $a_k = 1$ for any k such that $j \leq k < i$. Further, let $b_k = 0$ for all $k \neq j$. The remaining bits a_i and a_j are unassigned variables of value either 0 or 1. Note that the $i+1$ bit of $a+b$ is 1 if and only if a_i and b_j are both value 1. This setup constitutes our reduction of the Δ-communication problem to the addition problem as the adder TAC template with the specified bits hardcoded in constitutes a template for the Δ-communication problem that produces the AND of the input bit pair. We now specify explicitly how to generate a communication TAC from a given adder TAC.

For given n-bit adder TAC $\mathfrak{I} = (T, U = (F, W), V, \tau)$ with dimension d input template, we derive a Δ-communication TAC $\rho = (T, U^2 = (F^2, W^2), V^2, \tau)$ as follows. First, let $W^2(1) = W(i)$, and $W^2(2) = W(n+j)$. Note that as $\Delta_{W(i),W(n+j)} = \Omega(\sqrt[d]{n})$, W^2 satisfies the requirements for a Δ-communication input template for some $\Delta = \Omega(\sqrt[d]{n})$. Derive the frame of the template F^2 from F by adding tiles to F as follows: For any positive integer $k > i$, or $k < j$, or $k > n$ but not $k = n + j$, add a translation of t_0 (with label "0") translated to position $W(k)$. Additionally, for any k such that $j \leq k < i$, add a translation of t_1 (with label "1") at translation $W(k)$.

Now consider the Δ-communication TAC $\rho = (T, U^2 = (F^2, W^2), V^2, \tau)$ for some $\Delta = \Omega(\sqrt[d]{n})$. As assembly $U^2_{a_i,b_j} = U_{a_1 \ldots a_n, b_1 \ldots b_n}$, we know that the worst case run time of ρ is at most that of the worst case run time of $\mathfrak{I}$. Therefore, by Theorem 1, we have that $\mathfrak{I}$ has a run time of at least $\Omega(\sqrt[d]{n})$. □

Theorem 3 *Any n-bit multiplier TAC that has a dimension d input template for* $d = 1$, $d = 2$, *or* $d = 3$, *has a worst case run time of* $\Omega(\sqrt[d]{n})$.

Proof Consider some n-bit multiplier TAC $\mathfrak{I} = (T, U = (F, W), V, \tau)$ with d-dimension input template. By Lemma 2, some $W(i)$ and $W(n+j)$ must have distance at least $\Delta \geq \lceil \frac{1}{2} \lceil \sqrt[d]{2n} \rceil \rceil - 1$. Now consider input strings $a = a_n \ldots a_1$ and $b = b_n \ldots b_1$ to $\mathfrak{I}$ such that a_i and b_j are of variable value, and all other a_k and b_k have value 0. For such input strings, the $i+j$ bit of the product ab has value 1 if and only if $a_i = b_j = 1$. Thus, we can convert the n-bit multiplier system $\mathfrak{I}$ into a Δ-communication TAC with the same worst case run time in the same fashion as for Theorem 2, yielding a $\Omega(\sqrt[d]{n})$ lower bound for the worst case run time of $\mathfrak{I}$. □

As with addition, the lower bound implied by the limited dimension of the input template alone yields the general lower bound for d dimensional multiplication TACs.

3 Optimal 2D addition

We now present an adder TAC that achieves a run time of $O(\sqrt{n})$, which matches the lower bound from Theorem 2. This adder TAC closely resembles an electronic carry-select adder in that the addends are divided into sections of size $\sqrt{n}$ and the sum of the addends comprising each section is computed for both possible carry-in values. The correct result for the subsection is then selected after a carry-out has been propagated from the previous subsection. Within each subsection, the addition scheme resembles a ripple-carry adder.

Theorem 4 *There exists an n-bit adder TAC at temperature 2 with a worst case run-time of* $O(\sqrt[2]{n})$.

Proof Please see Sects. 3.1–3.2. For additional detail, see Keenan et al. (2013). □

3.1 Construction

Input/output template Figure 3a, b are examples of I/O templates for a 9-bit adder TAC. The inputs to the addition problem in this instance are two 9-bit binary numbers A and B with the least significant bit of A and B represented by A_0 and B_0, respectively. The north facing glues in the form of A_i or B_i in the input template must either be a 1 or a 0 depending on the value of the bit in A_i or B_i. The placement for these tiles is shown in Fig. 3a while a specific example of a possible input template is shown in Fig. 4a. The sum of $A + B$, C, is a ten bit binary number where C_0 represents the least significant bit. The placement for the tiles representing the result of the addition is shown in Fig. 3b.

To construct an n-bit adder in the case that n is a perfect square, split the two n-bit numbers into $\sqrt{n}$ sections each with $\sqrt{n}$ bits. Place the bits for each of these two numbers as per the previous paragraph, except with $\sqrt{n}$ bits per row, making sure to alternate between A and B bits. There will be the same amount of space between each row as seen in the example template Fig. 3a. All Z, N_C, and F', must be placed in the same relative locations. The solution, C, will be in the output template s.t. C_i will be three tile positions above B_i and a total of size $n + 1$.

Below, we use the adder tile set to add two nine-bit numbers: $A = 100110101$ and $B = 110101100$ to demonstrate the three stages in which the adder tile system performs addition.

Step one: Addition With the inclusion of the seed assembly (Fig. 4a) to the tile set (Fig. 5), the first subset of tiles able to bind are the addition tiles shown in Fig. 5a.

