FAST SIMULATION OF LARGE-SCALE GROWTH MODELS

TOBIAS FRIEDRICH AND LIONEL LEVINE

Abstract. We give an algorithm that computes the final state of certain growth models without computing all intermediate states. Our technique is based on a "least action principle" which characterizes the odometer function of the growth process. Starting from an educated guess for the odometer, we successively correct under- and overestimates and provably arrive at the correct final state. The degree of speedup depends on the accuracy of the initial guess.

Determining the size of the boundary fluctuations in internal diffusion-limited aggregation is a long-standing open problem in statistical physics. As an application of our method, we calculate the size of fluctuations over two orders of magnitude beyond previous simulations. Our data strongly support the conjecture that the fluctuations are logarithmic in the radius.

1. Introduction

In this paper we study the *abelian stack model*, a type of growth process on graphs. Special cases include *internal diffusion limited aggregation* (IDLA) and *rotor-router aggregation*. We describe a method for computing the final state of the process, given an initial guess. The more accurate the guess, the faster the computation.

IDLA. Starting with N particles at the origin of the two-dimensional square grid $\mathbb{Z}^2$, each particle in turn performs a simple random walk until reaching an unoccupied site. Introduced by Meakin and Deutch [26] and independently by Diaconis and Fulton [9], IDLA models physical phenomena such as solid melting around a heat source, electrochemical polishing, and fluid flow in a Hele-Shaw cell. Lawler, Bramson, and Griffeath [21] showed that as $N \to \infty$, the asymptotic shape of the resulting cluster of N occupied sites is a disk (and in higher dimensions, a Euclidean ball).

The boundary of an IDLA cluster is a natural model of a rough propagating front (Figure 1, left). From this perspective, the most basic question one could ask is, what is the scale of the fluctuations around the limiting circular shape? The answer is believed to be logarithmic in the radius r, but the current best rigorous bounds scale as $r^{1/3}$ times a logarithmic factor [20]. Determining the size of these boundary fluctuations is a long-standing open problem in statistical physics.

Rotor-router aggregation. James Propp [18] proposed the following way of derandomizing IDLA. At each lattice site in $\mathbb{Z}^2$ is a *rotor* that can point North, East, South or West. Instead of stepping in a random direction, a particle rotates the rotor at its current location counterclockwise, and then steps in the direction of this rotor. Each

Date: June 4, 2010.

2010 *Mathematics Subject Classification.* 82C24, 05C81, 05C85.

Key words and phrases. Cycle popping, internal diffusion limited aggregation, least action principle, low discrepancy random stack, odometer function, potential kernel, rotor-router model.

The second author was partly supported by a National Science Foundation Postdoctoral Fellowship.

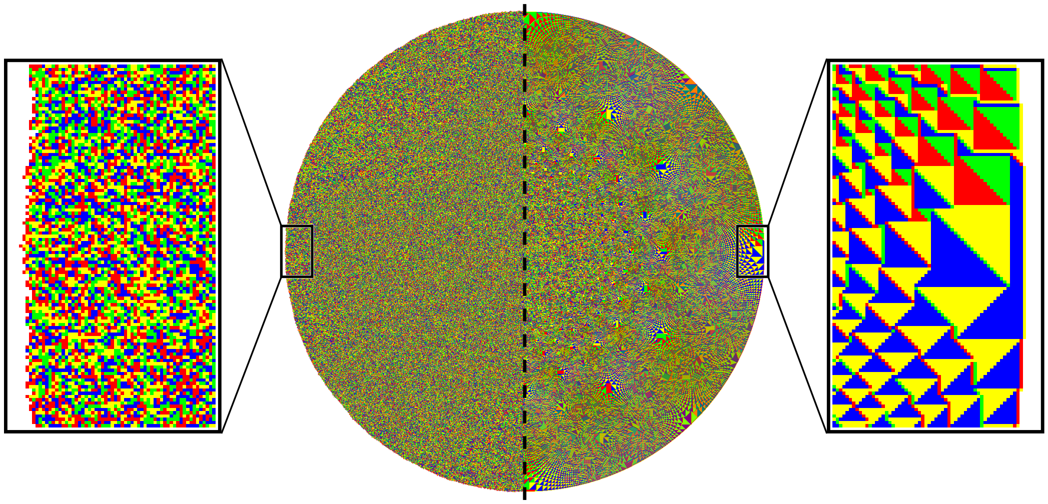


Figure 1. IDLA cluster (left) and rotor-router aggregation with counterclockwise rotor sequence (right) of $N = 10^6$ chips. Half of each aggregate is shown. Each site is colored according to the final direction of the rotor on top of its stack (yellow=W, red=S, blue=E, green=N). Note that the boundary fluctuations of the rotor-router aggregation are much smoother than for IDLA. Larger rotor-router aggregates of size up to $N = 10^{10}$ can be found on [1].

of N particles starting at the origin walks in this manner until reaching an unoccupied site. Given the initial configuration of the rotors (which can be taken, for example, all North), the resulting growth process is entirely deterministic. Regardless of the initial rotors, the asymptotic shape is a disk (and in higher dimensions, a Euclidean ball) and the inner fluctuations are proved to be $O(\log N)$ [23]. The true fluctuations appear to grow even more slowly, and may even be bounded independent of N.

Rotor-router aggregation is remarkable in that it generates a nearly perfect disk in the square lattice without any reference to the Euclidean norm $(x^2 + y^2)^{1/2}$. Perhaps even more remarkable are the patterns formed by the final directions of the rotors (Figure 1, right).

Low-discrepancy random stack. To better understand whether it is the regularity or the determinism which makes rotor-router aggregation so round, we follow a suggestion of James Propp and simulate a third model, *low-discrepancy random stack*, which combines the randomness of IDLA and the regularity of the rotor-router model.

Computing the odometer function. The central tool in our analysis of all three models is the *odometer function*, which measures the number of chips emitted from each site. The odometer function determines the shape of the final occupied cluster via a nonlinear operator that we call the *stack Laplacian*. Our main technical contribution is that even for highly non-deterministic models such as IDLA, one can achieve *fast exact calculation via intermediate approximation*. Approximating our three growth processes by an idealized model called the divisible sandpile, we can use the known asymptotic expansion of the potential kernel of random walk on $\mathbb{Z}^2$ to obtain an educated guess of the odometer function. We present a method for carrying out subsequent local corrections to provably transform this guess into the exact odometer function, and hence compute the shape of the occupied cluster.

Applications. Using our new algorithm of exact calculation via intermediate approximation, we are able to generate a large IDLA cluster relatively quickly. By generating many independent clusters, we estimate the order of fluctuations from circularity over two orders of magnitude beyond previous simulations. For rotor-router aggregation we achieve three orders of magnitude beyond previous simulations. Our data support the findings of [27] that the order of the maximum fluctuation for IDLA is logarithmic in the radius r of the occupied cluster (equivalently, logarithmic in N).

A second motivation for our work was to generate more fine-scaled examples of the intricate patterns that form the final rotors on the tops of the stacks at the end of the rotor-router aggregation process (Figure 1, right). These patterns remain poorly understood even on a heuristic level. We have used our algorithm to generate a four-color 10-gigapixel image [1] of the final rotors for $N = 10^{10}$ chips. This file is so large that we had to use a Google maps overlay to allow the user to zoom and scroll through the image. Indeed, the degree of speedup in our method was so dramatic that memory, rather than time, became the limiting factor.

Related Work. Unlike random walk, in a rotor-router walk each vertex serves its neighbors in a fixed order. The deterministic walk nevertheless closely resembles a random walk in several respects [4–6, 10, 16]. The rotor-router mechanism also leads to improvements in algorithmic applications. Examples include external mergesort [3], broadcasting information in networks [11], and iterative load-balancing [13].

Abelian stacks (defined in the next section) are a way of indexing the steps of a walk to a particular location rather than a particular particle. This fruitful idea goes back at least to Diaconis and Fulton [9, §4]. Wilson [29] (see also [28]) used this stack-based view of random walk in his algorithm for sampling a random spanning tree of a directed graph. The final cycle-popping phase of our algorithm is directly inspired by Wilson's algorithm. Our serial algorithm for IDLA also draws on ideas from the parallel algorithm of Moore and Machta [27].

Abelian stacks are a special case of the *abelian distributed processors* defined by Dhar [7]. In this viewpoint, each vertex is a processor. The chips are called "messages." When a processor receives a message, it can change internal state and also send one or more messages to neighboring processors according to its current internal state. We believe that it might be possible to extend our method to other types of abelian distributed processors, such as the Bak-Tang-Wiesenfeld abelian sandpile model [2]. Indeed, the initial inspiration for our work was the "least action principle" for sandpiles described in [12].

Organization of the paper. After formally defining the abelian stack model in §2, we describe the mathematics underlying our algorithm in §3. The main result of §3 is Theorem 1, which uniquely characterizes the odometer function by a few simple properties. In §4 we describe the algorithm itself, and use Theorem 1 to prove its correctness. §5 discusses how to find a good approximation function to use as input to the algorithm. Finally, §6 describes our implementation and experimental results.

2. Formal Model

The underlying graph for the abelian stack model can be any finite or infinite directed graph G. We will assume that G is *strongly connected*: for any two vertices x and y there are directed paths from x to y and from y to x. At each vertex x of G

is an infinite stack of *rotors* $(\rho_n(x))_{n\geqslant 0}$. Each rotor $\rho_n(x)$ is an edge of G emanating from x. We say that rotor $\rho_0(x)$ is "on top" of the stack.

A finite number of indistinguishable chips are dispersed on the vertices of G according to some prescribed initial configuration. For each vertex x, the first chip to visit x is absorbed there and never moves again. Each subsequent chip arriving at x first shifts the stack at x downward, so that the new stack is $\rho'_n(x) = \rho_{n+1}(x)$. After shifting the stack, the chip moves to the vertex y pointed to by the rotor $\rho'_0(x)$ now on top. We call this two-step procedure (shifting the stack and moving a chip) *firing* the site x. The effect of this rule is that the n-th time a chip is emitted from x, it travels along the edge $\rho_n(x)$.

We will generally assume that the stacks are *infinitive*: for each edge $e = (x, y)$, infinitely many rotors $\rho_n(x)$ are equal to e. If G is infinite, or if the total number of chips is at most the number of vertices, then this condition ensures that firing eventually stops, and all chips are absorbed.

We are interested in the set of *occupied sites*, that is, sites that absorb a chip. The *abelian property* [9, Theorem 4.1] asserts that this set does not depend on the order in which vertices are fired. This property plays a key role in our method; we discuss it further in §3.

If the rotors $\rho_n(x)$ are independent and identically distributed random edges emanating from x, then we obtain IDLA. The special case of IDLA in which all chips start at a fixed vertex o is more commonly described as follows. Let $A_1 = \{o\}$, and for $N \geqslant 2$ define a random set A_N of N vertices of G according to the recursive rule

$$A_{N+1} = A_N \cup \{x_N\} \tag{1}$$

where x_N is the endpoint of a random walk started at o and stopped when it first visits a site not in A_N. These random walks describe one particular sequence in which the vertices can be fired, for the initial configuration of N chips at o. The first chip is absorbed at o, and subsequent chips are absorbed in turn at sites $x_1, \ldots, x_{N-1}$. When firing stops, the set of occupied sites is A_N.

