COMPUTATIONAL COMPLEXITY OF BIASED DIFFUSION-LIMITED AGGREGATION [*]

NICOLAS BITAR [†], ERIC GOLES [‡], AND PEDRO MONTEALEGRE [‡]

Abstract. Diffusion-Limited Aggregation (DLA) is a cluster-growth model that consists in a set of particles that are sequentially aggregated over a two-dimensional grid. In this paper, we introduce a *biased* version of the DLA model, in which particles are limited to move in a subset of possible directions. We denote by k-DLA the model where the particles move only in k possible directions. We study the biased DLA model from the perspective of Computational Complexity, defining two decision problems The first problem is PREDICTION, whose input is a site of the grid c and a sequence S of walks, representing the trajectories of a set of particles. The question is whether a particle stops at site c when sequence S is realized. The second problem is REALIZATION, where the input is a set of positions of the grid, P. The question is whether there exists a sequence S that realizes P, i.e. all particles of S exactly occupy the positions in P. Our aim is to classify the PREDICITON and REALIZATION problems for the different versions of DLA. We first show that PREDICTION is **P**-Complete for 2-DLA (thus for 3-DLA). Later, we show that Prediction can be solved much more efficiently for 1-DLA. In fact, we show that in that case the problem is **NL**-Complete. With respect to REALIZATION, we show that restricted to 2-DLA the problem is in **P**, while in the 1-DLA case, the problem is in **L**.

Key words. Diffusion-Limited Aggregation, Computational Complexity, Space Complexity, NL-Completeness, P-Completeness

AMS subject classifications. 03D15, 68Q17, 68Q10

1. Introduction.

Diffusion-Limited Aggregation (DLA) is a kinetic model for cluster growth, first described by Witten and Sander [26], which consists of an idealization of the way dendrites or dust particles form, where the rate-limiting step is the diffusion of matter to the cluster. The original DLA model consists of a series of particles that are thrown one by one from the top edge of a d-dimensional grid, where $d \geq 2$. The sites in the grid can either be occupied or empty. Initially all the sites in the grid are empty, except for the bottom line which begins and remains occupied. Each particle follows a random walk in the grid, starting from a random position in the top edge, until it neighbors an occupied site, or the particle escapes from the top edge or one of the lateral edges. In case the particle finds itself neighboring an occupied site, the current position of the particle becomes occupied and the next particle is thrown. The set of occupied sites is called a *cluster*.

Clusters generated by the dynamics are highly intricate and fractal-like (see Figure 1); they have been shown to exhibit the properties of scale invariance and multifractality [12, 17]. DLA clusters have been observed to appear in electrodeposition, dielectrics and ion beam microscopy [7, 21, 22]. Nevertheless, perhaps the fundamental aspect of DLA is its profound connection to Hele-Shaw flow: it has been shown that DLA is its stochastic counterpart [9, 13].

In this article we study *restricted versions of DLA*, which consist in the limitation of the directions a particle is allowed to move within the grid. We ask what would be the consequences of restricting the particles movement in terms of computational

[*]Submitted to the editors December 20, 2021.

[†]Departamento de Ingeniería Matemática, Universidad de Chile, Santiago, Chile (nbitar@dim.uchile.cl).

[‡]Facultad de Ingeniería y Ciencias, Universidad Adolfo Ibáñez, Santiago, Chile (eric.chacc@uai.cl, p.montealegre@uai.cl).

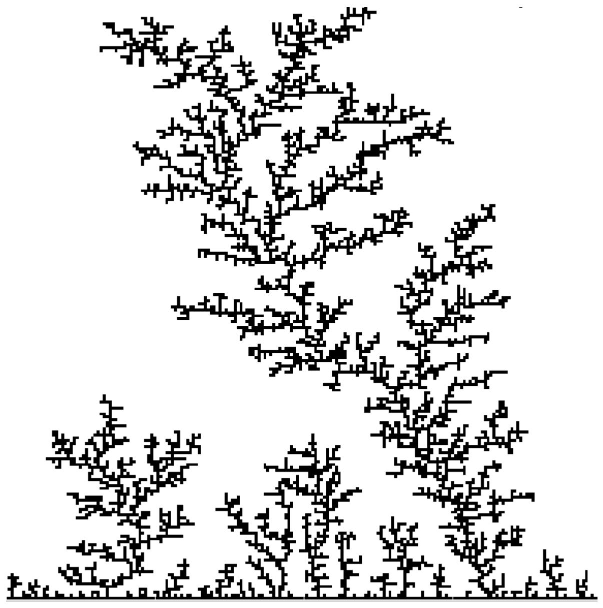


FIG. 1. *A realization of the dynamics for a* 200×200 *grid.*

complexity. More precisely, we consider four models, parameterized by their number of allowed movement directions, $k \in \{1, 2, 3, 4\}$. The 4-DLA model is simply the two-dimensional DLA model, i.e. when the particles can move in the 4 cardinal directions. The 3-DLA model is the DLA model when the particles can only move into the South, East or West direction. In the 2-DLA model, the directions are restricted to the South and East. Finally, in the 1-DLA model, the particles can only move downwards.

Even though the particles have restricted movements, it is possible to notice that the fractal-like structures are still present in the clusters obtained by the restricted DLA (see Figure 2).

It is interesting to note that, in fact, the 1-DLA model is a particular case of another computational model created to described processes in statistical physics, that of the Ballistic Deposition Model. In this model, there is a graph and a set of particles that are thrown into the vertices at some fixed initial height h. The particle's height decreases one unit at a time, until it reaches the bottom (height 0 or meets another particle, i.e. there is a particle in an adjacent vertex at the same height, or in the same vertex just behind. The 1-DLA model corresponds to the Ballistic Deposition model when the graph is an undirected path.

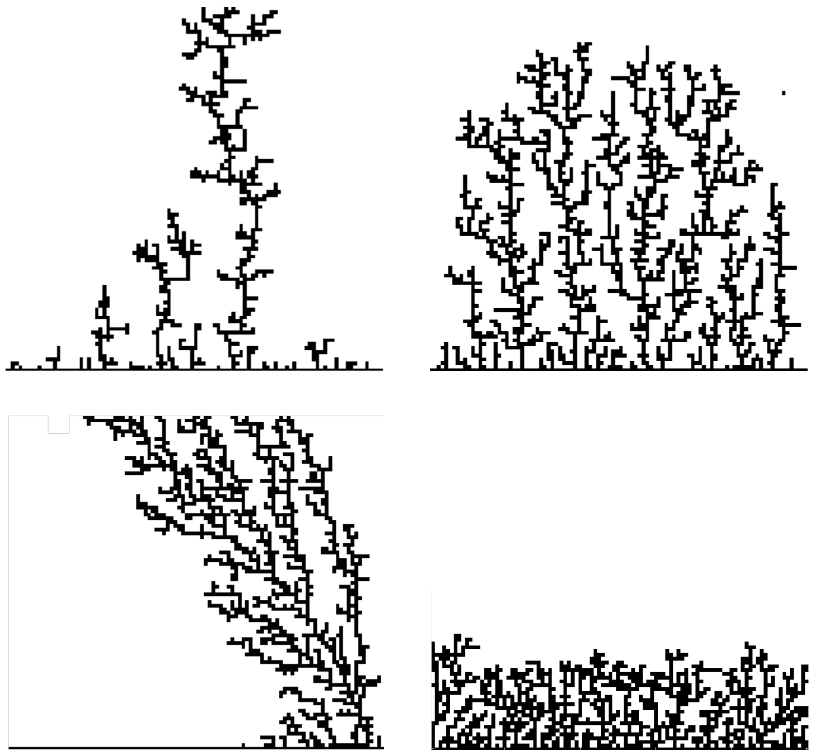


FIG. 2. *DLA simulation for four (top left), three (top right), two (bottom left) and one (bottom right) directions, when $N = 100$.*

Due to the generated cluster's properties, theoretical approaches to the DLA model are usually in the realm of fractal analysis, renormalization techniques and conformal representations [5]. In this article we consider a perhaps unusual approach to study the DLA model (and its restricted versions), related with its computational capabilities, the difficulty of simulating their dynamics, and the possibility of characterizing the patterns they produce. Machta and Greenlaw studied, within the framework of computational complexity theory, the difficulty of computing whether a given site on the grid becomes occupied after the dynamics have taken place, i.e. all the particles have stuck to the cluster or have been discarded [15]. Inspired by their work, we consider two decision problems:

- DLA-PREDICTION, which receives a sequence of trajectories for n particles (i.e. the trajectories are deterministic and explicit) and the coordinates of a site in the lattice as input. The question is whether the given coordinate is occupied by a particle after the n particles are thrown.

- DLA-REALIZATION, which receives an $n \times n$ sized pattern within the two-dimensional grid as input, and whose question is whether that pattern can be produced by the DLA model.

For each $k \in \{1, 2, 3\}$ we call k-DLA-PREDICTION and k-DLA-REALIZATION, respectively, the problems DLA-PREDICTION and DLA-REALIZATION restricted to the k-DLA model.

The computational complexity of a problem can be defined as the amount of resources, like time or space, needed to computationally solve it. Intuitively, the complexity of DLA-Prediction represent *how efficiently* (in terms of computational resources) we are able to simulate the dynamics of DLA. On the other hand, the complexity of DLA-Realization represent *how complex* are the patterns produced by DLA model.

We consider four fundamental complexity classes: **L**, **P**, **NL** and **NP**. Classes **L** and **P** contain the problems that can be solved in a *deterministic* Turing machine that use *logarithmic space* and take *polynomial time*, respectively. On the other hand **NL** and **NP** are the classes of problems that can be solved in a *non-deterministic* Turing machine that use *logarithmic space* and take *polynomial time*, respectively. For detailed definitions and characterizations of these classes we recommend the book of Arora and Barak [2]. A convention between computer theorists states that **P** is the class of problems that can be solved *efficiently* with respect to computation time. In that context, **NP** can be characterized as the classes of problems that can be *efficiently verified* with respect to computation time. Similarly, the same conventions hold for **L** and **NL** changing computation time for space.

It is easy to see that $\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP}$, though it is unknown if any of these inclusions is proper. Perhaps the most famous conjecture in Computational Complexity theory is $\mathbf{P} \neq \mathbf{NP}$. Put simply, this conjecture states that there are some problems whose solution can be efficiently verified but cannot be efficiently found. As mentioned in last paragraph, in this context *efficiently* means polynomial time. Similarly, it is conjectured that $\mathbf{L} \neq \mathbf{NL}$, where in this case efficiently refers to logarithmic space. It is also conjectured that $\mathbf{NL} \neq \mathbf{P}$, meaning that some problems can be computed efficiently with respect to computation time, but cannot be verified (or computed) efficiently with respect to space [2].

The problems in **P** that are the most likely to not belong to **NL** (hence not in **L**) are the **P**-Complete problems [11]. A problem is **P**-Complete if any other problem in **P** can be reduced to it via a *log-space reduction*, i.e. a function calculable in logarithmic space that takes *yes*-instances of one problem into the other. In a nutshell, it is unlikely that some **P**-Complete problem belongs to **NL**, because in that case we would have that $\mathbf{NL} = \mathbf{P}$. Similarly a problem is **NL**-Complete if any problem in **NL** can be reduced to it via a **L** reduction. **NL**-Complete problems are problems in **NL** that are the most likely to not belong to **L**, because if some **NL**-Complete problem were to belong in **L**, it would imply that $\mathbf{NL} = \mathbf{L}$ [2].

One **P**-Complete problem is the Circuit Value Problem (CVP), which consists in, given a Boolean circuit and an truth-assignment of its input gates, compute the output value of a given gate. Roughly, this problem is unlikely to be solvable (or verifiable) in logarithmic space because there is no better algorithm than simply sequentially computes truth values of each gate of the Boolean circuit, keeping in memory the values of all gates already evaluated. One **NL**-Complete problem is Reachability, which consists in, given a directed graph G and two vertices s and t, decide if there is a directed path between s and t. Roughly, this problem is unlikely to be solvable in logarithmic space because there is no way to remember all paths starting from s that do not reach t.

Within this context, it was shown by Machta and Greenlaw [15], that DLA-Prediction is **P**-Complete. The proof of this fact consists of reducing to it a version of the Circuit Value Problem, which is known to be **P**-Complete [11]. Within this proof, we noticed that the gadgets used to simulate the circuits rely heavily on the fact that in the DLA model, particles are free to move in any of the four cardinal directions. On the other hand, in the context of the study of the Ballistic deposition model it was proven by Matcha and Greenlaw [14] that 1-DLA-Prediction is in **NC**. The class **NC** is a complexity class that contains **NL** and is contained in **P**.

1.1. Our results. We begin our study of the complexity of biased DLA analyzing the complexity of the DLA-Prediction problem. By extending the results of Machta and Greenlaw to the 2-DLA model, we show that 2-DLA-Prediction is **P**-Complete. This result is obtained following essentially the same gadgets used for the non-restricted case, but carefully constructing them using only two directions. More precisely, the construction of Machta and Greenlaw consists in a representation of an instance of Circuit Value Problem as a sequence of particle throws, which final positions represent the input Boolean circuit with its gates evaluated on the given input. A gate evaluated *true* is represented by a path of particles, while the *false* signals are represented by the lack of such path. Since the construction of the circuit must be done in logarithmic space, the sequence of particles must be defined without knowing the actual output of the gates. Therefore, the trajectory of the particles considers that, if they are do not stick on a given position, they escape through the top edge of the grid. Since this escape movement is not possible in the 2-DLA (nor 3-DLA) model (because particles can not move upwards), we modify the circuit construction to build the gates in a specific way, in order to give the particles enough space to escape through the rightmost edge or deposit at the bottom of the grid without disrupting the ongoing evaluation of the circuit.

The fact that 2-DLA-Prediction is **P**-Complete directly implies that 3-DLA-Prediction is also **P**-Complete, settling the prediction problem for these two biased versions of the model.

We then study the 1-DLA model. Despite of what one might guess, the dynamics of the DLA model restricted to one direction are far from trivial (see Section 2.2 for examples of the patterns produced by this model). Indeed, we begin our study showing that this dynamics can simulate simple sorting algorithms like *Bead-Sort*. Then, we improve the result of Matcha and Greenlaw by showing that 1-DLA-Prediction is in **NL**. This is in fact an improvement, because they showed that 1-DLA-Prediction is in $\mathbf{NC}^2$ [14], and **NL** is a sub-class of $\mathbf{NC}^2$ [11]. Our result holds for the Ballistic Deposition model, i.e., when the graph is not restricted to a path but is an arbitrary graph given in the input (we call that problem BD-Prediction). We finish our study of the prediction problem showing that the complexity of BD-Prediction *cannot be improved*. Indeed, we show that BD-Prediction is **NL**-Complete.

After this, we study the DLA-Realization problem. We observe that k-DLA-Realization is in **NP** for all $k \in \{1, 2, 3, 4\}$. Moreover, the non-deterministic aspect of the **NP** algorithm solving DLA-Realization only needs to obtain the order of the sequence on which the particles are placed on the grid, rather than obtaining both the order and the trajectory that each particle follows. In fact, the trajectories can be computed in polynomial time, given the order in which the particles are placed in the grid.

We then show 1-DLA-Realization can be solved much more efficiently. In fact, we give a characterization of the patterns that the 1-DLA model can produce. Our characterization is based on a planar directed acyclic graph (PDAG) that represents the possible ways in which the particles are able to stick. Each occupied cell of the grid is represented by a node of the PDAG, plus a unique sink vertex that represents the *ground*. We show that a pattern can be constructed by 1-DLA if and only if there is a directed path from every vertex to the ground. We use our characterization to show that 1-DLA-Realization is in **L**, using a result of Allender *et al.* [1] solving Reachability in log-space, when the input graph is a single sink PDAG.

Finally, we give an efficient algorithm to solve the realization problem in the 2-DLA model, showing that 2-DLA-Realization is in **P**. Our algorithm uses the fact that in the 2-DLA model, the particles are placed into the grid in a very specific way. By establishing which particles force the order in which they are placed. We use this fact to efficiently compute the order in which the particles are thrown, obtaining a polynomial-time algorithm.

1.2. Related work.

For dynamical properties of the restricted versions of DLA, including Ballistic Deposition, we refer the reader to [4, 16, 18, 23].

Some problems of similar characteristics have been studied in this context, such as the Ising Model, Eden Growth, Internal DLA and Mandelbrot Percolation, to name a few [15, 19]. On the other hand, the problem of Sandpile Prediction is an example where increasing the degrees of freedom increases the computational complexity of the prediction problem. In particular, when the dimension is greater than 3, the prediction problem is **P**-Complete; but when the dimension is 1, the problem is in **NC** [20].

Another example of a complexity dichotomy that depends on the topology of the system is the Bootstrap Percolation model [8]. In this model, a set of cells in a d-dimensional grid are initially infected, in consecutive rounds, healthy sites that have more than the half of their neighbors infected become infected. In this model, the prediction problem consists in determining wether a given site becomes infected at some point in the evolution of the system. In [10] it is shown that this prediction problem is **P**-Complete in three or more dimensions, while in two dimensions it is in **NC**. Other problems related to Bootstrap percolation involve the maximum time that the dynamics takes before converging to a fixed point [6].

1.3. Structure of the article.

The first section formally introduces the different computational complexity classes, along side problems of known complexity used throughout the article. Next, the dynamics for the general case of DLA are presented, in addition to the formal definition of the two associated prediction problems that are discussed: Prediction and Realization. The third section focuses on the proof that DLA restricted to 2 or 3 directions is **P**-Complete and with the presentation of the non-deterministic log-space algorithm for the generalized version of the one direction DLA problem, Ballistic Deposition. The last section talks about the results concerning the Realization problem, where the one-directional case is shown to be solvable in **L**, and the two-directional case has a polynomial algorithm characterizing figures obtained from the dynamics.