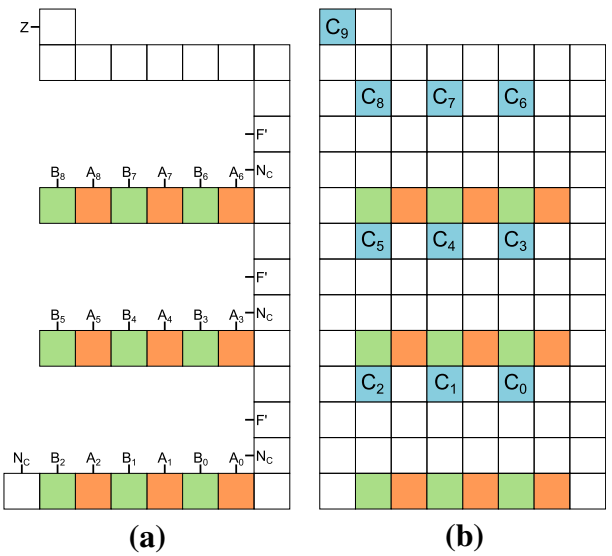

Fig. 3 These are example I/O templates for the worst case $O(\sqrt{n})$ time addition introduced in Sect. 3. **a** Addition input template. **b** Addition output template

These tiles sum each pair of bits from A and B (for example, $A_0 + B_0$) (Fig. 4b). Tiles shown in yellow are actively involved in adding and output the sum of each bit pair on the north face glue label. Yellow tiles also output a carry to the next more significant bit pair, if one is needed, as a west face glue label. Spacer tiles (white) output a B glue on the north face and serve to propagate carry information and the value of A_i to the adjacent B_i. Each row computes this addition step independently and outputs a carry or no-carry west face glue on the westernmost tile of each row (Fig. 4c). In a later step, this carry or no-carry information will be propagated northwards from the southernmost row in order to determine the sum. Note that immediately after the first addition tile is added to a row of the seed assembly, a second layer may form by the attachment of tiles from the increment tile set (Fig. 5c). While these two layers may form nearly concurrently, we separate them in this example for clarity and instead address the formation of the second layer of tiles in *Step Two: Increment* below.

Step two: Increment As the addition tiles bind to the seed, tiles from the incrementation tile set (Fig. 5c) may also begin to cooperatively attach. For clarity, we show their attachment following the completion of the addition layer. The purpose of the incrementation tiles is to determine the sum for each A and B bit pair in the event of a no-carry from the row below and in the event of a carry from the row below (Fig. 6). The two possibilities for each bit pair are presented as north facing glues on yellow increment tiles. These north face glues are of the form (x, y) where x represents the value of the sum in the event of no-carry from the row below while y represents the value of the sum in the event of a carry from the row below. White incrementation tiles are used as spacers, with the sole purpose of passing along carry or no-carry information via their east/west face glues F', which represents a no-carry, and F, which represents a carry.

Step three: Carry propagation and output The final step of the addition mechanism presented here propagates carry or no-carry information northwards from the southernmost row of the assembly using tiles from the tile set in Fig. 5b and then outputs the answer using the tile set in Fig. 5d. Following completion of the incrementation layers, tiles may begin to grow up the west side of the assembly as shown in Fig. 7a. When the tiles grow to a height such that the empty space above the increment row is presented with a carry or no-carry as in Fig. 7b, the output tiles may begin to attach from west to east to print the answer (Fig. 7c). As the carry propagation column grows northwards and presents carry or no carry information to each empty space above each increment layer, the sum may be printed for each row Fig. 7d, e. When the carry propagation column reaches the top of the assembly, the most significant bit of

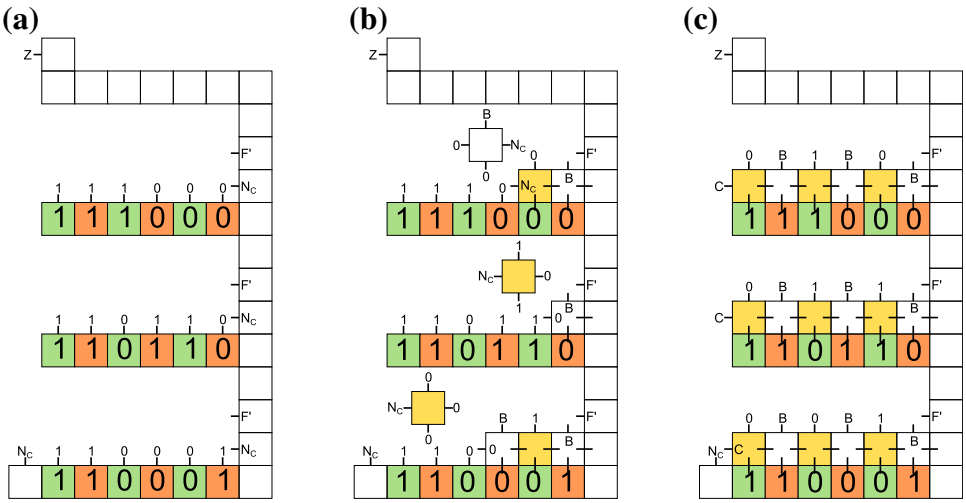

Fig. 4 Step 1: addition

the sum may be determined and the calculation is complete (Fig. 7f).

3.2 Time complexity

3.2.1 Parallel time

Using the parallel runtime model described in Sect. 2.4 we will show that the addition algorithm presented in this section has a worst case runtime of $O(\sqrt{n})$. In order to ease the analysis we will assume that each logical step of the algorithm happens in a synchronized fashion even though parts of the algorithm are running concurrently.

The first step of the algorithm is the addition of two $O(\sqrt{n})$ numbers co-located on the same row. This first step occurs by way of a linear growth starting from the leftmost bit all the way to the rightmost bit of the row. The growth of a line one tile at a time has a runtime on the order of the length of the line. In the case of our algorithm, the row is of size $O(\sqrt{n})$ and so the runtime for each row is $O(\sqrt{n})$. The addition of each of the $\sqrt{n}$ rows happens independently, in parallel, leading to a $O(\sqrt{n})$ runtime for all rows. Next, we increment each solution in each row of the addition step, keeping both the new and old values. As with the first step, each row can be completed independently in parallel by way of a linear growth across the row leading to a total runtime of $O(\sqrt{n})$ for this step. After we increment our current working solutions we must both generate and propagate the carries for each row. In our algorithm, this simply involves growing a line across the leftmost wall of the rows. The size of the wall is bounded by $O(\sqrt{n})$ and so this step takes $O(\sqrt{n})$ time. Finally, in order to output the result bits into their proper places we simply grow a line atop the line created by the increment step. This step has the same runtime properties as the addition and increment steps. Therefore, the output step has a runtime of $O(\sqrt{n})$ to output all rows.