A second interesting case is deterministic: the sequence $\rho_n(x)$ is periodic in n, for every vertex x. For example, on $\mathbb{Z}^2$, we could take the top rotor in each stack to point to the northward neighbor, the next to the eastward neighbor, and so on. This choice yields the model of rotor-router aggregation defined by Propp [18] and analyzed in [22, 23]. It is described by the growth rule (1), where x_N is the endpoint of a rotor-router walk started at the origin and stopped on first exiting A_N.

3. Least Action Principle

Let $G = (V, E)$ be a locally finite directed graph, which may have loops and multiple edges. Each edge $e \in E$ is oriented from its source vertex $\mathtt{s}(e)$ to its target vertex $\mathtt{t}(e)$. We assume that G is strongly connected.

A *rotor configuration* on G is a function

$$r \colon V \to E$$

such that $\mathtt{s}(r(v)) = v$ for all $v \in V$. A *chip configuration* on G is a function

$$\sigma \colon V \to \mathbb{Z}$$

with finite support. Note we do not require $\sigma \geqslant 0$. If $\sigma(x) = m > 0$, we say there are m chips at vertex x; if $\sigma(x) = -m < 0$, we say there is a hole of depth m at vertex x.

For an edge e and a nonnegative integer n, let

$$R_\rho(e,n) = \#\{1 \leqslant k \leqslant n \mid \rho_k(\mathtt{s}(e)) = e\} \tag{2}$$

be the number of times e occurs among the first n rotors in the stack at the vertex $\mathtt{s}(e)$ (excluding the top rotor $\rho_0(\mathtt{s}(e))$). When no ambiguity would result, we drop the subscript ρ.

Write $\mathbb{N}$ for the set of nonnegative integers. Given a function $u\colon V \to \mathbb{N}$, we would like to describe the net effect on chips resulting from firing each vertex $x \in V$ a total of $u(x)$ times. In the course of these firings, each vertex x emits $u(x)$ chips, and receives $R_\rho(e, u(\mathtt{s}(e)))$ chips along each incoming edge e with $\mathtt{t}(e) = x$. This motivates the following definition.

Definition. The *stack Laplacian* of a function $u\colon V \to \mathbb{N}$ is the function

$$\Delta_\rho u\colon V \to \mathbb{Z}$$

given by

$$\Delta_\rho u(x) = \sum_{\mathtt{t}(e)=x} R_\rho(e, u(\mathtt{s}(e))) - u(x). \tag{3}$$

The sum is over all edges e with target vertex $\mathtt{t}(e) = x$. We use the notation Δ_ρ to emphasize the dependence (via R_ρ) on the rotor stacks $\rho_k(x)$.

Given an initial chip configuration σ_0, the configuration σ resulting from performing $u(x)$ firings at each site $x \in V$ is given by

$$\sigma = \sigma_0 + \Delta_\rho u. \tag{4}$$

The rotor configuration on the tops of the stacks after these firings is also easy to describe. We denote this configuration by $\mathtt{Top}_\rho(u)$, and it is given by

$$\mathtt{Top}_\rho(u)(x) = \rho_{u(x)}(x).$$

We also write $E^u\rho$ for the collection of shifted stacks:

$$(E^u\rho)_k(x) = \rho_{k+u(x)}(x).$$

The stack Laplacian is not a linear operator, but it satisfies the relation

$$\Delta_\rho(u+v) = \Delta_\rho u + \Delta_{E^u\rho} v. \tag{5}$$

Vertices $x_1, \ldots, x_m$ form a *legal firing sequence* for σ_0 if

$$\sigma_j(x_{j+1}) > 1, \qquad j = 0, \ldots, m-1$$

where

$$\sigma_j = \sigma_0 + \Delta_\rho u_j$$

and

$$u_j(x) = \#\{i \leqslant j \colon x_i = x\}.$$

In words, the condition $\sigma_j(x_{j+1}) > 1$ says that after firing $x_1, \ldots, x_j$, the vertex x_{j+1} has at least two chips. We require at least two because in our growth model, the first chip to visit each vertex gets absorbed.

The firing sequence is *complete* if no further legal firings are possible; that is, $\sigma_m(x) \leqslant 1$ for all $x \in V$. If $x_1, \ldots, x_m$ is a complete legal firing sequence for the chip configuration σ_0, then we call the function $u := u_m$ the *odometer* of σ_0.

Abelian Property [9, Theorem 4.1] Given an initial configuration σ_0 and stacks ρ, every complete legal firing sequence for σ_0 has the same odometer function u.

It follows that the final chip configuration $\sigma_m = \sigma_0 + \Delta_\rho u$ and the final rotor configuration $\texttt{Top}(u)$ do not depend on the choice of complete legal firing sequence.

Given a chip configuration σ_0 and rotor stacks $\rho_k(x)$, our goal is to compute the final chip configuration σ_m without performing individual firings one at a time. By equation (4), it suffices to compute the odometer function u of σ_0. (In practice, it is usually easy to compute $\Delta_\rho u$ given u, an issue we address in §4.)

Our approach will be to start from an approximation of u and correct errors. In order to know when our algorithm is finished, the key mathematical point is to find a list of properties of u that characterize it uniquely. Our main result in this section, Theorem 1, gives such a list. As we now explain, the hypotheses of this theorem can all be guessed from certain necessary features of the final chip configuration σ_m and the final rotor configuration $\texttt{Top}_\rho(u)$. What is perhaps surprising is that these few properties suffice to characterize u.

Let $x_1, \ldots, x_m$ be a complete legal firing sequence for the chip configuration σ_0. We start with the observation that since no further legal firings are possible,

- $\sigma_m(x) \leqslant 1$ for all $x \in V$.

Next, let

$$A = \{x \in V : u(x) > 0\}$$

be the set of sites that fire. Since each site that fires must first absorb a chip, we have

- $\sigma_m(x) = 1$ for all $x \in A$.

Finally, observe that for any vertex $x \in A$, the rotor $r_x = \texttt{Top}_\rho(u)(x)$ at the top of the stack at x is the edge traversed by the last chip fired from x. In particular, for any finite subset A' of A, the top rotor at the vertex of A' that fired last points to a vertex not in A'.

- For any finite set $A' \subset A$, there exists $x \in A'$ with $\texttt{t}(r_x) \notin A'$.

We can state this last condition more succinctly by saying that the rotor configuration $r = \texttt{Top}(u)$ is *acyclic* on A; that is, the spanning subgraph $(V, r(A))$ has no directed cycles.

Theorem 1. *Let G be a finite or infinite directed graph, ρ a collection of rotor stacks on G, and σ_0 a chip configuration on G. Let u be the corresponding odometer function. Fix $u_* : V \to \mathbb{N}$, and let $A_* = \{x \in V : u_*(x) > 0\}$. Let $\sigma_* = \sigma_0 + \Delta_\rho u_*$, and suppose that*

- $\sigma_* \leqslant 1$;
- A_* *is finite;*
- $\sigma_*(x) = 1$ *for all* $x \in A_*$; *and*
- $\texttt{Top}_\rho(u_*)$ *is acyclic on* A_*.

Then $u_* = u$.

Remark. To ensure that u is finite (i.e., that there exists a finite complete legal firing sequence) it is common to place some minimal assumptions on ρ and σ_0. For example, if G is infinite and strongly connected, then it suffices to assume that the stacks ρ are infinitive. Theorem 1 does not explicitly make any assumptions of this kind; rather, if a function u_* exists satisfying the conditions listed, then u must be finite (and equal to u_*).

We break the proof into two inequalities. The first inequality can be seen as an analogue for the abelian stack model of the least action principle for sandpiles [12, Lemma 2.3].

Lemma 2. *(Least Action Principle) If $\sigma_* \leqslant 1$, then $u_* \geqslant u$.*

Proof. Perform legal firings in any order, without allowing any site x to fire more than $u_*(x)$ times, until no such firing is possible. Write $u'(x)$ for the number of times x fires during this procedure. We will show that $u' = u$.

Write $\sigma' = \sigma_0 + \Delta_\rho u'$. If $\sigma' \leqslant 1$, then $u' = u$ by the abelian property. Otherwise, choose y such that $\sigma'(y) > 1$. We must have $u'(y) = u_*(y)$, else it would have been possible to add another legal firing to u'. Therefore, if we now perform further firings $u_* - u'$, then since y does not fire, the number of chips at y cannot decrease. Hence

$$\sigma_*(y) \geqslant \sigma'(y) > 1$$

contradicting the assumption that $\sigma_* \leqslant 1$. □

Lemma 3. *Suppose that*

- *A_* is finite;*
- *$\sigma_*(x) \geqslant 1$ for all $x \in A_*$; and*
- *$\mathtt{Top}(u_*)$ is acyclic on A_*.*

Then $u_* \leqslant u$.

Proof. Let

$$m(x) = \min(u(x), u_*(x))$$

$$\psi = \sigma_0 + \Delta_\rho m$$

$$\sigma = \sigma_0 + \Delta_\rho u.$$

Then letting $\tilde{\rho} = E^m \rho$, we have from (5)

$$\begin{aligned}\sigma &= \sigma_0 + \Delta_\rho m + \Delta_{\tilde{\rho}}(u - m) \\ &= \psi + \Delta_{\tilde{\rho}}(u - m).\end{aligned}$$

Likewise, $\sigma_* = \psi + \Delta_{\tilde{\rho}}(u_* - m)$. Let

$$A = \{x \in V \mid u_*(x) > u(x)\}.$$

Since $u \geqslant 0$, we have $A \subset A_*$, hence A is finite. We must show that A is empty.

We have $\sigma_*(x) \geqslant 1$ for all $x \in A$ by hypothesis, while $\sigma(x) \leqslant 1$ by the definition of the odometer function u. So

$$\begin{aligned}0 &\leqslant \sum_{x \in A} (\sigma_*(x) - \sigma(x)) \\ &\leqslant \sum_{x \in A} \left(\Delta_{\tilde{\rho}}(u_* - m)(x) - \Delta_{\tilde{\rho}}(u - m)(x)\right).\end{aligned}$$

For $x \in A$ we have $u(x) = m(x)$, so $\Delta_{\tilde\rho}(u - m)(x) \geqslant 0$. Hence

$$\begin{aligned} 0 &\leqslant \sum_{x\in A} \Delta_{\tilde\rho}(u_* - m) \\ &= \sum_{x\in A}\Big(-(u_*(x) - m(x)) + \sum_{\mathtt{t}(e)=x} \#\{m(\mathtt{s}(e)) < k \leqslant u_*(\mathtt{s}(e)) \mid \rho_k(\mathtt{s}(e)) = e\}\Big). \end{aligned}$$

The terms of the inner sum corresponding to edges e such that $\mathtt{s}(e) \notin A$ vanish, since in that case $m(\mathtt{s}(e)) = u_*(\mathtt{s}(e))$. Hence

$$\begin{aligned} \sum_{x\in A}(u_*(x) - m(x)) &\leqslant \sum_{x\in A} \sum_{\substack{\mathtt{t}(e)=x \\ \mathtt{s}(e)\in A}} \#\{m(\mathtt{s}(e)) < k \leqslant u_*(\mathtt{s}(e)) \mid \rho_k(\mathtt{s}(e)) = e\} \\ &= \sum_{x\in A}\sum_{y\in A} \#\{m(y) < k \leqslant u_*(y) \mid \mathtt{t}(\rho_k(y)) = x\} \\ &= \sum_{y\in A} \#\{m(y) < k \leqslant u_*(y) \mid \mathtt{t}(\rho_k(y)) \in A\}. \end{aligned} \tag{6}$$

Now suppose for a contradiction that A is nonempty. Since $\mathtt{Top}(u_*)$ is acyclic on A, there exists a site $z \in A$ with $\mathtt{t}(\rho_k(z)) \notin A$, where $k = u_*(z)$. Therefore the sum on the right side of (6) is strictly less than $\sum_{y\in A}(u_*(y) - m(y))$, which gives the desired contradiction. $\square$

We conclude this section by observing a few consequences of Theorem 1. While our algorithm does not directly use the results below, we anticipate that they may be useful in further attempts to understand IDLA and rotor-router aggregation.