2. Preliminaries.

2.1. Complexity Classes and Circuit Value Problem.

In this subsection we will define the main background concepts in computational complexity required in this article. For a more complete and formal presentation we refer to the books of Arora and Barak [2] and Greenlaw et al. [11]. We assume that the reader is familiar with the basic concepts dealing with computational complexity. As we mentioned in the introduction, in this paper we will only consider complexity classes into which we classify the prediction problems. **P** is the class of problems solvable in a Turing machine that runs in polynomial time in the size of the input. More formally, if n is the size of the input, then a problem is polynomial time solvable if it can be solved in time $n^{\mathcal{O}(1)}$ in a deterministic Turing machine.

A logarithmic-space Turing machine consists in a Turing machine with three tapes: a read-only *input tape*, a write-only *output-tape* and a read-write *work-tape*. The Turing machine is allowed to move as much as it likes on the input tape, but can only use $\mathcal{O}(\log n)$ cells of the work-tape (where n is the size of the input). Moreover, once the machine writes something in the output-tape, it moves to the next cell and can not return. **L** is the class of problems solvable in a logarithmic-space Turing machine.

A non-deterministic Turing machine is a Turing machine whose transition function does not necessarily output a single state but one over a set of possible states. A computation of the non-deterministic Turing machine considers all possible outcomes of the transition function. The machine is required to stop in every possible computation thread, and we say that the machine *accepts* if at least one thread finishes on an accepting state. A non-deterministic Turing machine is said to run in *polynomial-time* if every computation thread stops in a number of steps that is polynomial in the size of the input. **NP** is the class of problems solvable in polynomial time in a non-deterministic Turing machine. A non-deterministic Turing machine is said to run in *logarithmic-space* if every thread of the machine uses only logarithmic space in the work-tape. **NL** is the class solvable in logarithmic space in a non-deterministic Turing machine.

A problem $\mathcal{L}$ is **P**-Complete if it belongs to **P** and any other problem in **P** can be reduced to $\mathcal{L}$ via a logarithmic-space (many-to-one or Turing) reduction. A **P**-Complete problem belongs to **NL** implies that $\mathbf{P} = \mathbf{NL}$.

One well-known **P**-Complete problem is the PLANAR-NOR-CIRCUIT-VALUE-PROBLEM. A *Planar NOR Boolean Circuit* is a planar directed acyclic graph C, where each vertex of C has two incoming and two outgoing edges, except for some vertices that have no incoming edges (called *inputs* of C) and others that have no outgoing edges (called *outputs* of C).

Each vertex of C has a Boolean value (TRUE or FALSE). A truth assignment of the inputs of C, called I, is an assignment of values of the input gates of C. The value of a non-input gate v is the NOR function (the negation of the disjunction) of the value of the two incoming neighbors of v. A truth assignment I of C defines a a truth-value of the output gates according to the truth values of the preceding gates. We call $C(I)$ the truth values of the output vertices of C when the input gates are assigned I.

The PLANAR-NOR-CIRCUIT-VALUE PROBLEM is defined as follows.

PLANAR-NOR-CIRCUIT-VALUE PROBLEM (PNORCVP)
Input: A NOR Boolean Circuit C of size n, a truth-assignment I of C and g an output gate of C.
Question: Is g TRUE in $C(I)$?

In [11] it is shown that this problem is **P**-Complete.

PROPOSITION 2.1 ([11]). PNORCVP *is* **P**-*Complete.*

A problem $\mathcal{L}$ is **NL**-Complete if $\mathcal{L}$ belongs to **NL** and any other problem in

NL can be reduced to $\mathcal{L}$ via a (many-to-one or Turing) reduction computable in logarithmic space. A **NL**-Complete problem belongs to **L** implies that **NL** = **L**.

One **NL** problem is REACHABILITY [2]. An instance of REACHABILITY is a directed graph G and two vertices s and t. The instance is accepted if there is a directed path from s to t in G. REACHABILITY is **NL**-Complete because the computation of a non-deterministic log-space Turing machine (which is the set of possible states of the machine plus contents of the working tape) can be represented by a directed graph of polynomial size, and the difficulty of finding a directed path in that graph is the difficulty of finding a sequence of transitions from the initial state to an accepting state.

For our reductions we will need specific variant of REACHABILITY, that we call LAYERED DAG EXACT REACHABILITY (LDE-REACHABILITY). In this problem, the input graph G is a directed acyclic graph (DAG), which is *layered*, meaning that vertices of a layer only receive inputs of a previous layer and only output to a next layer (but vertices with in-degree zero are not necessarily in the first layer). Also, besides G and s and t, the input considers a positive integer $k \leq |G|$, where $|G|$ is the number of vertices of the input graph. The question is whether there exists a path of length exactly k connecting vertices s and t. We show that this restricted version is also **NL**-Complete.

PROPOSITION 2.2. LDE-REACHABILITY *is* **NL**-*Complete.*

Proof. Let us first consider the problem EXACT-REACHABILITY. This problem receives as input a directed graph G, two vertices s and t and a positive integer $k \leq |G|$, and the question is whether there exists a directed path of length exactly k connecting vertices s and t. Is easy to see that EXACT-REACHABILITY is **NL**-Complete. Indeed, it belongs to **NL** because an algorithm can simply nondeterministically choose the right vertices to follow in a directed path of length exactly k from s to t. The verification of such path can be performed using $\mathcal{O}(\log n)$ simply verifying the adjacency of the vertices in the sequence, and keeping a counter of the length of the path, that uses $\mathcal{O}(\log n)$ space because $k \leq |G|$). Observe also that EXACT-REACHABILITY is **NL**-Complete because, if we have an algorithm $\mathcal{A}$ solving EXACT-REACHABILITY, we can solve REACHABILITY running $\mathcal{A}$ for $k \in \{0, \dots, |G|\}$.

Observe now that LDE-REACHABILITY is in **NL** for the same reasons than EXACT-REACHABILITY. We now show that, LDE-REACHABILITY is **NL**-Complete reducing EXACT-REACHABILITY to it. Let (G, s, t, k) be an instance of EXACT-REACHABILITY. Consider the instance (G', s', t', k) in LDE-REACHABILITY defined as follows. The set of vertices of G' is a set of k copies of $V(G)$, the set of vertices of G. We enumerate the copies from 1 to k, and call them $V_1, \dots, V_k$. Then the set of vertices of G' is $V(G') = V_1 \cup \cdots \cup V_k$. If v is a vertex of G, we call v_i the copy of v that belongs to V_i. There are no edges in G' between vertices in the same copy of V. Moreover, if u, v are two adjacent vertices in G, we add, for each $i \in \{1, \dots, k-1\}$, a directed edge from the i-th copy of u to the $(i+1)$-th copy v in G', formally:

$$(u, v) \in E(G) \iff (u_i, v_{i+1}) \in E(G'), \forall i \in \{1, \dots, k\}.$$

Finally, $s' = s_1$ and $t' = t_k$. By construction we obtain that G' is layered, and moreover (G, s, t, k) is a *yes*-instance of EXACT-REACHABILITY if and only if (G', s', t', k) is a *yes*-instance of LDE-REACHABILITY.

Finally, we remark that we can build the instance (G', s', t', k) in log-space from (G, s, t, k). Indeed, the algorithm has to simply make a counter j from 1 to k, and sequentially connect the vertices of the j-th copy of $V(G)$ with the vertices of the

$j+1$ copy of $V(G)$. We deduce that LDE-Reachability is **NL**-Complete. ∎

2.2. The DLA Model and its Restricted Counterpart. The dynamics of the computational model of DLA are the following: We begin with a sequence of particles which will undergo a random walk starting from a position at the top edge of a $N \times N$ lattice. The sequence specifies the order in which the particles are released. Each particle moves until it neighbors an occupied site at which point it sticks to its position, growing the cluster. We begin with an occupied bottom edge of the lattice. If the particles does not stick to the cluster and leaves the lattice (exiting through the top or lateral edges), it is discarded. A new particle begins its random walk as soon as the previous particle sticks to the cluster or is discarded. This process goes on until we run out of particles in our sequence.

To study this model from a computational perspective, it is convenient to consider a *determinisitic* version, where the sequence of sites visited by each released particle is predefined. The prediction problem presented by Machta and Greenlaw [15] for a d-dimensional DLA is:

DLA-Prediction
Input: Three positive integers: N, M, L, a site p in the two-dimensional lattice of size N^2, a list of random bits specifying M particle trajectories of length L defined by a site on the top edge of the lattice together with a list of directions of motion.
Question: Is site p occupied after the particles have been thrown into the lattice?

For this prediction problem, it is shown in [15] that DLA-Prediction is **P**-Complete. The proof consists of reducing a **P**-Complete variant of the Circuit Value Problem to the prediction problem. Their construction relies heavily on the fact that the particles can move in four directions (Up, Down, Left or Right).

The question we would like to answer is: what happens to the computational complexity of the prediction problem as we restrict the number of directions the particles are allowed to move along. Instead of the four permitted directions, we restrict the particle to move in three (left, right and downwards), two (right and downwards) and one (only downwards) directions.

We call the different directions by d_1 = Down, d_2 = Right, d_3 = Left and d_4 = Up. From this, we define the following class of prediction problems for $k \in \{1,2,3,4\}$:

k-DLA-Prediction
Input: Three positive integers: N, M, L, a site p in the $N \times N$ lattice, a list of random bits specifying M particle trajectories of length L defined by a site on the top edge of the lattice together with a list of directions of motion, where the allowed directions of motions are $\{d_1, ..., d_k\}$.
Question: Is site p occupied after the particles have been thrown into the lattice?

Where DLA-Prediction is the same as 4-DLA-Prediction.

In addition, we ask for the computational complexity of determining whether a given pattern or figure is obtainable through the different biased dynamics. To do this, we codify a pattern on the two-dimensional grid as a binary matrix, where 0 represents an unoccupied site, and 1 represents an occupied one (an example of this can be found in Section 4.2). We define the computational problem as follows:

k-DLA-Realization
Input: A 0-1 matrix M codifying a pattern on the two-dimensional grid.
Question: Does there exist a sequence of particle throws that can move only on the allowed directions of motions, $\{d_1, ..., d_k\}$, whose end figure is represented by M?

Intuitively, this problem is concerned with understanding the complexity of the figures that can be obtained through the different versions of the DLA dynamics, by looking at the computational resources that are required to understand their structure.

To our knowledge, this problem has not been studied from this angle before.

3. DLA-Prediction.

3.1. Two and three directions.

In this section we show that the 2-DLA-Prediction problem is **P**-Complete. This result directly implies that the 3-DLA-Prediction problem is also **P**-Complete.

Theorem 3.1. 2-DLA-Prediction *is* **P**-*Complete.*

Proof. We assume without loss of generality that the two directions in which the particles move are Down and Right. Our proof is an adaptation of the gadgets given by Machta and Greenlaw in [15] for the 4-directional case. The proof consists on using these directions of motion to simulate the evaluation of an instance of the Planar-NOR-Circuit-Value Problem.

In the construction of [15], the different parts of the circuits are simulated by a series of gadgets, representing the different parts of the input Boolean circuit. The gadgets are constructed as a series of particle walks, and the truth values are represented by prescribed locations occupied (or not) by a particle. When a gadget simulates the truth value False, some locations remain unoccupied, meaning that some particles visit these locations but do not get stuck in them. This is realized specifying that, for some particles, the walk consideres a trajectory in two phases. First, the particle follows a given trajectory in order to reach a prescribed destination. Then, if the particle is not sticked, the walk continues in the same trajectory backwards.

In this context, the difference between our constructions and that of [15] is that, in our limited version, the particles are not allowed to return through the same trajectory (in the case that they are not sticked in their destination). Therefore, our main challenge consists in designing a way to discard these particles.

What we must first tackle is the way in which the information is transmitted through the different gadgets. For this purpose, we create wires that transmit the respective truth values. A coordinate occupied by a particle will represent True, and an empty one will represent False. Through this representation, a wire carries the value True if a stack of particles grows along it, while wires that carry the value False remain unoccupied.

The wire is realized by having a pre-assigned particle for each site that makes it up. See Figure 3 for a representation of the gadget and Figure 4 for an example its dynamic. The path of each particle begins at the top of the lattice, and moves straight-down to its assigned location in the lattice. If the value the wire is transmitting is True, the particle will stick at the site. If the value is False, the particle will not stick. The particle is then instructed to move two positions to the right, and then to move indefinitely downwards to be discarded by means of getting stuck to the bottom, effectively transmitting the value of the wire. We note that the particle only realizes this trajectory in the case that it has not stuck, that is, the value transmitted by the wire is FALSE.

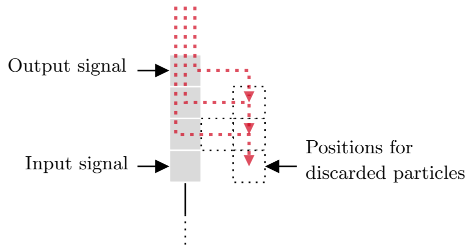


FIG. 3. *Gadget for the simulation of wires. A series of particles are thrown from the column aligned with the input position, following the trajectories depicted with red pointed arrows.*

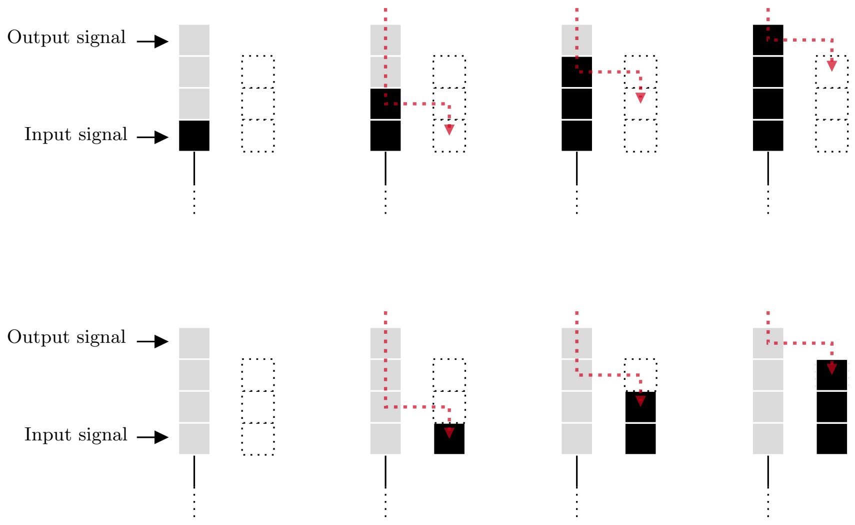


FIG. 4. *Dynamics of the wire when the input is* TRUE *(top) and when the input is* FALSE *(bottom). In the bottom case, the discarded particles are not necessarily stuck in that position and eventually fall further down.*

We must discard the particles transmitting the value FALSE this way because we cannot make use of the upwards direction to do so (through the top of the lattice, as was the case for the original 4-directional construction). This means, that when we finally put the circuit together, each wire must be isolated by a distance of at least four columns from the next, to permit discarding particles. This separation will later guarantee that the discarded particles do not interfere with the evaluation of the circuit: transmitting a FALSE value of length n corresponds to discard a stack of particles, will only reach a height of $n-1$. In a similar way, this construction can be adapted to allow wires that transmit information that turn a signal in a right or left direction, and to multiply signal as well.

We now explain the simulation of the NOR gate. It receives two inputs from the preceding layer. Each of these inputs are grown as mentioned before to the sites a

and b, as shown in Figure 5. It is important to remember that both input cables are separated by a distance of four columns, to allow for the discarding of particles. In addition, we grow a *power cable* to function the gate (it is a wire that always carries the TRUE value). Same as before, site c must be at least four columns away from site b. Once everything is in place, the gate is evaluated as follows (see Figure 6). : A particle makes the journey $a \to b \to c$. If input 1 is TRUE, then the particle will stick at a. The same goes for input 2 and site b. If both inputs are FALSE, the particle will then stick at site c. In any of the 3 cases, a new wire is grown starting from site d. This results in the simulation of a NOR gate.