There are four steps each taking $O(\sqrt{n})$ time leading to a total runtime of $O(\sqrt{n})$ for this algorithm. This upper bound meets the lower bound presented in Theorem 2 and the algorithm is therefore optimal.

Choice of $\sqrt{n}$ rows of $\sqrt{n}$ size The choice for dividing the bits up into a $\sqrt{n} \times \sqrt{n}$ grid is straightforward. Imagine that instead of using $O(\sqrt{n})$ bits per row, a much smaller growing function such as $O(\log n)$ bits per row is used. Then, each row would finish in $O(\log n)$ time. After each row finishes, we would have to propagate the carries. The length of the west wall would no longer be bound by the slow growing function $O(\sqrt{n})$ but would now be bound by the much faster growing function $O(\frac{n}{\log n})$. Therefore, there is a distinct trade off between the time necessary to add each row and the time necessary to propagate the carry with this scheme. The runtime of this algorithm can be viewed as the $max(row_{size}, westwall_{size})$. The best way to minimize this function is to divide the rows such that we have the same number of rows as columns, i.e. the smallest partition into the smallest sets. The best way to partition the bits is therefore into $\sqrt{n}$ rows of $\sqrt{n}$ bits.

3.2.2 Continuous time

To analyze the continuous run time of our TAC we first introduce a Lemma 4. This Lemma is essentially Lemma 5.2 of Woods et al. (2012) but slightly generalized and modified to be a statement about exponentially distributed random variables rather than a statement specific to the self-assembly model considered in that paper. To derive this Lemma, we make use of a well known Chernoff bound

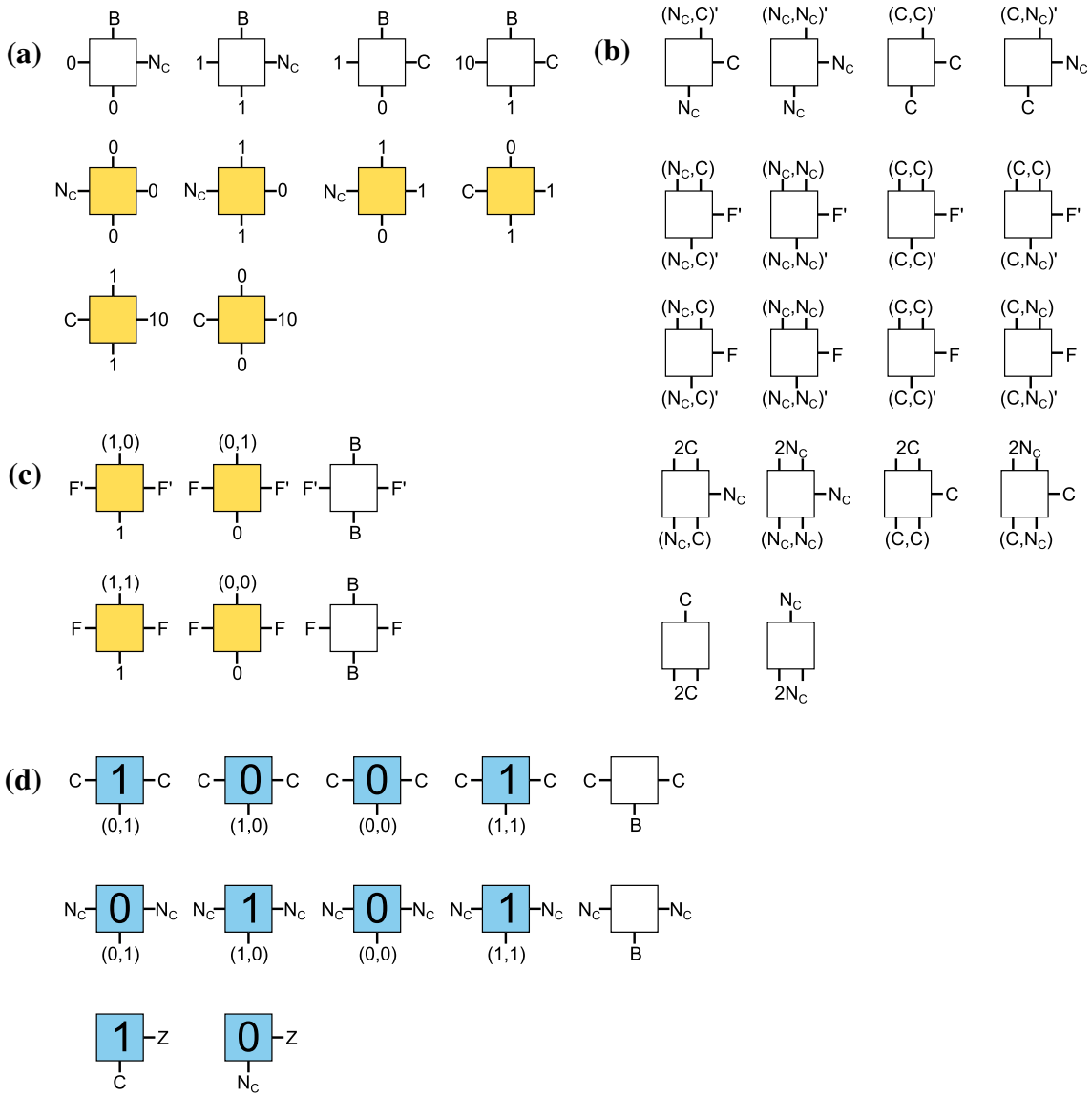

Fig. 5 The complete tile set. **a** Tiles involved in bit addition. **b** Carry propagation tiles. **c** Tiles involved in incrementation. **d** Tiles that print the answer