The stacks ρ and initial configuration σ_0 determine an odometer function $u = u(\rho, \sigma_0)$, which is the unique function satisfying the hypotheses of Theorem 1. In particular, given σ_0, the function u is completely characterized by properties of the chip configuration $\sigma_0 + \Delta_\rho u$ and the rotor configuration $\mathtt{Top}_\rho u$. Since permuting the stack elements $\rho_1(x), \ldots, \rho_{u(\rho,\sigma_0)(x)-1}$ does not change $\Delta_\rho u$ or $\mathtt{Top}_\rho u$, we obtain the following result.

Corollary 4. *(Exchangeability) Let σ be a chip configuration on G. Let $(\rho_k(x))_{x\in V, k\in\mathbb{Z}}$ and $(\rho'_k(x))_{x\in V,k\in\mathbb{Z}}$ be two collections of rotor stacks, with the property that for each vertex $x \in V$, the rotors*

$$\rho'_1(x), \ldots, \rho'_{u(\rho,\sigma)(x)-1}(x)$$

are a permutation of

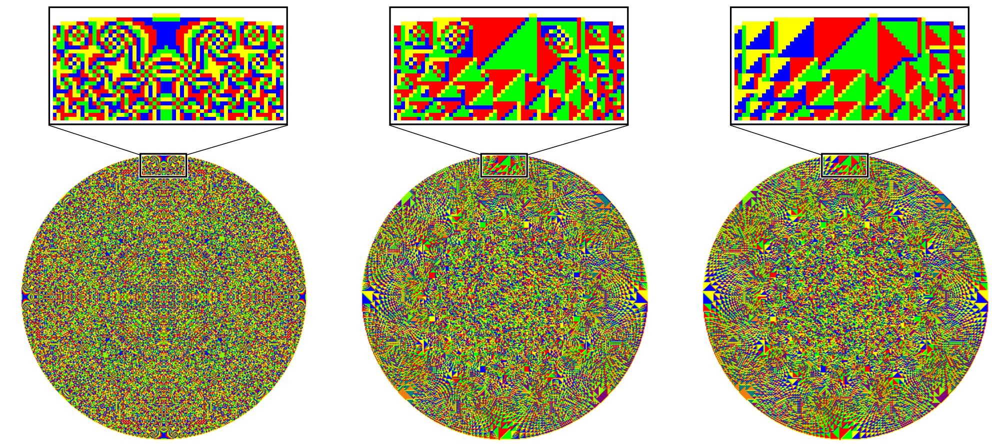
$$\rho_1(x), \ldots, \rho_{u(\rho,\sigma)(x)-1}(x).$$

Suppose moreover that

$$\rho_{u(\rho,\sigma)(x)}(x) = \rho'_{u(\rho,\sigma)(x)}(x).$$

Then $u(\rho', \sigma) = u(\rho, \sigma)$.

Edges $e_1, \ldots, e_m \in E$ form a *directed cycle* if $\mathtt{s}(e_{i+1}) = \mathtt{t}(e_i)$ for $i = 1, \ldots, m-1$ and $\mathtt{s}(e_1) = \mathtt{t}(e_m)$. The next result allows us to remove directed cycles of rotors from the stacks, without changing the final configuration.



(a) After odometer approximation (u_1) **(b)** After annihilation (u_2) **(c)** After cycle popping (u_3)

Figure 2. Classic rotor router aggregation of $N = 100,000$ chips with counterclockwise rotor sequence. The pictures show the direction of the rotors on top of the stacks after each step of the computation (yellow=`W`, red=`S`, blue=`E`, green=`N`).

Corollary 5. *(Cycle removal) Let σ be a chip configuration on G, and let $\mathcal{C} = \{e_1, \ldots, e_m\}$ be a directed cycle in G. Suppose that*

$$e_i = \rho_{k_i}(x_i), \qquad i = 1, \ldots, m$$

for vertices $x_1, \ldots, x_m$ and integers $k_1, \ldots, k_m$ satisfying $k_i \leqslant u(\rho, \sigma)(x_i) - 1$ for all $i = 1, \ldots, m$. Let ρ' be the rotor stacks obtained from ρ by removing the rotors $\rho_{k_i}(x_i)$ for all $i = 1, \ldots, m$. Then

$$u(\rho, \sigma) = u(\rho', \sigma) + \chi$$

where $\chi(x) = \#\{1 \leqslant i \leqslant m \mid x_i = x\}$. Moreover, the final configurations agree:

$$\sigma + \Delta_\rho[u(\rho, \sigma)] = \sigma + \Delta_{\rho'}[u(\rho', \sigma)].$$

Proof. Let $f = u(\rho, \sigma)$. The bound on k_i implies that $\mathtt{Top}_\rho f = \mathtt{Top}_{\rho'}(f - \chi)$. By Theorem 1, to complete the proof it suffices to check that $\Delta_\rho f = \Delta_{\rho'}(f - \chi)$. For any vertex x and edge e with $\mathtt{s}(e) = x$, we have

$$\begin{aligned} R_\rho(e, f(x)) &= \#\{1 \leqslant k \leqslant f(x) \mid \rho_k(x) = e\} \\ &= R_{\rho'}(e, f(x) - \chi(x)) + 1_{\{e \in \mathcal{C}\}}. \end{aligned}$$

Hence

$$\begin{aligned} \Delta_\rho f(x) &= -f(x) + \sum_{\mathtt{t}(e)=x} R_\rho(e, f(x)) \\ &= -f(x) + \sum_{\mathtt{t}(e)=x} R_{\rho'}(e, f(x) - \chi(x)) + \sum_{\mathtt{t}(e)=x} \mathbf{1}_{\{e \in \mathcal{C}\}}. \end{aligned}$$

Since $\mathcal{C}$ is a directed cycle, we have $\sum_{\mathtt{t}(e)=x} \mathbf{1}_{\{e\in\mathcal{C}\}} = \sum_{\mathtt{s}(e)=x} \mathbf{1}_{\{e\in\mathcal{C}\}} = \chi(x)$. So we obtain $\Delta_\rho f = \Delta_{\rho'}(f - \chi)$. $\square$

4. The Algorithm: From Approximation to Exact Calculation

In this section we describe how to compute the odometer function u exactly, given as input an approximation u_1. The running time depends on the accuracy of the approximation, but the correctness of the output does not. In the next section we explain how to find a good approximation u_1 for the example of N chips started at the origin in $\mathbb{Z}^2$.

We assume that G is strongly connected (finite or infinite), and that the initial configuration σ_0 satisfies $\sigma_0(x) \geqslant 0$ for all x, and $\sum_x \sigma_0(x) < \infty$. If G is finite, we assume that $\sum_x \sigma_0(x)$ is at most the number of vertices of G (otherwise, some chips would never get absorbed). The only assumption on the approximation u_1 is that it is nonnegative with finite support. Finally, we assume that the rotor stacks are infinitive, which ensures that the growth process terminates after finitely many firings: that is, $\sum_{x \in V} u(x) < \infty$.

For $x \in V$, write

$$d_{out}(x) = \#\{e \in E \mid \mathtt{s}(e) = x\}$$

$$d_{in}(x) = \#\{e \in E \mid \mathtt{t}(e) = x\}$$

for the out-degree and in-degree of x.

The odometer function u depends on the initial chip configuration σ_0 and on the rotor stacks $\rho_k(x)$. The latter are completely specified by the function $R(e,n)$ defined in §3. Note that for rotor-router aggregation, since the stacks are periodic, $R(e,n)$ has the simple explicit form

$$R(e,n) = \left\lfloor \frac{n + d_{out}(x) - j}{d_{out}(x)} \right\rfloor \tag{7}$$

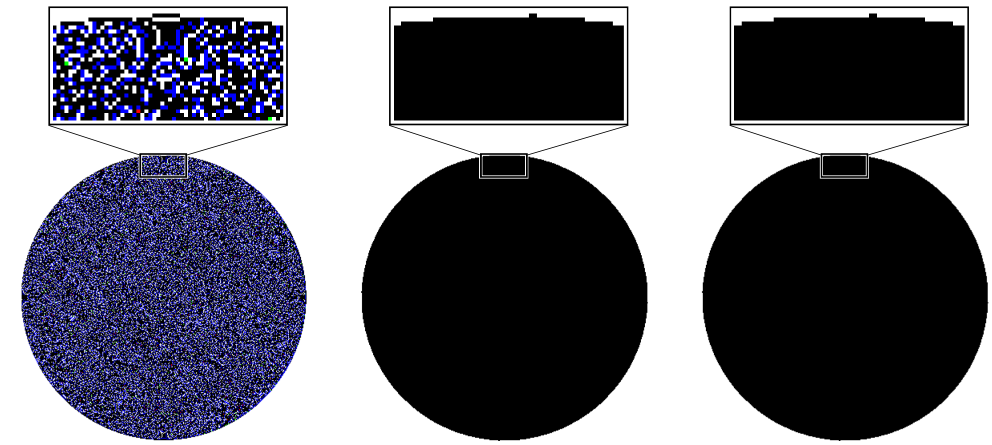
where j is the least positive integer such that $\rho_j(x) = e$. For IDLA, $R(e,n)$ is a random variable with the Binomial(n,p) distribution, where p is the transition probability associated to the edge e.

In this section we take $R(e,n)$ as known. From a computational standpoint, if the stacks are random, then determining $R(e,n)$ involves calls to a pseudorandom number generator. We address the issue of minimizing the number of such calls in §6.2.

Our algorithm consists of an approximation step followed by two error-correction steps: an annihilation step that corrects the chip locations, and a reverse cycle-popping step that corrects the rotors.

(1) **Approximation.** Perform firings according to the approximate odometer, by computing the chip configuration $\sigma_1 = \sigma_0 + \Delta_\rho u_1$. Using equation (3), this takes time $O(d_{in}(x) + 1)$ for each vertex x, for a total time of $O(\#E + \#V)$. This step is where the speedup occurs, because we are performing many firings at once: $\sum_x u_1(x)$ is typically much larger than $\#E + \#V$. Output σ_1.