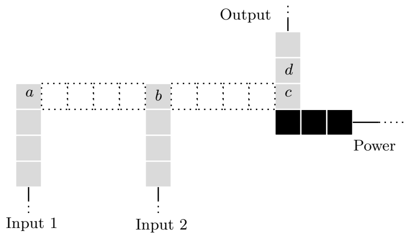


FIG. 5. *Gadget for the simulation of a NOR gate. Both inputs are grown until they reach the sites directly below sites a and b respectively. Two successive particles then follow the path $a \to b \to c$ following the dotted line, stopping according to the inputs. The output is then grown from site c.*

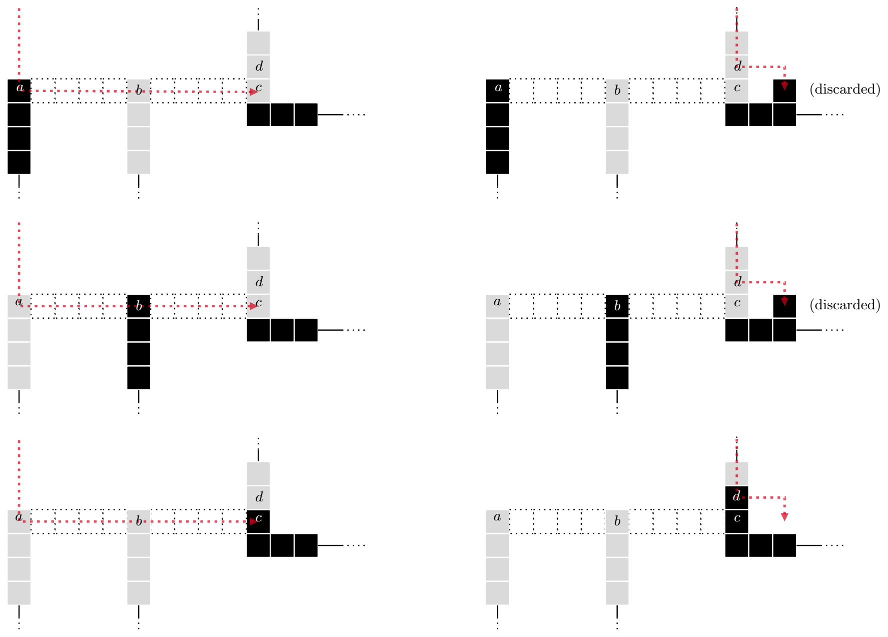


FIG. 6. *Examples of the dynamics of NOR gate when one of the two inputs is* TRUE *(first two lines), and when both inputs are* FALSE *(bottom). In the three lines in the right column is depicted in a red pointed line the trajectory of the particle that follows $a \to b \to c$. Notice that in the first two lines the particle is fixed before reaching position c, while in the third case the particle reaches position c. In the right column is depicted the trajectory of the wire that grows from d, which is only fixed in that position in the third case.*

The power cable is simply a path of occupied cells, growing from the rightmost part of the circuit. Here we find a second difference in our construction with respect the construction of [15]. In the latter, the power cable snakes around the configuration. Starting from the first layer, the power cable grows from right to left, then it grows through the second layer, and continues to grow left to right, and so on. The gates are constructed following the power cable. In our case, the gates also grow following the power cable. The difference is that we grow a new branch of the power cable for each layer of the circuit. The reason for this change is given by the restriction of the movement directions of the particles. Indeed, in our case, the gates of the same layer are evaluated always from right to left.

Once a NOR gate is evaluated, the power cable continues to grow in order to allow the evaluation of the next gate. In Figure 7 it is shown how to grow the power cable in order to cross the information over the NOR gate, independently of its evaluation.

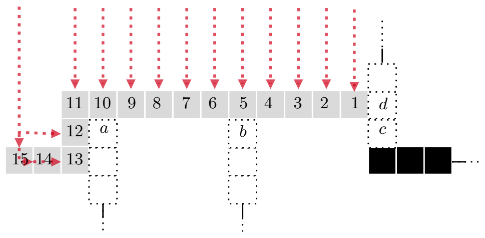


FIG. 7. *Growing the power cable after the evaluation of the NOR gate. One particle is thrown on every column in the order given by the numbers in the corresponding positions. The trajectories are straightforward, except for particles reaching positions 12, 13 and 14, which require a turn. As necessarily after the evaluation of the NOR gate one of a, b or d is occupied, after throwing this 15 particles, a particle is fixed in the last position.*

A given circuit might have gates with outputs in different layers, implying that in the simulation, we may found wires that cross a layer, possibly crossing power cables. A second gate is defined to cope with this problem, which we call *single input OR gate*, following the nomenclature of [15]. The construction will be described with the help of Figure 8. The input wire is grown up to site p. There, two particles make specific trajectories: the first visits $a \to b$, while the second one $c \to d$. After these particles have completed their trajectories, the power cable is grown starting at the site left of d. If the input is true, then the first of the walks will stop at a, and the second at c. The wire is grown from p, and the power cable as mentioned before, crossing the two of them. If the input is negative, then the walks will end at b and d respectively. As before, the wire and the power cable are grown, crossing them. We re-state the importance of the distance between the cables germinated at p, c and a, for the discarding of particles.

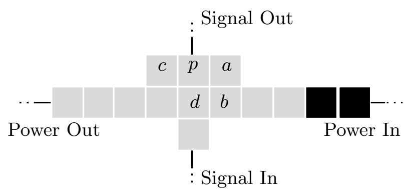


FIG. 8. *Gadget for the simulation of a OR gate. The input is grown up to site p. A first particle then follows the trajectory $a \to b$, stopping according to the truth value of the gadget's input. Then, a second particle makes the trajectory $c \to d$, also stopping according to the truth value of the input. To finalize, the power wire is grown starting from site d, and the output is grown from site p. This OR gate with a single input, in fact, simulates the crossing of the input wire with the power wire.*

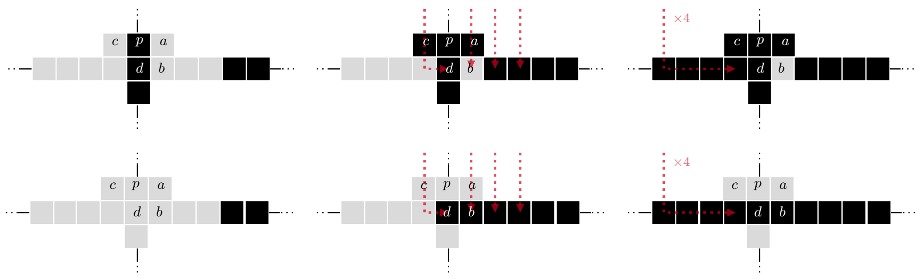


FIG. 9. *Dynamic associated to the OR gate when the input is* TRUE *(top) and* FALSE *(bottom).*

In the following, we give two examples of simple circuits in order to illustrate the resulting reduction.

Example. Let us look at a brief example of how the simulation works. Let us take a circuit consisting of a single NOR gate with inputs labeled x and y. In order to simplify the description, let us represent each particle in our simulation by a tuple $p \in \{1, ..., N\} \times \{D, R\}^*$, where the first coordinate represents the column into which the particle is released and the second is a word that codes the trajectory of the particle with D representing a downwards movement, and R a movement to the right. Following the description of the simulation of a NOR gate given above, the *skeleton* of the construction of this gate is given in Figure 10.

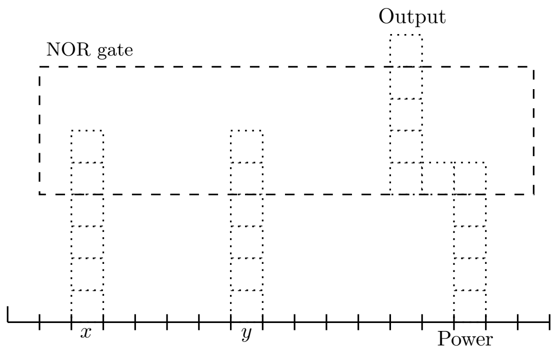


FIG. 10. *Skeleton of the construction of a circuit consisting in only one NOR gate.*

Observe that everything is contained in a 19 by 19 grid. We begin by initializing the truth values of the variables x and y. If $x =$ TRUE, we add the particles $(5, D^{19})$

and $(5, D^{19})$, else we add nothing. Similarly, if y = TRUE we add the particles $(10, D^{19})$ and $(10, D^{19})$, else we add nothing.

For the transmission of the truth values through wires we assign a position in the wire to each particle. For the position $t = (i, j)$ in the lattice, we add the particle p_t given by:

$$p_t = (\underbrace{i, D^j}_{\text{Position}} \circ \underbrace{R^2 D^{N-j}}_{\text{Discard}}).$$

For example, if the variable x has truth value TRUE, we add particles for every position $t_j = (5, 18 - j)$ with $j \in \{1, 2, 3, 4\}$. As we can see in Figure 11 , p_{t_1} will come down up to position t_1 and stick because of the particle at $(5, 18)$. Let us analyze the case when x = FALSE. In this case we add the same particles corresponding to positions t_j, but when they come down, they have nothing to attach to, so they are discarded, as shown in Figure 11.

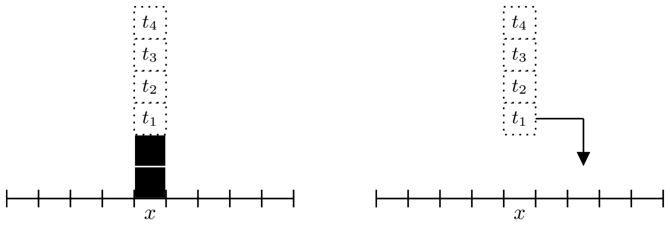


FIG. 11. *Evaluation of the wire transmitting the value of gate x, when x =* TRUE *(right) and x =* FALSE *(left).*

For the power cable, we simple add a particle for each position on the cable, without the suffix $R^2 D^{N-j}$ because they are never discarded. Adding everything up, if Figure 12 we show the state after all the particles have been released in the case where x = TRUE and y = FALSE. Now, we move to the evaluation of the gate. For this purpose we add a particle $P = (3, D^{13} R^{11})$ as was described above for the execution of the NOR gate. In our case, because position $(5, 14)$ will be occupied by the transmitted value from $x = 1$, P will stick at $(5, 13)$. Finally, we transmit the value for the output wire (0 in our particular case) through the same means described before.

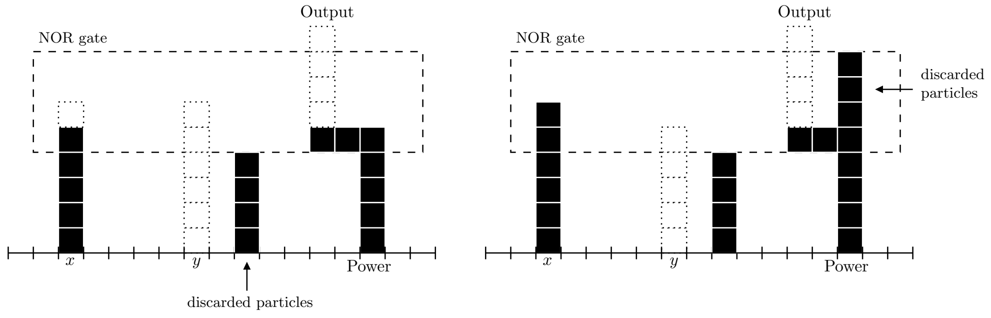


FIG. 12. *Left: the evaluation of the wires transmitting values x =* TRUE *and y =* FALSE*, as well as the power cable. Right: the evaluation of the NOR gate for that case.*

Therefore, our instance of the 2-DLA-PREDICTION problem will be all the particles mentioned along with the output site, which becomes occupied if and only if the corresponding circuit (in this case just one gate) outputs True.

Let us now consider the simulation of a slightly more complex circuit, consisting in three inputs and four NOR gates, as shown Figure 13. In the same figure, we give a schema of the simulation, including the power cables and the single input OR gates. Then, in Figure 14 we give a skeleton of the construction, where we illustrate some of the positions that the particles visit in their trajectories. Finally, in Figure 15 we show the relative order in which the particles are thrown for in the different layers.

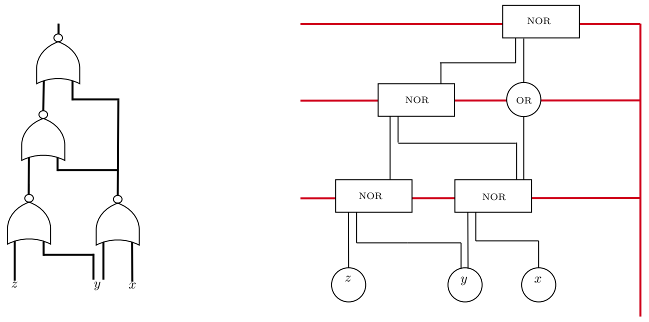


Fig. 13. *A second example of a circuit with tree inputs and two NOR gates. In left is depicted a planar NOR circuit with three inputs and four NOR gates. In right we have a schema of the reduction, where in red is represented the power cables.*

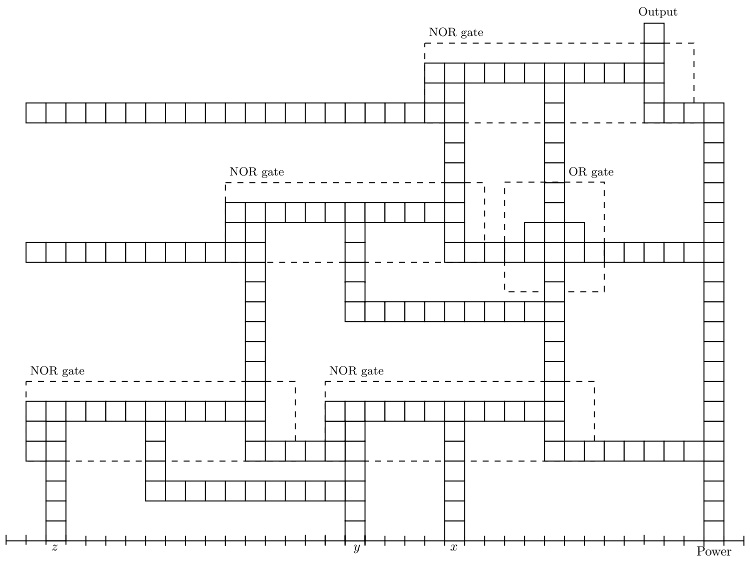
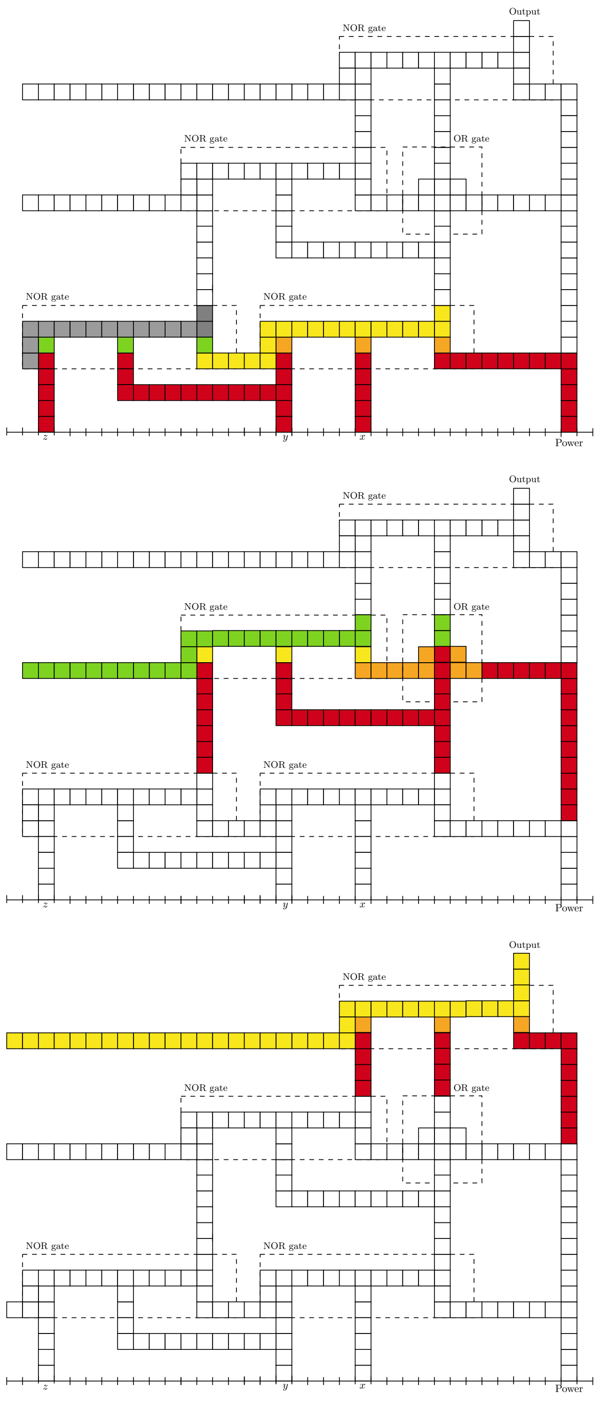


Fig. 14. *Schema of the positions of the particles in a simulation of the circuit given in Figure 13. For simplicity, we omit the positions of the discarded particles.*



□

FIG. 15. *Schema of the order in which the particles are thrown in a simulation of the circuit given in Figure 13, for the gates in the first layer (top left), the second layer (top right) and finally the third layer (bottom). The color code the following order: first red, then orange, then yellow, then green and finally gray. Particles of the same color are thrown from left to right, and/or respecting the order given in the definition gates.*

The reduction described above shows that the an instance of the Planar-NOR-Circuit-Value Problem is correctly evaluated by the growth of the 2-DLA cluster. A planar embedding of a planar circuit can be computed in **NC**. From such an embedding, we can compute the positions of the corresponding gadgets (inputs, gates and wires). Furthermore, the size of each gate is constant. We deduce that this is a **NC** reduction.The key point is that the choice of paths for the walks is independent of the evaluation of the circuit. The full layout of the walks is given globally by the planar layout of the original circuit All calculations required to compute these walks can be performed in **NC**.

Corollary 3.2. 3-DLA-Prediction *is* **P**-*Complete.*

Proof. The particles in this case are allowed to move downwards, to the left and to the right. Thus, the proof of the **P**-Completeness is straightforward. Because we already showed that given two directions, the prediction problem is **P**-Complete we can just ignore one of the lateral directions, and execute the same constructions shown in the previous theorem. ∎

Because the aforementioned proofs rely only on the use of two dimensions, both results are directly extended to the dynamics in an arbitrary number of dimensions:

Corollary 3.3. 2-DLA-Prediction *and* 3-DLA-Prediction *in* $\mathbb{Z}^d$*, are* **P**-*Complete.*

3.2. One Direction. By restricting the directions in which we allow particles to move, our problem statement simplifies. Because particles are only permitted to fall, there is no need to specify the whole trajectory of the particles, just the column of the $N \times N$ lattice we are throwing it down. Therefore, as a first method for representing the given behavior, we describe our input as a sequence of particle drops: $S = a_1 a_2 \dots a_{n-1} a_n$ where each $a_i \in [N]$ represents the column where the i-th particle is dropped, and $[n]$ denotes the set $\{1, \dots, n\}$ for each integer n.