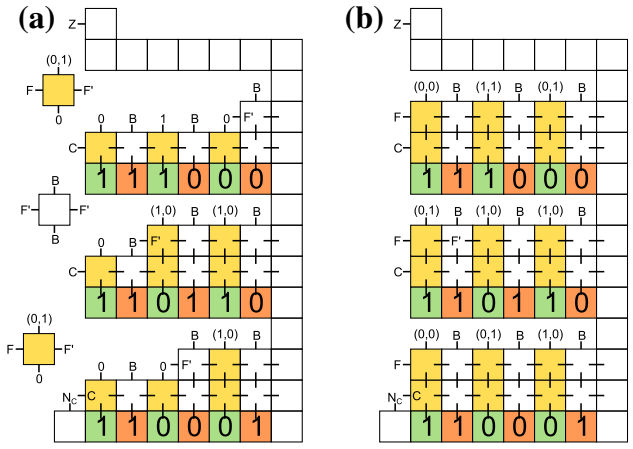

Fig. 6 Step 2: increment

for the sum of exponentially distributed random variables stated in Lemma 3.

Lemma 3 *Let* $X = \sum_{i=1}^{n} X_i$ *be a random variable denoting the sum of n independent exponentially distributed random variables each with respective rate* λ_i. *Let* $\lambda = \min\{\lambda_i | 1 \leq i \leq n\}$. *Then* $Pr[X > (1+\delta)n/\lambda] < (\frac{\delta+1}{e^{\delta}})^n$.

Lemma 4 *Let* $t_1, \ldots, t_m$ *denote m random variables where each* $t_i = \sum_{j=1}^{k_i} t_{i,j}$ *for some positive integer* k_i, *and the variables* $t_{i,j}$ *are independent exponentially distributed random variable each with respective rate* $\lambda_{i,j}$. *Let* $k = \max(k_1, \ldots k_m)$, $\lambda = \min\{\lambda_{i,j} | 1 \leq i \leq m, 1 \leq j \leq k_i\}$, $\lambda' = \max\{\lambda_{i,j} | 1 \leq i \leq m, 1 \leq j \leq k_i\}$, *and* $T = \max(t_1, \ldots t_m)$. *Then* $E[T] = O(\frac{k+\log m}{\lambda})$ *and* $E[T] = \Omega(\frac{k+\log m}{\lambda'})$.

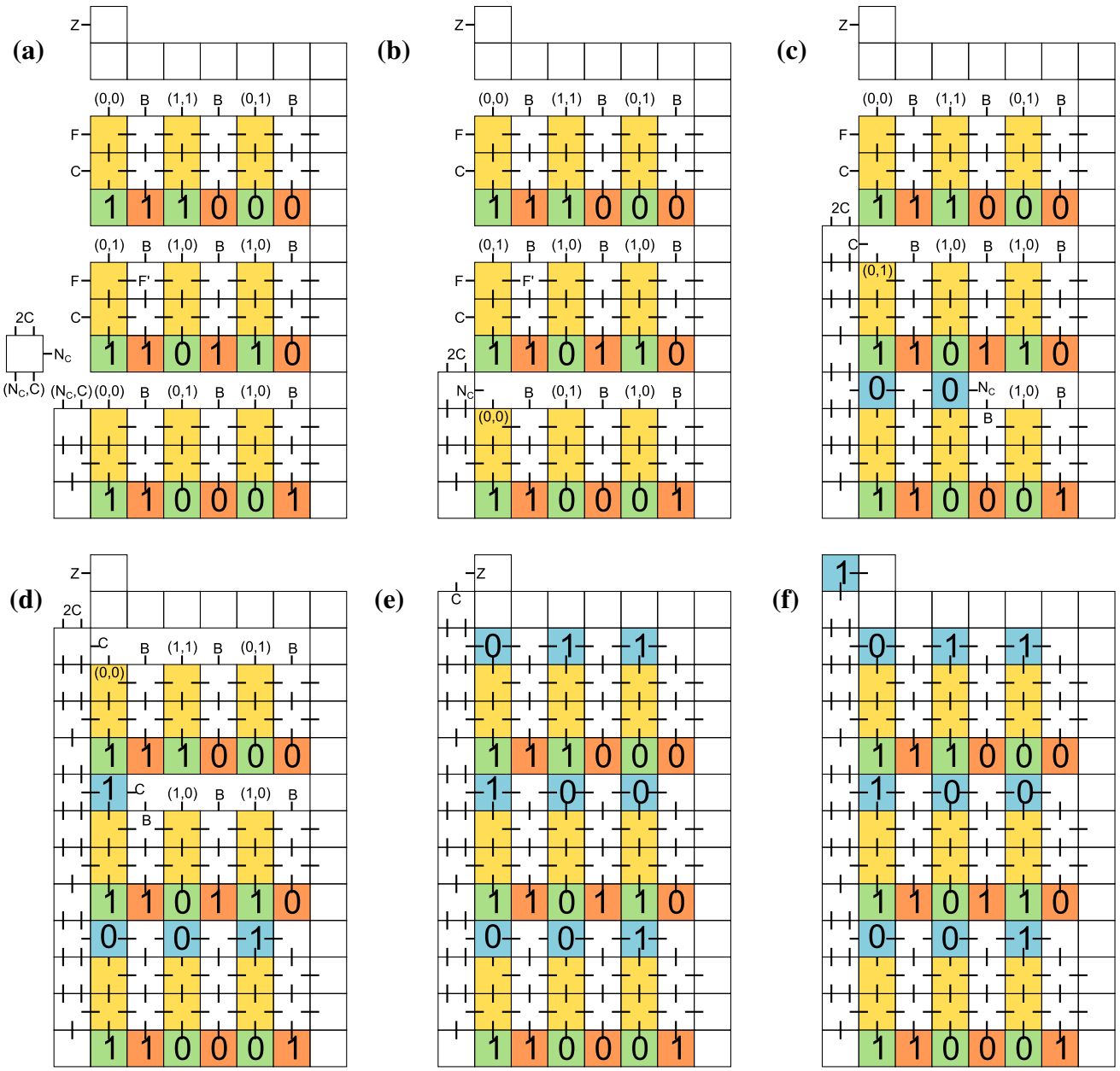

Fig. 7 Step 3: carry propagation and output

Proof First we show that $E[T] = O(\frac{k+\log m}{\lambda})$. For each t_i we know that $E[t_i] = \sum_{j=1}^{k_i} \lambda_{i,j} \leq k/\lambda$. Applying Lemma 3 we get that:

$$Pr[t_i > k/\lambda(1+\delta)] < \left(\frac{\delta+1}{e^{\delta}}\right)^{k}.$$

Applying the union bound we get the following for all $\delta \geq 3$:

$$Pr\left[T > \frac{(1+\delta)k}{\lambda}\right] \leq m\left(\frac{\delta+1}{e^{\delta}}\right)^{k} \leq me^{-k\delta/2}, \quad \text{for all } \delta > 3.$$

Let $\delta_m = \frac{2\ln m}{k}$. By plugging in $\delta_m + x$ for δ in the above inequality we get that:

$$Pr\left[T > \frac{(1+\delta_m+x)k}{\lambda}\right] \leq e^{-kx/2}.$$

Therefore we know that:

$$E[T] \leq \frac{k+2\ln m}{\lambda} + \int_{x=0}^{\infty} e^{-kx/2} = O\left(\frac{k+\log m}{\lambda}\right).$$

To show that $E[T] = \Omega(\frac{k+\log m}{\lambda'})$, first note that $T \geq \max_{1\leq i\leq m}\{t_{i,1}\}$ and that $E[\max_{1\leq i\leq m}\{t_{i,1}\}] =$

$\Omega(\frac{\log m}{\lambda'})$, implying that $E[T] = \Omega(\frac{\log m}{\lambda'})$. Next, observe that $T \geq t_i$ for each i. Since $E[t_i]$ is at least k/λ' for at least one choice of i, we therefore get that $E[T] = \Omega(k/\lambda')$. □

Lemma 5 helps us upper bound continuous run times by stating that if the assembly process is broken up into separate phases, denoted by a sequence of subassembly *waypoints* that must be reached, we may simply analyze the expected time to complete each phase and sum the total for an upper bound on the total run time.

Lemma 5 *Let $\Gamma = (T, S, \tau)$ be a deterministic tile assembly system and $A_0 \ldots A_r$ be elements of $PROD_\Gamma$ such that $A_0 = S$, A_r is the unique terminal assembly of Γ, and $A_0 \subseteq A_2 \subseteq \ldots A_r$. Let t_i be a random variable denoting the time for Γ_{MC} to transition from A_{i-1} to the first A'_i such that $A_i \subseteq A'_i$ for i from 1 to r. Then $\varsigma_\Gamma \leq \sum_{i=1}^{r} E[t_i]$.*

Proof This lemma follows easily from the fact that a given open tile position of an assembly is filled at a rate at least as high as any smaller subassembly. □

To bound the continuous run time of our TAC for any pair of length n-bit input strings we break the assembly process up into 4 phases. Let the phases be defined by 5 producible assemblies $S = A_0, A_1, A_2, A_3, A_4 = F$, where F is the final terminal assembly of our TAC. Let t_i denote a random variable representing the time taken to grow from A_{i-1} to the first assembly A' such that A' is a superset of A_i. Note that $E[t_1] + E[t_2] + E[t_3] + E[t_4]$ is an upper bound for the continuous time of our TAC according to Lemma 5. We now specify the four phase assemblies and bound the time to transition between each phase to achieve a bound on the continuous run time of our TAC for any input.

Let A_1 be the seed assembly plus the attachment of one layer of tiles above each of the input bits (see Fig. 4c for an example assembly). Let A_2 be assembly A_1 with the second layer of tiles above the input bits placed (see Fig. 6b). Let A_3 be assembly A_2 with the added vertical chain of tiles at the western most edge of the completed assembly, all the way to the top ending with the final blue output tile. Finally, let A_4 be the final terminal assembly of the system, which is the assembly A_3 with the added third layer of tiles attached above the input bits.

Time t_1 is the maximum of time taken to complete the attachment of the 1st row of tiles above input bits for each of the $\sqrt{n}$ rows. As each row can begin independently, and each row grows from right to left, with each tile position waiting for the previous tile position, each row completes in time dictated by a random variable which is the sum of $\sqrt{n}$ independent exponentially distributed random variables with rate $1/|T|$, where T is the tileset of our adder TAC. Therefore, according to Lemma 4, we have that $t_1 = \Theta(\sqrt{n} + \log \sqrt{n}) = \Theta(\sqrt{n})$. The same analysis applies to get $t_2 = \Theta(\sqrt{n})$. The value t_3 is simply the value of the sum of $4\sqrt{n} + 2$ exponentially distributed random variables and thus has expected value $\Theta(\sqrt{n})$. Finally, $t_4 = \Theta(\sqrt{n})$ by the same analysis given for t_1 and t_2. Therefore, by Lemma 5, our TAC has worst case $O(\sqrt{n})$ continuous run time.