(2) **Annihilation.** Start with $u_2 = u_1$ and $\sigma_2 = \sigma_1$. If $x \in V$ satisfies $\sigma_2(x) > 1$, then we call x a *hill*. If $\sigma_2(x) < 0$, or if $\sigma_2(x) = 0$ and $u_2(x) > 0$, then we call x a *hole*. For each $x \in \mathbb{Z}^2$,
   (a) If x is a hill, fire it by incrementing $u_2(x)$ by one and then moving one chip from x to $\mathtt{t}(\mathtt{Top}(u_2)(x))$.
   (b) If x is a hole, unfire it by moving one chip from $\mathtt{t}(\mathtt{Top}(u_2)(x))$ to x and then decrementing $u_2(x)$ by one.



(a) After odometer approximation (σ_1) **(b)** After annihilation (σ_2) **(c)** After cycle popping (σ_3)

Figure 3. Classic rotor router aggregation of $N = 100,000$ chips with counterclockwise rotor sequence. The pictures show the number of chips after each step of the algorithm. (Location x is colored red if $\sigma'(x) = -1$ white if $\sigma'(x) = 0$, black if $\sigma'(x) = 1$, blue if $\sigma'(x) = 2$, green if $\sigma'(x) = 3$.) Note that there are no locations with $\sigma'(x) < -1$ or $\sigma'(x) > 3$, and that no chips move during the final cycle-popping phase.

A hill can disappear in one of two ways: by reaching an unoccupied site on the boundary, or by reaching a hole and canceling it out. Moving each observed hill until it hits some hole is rather inefficient as these paths can be very long. We used the following multiscale approach to speed up this process. Let $L_1, L_2, \ldots$ be an exponentially growing sequence of integers. For each $i \geqslant 1$ do

- Substep i: fire each hill / unfire each hole until it either cancels out or reaches a site in $L_i\,\mathbb{Z} \times \mathbb{Z} \cup \mathbb{Z} \times L_i\mathbb{Z}$.

We used $L_1 = 1$ and $L_{i+1} = \lceil 1.9\, L_i \rceil$ for $i \geqslant 1$. When there are no more hills and holes, output u_2.

(3) **Reverse cycle-popping.** Start with $u_3 = u_2$ and

$$A_3 = \{x \in V \colon u_3(x) > 0\}.$$

If $\texttt{Top}(u_3)$ is not acyclic on A_3, then pick a cycle and unfire each of its vertices once. This may create additional cycles. Update A_3 (it may shrink, since u_3 has decreased) and repeat until $\texttt{Top}(u_3)$ is acyclic on A_3. Output u_3.

Next we argue that the algorithm terminates, and that its final output u_3 equals the odometer function u. Step 2 is simplest to analyze if we first fire all hills, and only after there are no more hills begin unfiring holes. (Experimentally, however, we find that the runtime is faster if we fire hills and unfire holes in tandem.)

At the beginning of step 2, all hills are contained in the set

$$S = \{x \in V \mid \sigma_1(x) > 0\}.$$

Since σ_0 and u_1 have finite support, $\sigma_1 = \sigma_0 + \Delta_\rho u_1$ has finite support, so S is finite. Since the total number of chips is conserved, we have

$$\sum_{x \in V} \sigma_1(x) = \sum_{x \in V} \sigma_0(x).$$

The right side is $\leqslant \#V$ by assumption. Therefore if $S = V$, we must have $\sigma_1(x) = 1$ for all $x \in V$; in this case there are no hills or holes, and we move on to step 3.

Suppose now that S is a proper subset of V. Let

$$h = \sum_{x \in S} (\sigma_1(x) - 1)$$

be the total height of the hills. Note that firing a hill cannot increase h. If a given vertex fires infinitely often, then since the rotor stacks are infinitive, each of its out-neighbors also fires infinitely often; since G is strongly connected, it would follow that every vertex fires infinitely often. Thus after firing finitely many hills, a chip must leave S. When this happens, h decreases. Thus after finitely many firings we reach $h = 0$ and there are no more hills.

Next we begin unfiring the holes. After all hills have been settled, we have $u_2(x) \geqslant 0$ for all $x \in V$. The sum $\sum_{x \in V} u_2(x)$ is finite, and each unfiring decreases it by one. To show that the unfiring step terminates, it suffices to show that for all $x \in V$ the unfiring of holes never causes $u_2(x)$ to become negative. Indeed, suppose that $u_2(x) = 0$ and $u_2(y) \geqslant 0$ for all neighbors y of x. Then the number of chips at x is $\sigma_0(x) + \Delta_\rho u_2(x) \geqslant 0$, so x is not a hole. Therefore the unfiring step terminates and its output u_2 is nonnegative.

After step 2 there are no hills or holes, i.e., $0 \leqslant \sigma_2(x) \leqslant 1$ for all x, and if $\sigma_2(x) = 0$ then $u_2(x) = 0$.

During step 3, we only unfire sites within A_3. Since $\sum_{x \in V} u_3(x)$ is finite and decreases with each unfiring, this step terminates and its output u_3 is nonnegative. When a cycle is unfired, each vertex in the cycle sends a chip to the previous vertex, so there is no net movement of chips: $\sigma_3 = \sigma_2$. In particular, there are no hills at the end of step 3. If $\sigma_3(x) = 0$, then $\sigma_2(x) = 0$; since there were no holes at the end of step 2, this means that $u_2(x) = 0$, and hence $u_3(x) = 0$. So there are still no holes at the end of step 3. By construction, $\texttt{Top}(u_3)$ is acyclic on A_3. Therefore all condtions of Theorem 1 are satisified, which shows that $u_3 = u$ as desired.

5. Approximating the Odometer Function

Next we describe how to find a good approximation to the odometer to use as input to the algorithm described in §4. Our main assumption will be that the rotor stacks are *balanced* in the sense that

$$R(e, n) \approx R(e', n)$$

for all $n \in \mathbb{N}$ and all edges e, e' with $\mathtt{s}(e) = \mathtt{s}(e')$. By definition, rotor-router aggregation obeys the strong balance condition

$$|R(e, n) - R(e', n)| \leqslant 1.$$

IDLA is somewhat less balanced: $|R(e, n) - R(e', n)|$ is typically on the order of $\sqrt{n}$. It turns out that this level of balance is still enough to get a fairly good approximation and hence a significant speedup in our algorithm.

If the rotor stacks are balanced, then the stack Laplacian Δ_ρ is well-approximated by the operator Δ on functions $u : V \to \mathbb{Z}$ defined by

$$\Delta u(z) = \sum_{\mathtt{t}(e)=z} \frac{u(\mathtt{s}(e))}{d_{out}(\mathtt{s}(e))} - u(z).$$

Note that Δ is the adjoint of the usual discrete Laplacian on G.

In this setting we can approximate the behavior of our stack-based aggregation with an idealized model called the *divisible sandpile* [23]. Instead of discrete chips, each vertex z has a real-valued "mass" $\sigma_0(z)$. Any site with mass greater than 1 can fire by keeping mass 1 for itself, and distributing the excess mass to its out-neighbors by sending an equal amount of mass along each outgoing edge. The resulting odometer function

$$v(z) = \text{total mass emitted from } z$$

satisfies the discrete variational problem

$$\begin{aligned} v &\geqslant 0 \\ \Delta v &\leqslant 1 - \sigma_0 \\ v(\Delta v - 1 + \sigma_0) &= 0. \end{aligned} \tag{8}$$

In words, these conditions say that each site emits a nonnegative amount of mass, each site ends with mass at most 1; and each site that emits a positive amount of mass ends with mass exactly 1. The conditions (8) can be reformulated as an obstacle problem, that of finding the smallest superharmonic function lying above a given function; see [24]. That formulation shows existence and uniqueness of the solution v.

If the rotor stacks are sufficiently balanced, we expect the divisible sandpile odometer function v to approximate closely our abelian stack odometer u. The next question is how to compute or approximate v. The obstacle problem formulation shows that v can be computed exactly by linear programming. Such an approach works well for small to moderate system sizes, but for the sizes we are interested in, the number of variables $v(z)$ is prohibitively large.

Fortunately, for specific examples it is sometimes possible to guess a near solution $w \approx v$. We briefly indicate how to do this for the specific example of interest to us, the initial configuration

$$\sigma_0 = N\delta_o$$

consisting of N chips at the origin $o \in \mathbb{Z}^2$. In that case the set of sites that are fully occupied in the final divisible sandpile configuration $\sigma_0 + \Delta v$ is very close to the disk

$$B_r = \{z \in \mathbb{Z}^2 \colon |z| < r\}$$

of radius $r = \sqrt{N/\pi}$; see [23, Theorem 3.3]. Here $|z| = (z_1^2 + z_2^2)^{1/2}$ is the Euclidean norm. Thus we are seeking a function $w \colon \mathbb{Z}^2 \to \mathbb{R}$ satisfying

$$\begin{aligned} \Delta w &= 1 - N\delta_o && \text{in } B_r \\ w &\approx 0 && \text{on } \partial B_r. \end{aligned}$$

An example of such a function is

$$w(z) = |z|^2 - Na(z) - r^2 + Na((r,0)) \tag{9}$$

where $a(z)$ is the potential kernel for simple random walk $(X_n)_{n\geqslant 0}$ started at the origin in $\mathbb{Z}^2$, defined as

$$a(z) = \sum_{n=1}^{\infty} \left(\mathbb{P}(X_n = o) - \mathbb{P}(X_n = z)\right).$$

Its discrete Laplacian is $\Delta a = \delta_o$.

As input to our algorithm we will use the function

$$w(z)^+ := \max(0, w(z))$$

where $w(z)$ is given by (9). One computational issue remains, which is how to compute the potential kernel $a(z)$. The potential kernel has the asymptotic expansion [14, Remark 2]

$$a(z) = \frac{2}{\pi}\ln|z| + \kappa + \frac{1}{6\pi}\frac{8\omega_1^2\omega_2^2 - 1}{|z|^2} + O(|z|^{-4}) \tag{10}$$

where $\omega = z/|z|$, and $\kappa = \frac{\ln 8 + 2\gamma}{\pi}$; here $\gamma \approx 0.577216$ is Euler's constant $\lim(\sum_{k=1}^{n}\frac{1}{k} - \ln n)$. Note that if θ is the argument of z, then

$$\begin{aligned} 8\omega_1^2\omega_2^2 - 1 &= 8\sin^2\theta\cos^2\theta - 1 \\ &= 2\sin^2 2\theta - 1 \\ &= \sin^2 2\theta - \cos^2 2\theta \\ &= -\cos 4\theta. \end{aligned}$$

Thus, identifying $\mathbb{Z}^2$ with $\mathbb{Z} + i\mathbb{Z} \subset \mathbb{C}$, we can write

$$a(z) = \frac{2}{\pi}\ln|z| + \kappa - \frac{1}{6\pi}\frac{\mathrm{Re}(z^4)}{|z|^6} + O(|z|^{-4}).$$

For z close to the origin the error term $O(|z|^{-4})$ becomes significant. Therefore, we use the McCrea-Whipple algorithm [25] (see also [19]) to precompute $a(z)$ exactly for $|z| < 100$. This algorithm uses the exact identity

$$a(n + in) = \frac{4}{\pi}\sum_{k=1}^{n}\frac{1}{2k-1}$$

for $n \geqslant 0$, together with the relation $\Delta a = \delta_o$ and reflection symmetry across the real and imaginary axes to compute $a(z)$ recursively. As this calculation is numerically very ill-conditioned, we performed it in advance by symbolic calculation with a computer algebra system.