This one-dimensional case is actually a particular instance of a more general model called *Ballistic Deposition* (BD), first introduced by Vold and Sutherland to model colloidal aggregation [24, 25]. The growth model takes the substrate to be an undirected graph $G = (V, E)$, where each vertex defines a column through a "height" function $h : V \to \mathbb{N}$, which represents the highest particle at the vertex. In addition, a sequence of particle throws is given by a list of vertices $S = v_1 v_2 \dots v_{n-1} v_n$, where a particle gets stuck at a height determined by its vertex, and all vertices neighboring it. It is easy to see that the one-dimensional DLA problem on a $N \times N$ square lattice is the special case when $G = ([N], \{(i, i+1) : i \in [N]\})$.

Our prediction problem is as follows:

BD Prediction
Input: A graph $G = (V, E)$, a sequence S of particle throws and a site $t = (h, v) \in \mathbb{N} \times V$, where v is a vertex and h a specified height for it.
Question: Is site t occupied after the particles have been thrown into the graph?

This problem was shown to be in $\mathbf{NC}^2$ by Matcha and Greenlaw using a Minimum-Weight Path parallel algorithm [14]. We improve this result to show that the problem is in fact **NL**-Complete.

3.2.1. Computational Capabilities of the Dynamics. As a first look at the computational capabilities of the model, and sticking to the 1-DLA version of Ballistic

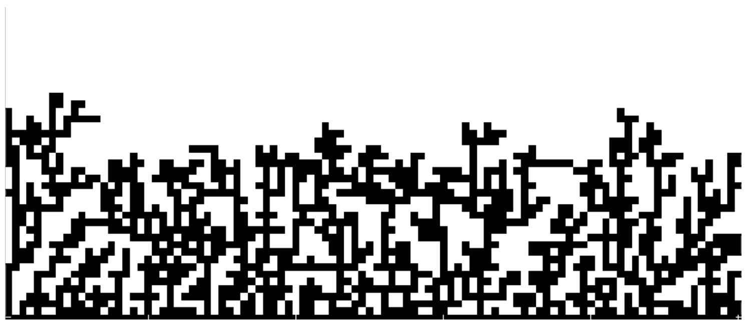


FIG. 16. *A realization of the one-directional dynamics of the system on a one-dimensional strip.*

Deposition, we show that we can sort natural numbers, simulating The Bead-Sort model described by Arulanandham et al. [3]. This model consists of sorting natural numbers through gravity: numbers are represented by beads on rods, like an abacus, and are let loose to be subjected to gravity. As shown in [3], this process effectively sorts any given set of natural numbers. It is reasonable to think that because of the dynamics and constraints of our model (one direction of movement for the particles), the same sorting method can be applied within our model, which is in fact the case.

LEMMA 3.4. *Bead-Sort can be simulated.*

Proof. Let A be a set of n positive natural numbers, with m being the biggest number in the set. We create a $2m \times 2m$ lattice where we will be throwing the particles. Here the k-th rod from the Bead-Sort model is represented by row $2k - 2$ on our lattice. Now, for each number $a \in A$ we create the sequence $S_a = 2\ 4\ 6 \ldots 2a$. The total sequence of launches S is created by concatenating all S_a for $a \in A$. We note that because of the commutativity of our model, the order in which the concatenation is made is not relevant. Thus, throwing sequence S into our lattice, effectively simulates the Bead-Sort algorithm. ◻

Let us give an example using the set $A = \{7, 4, 1, 10\}$. Following the proof, we must simulate 1-DLA on a 20×20 square lattice, and create the sequences $S_1 = 2$, $S_4 =$2 4 6 8, $S_7 =$ 2 4 6 8 10 12 14, and $S_{10} =$ 2 4 6 8 10 12 14 16 18 20. By releasing the sequence $S = S_1 \circ S_4 \circ S_7 \circ S_{10}$ into the lattice we obtain Figure 17, which is ordered increasingly, effectively sorting set A.

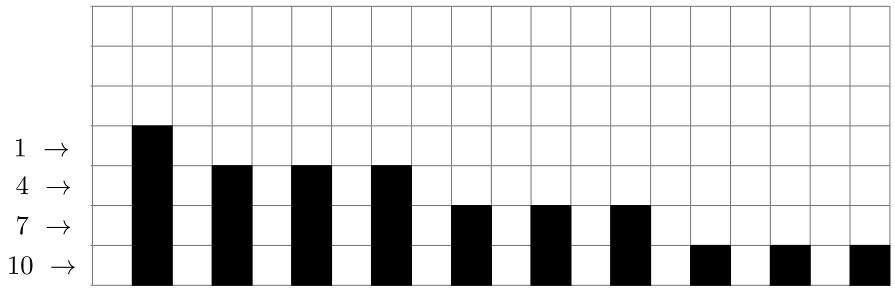


FIG. 17. *Figure obtained by trowing sequence S into the lattice. Set A is ordered decreasingly from the first row onwards.*

A key aspect of the present model is the commutativity of throws through nonconsecutive vertices of the graph. Given a figure, $\mathcal{F}$, we define $\varphi_v(\mathcal{F})$ as the figure that results after throwing a particle through vertex v. We notice, that because of the dynamics of our model, the point at which a given particle freezes is determined uniquely by the state of the vertex it has been dropped in, and the state of its neighbors. Therefore, given $v \in V$:

$$(\varphi_v \circ \varphi_u)(\mathcal{F}) = (\varphi_u \circ \varphi_v)(\mathcal{F}) \quad \forall u \notin N_G(v).$$

We will later use this fact to create a better algorithm for the prediction problem.

3.2.2. Non-deterministic Log-space Algorithm. There is a critical aspect of the dynamics that is exploitable to create an algorithm: if two particles are thrown on non-adjacent vertices, their relative order in the input sequence is reversible. By using this, our aim is to shuffle the sequence into another sequence with the same final configuration, but ordered in a way that allows us to quickly solve the prediction problem. Specifically, if we are able to reorganize the input sequence into one that releases particles according to the height they will ultimately end at, the remaining step to solve the problem is checking amongst the particles for the target height, if the target vertex appears.

Let S be the input sequence of BD PREDICTION. Formally, a sequence $S = s_1 \dots s_n$ is composed of the vertices onto which each particle will be released. From S, we are able define a sequence of particles $p_1, \dots, p_n$ that represents the same realization as the input sequence. We define a *particle* p, as a triple $(V(p), \mathrm{num}(p), \mathrm{pos}(p)) \in V \times [n] \times [n]$, where the first coordinate, $V(p)$, denotes the vertex onto which the particle is thrown, the second coordinate, $\mathrm{num}(p)$, is an integer representing the number of particles thrown onto vertex $V(p)$ before p, and the third, $\mathrm{pos}(p)$ is the position of the particle within sequence S. The particle description of S is easily obtained by setting $V(p_i) = s_i$, $\mathrm{num}(p_i) = |\{j \in [n] : s_j = s_i \wedge j \leq i\}|$, and $\mathrm{pos}(p_i) = i$.

Let us call $\mathbb{P} = \{p_1, \dots, p_n\}$ the *set of particles* of S.

To further break down the problem, we define the following sets:

$$A(p) := \{q \in \mathbb{P} : \mathrm{pos}(q) < \mathrm{pos}(p)\},$$

$$N(p) := \{q \in A(p) : V(q) \in N_G(V(p)) \cup \{V(p)\}\},$$
$$N^{=}(p) := \{q \in N(p) : V(q) = V(p)\}.$$

In words, $A(p)$ denotes the set of particles thrown before p, $N(p)$ denotes the set of particles that are thrown before p on vertices adjacent to p, and $N^{=}(p)$ denotes the subset of particles that belong to $N(p)$ and are in the same vertex as p.

For a particle $p \in \mathbb{P}$, the *row of* p, denoted $\mathrm{row}(p)$ is the height at which the particle ends up at after the dynamics have taken place. In other words, $\mathrm{row}(p) = h(V(p))$ after releasing the sequence $S' = s_1 \dots s_{\mathrm{pos}(p)}$. Relative to this definition, we call N^r the set of particles thrown before p in vertices adjacent than p that stick at row r, formally:

$$N^r(p) = \{q \in N(p) : \mathrm{row}(q) = r\}.$$

We translate the dynamics into this new notation in the following lemma:

LEMMA 3.5. *Let $p \in \mathbb{P}$ be a particle and let*

$$r = \begin{cases} 1 & \text{if} \quad N(p) = \emptyset, \\ \max\{\mathrm{row}(q) : q \in N(p)\} & \text{if} \quad N(p) \neq \emptyset. \end{cases}$$

Then,

$$\mathrm{row}(p) = \begin{cases} r+1 & \textit{if} \quad N^r(p) \cap N^=(p) \neq \emptyset, \\ r & \textit{if} \quad N^r(p) \cap N^=(p) = \emptyset. \end{cases}$$

Explicitly, if the particle is the first of its neighbors to be thrown ($N(p) = \emptyset$), its row is 1. If not, that is, if its neighbors are higher than the vertex it is thrown in ($N^r(p) \cap N^=(p) = \emptyset$), the particle sticks at their height. Lastly, if the vertex that the particle is thrown in is higher that its neighbors ($N^r(p) \cap N^=(p) \neq \emptyset$), the particle sticks one row higher than the last particle in the vertex.

Proof. Let p be a particle such that $N(p) = \emptyset$. This implies that p is the first particle thrown through vertex $V(p)$ and its adjacent vertices. From the commutativity property, we deduce that $\mathrm{row}(p) = 1$. On the other hand, $r = 1$ and $N^r(p) \cap N^=(p) = \emptyset$, so $\mathrm{row}(p) = r$.

Suppose now that $N(p) \neq \emptyset$, and let q be a particle in $N^r(p)$. Observe that $\mathrm{row}(q) = r$, and $\mathrm{row}(u) \leq r$ for all $u \in N(p)$. Then, when p is thrown, the first particle that it encounters is q. Suppose that we can pick q such that $V(q) = V(p)$ (i.e. $N^r(p) \cap N^=(p) \neq \emptyset$). Since $V(q) = V(p)$, we deduce that $\mathrm{row}(p) = \mathrm{row}(q) + 1 = r + 1$. On the other hand, if $N^r(p) \cap N^=(p) = \emptyset$ then the coordinate $(V(p), r)$ is empty when p is thrown, but some of (u, r), for $u \in N_G(V(p))$, are occupied. We deduce that $\mathrm{row}(p) = \mathrm{row}(q) = r$. ◻

We create a weighted graph that codifies the dependence of the particles to each other. Let G_S be a weighted directed graph defined from S as follows: the vertex set of G_S is the set of particles $\mathbb{P}$ plus one more vertex g, called the *ground* vertex. The edges of G_S have weights, given by the weight function W defined as:

$$W(p,q) = \begin{cases} 1 & \text{if } \ (p = g) \wedge (N(q) = \emptyset), \\ 1 & \text{if } \ p \in N^=(q), \\ 0 & \text{if } \ p \in N(q) \setminus N^=(q), \\ -\infty & \text{otherwise.} \end{cases}$$

Observe that if we keep only the edges with weight different than $-\infty$, the obtained graph has no directed cycle, i.e. it is a directed acyclic graph. Moreover, the set of incoming edges of vertex p is $N(p)$ if $N(p) \neq \emptyset$, and $\{g\}$ otherwise. For $p \in \mathbb{P}$, we call $\tilde{\omega}_{gp}$ the longest (maximum weight) path from g to p in G_S.

THEOREM 3.6. *For every $p \in \mathbb{P}$,* $\mathrm{row}(p) = \tilde{\omega}_{gp}$.

Proof. We reason by induction on $\mathrm{pos}(p)$. Let $p \in \mathbb{P}$ be the particle such that $\mathrm{pos}(p) = 1$. Observe that p has only one incoming edge, which comes from g, and $W(g, p) = 1$. Then $\tilde{\omega}_{gp} = 1 = \mathrm{row}(p)$.

Suppose now that $\mathrm{row}(p) = \tilde{\omega}_{gp}$ for every particle q such that $\mathrm{pos}(q) \leq k$ and let p be a particle with $\mathrm{pos}(p) = k + 1$. If $N(p) = \emptyset$, then, like in the base case, the only incoming edge of p is g, and from Lemma 3.5 we deduce that $\mathrm{row}(p) = 1 = \tilde{\omega}_{gp}$.

Suppose now that $N(p)$ is different than $\emptyset$. Let q be a particle in $N(p)$ such that $\mathrm{row}(q)$ is maximum, i.e. $\mathrm{row}(q) = \max\{\mathrm{row}(u) : u \in N(p)\}$. Observe that, from induction hypothesis and the choice of q, $\tilde{\omega}_{gq} \geq \tilde{\omega}_{gu}$ for all $u \in N(p) \setminus \{q\}$. Moreover, $\tilde{\omega}_{gp} \leq \tilde{\omega}_{gq} + 1$.

Suppose that q can be chosen to be such that $V(q) = V(p)$. Lemma 3.5 then implies that $\mathrm{row}(p) = \mathrm{row}(q) + 1$. On the other hand, the path from g to p that passes through q is of weight $\tilde{\omega}_{gq} + 1$. We deduce that $\tilde{\omega}_{gp} = \tilde{\omega}_{gq} + 1 = \mathrm{row}(q) + 1 = \mathrm{row}(p)$.

Suppose now that for all $u \in N^=(p)$, $\mathrm{row}(u)$ is strictly smaller than $\mathrm{row}(q)$. In this case, Lemma 3.5 implies that $\mathrm{row}(p) = \mathrm{row}(q)$. On the other hand, the path from

g to p that passes through q is of weight $\tilde{\omega}_{gq}$, which is greater or equal than $\tilde{\omega}_{gu}$, for all $u \in N(p) \setminus N^{=}(p)$ and strictly greater than $\tilde{\omega}_{gu}$, for all $u \in N^{=}(p)$. We deduce that $\tilde{\omega}_{gp} = \tilde{\omega}_{gq} = \text{row}(q) = \text{row}(q)$. □

We now present an **NL**-algorithm for the prediction problem.

Given a site (h, v), we want to non-deterministically obtain a path through graph G_S that will guarantee the site will be occupied by a particle. Each step of our algorithm we will non-deterministically guess a pair consisting of the next particle, and the corresponding weight of the transition between the last particle and the new one, such that the sum of the weights is the maximum weight to the final particle. We say a particle p is valid for an input sequence S, if $p \in \mathbb{P}$. The following log-space algorithm verifies if a particle is valid.

Algorithm 3.1

```
Input: A sequence $S$ and a particle $p \in V \times [n] \times [n]$
Output: Accept if particle $p$ is valid.
Check if $V(p) = s_{\text{pos}(p)}$
Sum ← 0
for $i \in \{1, ..., \text{pos}(p)\}$ do
    if $s_i = V(p)$ then
        Sum ← Sum + 1
    end
end
if Sum = num($p$) then
    Accept
else
    Reject.
end
```

At each step of the for loop of the algorithm, we must remember the value of Sum, the particles vertex, $V(p)$, and the current index of the iteration. This amounts to using $\mathcal{O}(\log(n))$ space.

We also present a log-space algorithm to determine wether the obtained transition weight corresponds to the value of the weight function

Algorithm 3.2

```
Input: A sequence $S$, two valid particles $p, q \in V \times [n] \times [n]$, and $w \in \{0,1\}$
Output: Accept if $W(p,q) = w$.
if $p = g$ then
    for $i < \text{pos}(q)$ do
        Check that $s_i$ is not adjacent to $V(q)$
    end
    Accept
end
if $w = 1$ then
    Check that $\text{pos}(p) < \text{pos}(q)$ and $V(p) = V(q)$
    Accept
end
if $w = 0$ then
    Check that $\text{pos}(p) < \text{pos}(q)$
    Check that $V(p) \neq V(q)$
    Check that $V(p)$ is adjacent to $V(q)$
    Accept
end
Reject
```

For this algorithm, the only case where information needs to be stored is when $p = g$. For this instance, each iteration of the for loop must remember the index of the iteration, and the vertex $V(q)$. Therefore, this algorithm uses $\mathcal{O}(\log(n))$ space.

Combining these two subroutines, we are now ready to present the main algorithm.

Algorithm 3.3 NL algorithm for BD Prediction

```
Input: A graph $G = (V, E)$, a sequence $S$ and a site $t = (x, v) \in \mathbb{N} \times V$
Output: Accept if a particle occupies site $t$ and reject otherwise.
Non-deterministically obtain $m$, the number of particles, and $p_1$. Write them down.
Check if $p_1$ is valid and that $W(g, p_1) = 1$
Sum $\leftarrow 1$
Write down Sum.
for $j \in \{2, ..., m\}$ do
    Non-deterministically obtain particle $p_j$ and the transition weight $w_{j-1,j}$, and
      write them down.
    Check if $p_j$ is valid.
    Check if $W(p_{j-1}, p_j) = w_{j-1,j}$
    Sum $\leftarrow$ Sum + $w_{j-1,j}$
    Erase $p_{j-1}$ and $w_{j-1,j}$
end
if Sum $= x$ and $V(p_m) = v$ then
    Accept
else
    Reject
end
```

At the j-th step of this algorithm, we must retain the following information: the sum of the weights so far, particles p_{j-1} and p_j, the current weight $w_{j-1,j}$. This means that around $4\log(n) = \mathcal{O}(\log(n))$ space is used on the tape.

Proposition 3.7. BD-Prediction *is in* **NL**.

Proof. Let us show that Algorithm 3.2.2 decides BD Prediction. Let S be an input sequence and $P = (h, v)$ an input site.