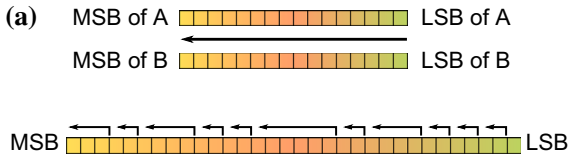

Fig. 8 *Arrows* represent carry origination and propagation direction. **a** This schematic represents the previously described $O(n)$ worst case addition for addends A and B Brun (2007). The least significant and most significant bits of A and B are denoted by LSB and MSB, respectively. **b** The average case $O(\log n)$ construction described in this paper is shown here. Addends A and B populate the linear assembly with bit A_i immediately adjacent to B_i. Carry propagation is done in parallel along the length of the assembly

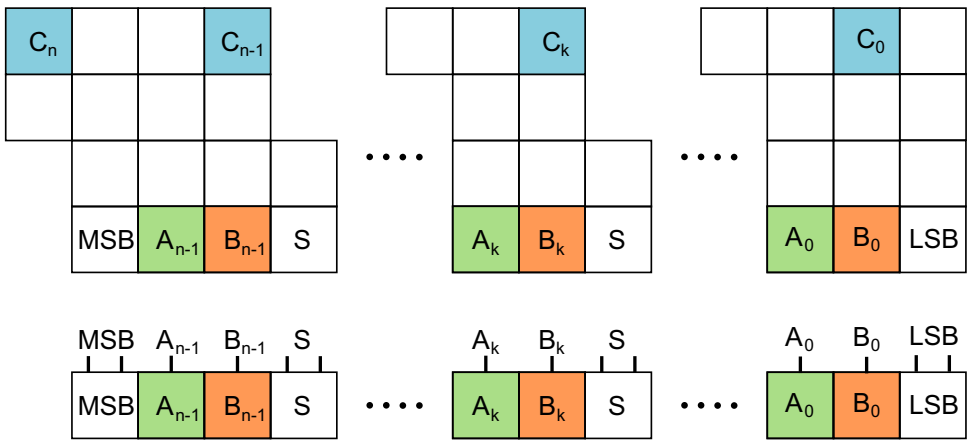

Fig. 9 *Top*: output template displaying addition result C for $O(\log n)$ average case addition construction. *Bottom*: input template composed of n blocks of three tiles each, representing n-bit addends A and B