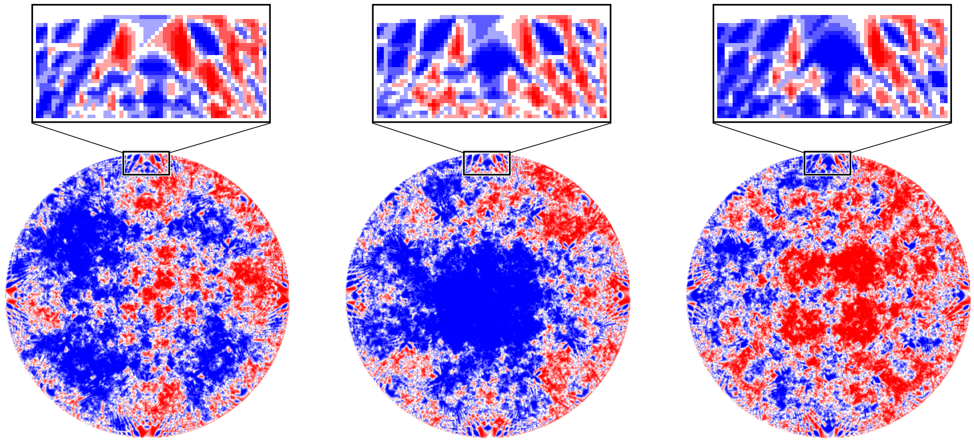
Now we can describe the function u_1 that we used as input to the first step of our algorithm. Let $r = \sqrt{N/\pi}$. Approximating the term $a((r, 0))$ in (9) by $\frac{2}{\pi}\log r + \kappa$, we set

$$u_1(z) = \left\lfloor |z|^2 + r^2\left(2\ln r - 1 + \pi\kappa - \pi a(z)\right)\right\rceil, \qquad |z| < 100.$$

Here $\lfloor t\rceil = \lfloor t + \frac{1}{2}\rfloor$ denotes the closest integer to $t \in \mathbb{R}$. For $|z| \geqslant 100$ we use the asymptotic expansion for $a(z)$ in (9), which gives

$$u_1(z) = \left\lfloor |z|^2 + r^2\left(2\ln\frac{r}{|z|} - 1 + \frac{\mathrm{Re}(z^4)}{6\,|z|^6}\right)\right\rceil^+, \qquad |z| \geqslant 100,$$

where $t^+ := \max(t, 0)$. Including more terms of the asymptotic expansion of $a(z)$ from [19] improves the approximation very slightly, but increases the overall runtime.



(a) $N = 100000$. **(b)** $N = 100100$. **(c)** $N = 100200$.

Figure 4. Classic rotor router aggregation with counterclockwise rotor sequence. The pictures show the quality of the odometer approximation for different values of N, as measured by the difference $u_1 - u$. The site x is colored blue if $u_1(x) > u(x)$, red if $u_1(x) < u(x)$, and white if $u_1(x) = u(x)$. The dramatic dependence on N suggests that our approximation u_1 captures substantially all of the large-scale regular structure in u.

6. Experimental Results

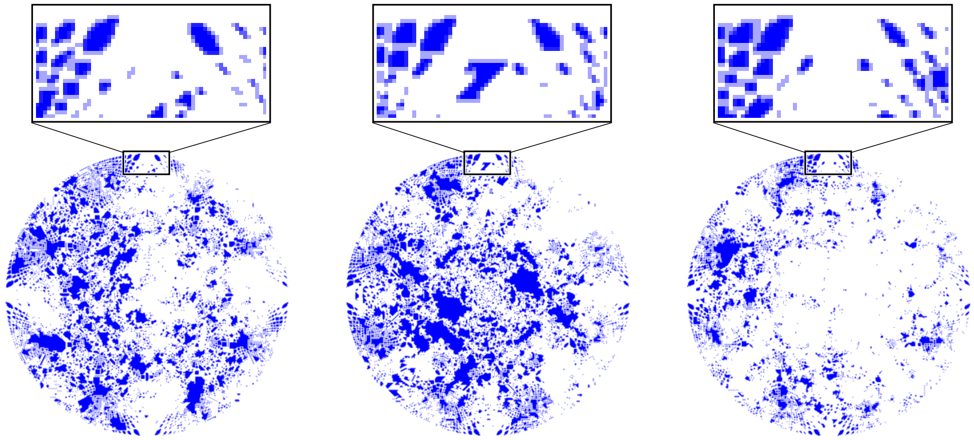
We implemented our algorithm for three different growth models in $\mathbb{Z}^2$: rotor-router aggregation, IDLA, and a hybrid of the two which we call "low-discrepancy random stack." In this section we discuss some details of the implementation, comment on the observed runtime, and present our findings on the scale of the fluctuations of the cluster A_N from circularity for large N.

As a basis for comparison to our algorithm, consider the time it takes to compute the occupied cluster A_N for rotor-router aggregation by the traditional method of firing one vertex at a time. If $z_1, \ldots, z_N \in \mathbb{Z}^2$ are the locations of the N chips, define the *quadratic weight* $Q(\mathbf{z}) = \sum_{i=1}^N |z_i|^2$, where $|(x,y)| = (x^2+y^2)^{1/2}$ is the Euclidean norm. Firing a given vertex z four times results in exactly one chip being sent to each of the four neighbors $z \pm e_1$, $z \pm e_2$. The net effect of these four firings on the quadratic weight is to increase Q by

$$|z+e_1|^2 + |z-e_1|^2 + |z+e_2|^2 + |z-e_2|^2 - 4|z|^2 = 4.$$

Thus, the total number of firings needed to produce the final occupied cluster A_N is approximately $\sum_{z \in A_N} |z|^2$. Since A_N is close to a disk of area N, this sum is about $N^2/2\pi$.

Traditional step-by-step simulation therefore requires quadratic time to compute the occupied cluster. Step-by-step simulation of IDLA also requires quadratic time, as observed in [21, 27]. We find experimentally that our algorithm runs in significantly shorter time: about $N \log N$ for the rotor-router model (Table 1), and about $N^{1.5}$ for IDLA (Table 2).



(a) $N = 100000$. **(b)** $N = 100100$. **(c)** $N = 100200$.

Figure 5. Classic rotor router aggregation with counterclockwise rotor sequence. The pictures show the quality of the odometer approximation after the annihilation phase, for different values of N, as measured by the difference $u_2 - u$. The site x is colored blue if $u_2(x) > u(x)$, white if $u_2(x) = u(x)$. Note that after annihilation, there are no longer any sites satisfying $u_2(x) < u(x)$. The remaining odometer difference also shows how many cycles are then popped in the last phase of our algorithm. The darker the color, the more cycles run through this location.

6.1. **Rotor-router aggregation.** In the classic rotor-router model, the rotor stack is the cyclic sequence of the four cardinal directions in counterclockwise order. Table 1 shows some statistical data of our computation, which was performed on a Sun Fire X4600 M2 Server with 256 GB main memory using all four AMD Opteron 8222 processors. The absolute error in our odometer approximation

$$\|u_1 - u\|_1 = \sum_x |u_1(x) - u(x)|$$

appears to scale linearly with N. This quantity is certainly a lower bound for the running time of our algorithm. The measured runtimes indicate close-to-linear runtime behavior, which suggests that our multiscale approach to canceling out hills and holes is relatively efficient.

Figure 4 depicts the odometer difference $u_1(x) - u(x)$ for three different values of N. Figure 5 depicts the odometer difference $u_2(x) - u(x)$ after the annihilation step of the algorithm.

The asymptotic shape of rotor-router aggregation is a disk [22, 23]. To measure how close A_N is to a disk, we define the *inradius* and *outradius* of a set $A \subset \mathbb{Z}^2$ by

$$r_{in}(A) = \min\{|x| \colon x \notin A\}$$

and

$$r_{out}(A) = \max\{|x| \colon x \in A\}.$$

We then define

$$\mathrm{diff}(N) = r_{out}(A_N) - r_{in}(A_N).$$

Kleber [18] observed that $\mathrm{diff}(3{\cdot}10^6) \approx 1.6106$. We can now extend the measurement of $\mathrm{diff}(N)$ up to $N = 4{\cdot}10^9$. Our algorithm takes about half an hour for this value of N; by comparison, a step-by-step simulation of this size would take a several centuries

Number of chips N	**Runtime**	**Radius Difference** absolute	recentered	$\|u_1 - u\|_1/N$	$\max \|u_1 - u\|$	**highest hill**	**deepest hole**
2^{10}=1,024	1.29 ms	1.324	0.278	1.800	6	3	-1
2^{11}=2,048	0.82 ms	1.490	0.273	1.142	5	3	-1
2^{12}=4,096	1.69 ms	1.523	0.138	3.370	10	3	-1
2^{13}=8,192	2.83 ms	1.606	0.235	3.599	11	3	-1
2^{14}=16,384	5.03 ms	1.579	0.166	2.417	12	3	-1
2^{15}=32,768	9.28 ms	1.567	0.274	2.405	10	3	-1
2^{16}=65,536	24.0 ms	1.611	0.429	4.461	17	3	-1
2^{17}=131,072	41.8 ms	1.652	0.279	2.486	16	3	-1
2^{18}=262,144	80.8 ms	1.565	0.346	2.919	16	3	-1
2^{19}=524,288	0.18 sec	1.463	0.237	2.955	16	3	-1
2^{20}=1,048,576	0.40 sec	1.642	0.362	4.323	23	3	-1
2^{21}=2,097,152	0.83 sec	1.591	0.297	3.594	26	3	-1
2^{22}=4,194,304	1.72 sec	1.596	0.316	4.220	29	3	-1
2^{23}=8,388,608	2.81 sec	1.674	0.418	3.141	34	3	-1
2^{24}=16,777,216	5.85 sec	1.614	0.396	3.974	45	3	-1
2^{25}=33,554,432	0.20 min	1.538	0.229	3.142	43	3	-1
2^{26}=67,108,864	0.43 min	1.658	0.368	4.695	62	3	-1
2^{27}=134,217,728	0.77 min	1.563	0.184	3.344	67	3	-1
2^{28}=268,435,456	1.60 min	1.639	0.340	4.463	83	3	-1
2^{29}=536,870,912	3.76 min	1.679	0.402	4.495	85	3	-1
2^{30}=1,073,741,824	6.90 min	1.635	0.414	4.309	91	3	-1
2^{31}=2,147,483,648	0.21 hours	1.521	0.304	3.932	127	3	-1
2^{32}=4,294,967,296	0.50 hours	1.650	0.366	4.383	172	4	-2

Table 1. Simulation results for classic rotor-router aggregation with counterclockwise rotor sequence. The given runtime is the total runtime of the calculation of one rotor-router aggreation of the given size on a Sun Fire X4600 M2. The next two columns show the difference between the outradius and inradius of the occupied cluster A_N, measured with respect to the origin ("absolute") and with respect to the putative center of mass $\left(\frac{1}{2}, \frac{1}{2}\right)$ ("recentered"). The next two columns give two measurements of the error of our odometer approximation u_1, the total absolute error and maximum pointwise error. In the last two columns, "highest hill" and "deepest hole" refer respectively to $\max_x \sigma_1(x)$ and $\min_x \sigma_1(x)$.

on a computer with one billion operations per second. In our implementation, the limiting factor is memory rather than time.