If the release of S on to the underlying graph results on site P being occupied, by Theorem 3.6 we know that there exists a particle $q \in \mathbb{P}$ such that $h = \text{row}(q) = \tilde{\omega}_{gq}$ and $V(q) = v$. Let $C = g\ p_1\ p_2\ ...\ p_{m-1}\ p_m$ be the maximum weight path of weight $\tilde{\omega}_{gq}$, where $p_m = q$. Then, for the j-th non-deterministic choice the algorithm makes, it obtains the pair: p_j and $W(p_{j-1}, p_j)$.

If the algorithm accepts for S and P, we will obtain a sequence of particles such that the sum of the transition weights is exactly h and that $V(p_m) = v$. This means that the weight from the ground to the last particle will indeed be $h = \tilde{\omega}_{gp_m}$. Due to Theorem 3.6, this means that $\text{row}(p_m) = \tilde{\omega}_{gp_m} = h$. Therefore, particle p_m indeed occupies site P. ∎

Theorem 3.8. BD-Prediction *is* **NL**-*Complete.*

Proof. Given proposition 2.2, in order to show that the problem is **NL**-Hard, we will reduce an instance of LDE-Reachability to the Ballistic Deposition problem.

Let (G, s, t, k) be an instance of LDE-Reachability, where m is the number of layers of G, and let $i \in [m]$ be the index such that $s \in V_i$. The idea is to throw two particles for each vertex in all layers from i to $i + k$. This way, the height at which the particles freeze will increase with each layer. Formally, for every $i < j \leq i+k$ and every $u \in V_j$ we create the sequence $S_u = uu$ (two particles are thrown in vertex u). Concatenating these sequences, we obtain a sequence of throws on the whole layer $S_j = \bigcirc_{u \in V_j} S_u$. We note that due to the structure of the graph and the commutativity of the dynamics, the order in which these sequences are concatenated does not matter because no two vertices in the same layer are adjacent.

Finally, our input sequence will be the concatenation of the sequences associated to every layer from the i-th layer to the $i+k$-th one, $S = S_s \circ \bigcirc_{j=i+1}^{i+k} S_j$. The order in which these sequences are concatenated is important, and must be done in increasing order according to their index. At any point in this process the only information retained is the vertex for which we are currently creating the sequence S_u. This only requires $\log(n)$ space to store, making this process a log-space reduction.

By defining the site $P = (k + 1, t)$, we create an instance of BD-Prediction: (G, S, P). Let us prove that this is indeed a reduction.

If $(G, s, t, k) \in$ LDE-Reachability, then there exists a directed path $C = v_0\ ...\ v_k$ in G, where $s = v_0$ and $t = v_k$. Because of the layered structure of the graph, if $s \in V_i$ then $v_j \in V_{i+j}$. Then, because by construction, for every vertex on C two particles will be dropped, the height of the last particle dropped in v_j will be $j+1$, meaning that the last particle dropped on $v_k = t$ will have a height of $k+1$. This means that site P will in fact be occupied, and therefore $(G, S, P) \in$ BD-Prediction.

If $(G, S, P) \in$ BD-Prediction, site $P = (k + 1, t)$ is occupied after the sequence of particles, S, has been released onto G. Due to our construction, if site P is occupied, site (k, t) must also be occupied by a particle. Let $l \in [m]$ be such that $t \in V_l$. Because only two particles are thrown at each vertex, for the latter site to be occupied there must exist a vertex $v_1 \in V_{l-1}$ adjacent to t such that sites (k, v_1) and $(k-1, v_1)$ are occupied.

Iterating this process, we obtain a sequence $v_1 \dots v_{k-1}$ such that v_i is adjacent to v_{i-1} and sites $(k-i+1, v_i)$ and $(k-i, v_i)$ are occupied, for every $i \in \{2, \dots, k-1\}$. By virtue of the construction of S, the only posible way in which site $(1, v_{k-1})$ is occupied is that $s = v_{k-1}$. This proves that $(G, s, t, k) \in$ LDE-Reachability, concluding our proof. ∎

4. Shape Characterizations and Realization.

Although the figures obtained by simulating the dynamics are complex and fractal-like, not every shape is obtainable as an end product. This naturally leads to the problem of characterizing the figures which are obtainable through the dynamics, for the different restrictions of the DLA model.

A crucial observation is the fact that not all connected shapes are realizable. Figure 18 shows shapes that are not achievable for each of the restricted versions.

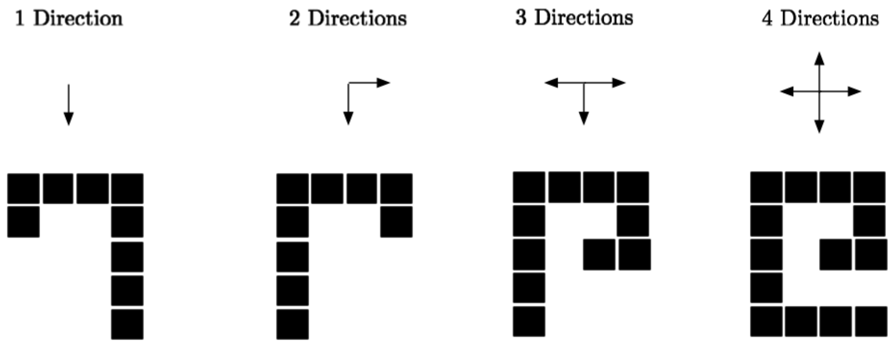


Fig. 18. *Four non-constructible figures with the respective maximum number of directions where it is not constructible. The last figure is not constructible even with four directions.*

Given a fixed number of allowed directions, determining whether a given figure is realizable is an **NP** problem, where the non-deterministic choices of the algorithm are the order in which the particles are released. Having the order, it is possible to compute the trajectories that each particle takes in polynomial time, by finding a path that does not neighbor the already constructed cluster, from the place at which it is released to its final destination.

We give better algorithms for the shape characterization problems for both 1-DLA and 2-DLA, showing that the former belongs to the class of log-space solvable problems, **L**, and the latter to **P**.

4.1. One Direction.

To characterize shapes created by the one-directional dynamics, given a sequence of drops S, and its corresponding shape $\mathcal{F}(S) \in \{0,1\}^{m \times n}$, we construct a planar directed acyclic graph (DAG), $G = (V, E)$, that takes into account the ways in which a shape can be constructed with the dynamics. We construct G in two steps.

We begin by constructing $G_a = (V_a, E_a)$, where:

$$\begin{aligned}
V_a &= \{ij : \mathcal{F}(S)_{ij} = 1\} \cup \{g\},\\
E_1 &= \{(ij, ij+1) : \mathcal{F}(S)_{ij} = \mathcal{F}(S)_{ij+1} = 1\},\\
E_2 &= \{(ij, ij-1) : \mathcal{F}(S)_{ij} = \mathcal{F}(S)_{ij-1} = 1\},\\
E_3 &= \{(ij, i+1j) : \mathcal{F}(S)_{ij} = \mathcal{F}(S)_{i+1j} = 1\},\\
E_4 &= \{(nj, g) : \mathcal{F}(S)_{nj} = 1\},
\end{aligned}$$

and $E_a = E_1 \cup E_2 \cup E_3 \cup E_4$. The intuition is the following: we create a vertex for each block in the shape and one representing the ground where the initial particles stick. E_1 and E_2 account for the particles that stick through their sides, E_3 accounts for particles falling on top of each other and finally E_4 connects the first level to the ground.

For example, given the sequence $S = 2\ 7\ 7\ 2\ 6\ 3\ 4\ 4\ 4\ 5\ 6\ 3\ 2\ 6\ 2$, we depict the obtained shape and corresponding graph in Figure 19.

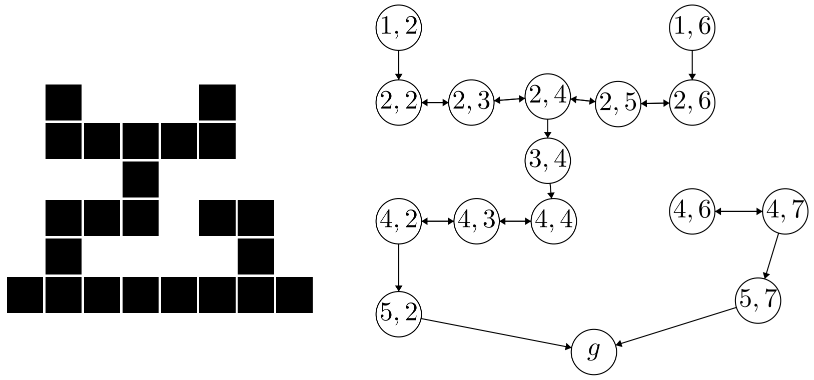


FIG. 19. *Shape and graph obtained from sequence $S = 2\ 7\ 7\ 2\ 6\ 3\ 4\ 4\ 4\ 5\ 6\ 3\ 2\ 6\ 2$.*

LEMMA 4.1. *A configuration is constructible iff $\forall ij \in V_a \setminus \{g\}$ there exists a directed path between ij and g.*

Proof. Given a constructible configuration, by virtue of the definition, there is a sequence S of particle drops that generates the configuration. Therefore, by the dynamics of our system and the construction of the graph, for each node in our graph, there exists a directed path to the *ground*, represented by node g.

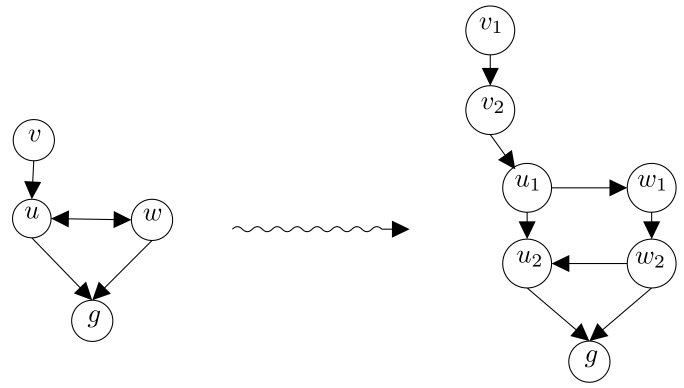
Now, let $G = (V, E)$ be the graph corresponding to a given configuration on the grid. If we have that $\forall ij \in V \setminus \{g\}$ there exists a directed path between ij and g, we start by reversing the direction of the arcs in our graph, and running a Breadth-first search-like algorithm starting from node g, to determine the minimum distance from g to each of the other nodes. For all nodes with distance 1 from g, $\{i_k j_k\}_{k=1}^{n}$, we create the sequence $S_1 = j_1 ... j_n$. Then for all non-visited nodes, with distance d from g, $\{a_k b_k\}_{k=1}^{m}$, we create the sequence $S_d = b_1\ ...\ b_m$. Let us show that the sequence $S = S_1 \circ S_2 \circ ... \circ S_M$, with $M = \max\{\text{dist}(ij, p) : ij \in V \setminus \{g\}\}$ corresponds to the configuration. We do this by induction over $|S| = n$.

For $n = 1$, we have only one occupied state on our configuration, which is the only particle present in S. It is straightforward to see that $\mathcal{F}(S)$ actually corresponds to the configuration.

Assuming the sequence is correct for n, let us prove that it is also correct for $n+1$. Because of the way S was constructed, the only possibility for the configuration not to be achieved is that S_{n+1} ends up higher or lower in $\mathcal{F}(S)$ on its column that in the given configuration. Due to the dynamics, the only way for a particle to become inmobile is to stick to another particle, this means that either it sticks to a particle in its own column, or to a particle in one of the neighboring ones. By the induction hypothesis, $S \setminus S_{n+1}$ actually corresponds to the configuration of the first n particles. Therefore, if, without loss of generality, S_{n+1} ends up lower, this means that when generating the sequence S a node that was at a smaller distance from g than the particle with which it sticks, was added after all other nodes that at the same distance, which is a contradiction. ∎

Next, we obtain G from G_a through the following procedure:

1. For every $v \in V_a \setminus \{g\}$, we create two vertices v_1 and v_2, connected by an arc from the former to the latter.
2. For every $e = (v, u) \in E_1$, we create an edge (v_1, u_1).
3. For every $e = (v, u) \in E_2$, we create an edge (v_2, u_2).
4. For every $e = (v, u) \in E_3$, we create an edge (v_2, u_1).
5. For every $e = (v, g) \in E_4$, we create an edge (v_2, g).



This procedure turns G_a into a planar DAG.

It is straightforward to see that there is a directed path from a vertex $v \in V_a$ to g on graph G_a if and only if there is a path from vertex v_1 to g on graph G.

Due to our construction, graph G is what is known as a Multiple Source Single Sink Planar DAG (MSPD): there are multiple vertices with in-degree zero, and one vertex with out-degree zero. Allender et al. showed that the reachability problem on MSPD is in fact log-space solvable [1].

THEOREM 4.2 ([1], Theorem 5.7). *MSPD reachability is in* **L**.

PROPOSITION 4.3. *Determining whether a given figure is a valid configuration for 1-DLA, is in* **L**.

The proof of this proposition consists on creating a log-space reduction from our realization problem to MSPD reachability. This stems from the fact that if a problem is log-space reducible to a log-space solvable problem, it is itself log-space solvable (the proof of this fact can be found in [2]).

Proof. Let M be a matrix representing the configuration, using the same convention as the definition shape. We can see that the directed graph is constructible from a shape in $\mathbf{NC}^1$. By assigning a processor for each pair of coordinates in the matrix, that is $\mathcal{O}(n^2)$ processors, to construct the nodes and arcs of our graph. Because $\mathbf{NC}^1 \subseteq \mathbf{L}$, this procedure is realizable in log-space. This procedure creates the MSPD G.

For each vertex $v \in V$, we can solve the MSPD reachability problem for the instance (G, v, g) in log-space. Due to Lemma 4.1, this solves the one-directional realization problem.

4.2. Two directions. To characterize shapes generated by the two-directional dynamics, we follow a similar approach than we did for one-directional dynamics. We define for each figure a directed graph called the *dependency graph*, which has as vertex set the set of all particles in the input figure F. The dependency graph satisfies that, if a particle u is an in-neighbor of a particle v, then u must be fixed before v on every realization of the figure. Moreover, we show that the dependency graph characterizes the realizable figures. More precisely, we show that a figure is realizable if and only if its is acyclic.

Let M be the input matrix of an instance of 2-DLA-Realization. For simplicity, in this section we assume that M is a matrix of dimensions $[N+1] \times [N]$ where all the coordinates in row $N+1$ have value 1 (representing the *ground*). Moreover, we assume that N is large enough so M has a border of zeroes. More precisely, every coordinate of M that equals 1 and that is not in row $N+1$, is in a row and column of M that is greater than 3, and in a column fewer than $N-3$. More formally, we assume that $M = (m_{ij})$ satisfies:

$$i \leq 3 \vee j \leq 3 \vee j \geq N-3 \Rightarrow m_{ij} = 0.$$

We will say a coordinate $(i,j) \in [N] \times [N]$ is a *particle* if $M_{ij} = 1$. The set of all particles is called a *figure* F. We denote the row and column of a particle $p \in F$ by $\mathrm{row}(p)$ and $\mathrm{col}(p)$ respectively. Two particles p and q are said to be *adjacent* if $|\mathrm{col}(p) - \mathrm{col}(q)| + |\mathrm{row}(p) - \mathrm{row}(q)| = 1$. We call n the total number of particles, i.e. $n = |F|$.

Definition 4.4. *Let $F \subseteq \{0,1\}^{N \times N}$ be a figure. A 2DLA-realization of F is a bijective function $\varphi : F \to [n]$ such that, for every $t \in [n]$:*

- *Each particle is fixed to an adjacent particle already fixed, i.e., $\varphi^{-1}(t)$ has an adjacent particle in $\varphi^{-1}(\{1, \dots, t-1\})$ or $\mathrm{row}(\varphi^{-1}(t)) = N$.*
- *There exists an* available path *for $\varphi^{-1}(t)$, that is to say, a path $P = p_0, \dots p_k$ such that*
  - *The path starts in $(0,0)$ and reaches p_k, i.e. , $p_0 = (0,0)$ and $p_k = \varphi^{-1}(t)$,*
  - *The particle is not fixed before its final destination, i.e., $0 < \ell < k$, cell p_ℓ is not adjacent to any cell in $\varphi^{-1}(\{1, \dots, t-1\})$ and $\mathrm{row}(p_\ell) < N$.*
  - *The path reaches the cell using only two directions, i.e, for each $0 \leq \ell < k$ cell $p_{\ell+1} = (\mathrm{row}(p_\ell) + 1, \mathrm{col}(p_\ell))$ or $p_{\ell+1} = (\mathrm{row}(p_\ell), \mathrm{col}(p_\ell) + 1)$*

Let us better understand how the relative position of particles affect the order in which they must be placed. For a particle $u \in F$, we define the *shadow* of the particle as:

$$S(u) = \{(i,j) : \mathrm{row}(u) > i \ \wedge \ \mathrm{col}(u) > j\}$$

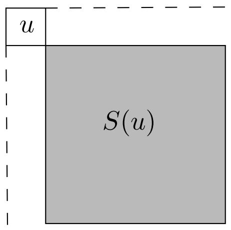


FIG. 20.

DEFINITION 4.5. *Let F be a 2-DLA constructible figure. A realization φ of F is said to be canonical if*

$$\forall u, v \in F : \quad u \in S(v) \implies \varphi(u) < \varphi(v).$$

The following lemma states that every figure that is realizable admits a canonical realization.

LEMMA 4.6. *F admits a 2-DLA realization if and only if F has a canonical realization.*

Proof. If F has a canonical realization it is trivially 2-DLA constructible.