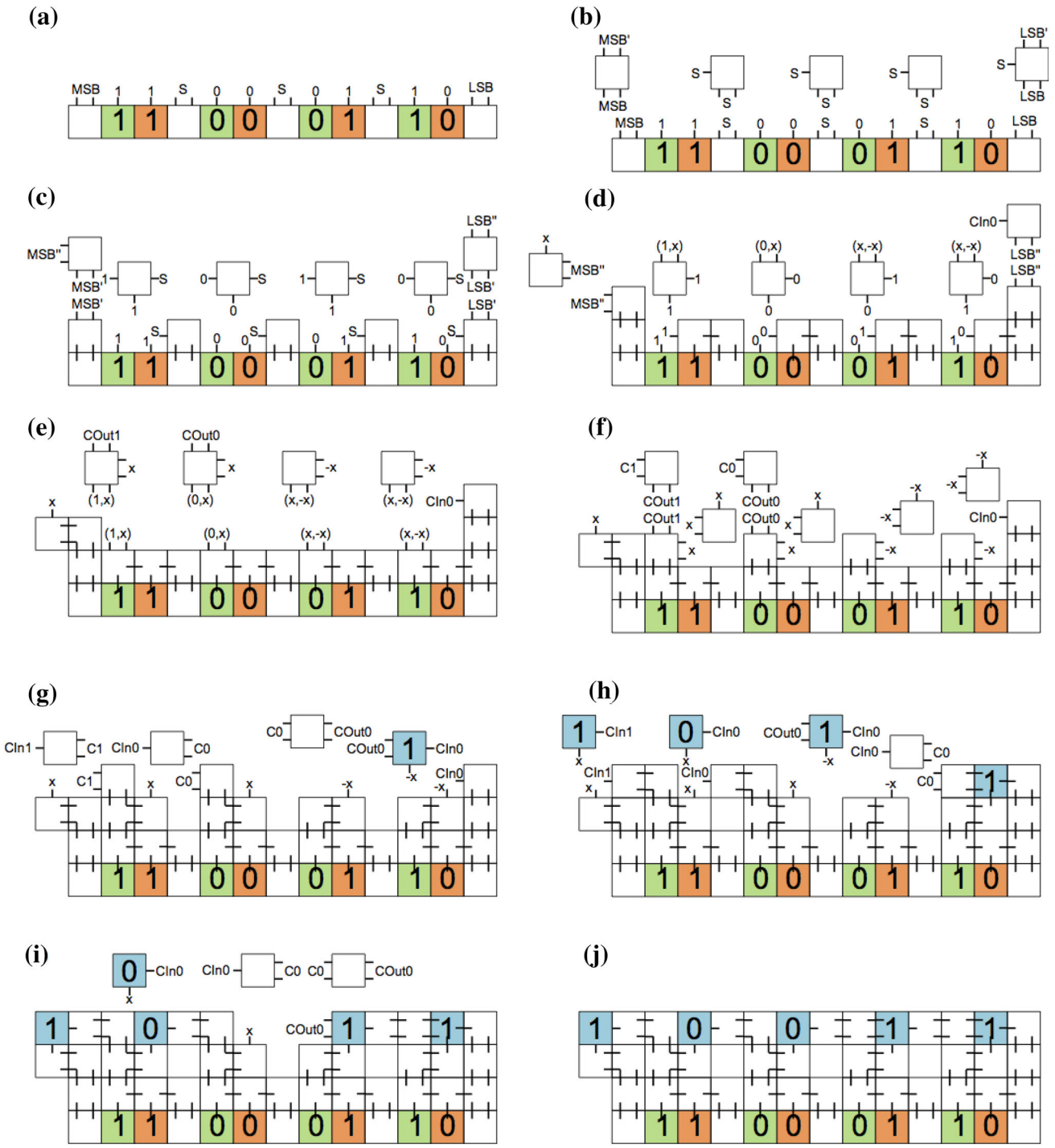

Fig. 10 Example tile assembly system to compute the sum of 1001 and 1010 using the adder TAC presented in Sect. 4. **a** The template with the two 4-bit binary numbers to be summed. **b** The first step is the parallel binding of tiles to the *S*, *LSB*, and *MSB* tiles. **c** Tiles bind cooperatively to west-face *S* glues and glues representing bits of *B*. The purpose of this step is to propagate bit B_i closer to A_i so that in **d** a tile may bind cooperatively, processing information from both A_i and B_i. **e** Note that addend-pairs consisting of either both 1s or both 0s have a tile with a north face glue consisting of either (1,x) or (0,x) bound to the A_i tile. This glue represents a carry out of either 1 or 0 from the addend-pair. In **e**–**g** the carry outs are propagated westward via tile additions for those addend-pairs where the carry out can be determined. Otherwise, spacer tiles bind. **h** Tiles representing bits of *C* (the sum of *A* and *B*) begin to bind where a carry in is known. **i**, **j** As carry bits propagate through sequences of (0,1) or (1,0) addend pairs, the final sum is computed

4 Addition in average case logarithmic time

We construct an adder TAC that resembles an electronic carry-skip adder in that the carry-out bit for addend pairs where each addend in the pair has the same bit value is generated in a constant number of steps and immediately propagated. When each addend in a pair of addends does not have the same bit value, a carry-out cannot be deduced until the value of the carry-in to the pair of addends is known. When such addends combinations occur in a contiguous sequence, the carry must ripple through the sequence from right-to-left, one step at a time as each position is evaluated. Within these worst-case sequences, our construction resembles an electronic ripple-carry adder. We show that using this approach it is possible to construct an n-bit adder TAC that can perform addition with an average runtime of $O(\log n)$ and a worst-case runtime of $O(n)$ (Fig. 8).

Lemma 6 *Consider a non-negative integer N generated uniformly at random from the set $\{0, 1, \ldots, 2^n - 1\}$. The expected length of the longest substring of contiguous ones in the binary expansion of N is $O(\log n)$.*

For a detailed discussion of Lemma 6 please see Schilling (1990).

Theorem 5 *For any positive integer n, there exists an n-bit adder TAC (tile assembly computer) that has worst case run time $O(n)$ and an average case run time of $O(\log n)$.*

The proof of Theorem 5 follows from the construction of the adder in Sect. 4.1. Additional detail, including detailed proofs of the time complexity in both timing models, can be found in Keenan et al. (2013).

4.1 Construction

We summarize the mechanism of addition presented here in an abbreviated example. A more detailed example may be found in Keenan et al. (2013), along with the complete tile set.

Input Template The input template, or seed, for the construction of an adder with an $O(\log n)$ average case is shown in Fig. 9. This input template is composed of n blocks, each containing three tiles. Within a block, the easternmost tile is the S labeled tile followed by two tiles representing A_k and B_k, the kth bits of A and B respectively. Of these n blocks, the easternmost and westernmost blocks of the template assembly are unique. Instead of an S tile, the block furthest east has an *LSB*-labeled tile which accompanies the tiles representing the least significant bits of A and B, A_0 and B_0. The westernmost block of the template assembly contains a block labeled *MSB* instead of the S block and accompanies the most significant bits of A and B, A_{n-1} and B_{n-1}.

Computing carry out bits For clarity, we demonstrate the mechanism of this adder TAC through an example by selecting two 4-bit binary numbers A and B such that the addends A_i and B_i encompass every possible addend combination. The input template for such an addition is shown in Fig. 10a where orange tiles represent bits of A and green tiles represent bits of B. Each block begins the computation in parallel at each S tile. After six parallel steps (Fig. 10b–g), all carry out bits, represented by glues $C0$ and $C1$, are determined for addend-pairs where both A_i and B_i are either both 0 or both 1. For addend pairs A_i and B_i where one addend is 0 and one addend is 1, the carry out bit cannot be deduced until a carry out bit has been produced by the previous addend pair, A_{i-1} and B_{i-1}. By step seven, a carry bit has been presented to all addend pairs that are flanked on the east by an addend pair comprised of either both 0s or both 1s, or that are flanked on the east by the *LSB* start tile, since the carry in to this site is always 0 (Fig. 10h). For those addend pairs flanked on the east by a contiguous sequence of size j pairs consisting of one 1 and one 0, $2j$ parallel attachment steps must occur before a carry bit is presented to the pair.

5 Towards faster addition

Our next result combines the approaches described in Sects. 4 and 3 in order to achieve both $O(\log n)$ average case addition and $O(\sqrt{n})$ worst case addition with the same algorithm (Fig. 11a). This construction resembles the construction described in Sect. 3 in that the numbers to be

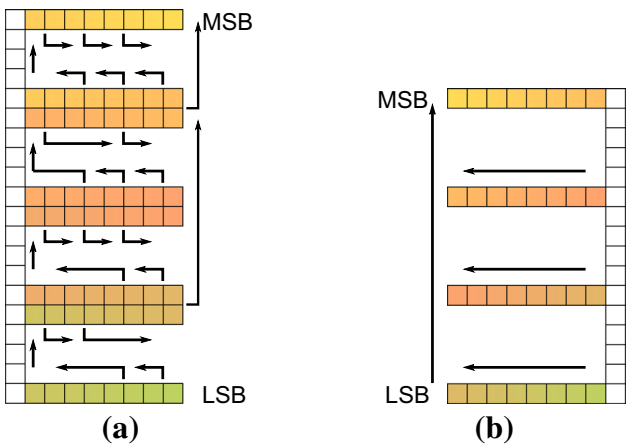

Fig. 11 *Arrows* represent carry origination and direction of propagation for **a** $O(\log n)$ average case, $O(\sqrt{n})$ worst case combined construction and **b** $O(\sqrt{n})$ worst case construction

added are divided into sections and values are computed for both possible carry-in bit values. Additionally, the construction described here lowers the average case run time by utilizing the carry-skip mechanism described in Sect. 4 within each section and between sections.