Up to dihedral symmetry, there are three different balanced period-4 rotor sequences for $\mathbb{Z}^2$: `WENS`, `WNSE` and `WNES`. The notation `WENS` means that the first four rotors in each stack point respectively west, east, north and south.

Figure 6a shows the radius difference $\text{diff}(N)$ for various N for the three different rotor sequences. As these values are rather noisy, we have also calculated and plotted the averages

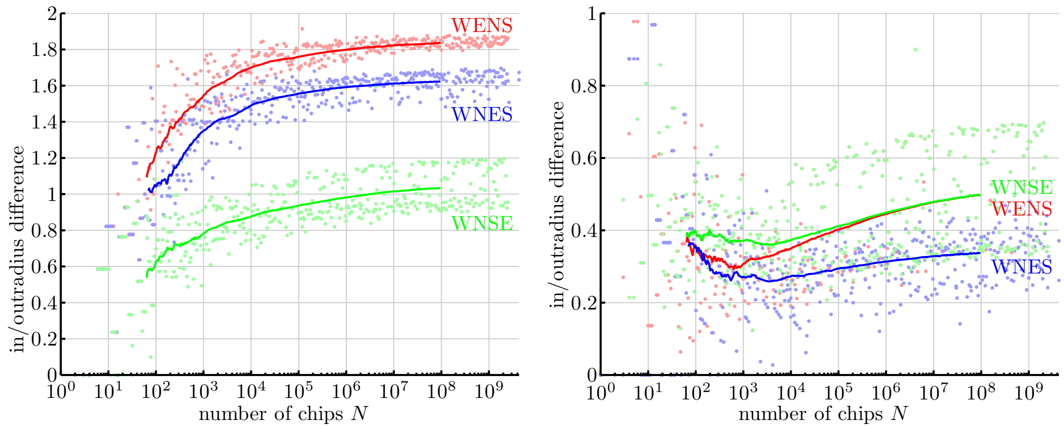
$$\overline{\text{diff}}(N) := \frac{1}{|I(N)|} \sum_{N' \in I(N)} \text{diff}(N') \tag{11}$$

with

$$I(N) = \begin{cases} \left[\frac{N}{2}, \frac{3N}{2}\right] & \text{for } N \leqslant 10^6, \\ [N - 5 \cdot 10^5, N + 5 \cdot 10^5] & \text{for } N > 10^6. \end{cases}$$

Note that in Figure 6a, the radius difference $\overline{\text{diff}}(N)$ grows extremely slowly in N. In particular, it appears to be sublogarithmic.

We observe a systematic difference in behavior for the three different rotor sequences. The observed radius differences are lowest for `WNSE`, intermediate for `WNES`, and highest



(a) Radius difference around the origin. **(b)** Radius difference around putative center.

Figure 6. Difference between the inradius and outradius of the rotor-router aggregate, for different numbers of chips N. The single dots are individual values of $\mathrm{diff}(N)$ (left) and $\mathrm{diff}'(N)$ (right). The darker curves show the averages $\overline{\mathrm{diff}}(N)$ and $\overline{\mathrm{diff}}'(N)$ as defined in equations (11) and (12). (Color scheme: `WNES`=blue, `WNSE`=green, `WENS`=red).

for `WENS`. For example,

$$\overline{\mathrm{diff}}(10^8) \approx \begin{cases} 1.034 & \text{for } \texttt{WNSE}, \\ 1.623 & \text{for } \texttt{WNES}, \\ 1.837 & \text{for } \texttt{WENS}. \end{cases}$$

This difference can be partially explained by considering the center of mass of the aggregate. Recall that our convention is "retrospective," not "prospective" rotor notation: that is, the rotor currently on top of the stack indicates where the last chip has gone rather than where the next chip will go. Hence for `WNES` rotors, the first time each site fires it sends a chip north, the next time east, then south, then west. As about $1/4$ of the sites end up in each of the four rotor states, for `WNES` rotors about half of the sites send one more chip N than S, and (a different but overlapping) half send one more chip E than W. As a result, the center of mass of the set of occupied sites is close to $(1/2, 1/2)$. For `WENS` the center of mass is close to $(3/4, 1/4)$, and for `WNSE` it's close to $(1/4, 1/4)$.

In some sense a better measure of circularity than $\mathrm{diff}(N)$ is the radius difference relative to the center of mass. Thus we define

$$\mathrm{diff}'(N) = r_{out}(A_N - \mathbf{c}) - r_{in}(A_N - \mathbf{c})$$

where $\mathbf{c}$ is one of $(1/2, 1/2)$, $(3/4, 1/4)$ or $(1/4, 1/4)$ chosen according to the rotor sequence used. Let

$$\overline{\mathrm{diff}}'(N) := \frac{1}{|I(N)|} \sum_{N' \in I(N)} \mathrm{diff}'(N). \tag{12}$$

These values are plotted for various N in Figure 6b. We find

$$\overline{\mathrm{diff}}'(10^8) \approx \begin{cases} 0.499 & \text{for } \texttt{WNSE} \text{ and } \texttt{WENS}, \\ 0.338 & \text{for } \texttt{WNES}. \end{cases}$$

Number of chips N	Average Runtime	Radius Difference	$\|u_1 - u\|_1/N^{3/2}$	$\max\|u_1 - u\|$	Number of runs
2^{10}=1,024	20.6 ms	3.201±0.569	0.490±0.057	134±27	10^5
2^{11}=2,048	51.7 ms	3.570±0.547	0.515±0.054	219±41	10^5
2^{12}=4,096	0.12 sec	3.949±0.554	0.541±0.051	355±62	10^5
2^{13}=8,192	0.28 sec	4.307±0.553	0.565±0.049	568±93	10^5
2^{14}=16,384	0.70 sec	4.664±0.565	0.588±0.047	900±139	10^5
2^{15}=32,768	1.90 sec	5.028±0.579	0.610±0.045	1,418±207	10^5
2^{16}=65,536	5.42 sec	5.394±0.577	0.631±0.043	2,215±308	10^5
2^{17}=131,072	0.26 min	5.761±0.593	0.652±0.042	3,440±456	$5 \cdot 10^3$
2^{18}=262,144	0.76 min	6.124±0.583	0.672±0.042	5,325±674	$5 \cdot 10^3$
2^{19}=524,288	2.25 min	6.495±0.601	0.693±0.040	8,180±992	$5 \cdot 10^3$
2^{20}=1,048,576	6.69 min	6.851±0.577	0.712±0.039	12,507±1,421	$5 \cdot 10^3$
2^{21}=2,097,152	0.33 hours	7.198±0.606	0.729±0.037	19,073±2,151	$2 \cdot 10^3$
2^{22}=4,194,304	0.99 hours	7.595±0.610	0.748±0.036	28,996±3,081	$2 \cdot 10^3$
2^{23}=8,388,608	3.92 hours	7.940±0.564	0.766±0.039	43,885±4,482	$2 \cdot 10^2$
2^{24}=16,777,216	14.9 hours	8.335±0.646	0.779±0.030	66,503±7,042	10^2

Table 2. Simulation results for IDLA. These calculations were performed on a compute cluster of 96 Sun Fire V20z with AMD Opteron 250 processors. The given runtime is the total time taken for the calculation of one IDLA cluster of the given size on a single core. To fit within 4 GB main memory, we used $\lambda = 0$ for $N \leqslant 2^{22}$, $\lambda = 2$ for $N = 2^{23}$, and $\lambda = 5$ for $N = 2^{24}$. The next column shows the difference between the outradius and inradius of the occupied cluster A_N. The fourth and fifth column give two measurements of the error of our odometer approximation u_1, the total absolute error and maximum pointwise error. The values shown are averages and standard deviations over many independent trials; the last column shows the number of trials.

The differences are now significantly smaller, and the two non-cyclic rotor sequences `WNSE` and `WENS` have nearly the same radius difference for large N. To see why, note that `WENS` is obtained from `WNSE` by a shift in the stacks (to `EWNS`) followed by interchanging the directions east and west. Thus the observed difference in $\mathrm{diff}(N)$ between these two rotor sequences is entirely due to the effect of the initial condition of rotors primed to send chips west. By adjusting for the center of mass, we have largely removed this effect in $\mathrm{diff}'(N)$.

6.2. **Internal Diffusion Limited Aggregation (IDLA).** In IDLA, the rotor directions $\rho_k(x)$ for $x \in \mathbb{Z}^2$ and $k \in \mathbb{Z}$ are chosen independently and uniformly at random from among the four cardinal directions. In the course of firing and unfiring during steps 2 and 3 of our algorithm, the same rotor $\rho_k(x)$ may be requested several times. Therefore, we need to be able to generate the same pseudorandom value for $\rho_k(x)$ each time it is used. Generating and storing all rotors $\rho_k(x)$ for all x and all $1 \leqslant k \leqslant u_1(x)$ is out of the question, however, since it would cost $\Omega(N^2)$ time and space.

Moore and Machta [27] encountered the same issue in developing a fast parallel algorithm for IDLA. Rather than store all of the random choices, they chose to store only certain seed values for the random number generator and generate random walk steps online as needed. Next we describe how to adapt this idea to our setting for fast serial computation of IDLA.

As a source of pseudorandom numbers, we used the cryptographically secure random number generator "Advanced Encryption Standard" (AES) [15], which is the official successor of the well-known "Data Encryption Standard" (DES). We used a key size of 256 bits with the Rijndael cipher implementation by Rijmen, Bosselaers and Barreto, which is also used in OpenSSH. The generator takes as input a block of 128 bits and "encrypts" it, outputting a block of 128 pseudorandom bits. We interpret the output block as the binary expansion of a number in the interval $[0, 1)$.

Let $rnd(b)$ be the pseudorandom number generated from input block b. Let

$$U_k(x) = rnd(block(x, k, a)),$$

where $block(x, k, a)$ is a simple deterministic function that assumes distinct values for for each triple (x, k, a) of site x, odometer value $1 \leqslant k \leqslant K$, and integer $1 \leqslant a \leqslant A$. The integer a is fixed for each run of the algorithm, and A is the total number of runs of the algorithm; this way, each run generates an independent IDLA cluster. The bound K is chosen safely larger than the maximal odometer value $u_1(o) \approx 2r^2 \ln r$.