Let us suppose that F is 2-DLA constructible but has no canonical realization. We will say a pair of particles u, v is a *bad pair* of a realization φ if $\varphi(u) > \varphi(v)$ and $u \in S(v)$. Let φ be the realization with the least amount of bad pairs. We will show that it is possible to define a realization with a lower amount of bad pairs, giving us a contradiction.

From all possible bad pairs (v, u), let us take one that minimizes $\text{col}(v)$. Fixing such v, let us take u as the first cell to be placed in $S(v)$ after v. In order to reach a contradiction, we consider two cases.

Case 1: Let us suppose that u is strictly in the interior of $S(v)$, that is, $\text{col}(u) > \text{col}(v) + 1$ and $\text{row}(u) > \text{row}(v) + 1$ as see on Figure 21.

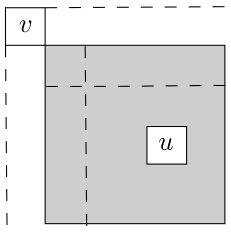


FIG. 21. *Case 1: u is in the interior of $S(v)$.*

For $w \in F$ let us call $F_w(\varphi)$ the set of particles such that

$$F_w(\varphi) = \{z \in F : \varphi(z) \leq \varphi(w)\}$$

It is clear that in this case, we must have that u must be adjacent to a particle in $F_v(\varphi)$. If not, we would have a contradiction with the fact that u is the fist element to be placed after v and that means it would be placed without a contact point.

We now define $\tilde{\varphi}$ as the realization that equals φ, except that we place u right before v, more formally:

$$\tilde{\varphi}(p) = \begin{cases} \varphi(p) & \text{if } \ \varphi(p) < \varphi(v) \\ \varphi(v) & \text{if } \ p = u \\ \varphi(p) + 1 & \text{if } \ \varphi(v) \leq \varphi(p) < \varphi(u) \\ \varphi(p) & \text{if } \ \varphi(p) > \varphi(u) \end{cases}$$

Let us see that $\tilde{\varphi}$ is a realization. Let w be any particle of F. Let p_w denote the trajectory the particle takes to reach cell w in the realization given by φ. Clearly if $\varphi(w) < \varphi(v)$ or $\varphi(w) > \varphi(u)$ then $F_w(\varphi) = F_w(\tilde{\varphi})$ and then p_w is still available as a trajectory for the new realization $\tilde{\varphi}$. If $w = u$ then $F_w(\tilde{\varphi}) \subseteq F_w(\varphi)$ and then P_w is also available as a trajectory for the new realization $\tilde{\varphi}$. Suppose then that $\varphi(v) \leq \varphi(w) < \varphi(u)$. If u is not adjacent to any of the coordinates on p_w (for example when $w = v$), this path is also valid as a trajectory for the new realization $\tilde{\varphi}$. We can observe that this is always the case: if $w \notin S(v)$, p_w can never be adjacent to u. If $w \in S(v)$, because this means that $\varphi(u) > \varphi(w)$, u cannot be adjacent to p_w.

Therefore, $\tilde{\varphi}$ is a realization of F. Moreover, $\tilde{\varphi}$ has a lower number of bad pairs. Indeed, on one hand (v, u) is no longer a bad pair of $\tilde{\varphi}$. On the other, suppose that (w_1, w_2) is a bad pair of $\tilde{\varphi}$ that is not a bad pair of φ. Then necessarily $w_1 = u$ and $w_2 \in S(u)$ is such that $\varphi(v) < \varphi(w_2) < \varphi(u)$. We obtain a contradiction with the fact that $S(u) \subset S(v)$ and u was the first particle in $S(v)$ to be placed after v. We deduce that $\tilde{\varphi}$ is a realization of F with a lower number of bad pairs than φ, which is a contradiction with the choice of φ.

Case 2: Let us now suppose that u is in the border of $S(v)$, that is, $\text{col}(u) = \text{col}(v) + 1$ or $\text{row}(u) = \text{row}(v) + 1$. Without loss of generality let us assume that $\text{col}(u) = \text{col}(v) + 1$.

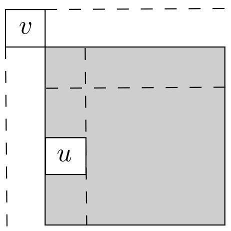


FIG. 22. *Case 2: when u is in the border of v* ($\text{col}(u) = \text{col}(v) + 1$)

Let p_u denote the trajectory the particle takes to reach cell u in the realization given by φ. We have that p_u must necessarily contain a coordinate (i, j) such that $j = \text{col}(v)$ and $i < \text{row}(v)$. Let $C(\varphi; u)$ the connected component of $[N] \times [N] \setminus (S(v) \cup p_u)$ that contains $z = (\text{col}(v), N)$.

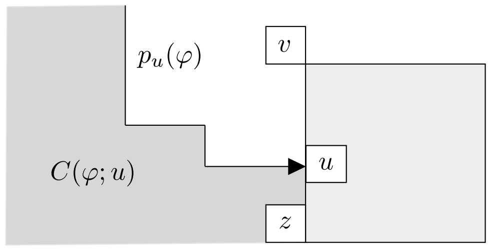


FIG. 23. *Construction of* $C(\varphi; u)$

Now let us define the set $C(u, v)$ as the set of particles that are placed in $C(\varphi; u)$ after v and before u according to φ. More precisely:

$$C(u, v) = \{w \in P(C(\varphi; u)) : \varphi(v) < \varphi(w) < \varphi(u)\},$$

Finally, let us define the realization $\tilde{\varphi}$ as follows:

- First set all elements in $F_v \setminus \{v\}$preserving the order given by φ.
- Then, set all the elements of $C(u, v)$ preserving the order given by φ.
- Then, set u followed by v.
- Finally set the rest of the elements of F preserving the order given by φ.

Once again let us prove that $\tilde{\varphi}$ is in fact a realization of F. Let $w \in F$ be a particle. If $\tilde{\varphi}(w) < \varphi(v)$ or $\tilde{\varphi}(w) > \varphi(u)$, then it is clear that p_w is still available in $\tilde{\varphi}$, because $F_w(\tilde{\varphi}) = F_w(\varphi)$. For the particles $w \in C(u, v)$ we have that $F_w(\tilde{\varphi}) \subseteq F_w(\varphi)$ and then the path p_w is also available in $\tilde{\varphi}$. The same argument goes for $w = u$.

For the remaining options, let us define

$$X = \{w \in F : \varphi(v) \leq \varphi(w) < \varphi(u)\} \setminus C(u, v).$$

This set is not empty because it contains v. Let us take $w \in X$, and suppose that no coordinate in p_w is adjacent to $C(u, v)$. In this case p_w is also available in $\tilde{\varphi}$. Suppose then that p_w intersects $C(u, v)$. Since w is not contained in $C(u, v)$, necessarily p_w intersects p_u (because p_u is the frontier of the component $C(\varphi; u)$). Let $y \in p_u \cap p_w$ be the coordinate of the last time p_u intersects p_w. Then define the trajectory $\tilde{p}_w$ of w in $\tilde{\varphi}$ as p_1 and p_2, where p_1 equals to p_u from the top of the grid up until reaching y and p_2 is equal to p_w from y until reaching w. We obtain that $\tilde{p}_w$ is an available trajectory for w in the realization $\tilde{\varphi}$. We deduce that $\tilde{\varphi}$ is a realization of F.

To reach a contradiction, now we show that $\tilde{\varphi}$ has a lower number of bad pairs than φ. Obviously (v, u) is not a bad pair of $\tilde{\varphi}$. Let us suppose that there is a bad pair (p, q) for $\tilde{\varphi}$, that is not a bad pair for φ. The only possibility for this to happen is that

$$q \in C(u, v) \cup \{u\} \quad \text{and} \quad p \in \{z \in F : z \notin C(u, v) \ \wedge \ \varphi(v) \leq \varphi(z) < \varphi(v)\}.$$

In this case we would obtain a bad pair for φ where $\text{col}(q) < \text{col}(v)$, which is not possible due to the choice of v. We conclude that in this case, $\tilde{\varphi}$ is a realization that has strictly less number of bad pairs than φ, which is a contradiction with the existence of φ. ◻

This Lemma allows us to better structure the proof that 2-DLA constructible figures can be characterized through canonical realizations. Given a particle $p \in F$, we define the particles scope as

$$L(p) = \{(i, j) : \text{row}(p) < i \ \wedge \ \text{col}(p) < j\} = \{q \in F : p \in S(q)\}.$$

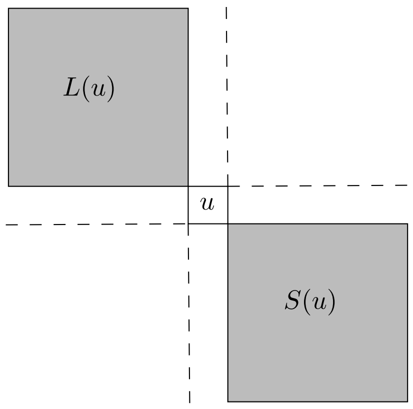


FIG. 24. *The shadow $S(v)$ and scope $L(v)$ of a particle v*

With this at hand, we reinterpret Lemma 4.6 as follows: a figure F is 2DLA-Realizable if and only if there exists a canonical realization φ, that is a realization such that, for every $u \in F$:

$$\varphi(z) < \varphi(u) < \varphi(x),\ \forall z \in S(u),\ \forall x \in L(u).$$

In the following we only focus on canonical realizations. Because of previous result, when constructing a figure and want to add a particle $x = (i, j)$ we can guarantee that there will be an interrupted path from the top of the grid to the coordinate $(i-2, j-2) \in L(x)$.

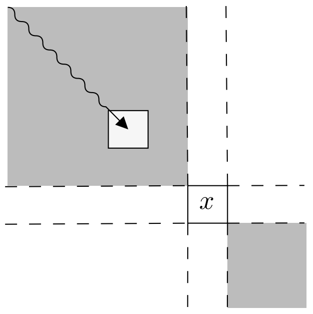


FIG. 25. *We assume that for every occupied site $x = (i, j)$, when a particle is placed in x there is an available path that starts in $(1, 1)$ and reaches $(i-2, j-2)$.*

This means that, to build a realization of a figure, it is sufficient to select one that satisfies that, for each particle (i, j), there is a path from $(i-2, j-2)$ to (i, j) that avoids the collision with other particles already fixed. For a particle $x \in F$, we define $V(x)$ the vicinity of x as the set of sites depicted on Figure 26. More formally, if $x = (i, j)$ then $V(x) = V_1(x) \cup V_2(x) \cup V_3(x) \cup V_4(x)$ where

$$V_1((i,j)) = \{(i-3,j),(i-2,j),(i-1,j),(i-2,j+1),(i-1,j+1)\},$$

$$V_2((i,j)) = \{(i,j-3),(i,j-2),(i,j-1),(i+1,j-2),(i+1,j-1)\},$$

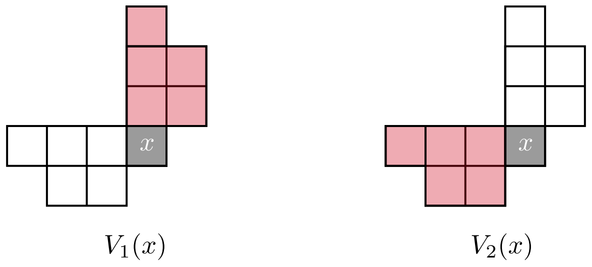


FIG. 26. *Visual representation of $V(x)$ and sets $V_1(x)$ and $V_2(x)$ highlighted in red*

Due to the nature of the dynamics, we have to consider only four different possible paths a particle can take to reach x from $y = (\text{row}(x) - 2, \text{col}(x) - 2)$. We denote these paths as Q_1, Q_2, Q_3 and Q_4, which are shown in the following figure.

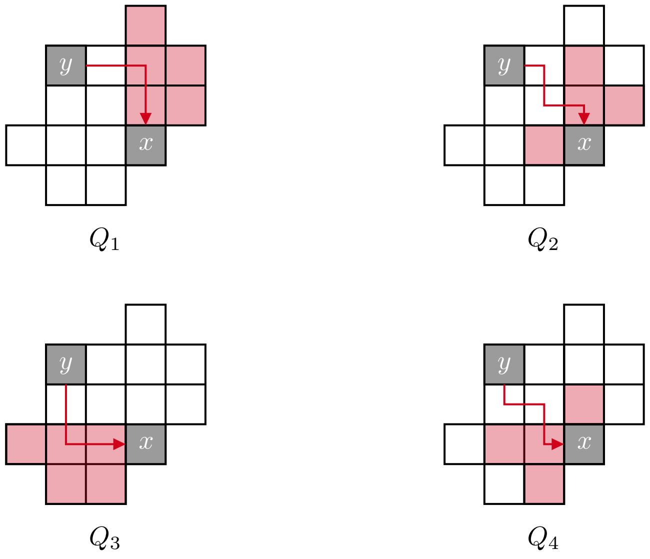


FIG. 27. *A representation of the four choices of paths starting from $y = (i-2, j-2)$ and reaching coordinate $x = (i, j)$.*

More formally, in Table 1 are given the four possible choices of paths starting from $(i-2, j-2)$ and reaching coordinate (i, j), with the list of visited cells and the cells adjacent to each path. For $k \in \{1, 2, 3, 4\}$ we call $Q_k((i, j))$ the union of the set of visited and adjacent cells of path $Q_k((i, j))$. Given a realization φ of F, we say that a path Q_k is *available* if all cells c in the second of and third columns of the k-th row of the table are satisfy that $\varphi(c) > \varphi((i, j))$. Otherwise, we say that Q_k is *unavailable.*

Path	Sequence of visited cells	Adjacent cells in $V(x)$
$Q_1((i,j))$	$(i-2,j-2),(i-2,j-1),$ $(i-2,j),(i-1,j)$	$(i-3,j),(i-2,j+1),$ $(i-1,j+1)$
$Q_2((i,j))$	$(i-2,j-2),(i-2,j-1),$ $(i-1,j-1),(i-1,j)$	$(i-2,j),(i-1,j+1),$ $(i,j-1)$
$Q_3((i,j))$	$(i-2,j-2),(i-1,j-2),$ $(i,j-2),(i,j-1)$	$(i,j-3),(i+1,j-1),$ $(i+1,j-1)$
$Q_4((i,j))$	$(i-2,j-2),(i-1,j-2),$ $(i-1,j-1),(i,j-1)$	$(i,j-2),(i+1,j-1),$ $(i-1,j)$

TABLE 1
Definition of possible choices of paths starting from $(i-2,j-2)$ and reaching coordinate (i,j), with the list of visited cells and the cells adjacent to each path. In a realization, we say that a path Q_i is available *if all cells in the second of and third columns of the i-th row are empty.*

Remark 4.7. There exist two other possible choices for paths reaching cell (i,j) from cell $(i-2,j-2)$, however, namely the paths $(i-2,j-2),(i-2,j-1),(i-1,j-1),(i,j-1),(i,j)$ and $(i-2,j-2),(i-1,j-2),(i-1,j-1),(i-1,j),(i,j)$. However, these paths require have more available cells than Q_2 and Q_3, respectively. Therefore, any trajectory that uses any of such paths have Q_2 or Q_4 available.

4.2.1. The dependency graph. We are now ready to define the *dependency graph* of a figure. The dependency graph H of a figure F is a directed graph satisfying that, if $u,v \in F$ and $(u,v) \in E(H)$, then u must be fixed before v on every canonical realization of v. To precisely define the set of edges, we apply a number of applications of a set of *rules*, described below.

First, let us denote Ground as the set $\{N+1\} \times [N]$, that is to say, the set of particles in the bottom of the figure. The vertex set of the dependency graph H is $F \cup \mathsf{Ground}$. The construction starts with the set of edges

$$E_0 = (\mathsf{Ground} \times F \setminus \mathsf{Ground}) \cup \{(u,v) \in F \times F : u \in S(v)\}.$$

We start this way because all the edges in the bottom must be fixed (are fixed from the beginning) before all particles in F. Moreover, as we are considering only canonical realizations, and all the particles in $S(v)$ are fixed before v. Then, we iteratively add edges to E_0 according to two rules defined below, obtaining this way sets $E_1, E_2, \dots$. The process stops when no new edges can be with any of the two rules. The resulting set is the set of edges of H.

For $\ell \geq 0$ and a node $v \in V(H)$, let us consider the sets

$$N(v) = \{(\text{row}(v), \text{col}(v) \pm 1), (\text{row}(v) \pm 1, \text{col}(v))\} \cap V(H)$$

$$D^-_\ell(v) = \{w \in V(H) : (w,v) \in E_\ell\},$$
$$D^+_\ell(v) = \{w \in V(H) : (v,w) \in E_\ell\},$$

which are called, respectively, the sets of *adjacent nodes* and the sets of *predecessors* and *successors* of v with respect to edge set E_ℓ. Given E_ℓ, we construct $E_{\ell+1}$ adding edges to E_ℓ according to the following two rules.

The first rule is used to force that a particle that is not adjacent to the ground must have at least one adjacent particle that is potentially fixed before it. This rule is applied over vertices $v \in F$ such that $D^-_\ell(v) = \emptyset$. First, let us call $R_\ell(v)$ the set of neighbors of v defined by:

$$R_\ell(v) = N(v) \setminus \left(\bigcup_{w \in D^+_\ell(v)} D^+_\ell(w) \cup D^+_\ell(v) \right)$$

In words, $R_\ell(v)$ is the set of particles adjacent to v that are not successors of v, neither successors of its successors. If $R_\ell(v) = \emptyset$, then all neighbors of v must be fixed after v, and therefore the figure is not realizable. To represent that, we add edge (v, v) to $E_{\ell+1}$. Now suppose that $R_\ell(v)$ is nonempty and suppose that there exists $u \in R_\ell(v)$ such that $R_\ell(v) \setminus (D^+_\ell(u) \cap \{u\}) = \emptyset$. Then, necessarily u must be a predecessor of v. We conclude the following definition of **Rule 1**.