Theorem 6 *There exists a 2-dimensional n-bit adder TAC with an average run-time of* $O(\log n)$ *and a worst case run-time of* $O(\sqrt{n})$.

Given this 2D construction, it is possible to stack multiple linked instances of this construction into the third dimension to achieve an analogous optimal result for 3D addition.

Theorem 7 *There exists a 3-dimensional n-bit adder TAC with an average run-time of* $O(\log n)$ *and a worst case run-time of* $O(\sqrt[3]{n})$.

For a detailed presentation of these results please see Keenan et al. (2013).

Acknowledgments We would like to thank Ho-Lin Chen and Damien Woods for helpful discussions regarding Lemma 4 and Florent Becker for discussions regarding timing models in self-assembly. We would also like to thank Matt Patitz for helpful discussions of tile assembly simulation. The authors were supported in part by National Science Foundation Grant CCF-1117672.

References

Abel Z, Benbernou N, Damian M, Demaine E, Demaine M, Flatland R, Kominers S, Schweller R (2010) Shape replication through self-assembly and RNase enzymes. In: SODA 2010: proceedings of the twenty-first annual ACM-SIAM symposium on discrete algorithms (Austin, Texas), Society for Industrial and Applied Mathematics

Adleman L, Cheng Q, Goel A, Huang M-D (2001) Running time and program size for self-assembled squares. In: Proceedings of the thirty-third annual ACM symposium on theory of computing (New York, NY, USA), ACM, pp 740–748

Adleman LM, Cheng Q, Goel A, Huang M-DA, Kempe D, de Espanés PM, Rothemund Paul WK (2002) Combinatorial optimization problems in self-assembly. In: Proceedings of the thirty-fourth annual ACM symposium on theory of computing, pp 23–32

Becker F, Rapaport I, Rémila E (2006) Self-assembling classes of shapes with a minimum number of tiles, and in optimal time. Foundations of Software Technology and Theoretical Computer Science (FSTTCS), pp 45–56

Brun Y (2007) Arithmetic computation in the tile assembly model: addition and multiplication. Theor Comput Sci 378:17–31

Bryans N, Chiniforooshan E, Doty D, Kari L, Seki S (2011) The power of nondeterminism in self-assembly. In: SODA 2011: proceedings of the 22nd annual ACM-SIAM symposium on discrete algorithms, SIAM, pp 590–602

Chandran H, Gopalkrishnan N, Reif JH (2009) The tile complexity of linear assemblies. In: 36th international colloquium on automata, languages and programming, vol 5555

Cheng Q, Aggarwal G, Goldwasser MH, Kao M-Y, Schweller RT, de Espanés PM (2005) Complexities for generalized models of self-assembly. SIAM J Comput 34:1493–1515

Cheng Q, Goel A, de Espanés PM (2004) Optimal self-assembly of counters at temperature two. In: Proceedings of the first conference on foundations of nanoscience: self-assembled architectures and devices

Cook M, Fu Y, Schweller RT (2011) Temperature 1 self-assembly: deterministic assembly in 3d and probabilistic assembly in 2d. In: Dana R (ed) Proceedings of the twenty-second annual ACM-SIAM symposium on discrete algorithms, SODA 2011. SIAM, pp 570–589

Demaine E, Patitz M, Rogers T, Schweller R, Summers S, Woods D (2013) The two-handed tile assembly model is not intrinsically universal. In: Proceedings of the 40th international colloquium on automata, languages and programming (ICALP 2013)

Demaine ED, Demaine ML, Fekete SP, Patitz MJ, Schweller RT, Winslow A, Woods D (2014) One tile to rule them all: simulating any tile assembly system with a single universal tile. In: ICALP 2014: proceedings of the 41st international colloquium on automata, languages and programming

Doty D (2010) Randomized self-assembly for exact shapes. SIAM J Comput 39(8):3521–3552

Doty D, Patitz MJ, Reishus D, Schweller RT, Summers SM (2010) Strong fault-tolerance for self-assembly with fuzzy temperature. In: Proceedings of the 51st annual IEEE symposium on foundations of computer science (FOCS 2010), pp 417–426

Doty D, Lutz JH, Patitz MJ, Schweller R, Summers SM, Woods D (2012) The tile assembly model is intrinsically universal. In: FOCS 2012: proceedings of the 53rd IEEE conference on foundations of computer science

Fu B, Patitz MJ, Schweller R, Sheline R (2012) Self-assembly with geometric tiles. In: ICALP 2012: proceedings of the 39th international colloquium on automata, languages and programming (Warwick, UK)

Kao M-Y, Schweller RT (2008) Randomized self-assembly for approximate shapes. In: International colloqium on automata, languages, and programming, lecture notes in computer science, vol 5125. Springer, pp 370–384

Keenan A, Schweller RT, Sherman M, Zhong X (2013) Fast arithmetic in algorithmic self-assembly. CoRR arXiv:1303.2416

Mao C, LaBean TH, Relf JH, Seeman NC (2000) Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. Nature 407(6803):493–496

Meunier P-E, Patitz MJ, Summers SM, Theyssier G, Winslow A, Woods D (2014) Intrinsic universality in tile self-assembly requires cooperation. In: Proceedings of the twenty-fifth annual ACM-SIAM symposium on discrete algorithms

Schilling M (1990) The longest run of heads. Coll Math J 21(3):196–207

Schweller R, Sherman M (2013) Fuel efficient computation in passive self-assembly. In: SODA 2013: proceedings of the 24th annual ACM-SIAM symposium on discrete algorithms, SIAM, pp 1513–1525

Winfree E (1998) Algorithmic self-assembly of DNA. Ph.D. thesis, California Institute of Technology, June 1998

Woods D, Chen H-L, Goodfriend S, Dabby N, Winfree E, Yin P (2012) Efficient active self-assembly of shapes. Manuscript