Writing $\uparrow$, $\rightarrow$, $\downarrow$, $\leftarrow$ for the four outgoing edges from site x, we set

$$\rho_k(x) := \begin{cases} \uparrow & \text{if } 0 \leqslant U_k(x) < 1/4, \\ \rightarrow & \text{if } 1/4 \leqslant U_k(x) < 1/2, \\ \downarrow & \text{if } 1/2 \leqslant U_k(x) < 3/4, \\ \leftarrow & \text{if } 3/4 \leqslant U_k(x) < 1. \end{cases} \tag{13}$$

The first step of the algorithm described in §4 is to calculate σ_1 from the odometer approximation u_1. In this calculation, the definition of $R(e, n)$ given in equation (2) involves evaluating $\rho_k(x)$ for all $1 \leqslant k \leqslant n$. As this is much too expensive, we instead use the fact that $R(e, n)$ is a random variable with the Binomial$(n, 1/4)$ distribution. In steps 2 and 3 of the algorithm, we need to sample some individual rotors $\rho_k(x)$, but typically not too many: on the order of $\sqrt{u_1(x)}$. The distribution of these rotors depends on the binomials already drawn. We think of first populating an urn with balls of 4 colors $\uparrow$, $\rightarrow$, $\downarrow$, $\leftarrow$. When the algorithm asks for an individual rotor, we draw a ball at random from the urn using our knowledge of how many balls of each color remain.

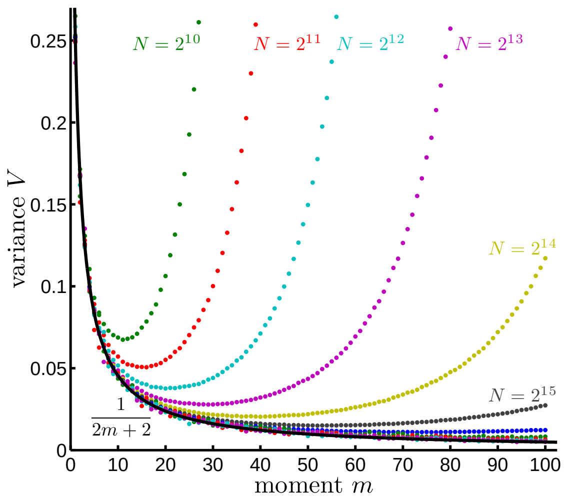
This approach works well for small and moderate system sizes, but for large N it is too memory-intensive. The memory usage comes from the need to store the rotors previously drawn from the urn. in order to keep track of how many balls of each color remain in the urn. Note that if each rotor were only needed once, then a simple count would suffice to keep track of how many balls of each color remain in the urn; but because the algorithm may request a single rotor multiple times, each rotor drawn from the urn needs to be stored in case it is needed later.

Fix a parameter $\lambda \geqslant 0$ representing the tradeoff between time and memory. A larger value of λ will result in saving memory at the cost of additional time. Let

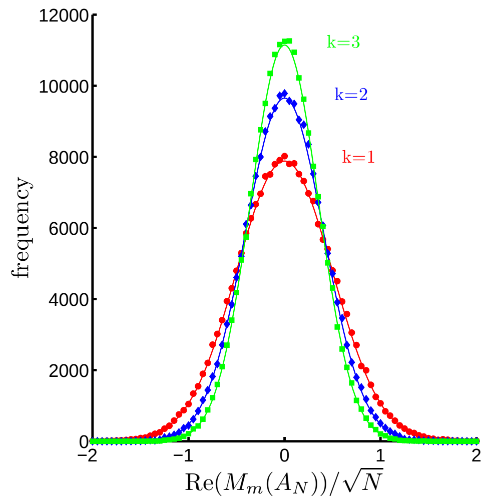
$$f(x) = \left(u_1(x) - \lambda\sqrt{u_1(x)}\right)^+.$$

For each site x with $f(x) > 0$, we sample three binomial random variables

$$\begin{aligned} B &\sim \text{Binomial}\big(f(x),\, \tfrac{1}{4}\big), \\ B' &\sim \text{Binomial}\big(f(x) - B,\, \tfrac{1}{3}\big), \\ B'' &\sim \text{Binomial}\big(f(x) - B - B',\, \tfrac{1}{2}\big). \end{aligned}$$



(a) Sample variance of the first 100 moments.



(b) Histograms of the first three moments.

Figure 7. Complex moments of the IDLA cluster. Left: The sample variance $V(m) = \mathbb{E}\,\mathrm{Re}(M_m(A_N)/\sqrt{N})^2$ of the real parts of the first 100 moments, for $N = 2^{10}, \ldots, 2^{20}$. As N increases, the variance of the real part of the m-th moment approaches $1/(2m+2)$, in agreement with the results of [17]. Right: Histogram of the real part of the first three moments for $N = 2^{16}$. The histogram shows 200,000 independent runs in bins of size 0.05. Data for the imaginary parts is similar.

We then set

$$\begin{aligned} R(\uparrow, f(x)) &= B \\ R(\rightarrow, f(x)) &= B' \\ R(\downarrow, f(x)) &= B'' \\ R(\leftarrow, f(x)) &= f(x) - B - B' - B''. \end{aligned}$$

Next, to implement step 1 of the algorithm described in §4, we need to know $R(e, u_1(x))$. So we compute

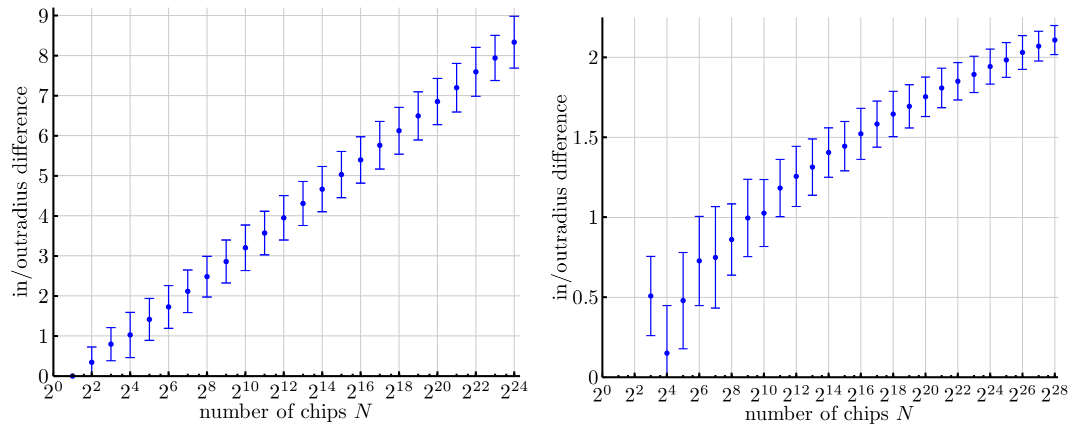
$$R(e, u_1(x)) = R(e, f(x)) + \#\{f(x) < k \leqslant u_1(x) \mid \rho_k(x) = e\}.$$

Note that if λ is large, then this calculation is expensive in time, since it involves calling the pseudorandom number generator to draw all of the rotors

$$\rho_k(x), \qquad f(x) < k \leqslant u_1(x)$$

using equation (13). But, crucially, these rotors do not need to be stored.

During steps 2 and 3 of the algorithm, we sample any rotors $\rho_k(x)$ for $k > f(x)$ as needed using (13). Rotors $\rho_k(x)$ for $k \leqslant f(x)$ can be sampled online as needed



(a) IDLA. **(b)** Low-discrepancy random stack.

Figure 8. Difference between inradius and outradius for different numbers of chips N for IDLA (§6.2) and the low discrepancy random stack model (§6.3). The respective data can be found in Tables 2 and 3.

according to the distribution

$$\rho_k(x) := \begin{cases} \uparrow & \text{if } U_k(x) \in [0,\, p_k(\uparrow)), \\ \rightarrow & \text{if } U_k(x) \in [p_k(\uparrow),\, p_k(\uparrow) + p_k(\rightarrow)), \\ \downarrow & \text{if } U_k(x) \in [p_k(\uparrow) + p_k(\rightarrow),\, p_k(\uparrow) + p_k(\rightarrow) + p_k(\downarrow)), \\ \leftarrow & \text{if } U_k(x) \in [p_k(\uparrow) + p_k(\rightarrow) + p_k(\downarrow),\, 1). \end{cases}$$

Here $p_k(e) = R(e,k)/k$. Initially, the values $R(e,k)$ are known only for $k = f(x)$. We generate the rotors $\rho_k(x)$ as needed in order of decreasing index k, starting with $k = f(x)$. Upon generating a new rotor $\rho_k(x) = e$, we inductively set

$$R(e, k-1) = R(e,k) - 1$$

and $R(e', k-1) = R(e', k)$ for $e' \in \{\uparrow, \rightarrow, \downarrow, \leftarrow\} - \{e\}$. These values specify the distribution for the next rotor $\rho_{k-1}(x)$ in case it is needed later.

The results of our large-scale simulations of IDLA are summarized in Table 2, extending the experiments of Moore and Machta [27] ($N \leqslant 10^{5.25}$ with 100 trials) to over $100,000$ trials for $N \leqslant 2^{16}$ and over 100 trials for $N \leqslant 2^{24} \approx 10^{7.2}$. The observed runtime of our algorithm for IDLA is about $N^{1.5}$; in contrast, serial simulation of each random walk used to build the cluster takes expected time order N^2 (cf. [27, Fig. 3]).

An interesting question is whether the runtime could be reduced further by starting from a random odometer approximation $\widetilde{u}_1$ instead of the deterministic approximation u_1. One approach is to draw binomials as above (taking $\lambda = 0$), and use them to define a "warped" Laplacian operator $\widetilde{\Delta}$, given by

$$\widetilde{\Delta} f(x) = \sum_{y \sim x} \frac{B_{yx}}{B_y} f(y) - f(x).$$

Here $B_y = u_1(y)$, and $B_{yx} = R((y,x), u_1(y))$ is the binomial associated to the directed edge (y,x). We then take $\widetilde{u}_1$ to be the solution to the variational problem (8), with Δ replaced by $\widetilde{\Delta}$. This problem can be formulated as a linear program: minimize $\sum_x \widetilde{u}_1(x)$ subject to the constraints $\widetilde{u}_1 \geqslant 0$ and $\widetilde{\Delta}\widetilde{u}_1 \leqslant 1 - N\delta_o$. One could even iterate this construction, using $\widetilde{u}_1$ to draw new binomials and get a new warping $\widetilde{\widetilde{\Delta}}$, and a new approximation $\widetilde{\widetilde{u}}_1$. A small number of iterations should suffice to bring the approximation very close to the true odometer. The main computational issue is how to quickly solve (or even approximately solve) these linear programs, which are sparse but quite large: the number of variables is about N. We achieved some modest speedup with this kind of approach, but not enough to justify the additional complexity.

To measure the circularity of the IDLA cluster, we computed the complex moments

$$M_m(A_N) = \sum_{z \in A_N} \left(\frac{z}{r}\right)^m$$

for $m = 1, \ldots, 100$. Here $r = \sqrt{N/\pi}$, and we view $z \in A_N$ as a point in the complex plane by identifying $\mathbb{Z}^2$ with $\mathbb{Z}+i\mathbb{Z}$. These moments obey a central limit theorem [17]: $M_m(A_N)/\sqrt{N}$ converges in distribution as $N \to \infty$ to a complex Gaussian with variance $1/(m+1)$. The distribution of the real part of $M_m(A_N)/\sqrt{N}$ is shown in Figure 7.