Definition 4.8 (**Rule 1).** *For each $v \in F$ such that $D^-_\ell(v) = \emptyset$:*

- *If $R_\ell(v) = \emptyset$, then add edge (v, v) to $E_{\ell+1}$.*
- *If $R_\ell(v) \neq \emptyset$ and there exists $u \in R_\ell(v)$ satisfying that $R_\ell(v) \setminus (D^+_\ell(u) \cup \{u\}) = \emptyset$ then we add (u, v) to $E_{\ell+1}$.*

The second rule states that, when a particle is fixed there must exist an available path reaching v from $(\text{row}(v) - 2, \text{col}(v) - 2)$. This implies that we must fix v before the cells that block all the available paths of v. More precisely, for a particle $v \in F$, we call $B_\ell(v)$ the set of currently unavailable paths of v, formally:

$$B_\ell(v) = \{k \in \{1, 2, 3, 4\} : Q_k(v) \cap D^-_\ell(v) \neq \emptyset\},$$

The set of available paths of v is $A_\ell(v) = \{1, 2, 3, 4\} \setminus B_\ell(v)$. Now let us define $\mathcal{I}_\ell(v)$ as the set of particles in the intersection of all available paths, formally

$$\mathcal{I}_\ell(v) = \bigcap_{k \in A_\ell(v)} Q_k(v).$$

Then, **Rule 2** states that all particles in $\mathcal{I}_v$ must be fixed after v. We deduce the following definition.

Definition 4.9 (**Rule 2).**
For each $v \in F$:

- *If $A_\ell(v) = \emptyset$, then add edge (v, v) to $E_{\ell+1}$.*
- *If $u \in \mathcal{I}_v$, then we add (v, u) to $E_{\ell+1}$.*

Therefore, the construction of the dependency graph consists in the applications of **Rule 1** and **Rule 2** until no new edges are created. Since a figure contains n particles, there are at most n^2 possible edges between them. Therefore, the construction finalizes in at most n^2 applications of the rules. Observe also that each rule can be applied in polynomial time. We formalize previous construction in Algorithm 4.1.

Algorithm 4.1 Constructing the edges of the dependency graph of figure F

```
Input: A figure $F$ represented by a set of coordinates in $[N] \times [N]$
Output: A set of edges $E$
$\mathsf{Ground} \leftarrow (\{N+1\} \times [N])$ // the cells in the bottom
 $E_0 = \leftarrow \{(u,v) : u \in S(v)\} \cup (\mathsf{Ground} \times (F \setminus \mathsf{Ground}))$
for $u = (i,j) \in F$ do
    $N(u) \leftarrow \{(i+1,j),(i-1,j),(i,j+1),(i,j-1)\} \cap (F \cup \mathsf{Ground})$
    $Q_1(u) = \{(i-3,j),(i-2,j),(i-1,j),(i-2,j+1),(i-1,j+1)\} \cap (F \cup \mathsf{Ground})$
    $Q_2(u) = \{(i-2,j),(i-1,j),(i-1,j+1),(i,j-1)\} \cap (F \cup \mathsf{Ground})$
    $Q_3(u) = \{(i,j-3),(i,j-2),(i,j-1),(i+1,j-2),(i+1,j-1)\} \cap (F \cup \mathsf{Ground})$
    $Q_4(u) = \{(i,j-2),(i,j-1),(i+1,j-1),(i-1,j)\} \cap (F \cup \mathsf{Ground})$
end
for  $\ell \in \{1, \dots n^2\}$ do
    $E_{\ell+1} \leftarrow E_\ell$
    for $x \in F$ do
        $D^+(x) \leftarrow \{w \in F : (x,w) \in E_\ell\}$
        $D^-(x) \leftarrow \{w \in F : (w,x) \in E_\ell\}$
        $R_\ell(x) \leftarrow N(x) \setminus \left(\bigcup_{w \in D_\ell^+(x)} D_\ell^+(w) \cup D_\ell^+(x)\right)$
        $B_\ell(x) \leftarrow \{k \in \{1,2,3,4\} : D^-(x) \cap Q_k(x) \neq \emptyset\}$
        $A_\ell(x) \leftarrow \{1,2,3,4\} \setminus B_\ell(x)$.
        $\mathcal{I}_\ell(x) \leftarrow \bigcap_{k \in A_\ell(x)} Q_k(x)$
    end
    for $v \in F$ do
        // Rule 1
        if $D_\ell^-(v) = \emptyset$ and $R_\ell(v) = \emptyset$ then
            $E_{\ell+1} \leftarrow E_{\ell+1} \cup \{(v,v)\}$
        end
        if $D_\ell^-(v) = \emptyset$ and $R_\ell(v) \neq \emptyset$ then
            for $u \in R_\ell(v)$ do
                if $R_\ell(v) \setminus (D^+(u) \cup \{u\}) = \emptyset$ then
                    $E_{\ell+1} \leftarrow E_{\ell+1} \cup \{(u,v)\}$
                end
            end
        end
        // Rule 2
        if $A_\ell(v) = \emptyset$ then
            $E_{\ell+1} \leftarrow E_{\ell+1} \cup \{(v,v)\}$
        else
            for $u \in F \cup \mathit{Ground}$ do
                if $u \in \mathcal{I}_v$ then
                    $E_{\ell+1} \leftarrow E_{\ell+1} \cup \{(v,u)\}$
                end
            end
        end
    end
end
return $E_{n^2} \setminus (\mathsf{Ground} \times F \setminus \mathsf{Ground}))$
```

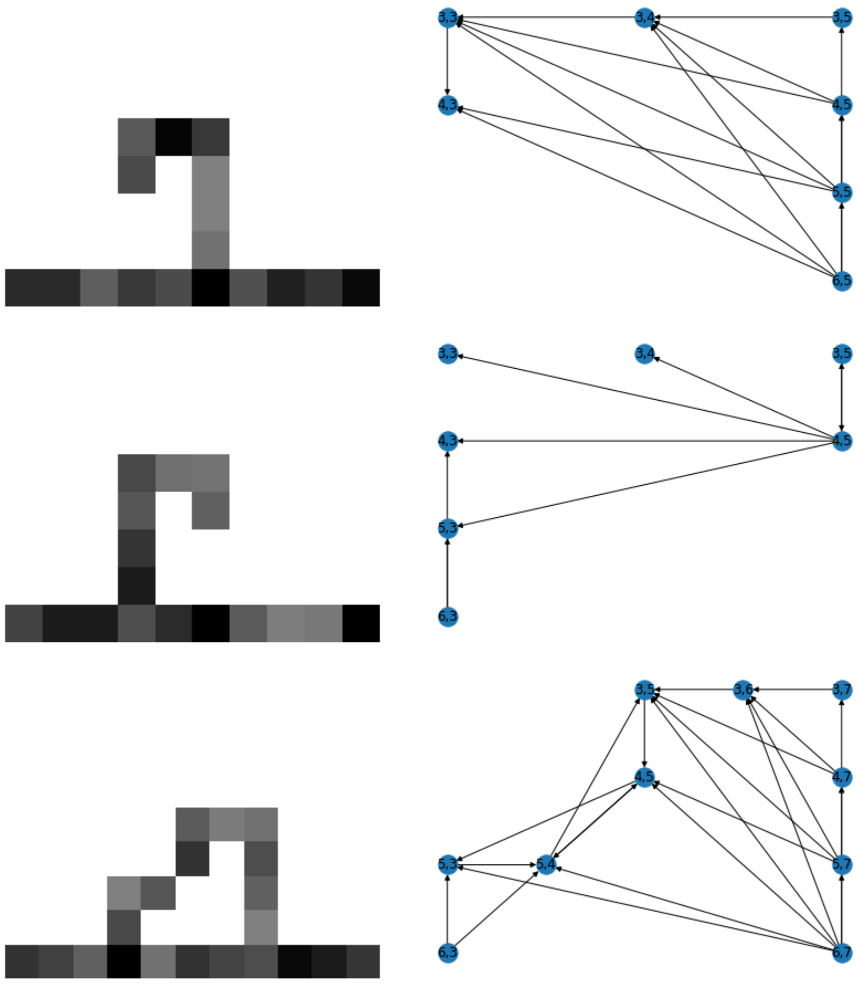


FIG. 28. *Three examples of figures and their corresponding dependency graphs. Observe that the middle and bottom figures are not 2DLA-realizable, and their dependency graphs are not acyclic.*

DEFINITION 4.10. *The* dependency graph *of a figure F is the graph with vertex set F and set of edges obtained as the output of Algorithm 4.1 on input F.*

In Figure 28 is depicted an example of a figure F and its corresponding dependency graph.

LEMMA 4.11. *Let $H = (F, E)$ be the dependency graph of a 2DLA-realizable figure F. For every pair of particles $u, v \in F$, if $(u, v) \in E$ then u is fixed before v on every canonical realization of F.*

Proof. We prove the result by induction on ℓ, showing that every edge in E_ℓ have the above property. The base case $\ell = 0$ is trivial because E_0 contains only the edges (u, v) such that $u \in S(v)$. Now suppose that for a given $\ell \geq 0$ all the edges in (u, v) satisfy that u is fixed before v on every canonical realization of F. If $E_{\ell+1} = E_\ell$ then $E(H) = E_\ell$ and we are done. Therefore, suppose that $E_{\ell+1}$ contains an edge (u, v) not in E_ℓ and assume that there exists a canonical realization φ that fixes v before u.

We will reach a contradiction by showing that (u, v) can not created with **Rule 1** or **Rule 2**.

First, as v can fixed before u in φ, there must exist a neighbor $w \in N(v)$ that is fixed before v and u. From the induction hypothesis, this w must be contained in $R_\ell(v)$ and moreover, $w \notin D^+_\ell(u) \cup \{u\} \cup D^+_\ell(v)$. Then (u, v) was not created according to **Rule 1**. Second, since u can be fixed after v in φ, there must exist an available path $k \in A_\ell(u)$ such that $v \notin Q_k(u)$. We deduce that (u, v) can not be created according to **Rule 2**. We deduce that, if (u, v) is an edge of $E_{\ell+1}$ then u must be fixed before v on every canonical realization of F. □

Our approach consists in constructing a canonical realization of F by removing one by one the particles of F, applying the following recursive algorithm. First, if $|F| = 1$, i.e., the input figure consists in a single particle, the algorithm returns a realization consisting in this single particle. If $|F| > 1$, we compute the set of edges E of the dependency graph of F. Then, we look for a *removable particle* u, defined as follows.

Definition 4.12. *A removable particle is a particle satisfying the following conditions.*

- $C(u) = \{k \in \{1, 2, 3, 4\} : (F \cup \textsf{Ground}) \cap Q_k(u) = \emptyset\} \neq \emptyset$, *meaning that there is at least one available path u not touching any particle in $F \cup$ Ground.*
- $D^+(u) = \{w \in F : (t, w) \in E\} = \emptyset$, *meaning that no other particle needs to be fixed after u.*
- *The dependency graph of $F \setminus \{u\}$ is acyclic.*

If no such particle u is found, the algorithm rejects. Otherwise, we recursively run the algorithm in $F \setminus \{u\}$. Finally, if the recursive call of the algorithm does not reject, the output is a realization $\tilde{\varphi}$ of $F \setminus \{u\}$. The realization of F is obtained by fixing u after all the particles in $F \setminus \{u\}$ according to $\tilde{\varphi}$, obtaining this way a canonical realization φ. The pseudocode of algorithm is written in Algorithm 4.2.

Lemma 4.13. *Let F be a figure with an acyclic dependency graph H, then H contains a removable particle.*

Proof. Let F be a figure such that its corresponding dependency graph H is acyclic. Let W be the set of particles with out-degree zero in H, formally $W = \{v \in F : D^+(v) = \emptyset\}$. This set is non-empty as H is acyclic. Now let us pick the particle in $u \in W$ such that $\mathrm{col}(u)$ is minimum, and over all possible choices, we pick the one such that $\mathrm{row}(u)$ is maximum. In other words, u is the bottom-left most particle in W.

We are going to show that u is a removable particle, by showing a series of claims. We define before the following four sets relatively to u:

$$S_{NE}(u) = \{w \in F : \mathrm{row}(w) \leq \mathrm{row}(u) \text{ and } \mathrm{col}(w) \geq \mathrm{col}(u)\} \setminus \{u\}$$
$$S_{NW}(u) = \{w \in F : \mathrm{row}(w) < \mathrm{row}(u) \text{ and } \mathrm{col}(w) < \mathrm{col}(u)\}$$
$$S_{SE}(u) = \{w \in F : \mathrm{row}(w) > \mathrm{row}(u) \text{ and } \mathrm{col}(w) > \mathrm{col}(u)\}$$
$$S_{SW}(u) = \{w \in F : \mathrm{row}(w) \geq \mathrm{row}(u) \text{ and } \mathrm{col}(w) \leq \mathrm{col}(u)\} \setminus \{u\}$$

Observe that $S_{NW}(u)$ is the scope of u and $S_{SE}(u)$ is the shadow of u. Observe that the scope of u is empty, because u has out degree zero in H.

Algorithm 4.2 Constructing the realization of F

```
Input: A figure $F \subseteq [N] \times [N]$ of size $n$ with an acyclic dependency graph.
Output: A realization of $F$.
if $|F| = \{u\}$ then
    return a list $\varphi$ containing $u$ as a single element.
else
    Compute $E$ as the output of Algorithm 4.1 on input $F$
    for $x \in F$ do
        $D^+(x) \leftarrow \{w \in F : (x, w) \in E\}$
        $C_x \leftarrow \{k \in \{1, 2, 3, 4\} : F \cap Q_k(x) \neq \emptyset\}$
    end
    $\alpha \leftarrow \{u \in F : C_x \neq \emptyset\}$
    $\beta \leftarrow \{u \in F : D^+(u) = \emptyset\}$
    $\gamma \leftarrow \alpha \cap \beta$
    for $u \in \gamma$ do
        Compute $E'$ as the output of Algorithm 4.1 on input $F \setminus \{u\}$
        if  $H = (F \setminus \{u\}, E')$ contains a cycle then
            $\gamma \leftarrow \gamma \setminus \{u\}$
        end
    end
    Pick any $u \in \gamma$
    $\tilde{\varphi} \leftarrow$ output of Algorithm 4.2 on input $F \setminus \{u\}$
    $\varphi \leftarrow$ list $\tilde{\varphi}$ adding $u$ in the end.
    return $\varphi$
end
```

Claim 1: All nodes in $S_{SW}(u)$ are ancestors of u in H. Formally, for every $w \in S_{SW}(u)$ there is a w, u-directed path in H.

Proof of Claim 1. Let w_0 an arbitrary particle in $S_{SW}(u)$. From the choice of u, we know that w_0 has at least one out-neighbor in H. Let $P = w_0, w_1, \ldots, w_k$ be a longest directed path in H starting from w_0. Then w_k has out degree zero, implying that it is not contained in $S_{SW}(u)$. Let w_ℓ, with $0 < \ell \leq k$ be the first node in P not contained in $S_{SW}(u)$. If $w_\ell = u$ we are done. If w_ℓ belongs to $S_{SE}(u)$ then $(w_ell, u) \in E(H)$ and $P' = w_0, \ldots, w_\ell, u$ is a directed path in H. Finally, suppose that w_ℓ belongs to S_{NE}. Then necessarily $(w_{\ell-1}, w_\ell)$ is created by **Rule 2** and w_ℓ belongs to $V_1(w_{\ell-1})$. Observe that u is also contained in $V_1(w_{\ell-1})$ and then $(w_{\ell-1}, u)$ is also an edge created by **Rule 2**. We conclude that $P'' = w_0, \ldots, w_{\ell-1}, u$ is a directed path of H.

Claim 2: u has an available path in F, i.e. $C(u) \neq \emptyset$.

Proof of Claim 2. Suppose that $Q_3(u)$ intersects F and pick $w \in Q_3(u) \cap F$. Observe that $w \in S_{SW}(u)$. Therefore, by **Claim 1** there is a directed path from w to u, and moreover, (w, u) is an edge in $E(H)$. Then, by **Rule 2** there is an edge connecting u with all the particles in $V_1(u)$. Since u has out-degree zero, necessarily $V_1(u) \cap F$ is empty. We deduce that $Q_1(u)$ is an available path for u and then $1 \in C(u)$. We conclude that $\{1, 3\} \cap C(u) \neq \emptyset$.

Claims **1** and **2** imply that u satisfies the first two conditions of the definition of removable particle. It remains to show that the dependency graph of $F \setminus \{u\}$ is acyclic. In the following we call $E(H-u)$ the dependency graph of $F \setminus \{u\}$.

Claim 3: $E(H) \setminus (F \times \{u\})$ is a subset of $E(H-u)$.

Proof of Claim 3. Suppose that $E(H) \setminus (F \times \{u\})$ contains an edge $e = (w_1, w_2)$ not in $E(H-u)$. From all possible choices, let us pick the one that is created in the earliest iteration of Algorithm 4.1. Observe that e can not be created by **Rule 1**. Indeed, if $R_\ell(w_2)$ does not contains u then necessarily e is also an edge of $E(H-u)$, and if $R_\ell(w_2)$ contains u then necessarily u also belongs to $D^+(w_1)$ (u must be different than w_1 because u has out-degree zero). In both cases we have that if e is created by **Rule 1**, then e must be an edge of $E(H-u)$. Then necessarily e is created with **Rule 2**. However, as u has out-degree zero, it cannot block any path of another node. We deduce that e can not exist and then $E(H) \setminus (F \times \{u\}) \subseteq E(H-u)$.