The expected value of the difference between outradius and inradius grows logarithmically in N: The data in the third column of Table 2, graphed in Figure 8(a), fit to

$$\mathbb{E}\,\mathrm{diff}(N) = 0.526 \ln(N) - 0.442$$

with a coefficient of determination of $R^2 = 0.99996$.

We remark that the above approach to calculating IDLA clusters depends strongly on the availability of a high-quality pseudorandom number generator such as AES. Small trials using C's built-in `rand(·)` function, which has a small period, produced a noticeably smaller difference between inradius and outradius.

6.3. **Low-Discrepancy Random Stack.** In the rotor-router model (§6.1), the neighbors are served in a maximally balanced manner, while in IDLA (§6.2), the rotor stack is completely random. Following a suggestion of James Propp, we examine a model which combines both features by using low-discrepancy random stacks. In this model the neighbors are served in similarly balanced manner as the rotor-router model. The rotor stacks consist of blocks of length 4, chosen independently and uniformly at random from among the 24 permutations of `NESW`. Hence the rotor stack is random, but still satisfies $|R(e,n) - R(e',n)| \leqslant 1$ for all n and all edges e and e' such that $\mathtt{s}(e) = \mathtt{s}(e')$.

This model can be implemented with our method in the same way as IDLA. Figure 8b gives averages and standard deviations for the radius difference $\mathrm{diff}(N)$ up to $N = 2^{28} = 268,435,456$. In contrast to IDLA, the difference between inradius and outradius now grows slower than logarithmically in N, and is not much larger than the corresponding difference for the rotor-router model. In fact, the data points $\mathbb{E}\,\mathrm{diff}(N)$ of Table 3 fit to

$$\mathbb{E}\,\mathrm{diff}(N) = 1.018 \ln\ln(N) - 0.919$$

with a coefficient of determination of $R^2 = 0.998$. Of course, it is very hard to distinguish empirically between slowly growing functions such as $\ln\ln(N)$ and $\sqrt{\ln(N)}$, so we can not be sure of the exact growth rate; among several functions we tried,

Number of chips N	**Average Runtime**	**Radius Difference**	$\|u_1 - u\|_1/N$	$\max \|u_1 - u\|$	**Number of runs**
2^{10}=1,024	3.16 ms	1.026±0.209	1.34±0.16	6.00±0.90	$5 \cdot 10^5$
2^{11}=2,048	6.21 ms	1.183±0.180	1.47±0.17	6.83±0.94	$5 \cdot 10^5$
2^{12}=4,096	12.0 ms	1.256±0.188	1.60±0.18	7.65±1.00	$5 \cdot 10^5$
2^{13}=8,192	23.9 ms	1.314±0.176	1.73±0.19	8.52±1.04	$5 \cdot 10^5$
2^{14}=16,384	49.7 ms	1.405±0.154	1.86±0.20	9.40±1.07	$5 \cdot 10^5$
2^{15}=32,768	0.10 sec	1.444±0.154	1.99±0.21	10.3±1.1	$5 \cdot 10^5$
2^{16}=65,536	0.21 sec	1.522±0.160	2.11±0.22	11.2±1.2	$5 \cdot 10^5$
2^{17}=131,072	0.45 sec	1.583±0.144	2.23±0.23	12.2±1.2	$5 \cdot 10^5$
2^{18}=262,144	0.93 sec	1.646±0.142	2.35±0.24	13.2±1.2	$5 \cdot 10^5$
2^{19}=524,288	1.90 sec	1.694±0.135	2.46±0.24	14.1±1.3	$5 \cdot 10^5$
2^{20}=1,048,576	3.88 sec	1.753±0.124	2.59±0.26	15.1±1.3	$5 \cdot 10^5$
2^{21}=2,097,152	7.96 sec	1.808±0.124	2.73±0.28	16.2±1.4	$5 \cdot 10^4$
2^{22}=4,194,304	0.27 min	1.850±0.117	2.86±0.29	17.3±1.4	$5 \cdot 10^4$
2^{23}=8,388,608	0.55 min	1.893±0.114	2.98±0.30	18.4±1.4	$5 \cdot 10^4$
2^{24}=16,777,216	1.13 min	1.942±0.109	3.11±0.31	19.4±1.5	$5 \cdot 10^3$
2^{25}=33,554,432	2.32 min	1.983±0.109	3.25±0.33	20.6±1.5	$5 \cdot 10^3$
2^{26}=67,108,864	4.74 min	2.030±0.106	3.35±0.32	21.6±1.5	$5 \cdot 10^3$
2^{27}=134,217,728	9.72 min	2.070±0.093	3.51±0.35	22.9±1.5	$5 \cdot 10^2$
2^{28}=268,435,456	0.33 hours	2.108±0.091	3.61±0.36	24.1±1.6	$5 \cdot 10^2$

Table 3. Simulation results for low-discrepancy random stack. The given runtime is the total runtime of the calculation of one cluster of the given size on one core of a AMD Opteron processor 8222. The next column shows the difference between the outradius and inradius of the occupied cluster A_N. The fourth and fifth column give two measurements of the error of our odometer approximation u_1, the total absolute error and maximum pointwise error. The values shown are averages and standard deviations over many independent trials; the last column shows the number of trials.

$\ln\ln(N)$ had the best fit. The very slow growth of $\mathrm{diff}(N)$ for low-discrepancy random stack suggests that the extremely good circularity of the rotor-router model is mainly due to its low discrepancy rather than its deterministic nature.

7. Further Directions

We have shown that the abelian stack model can be predicted exactly by correcting an initial approximation, and that such prediction is very fast if a good initial approximation is available. It would be interesting to investigate what other classes of cellular automata can be predicted quickly and exactly by correcting an initial approximation.

The abelian sandpile model in $\mathbb{Z}^2$ produces beautiful examples of pattern formation that remain far from understood [8, 12]. Using Lemma 2.3 of [12], it should be possible to characterize the sandpile odometer function in a manner similar to Theorem 1. In this characterization, the recurrent sandpile configurations play a role analogous to the acyclic rotor configuration in Theorem 1. The remaining challenge would then be to find a good approximation to the sandpile odometer function.

Acknowledgments

We thank James Propp for many enlightening conversations and for comments on earlier drafts of this article. We thank David Wilson suggesting the AES random number generator and for tips about how to use it effectively.

References

[1] available at `http://rotor-router.mpi-inf.mpg.de`.

[2] P. Bak, C. Tang, and K. Wiesenfeld. Self-organized criticality: an explanation of the $1/f$ noise. *Phys. Rev. Lett.*, 59(4):381–384, 1987.

[3] R. D. Barve, E. F. Grove, and J. S. Vitter. Simple randomized mergesort on parallel disks. *Parallel Computing*, 23(4-5):601–631, 1997.

[4] J. Cooper and J. Spencer. Simulating a random walk with constant error. *Combinatorics, Probability & Computing*, 15:815–822, 2006.

[5] J. Cooper, B. Doerr, J. Spencer, and G. Tardos. Deterministic random walks on the integers. *European Journal of Combinatorics*, 28:2072–2090, 2007.

[6] J. Cooper, B. Doerr, T. Friedrich, and J. Spencer. Deterministic random walks on regular trees. *Random Structures and Algorithms*, 2010+. To appear; Preliminary version appeared in *19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'08)*, pages 766–772, 2008.

[7] D. Dhar. Theoretical studies of self-organized criticality. *Physica A*, 369:29–70, 2006. See also arXiv:cond-mat/9909009.

[8] D. Dhar, T. Sadhu, and S. Chandra. Pattern formation in growing sandpiles. *Europhys. Lett.*, 85, 2009. arXiv:0808.1732.

[9] P. Diaconis and W. Fulton. A growth model, a game, an algebra, Lagrange inversion, and characteristic classes. *Rend. Sem. Mat. Univ. Politec. Torino*, 49(1):95–119, 1991.

[10] B. Doerr and T. Friedrich. Deterministic random walks on the two-dimensional grid. *Combinatorics, Probability & Computing*, 18(1-2):123–144, 2009.

[11] B. Doerr, T. Friedrich, and T. Sauerwald. Quasirandom rumor spreading. In *19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'08)*, pages 773–781, 2008.

[12] A. Fey, L. Levine, and Y. Peres. Growth rates and explosions in sandpiles. *J. Stat. Phys.*, 138:143–159, 2010. arXiv:math/0901.3805.

[13] T. Friedrich, M. Gairing, and T. Sauerwald. Quasirandom load balancing. In *21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*, pages 1620–1629, 2010.

[14] Y. Fukai and K. Uchiyama. Potential kernel for two-dimensional random walk. *Ann. Probab.*, 24(4):1979–1992, 1996.

[15] P. Hellekalek and S. Wegenkittl. Empirical evidence concerning AES. *ACM Trans. Model. Comput. Simul.*, 13(4):322–333, 2003.

[16] A. E. Holroyd and J. Propp. Rotor walks and Markov chains. *Algorithmic Probability and Combinatorics*, 2010. To appear. arXiv:0904.4507.

[17] D. Jerison, L. Levine, and S. Sheffield. Internal DLA and the Gaussian free field, 2010. Manuscript in preparation.

[18] M. Kleber. Goldbug variations. *Math. Intelligencer*, 27(1):55–63, 2005.

[19] G. Kozma and E. Schreiber. An asymptotic expansion for the discrete harmonic potential. *Electron. J. Probab.*, 9(1):1–17, 2004.

[20] G. F. Lawler. Subdiffusive fluctuations for internal diffusion limited aggregation. *Ann. Probab.*, 23(1):71–86, 1995.

[21] G. F. Lawler, M. Bramson, and D. Griffeath. Internal diffusion limited aggregation. *Ann. Probab.*, 20(4):2117–2140, 1992.

[22] L. Levine and Y. Peres. Spherical asymptotics for the rotor-router model in $\mathbb{Z}^d$. *Indiana Univ. Math. J.*, 57(1):431–450, 2008.

[23] L. Levine and Y. Peres. Strong spherical asymptotics for rotor-router aggregation and the divisible sandpile. *Potential Anal.*, 30:1–27, 2009.

[24] L. Levine and Y. Peres. Scaling limits for internal aggregation models with multiple sources. *J. d'Analyse Math.*, 2010. To appear. arXiv:0712.3378.

[25] W. H. McCrea and F. J. Whipple. Random paths in two and three dimensions. *Proc. Royal Soc.*, 60:281–298, 1940.

[26] P. Meakin and J. M. Deutch. The formation of surfaces by diffusion-limited annihilation. *J. Chem. Phys.*, 85, 1986.

[27] C. Moore and J. Machta. Internal diffusion-limited aggregation: parallel algorithms and complexity. *J. Stat. Phys.*, 99(3-4):661–690, 2000.

[28] J. G. Propp and D. B. Wilson. How to get a perfectly random sample from a generic Markov chain and generate a random spanning tree of a directed graph. *J. Algorithms.*, 27:170–217, 1998.

[29] D. B. Wilson. Generating random spanning trees more quickly than the cover time. In *28th Annual ACM Symposium on the Theory of Computing (STOC '96)*, pages 296–303, 1996.