Claim 4: Let $e = (w_1, w_2)$ be an edge of $E(H-u) \setminus E(H)$. Then $w_1 \neq w_2$ (i.e. e is not a self-loop), and w_1 belongs to $S_{NE}(u)$. Moreover, if w_2 also belongs to $S_{NE}(u)$, then w_2 is closer to u than w_1 in Manhattan distance, i.e. $|\mathrm{row}(w_2) - \mathrm{row}(u)| + |\mathrm{col}(w_2) - \mathrm{col}(u)| < |\mathrm{row}(w_1) - \mathrm{row}(u)| + |\mathrm{col}(w_1) - \mathrm{col}(u)|$.

Proof of Claim 4. Let $\ell_1 < \cdots < \ell_t$ be the iterations of Algorithm 4.1 in which an edge of $E(H-u) \setminus E(H)$ is created. If $e = (w_1, w_2)$ is created in iteration ℓ_1, then necessarily e is created by **Rule 1** (as u has out-degree 0 it can not block any path of another node). Then necessarily w_2 is a neighbor of u such that $D^+(w_2) = \emptyset$. Observe that $R_{\ell_1}(w_2) \setminus \{u\} \neq \emptyset$ because otherwise (w_2, u) is an edge of $E(H)$. Then $w_1 \neq w_2$ and w_2 is closer to u than w_1 in Manhattan distance.

Now suppose that the claim is true for all edges created in steps up to ℓ_i, with $i \in \{1, \dots, t-1\}$, and suppose that $e = (w_1, w_2)$ is created in iteration ℓ_{i+1}.

If (w_1, w_2) is created by **Rule 1**, then necessarily w_2 has an out-neighbor w_3 not present in $E(H)$, that was created in iterations $\ell_1, \dots, \ell_i$. Then w_2 belongs to $S_{NE}(u)$ by induction hypothesis. Observe that $R_{\ell_{i+1}}(w_2) \setminus w_3 \neq \emptyset$, otherwise (w_2, w_3) is an edge of $E(H)$. Then $w_1 \neq w_2$. Suppose by contradiction that w_1 is not contained in $S_{NE}(u)$. Then necessarily w_1 belongs to $S_{SE}(u)$, with $w_1 = (\mathrm{row}(w_2)+1, \mathrm{col}(w_2))$. Moreover, w_3 belongs to $S_{NE}(u)$. Then, by induction hypothesis, w_3 is closer to u than w_2, meaning that $w_3 = (\mathrm{row}(w_2), \mathrm{col}(w_2)-1)$. This implies that (w_1, w_3) is an edge of $E(H)$ and then (w_1, w_2) must be also an edge of $E(H)$. We deduce that w_1 is contained in $S_{NE}(u)$. As the distance from w_2 to w_3 is fewer than the distance of w_1 to w_3 we deduce by the induction hypothesis that the distance from w_2 to u is fewer than the distance from w_1 to u.

If (w_1, w_2) is created by **Rule 2**. Then necessarily w_1 has an incoming edge from a node $w_0 \in V(w_1)$ such that (w_0, w_1) is not in $E(H)$ and was created in iterations $\ell_1, \dots, \ell_i$. From the induction hypothesis we know that w_0 belongs to $S_{NE}(u)$ and is closer to u than w_1. Then w_0 belongs to $V_1(w_1)$. If (w_0, w_1) is created with **Rule 1**, then there is a path P from w_1 to one of the adjacent particles of u where all the edges in that path are edges in $E(H-u) \setminus E(H)$ created by **Rule 1** (this is proven in the previous paragraph). Then P either contains w_2 or contains a node w' such that. $\mathrm{row}(w') = \mathrm{row}(w_2)$ and $\mathrm{col}(w') > \mathrm{col}(w_2)$. Indeed, in the other possibility is that (v_0, v_1) and (v_1, v_2) are edges of P, with $v_0 = (\mathrm{row}(w_2)-1, \mathrm{col}(w_2)+1)$, $v_1 = (\mathrm{row}(w_2)-1, \mathrm{col}(w_2))$ and $v_2 = (\mathrm{row}(w_2)-1, \mathrm{col}(w_2)-1)$. However, this is impossible because in that case (w_2, v_1) must be an edge of $E(H)$ and then (v_0, v_1) is also an edge of $E(H)$. Then, there is a directed path of *rule 1 edges* connecting w_1 and a neighbor of u. This implies that w_2 is ether in $S_{NE}(u)$ or in $S_{SW}(u)$. In any case we have that (w_1, w_2) satisfies the conditions of the claim. It remains the case when (w_0, w_1) is created with **Rule 2** in some iteration $\ell_1, \dots, \ell_i$. Let $P' = v_0, \dots, v_k$ be the longest directed path of edges in $E(H-u) \setminus E(H)$ that are created with **Rule 2** in iterations $\ell_1, \dots, \ell_i$, such that $v_k = w_0$. Then v_0 must have an incoming neighbor z_0 such that (z_0, w_0) is an edge of $E(H-u) \setminus E(H)$ created by **Rule 1**, and such that $z_0 = (\text{row}(v_0) - 1, \text{col}(v_0))$. Following the same argument than in the previous case (i.e. taking the path of edges of $E(H-u) \setminus E(H)$ created by **Rule 1** connecting w_0 with a particle adjacent to u) we deduce that (w_1, w_2) satisfies the conditions of the claim. This finishes the proof of **Claim 4**.

Observe that **Claim 4** implies that the edges of $E(H-u) \setminus E(H)$ form a directed acyclic graph. On the other hand, we know that H is acyclic. Finally, it is impossible to have a cycle containing a combination of edges in $E(H-u) \setminus E(H)$ and $E(H)$, because all the out-going neighbors of nodes in $E(H-u) \setminus E(H)$ also belong to $E(H-u) \setminus E(H)$. We deduce that the dependency graph of $F \setminus \{u\}$ is acyclic. We conclude that u is a removable particle of F.

We remark that the third condition of Definition 4.12 is not necessarily deduced from the first two conditions, in the sense that there are figures with particles u satisfying that $C(u) \neq \emptyset$ and $D^+(u) = \emptyset$, but that are not removable. For instance, the realizable figure depicted in Figure 29 contains a particle u such that $C(u) \neq \emptyset$, $D^+(u) = \emptyset$ and $F \setminus \{u\}$ is not realizable.

We are now ready to give the main result of this section.

THEOREM 4.14. *A figure F is 2DLA-realizable if and only its dependency graph H is acyclic.*

Proof. Let us assume that H contains a cycle, and let u be a node in a cycle of H. Lemma 4.11 implies that u must be placed before himself on any canonical realization. Therefore, F does not admit a canonical realization. From Lemma 4.6 we deduce that F is not realizable.

Conversely, suppose that $H = (F, E)$ is acyclic. Then, Lemma 4.13 imply that we can successively decompose F finding removable nodes. The obtained canonical realization of F is the output of Algorithm 4.2. ◻

COROLLARY 4.15. *2*-DLA-REALIZATION *is solvable in polynomial time.*

Remark 4.16. An implementation of the algorithm solving 2-DLA-REALIZATION can be found in https://github.com/pedromontealegre/2DLA-Realization.

5. Conclusion.

The introduction of restrictions to discrete dynamical systems, coming from statistical physics, has been shown to change the computational complexity of its associated decision problem. We have once again observed this phenomenon with the changes in computational complexity when restricting the directions a particle can move in the DLA model. By adapting the **P**-Complete proof of the 4-DLA-PREDICTION we showed that both 3-DLA-PREDICTION and 2-DLA-PREDICTION are **P**-Complete. For 1-DLA-PREDICTION, we tackled the generalized problem, BALLISTIC DEPOSITION and showed that by exploiting the commutativity exhibited by the dynamics of the system, we created a non-deterministic log-space algorithm to solve the problem, and show that 1-PREDICTION is **NL**-Complete. What is interesting to note is that the algorithm does not depend on the topological properties of the model, it exclusively works on the input word (the sequence of particle throws in this case).

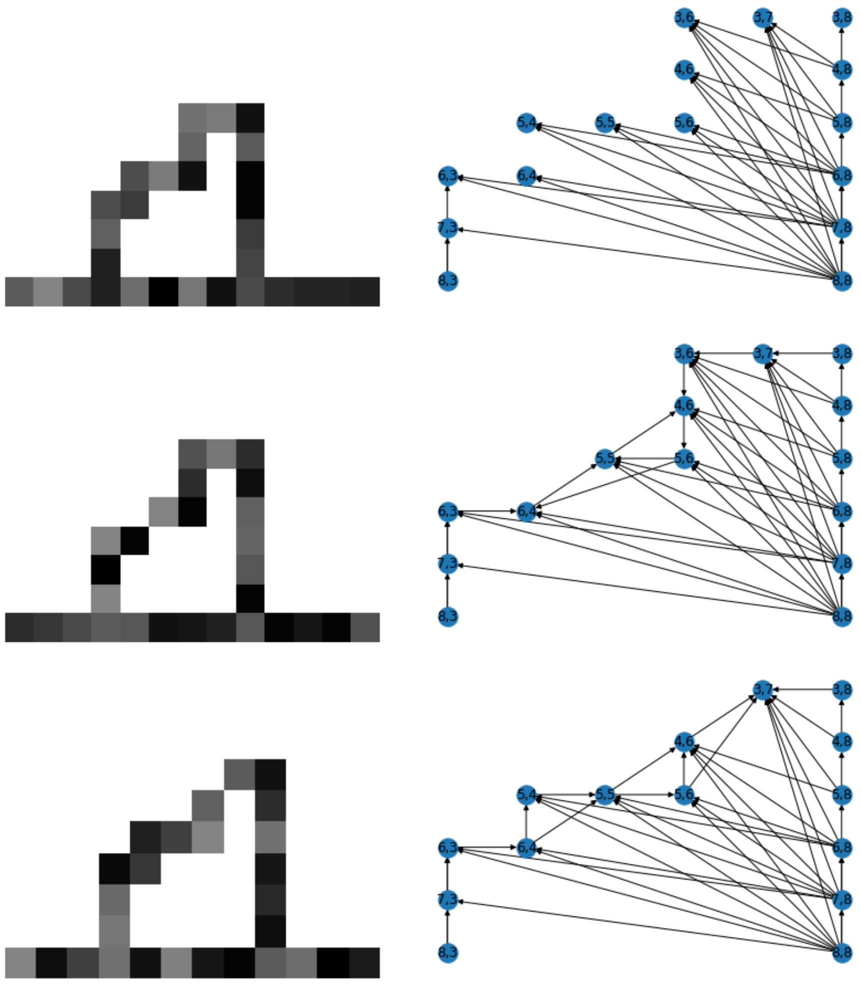


FIG. 29. *An example of a figure containing a node of coordinates $u = (5,4)$ such that $C(u) \neq \emptyset$ (as $Q_1(u)$ is available), $D^+(u) = \emptyset$ but $F \setminus \{u\}$ contains a cycle. In the first row is depicted the figure F and its dependency graph. And in the second row is depicted $F \setminus \{u\}$ and its corresponding dependency graph. Nodes $(5,5),(4,6),(5,6)$ form a cycle in this graph. Observe that F does contain a removable particle, for instance $v = (3,6)$. In the third row is depicted $F \setminus \{v\}$ and its corresponding (acyclic) dependency graph.*

We finally showed that characterizing the shapes that are obtainable through the dynamics is an interesting problem, and exhibited that the computational problem associated is in **L** for the one-directional dynamics, and in **P** for the two-directional one. This is interesting due to the fact that the figures produced when executing the probabilistic versions of the dynamics, as shown in Figure 2, the computational complexity of examining these shapes is efficient.

5.1. Future Work. An interesting extension to the presented problem is the one of determining, given a figure, what is the minimum amount of directions necessary to produce it, if it is achievable at all. We have shown that figures generated by the 1-PREDICTION model are characterizable in **L**, and those of the two-directional model in **P**. It remains to see if the latter can be improved into an **NC** algorithm, or the problem is in fact **P**-Complete. The definition of the dependency graph seems intrinsically sequential, and we believe that is unlikely that the problem could be solved by a fast-parallel algorithm

In addition, the complexity classification of realization problem for three and four directions remains open. We believe that the definition of a dependency graph for these cases is plausible, but several assumptions (such as the existence of canonical realizations) must be redefined.

A related problem, not treated in this article, consists in the *reachability*, i.e. given a figure F and a subfigure F', decide if it is possible to construct F given that F' is already fixed. This problem can be also studied restricting the number of directions the particles are allowed to move in. Our belief is that there is room to create more complex gadgets, indicating that these problems are possibly **NP**-Complete at least for three or more movement directions.

Acknowledgments. We would like to thank the anonymous reviewers for letting us know about the Ballistic Deposition model, which we had no previous knowledge of.

This work has been partially supported by ANID, via PAI + Convocatoria Nacional Subvención a la Incorporación en la Academia Año 2017 + PAI77170068 (P.M.), FONDECYT 11190482 (P.M.) FONDECYT 1200006 (E.G. and P.M.) and via CONICYT-PFCHA/MagísterNacional/2019 - 22190497 (N.B.).

REFERENCES

[1] E. Allender, T. Chakraborty, D. A. M. Barrington, S. Datta, and S. Roy, *Grid graph reachability problems*, in 21st Annual IEEE Conference on Computational Complexity (CCC'06), IEEE, 2006, pp. 15–29.

[2] S. Arora and B. Barak, *Computational complexity: a modern approach*, Cambridge University Press, 2009.

[3] J. Arulanandham, C. Calude, and M. Dinneen, *Bead–sort: A natural sorting algorithm*, Bulletin of the European Association for Theoretical Computer Science, 76 (2002), pp. 153–162.

[4] A. Asselah, E. N. Cirillo, B. Scoppola, E. Scoppola, et al., *On diffusion limited deposition*, Electronic Journal of Probability, 21 (2016).

[5] F. Barra, B. Davidovitch, and I. Procaccia, *Iterated conformal dynamics and laplacian growth*, Physical Review E, 65 (2002), p. 046144.

[6] F. Benevides and M. Przykucki, *Maximum percolation time in two-dimensional bootstrap percolation*, SIAM Journal on Discrete Mathematics, 29 (2015), pp. 224–251, https://doi.org/10.1137/130941584.

[7] R. Brady and R. Ball, *Fractal growth of copper electrodeposits*, Nature, 309 (1984), pp. 225–229.

[8] J. Chalupa, P. L. Leath, and G. R. Reich, *Bootstrap percolation on a bethe lattice*, Journal of Physics C: Solid State Physics, 12 (1979), pp. L31–L35, https://doi.org/10.1088/0022-3719/12/1/008.

[9] D. Y. Chan, B. D. Hughes, L. Paterson, and C. Sirakoff, *Simulating flow in porous media*, Physical Review A, 38 (1988), p. 4106.

[10] E. Goles, P. Montealegre-Barba, and I. Todinca, *The complexity of the bootstraping percolation and other problems*, Theoretical Computer Science, 504 (2013), pp. 73–82, https://doi.org/10.1016/j.tcs.2012.08.001.

[11] R. Greenlaw, H. Hoover, and W. Ruzzo, *Limits to Parallel Computation: P-completeness Theory*, Oxford University Press, Inc., New York, NY, USA, 1995.

[12] T. C. Halsey, P. Meakin, and I. Procaccia, *Scaling structure of the surface layer of diffusion-limited aggregates*, Physical review letters, 56 (1986), p. 854.

[13] Z. Koza, *The equivalence of the dla and a hydrodynamic model*, Journal of Physics A: Mathematical and General, 24 (1991), p. 4895.

[14] J. Machta and R. Greenlaw, *The parallel complexity of growth models*, Journal of Statistical Physics, 77 (1994), pp. 755–781.

[15] J. Machta and R. Greenlaw, *The computational complexity of generating random fractals*, Journal of Statistical Physics, 82 (1996), pp. 1299–1326.

[16] T. Mansour, R. Rastegar, and A. Roitershtein, *On ballistic deposition process on a strip*, arXiv preprint arXiv:1903.12548, (2019).

[17] P. Meakin, *Diffusion-controlled cluster formation in 2-6-dimensional space*, Physical Review A, 27 (1983), p. 1495.

[18] P. Meakin, P. Ramanlal, L. M. Sander, and R. Ball, *Ballistic deposition on surfaces*, Physical Review A, 34 (1986), p. 5091.

[19] C. Moore and J. Machta, *Internal diffusion-limited aggregation: parallel algorithms and complexity*, Journal of Statistical Physics, 99 (2000), pp. 661–690.

[20] C. Moore and M. Nilsson, *The computational complexity of sandpiles*, Journal of statistical physics, 96 (1999), pp. 205–224.

[21] L. Niemeyer, L. Pietronero, and H. Wiesmann, *Fractal dimension of dielectric breakdown*, Physical Review Letters, 52 (1984), p. 1033.

[22] J. Nittmann, G. Daccord, and H. E. Stanley, *Fractal growth viscous fingers: quantitative characterization of a fluid instability phenomenon*, Nature, 314 (1985), pp. 141–144.

[23] M. D. Penrose, *Growth and roughness of the interface for ballistic deposition*, Journal of Statistical Physics, 131 (2008), pp. 247–268.

[24] D. Sutherland, *Comments on vold's simulation of floc formation*, Journal of Colloid and Interface Science, 22 (1966), pp. 300–302.

[25] M. J. Vold, *Computer simulation of floc formation in a colloidal suspension*, Journal of Colloid Science, 18 (1963), pp. 684–695.

[26] T. A. Witten and L. M. Sander, *Diffusion-limited aggregation, a kinetic critical phenomenon*, Phys. Rev. Lett., 47 (1981), pp. 1400–1403, https://doi.org/10.1103/PhysRevLett.47.1400.