A Linear Acceleration Theorem for 2D Cellular Automata on All Complete Neighborhoods

Anaël Grandjean[1] and Victor Poupet[2]

1 **LIRMM, Université Montpellier, Montpellier, France**
`anael.grandjean@lirmm.fr`
2 **LIRMM, Université Montpellier, Montpellier, France**
`victor.poupet@lirmm.fr`

Abstract

Linear acceleration theorems are known for most computational models. Although such results have been proved for two-dimensional cellular automata working on specific neighborhoods, no general construction was known. We present here a technique of linear acceleration for all two-dimensional languages recognized by cellular automata working on complete neighborhoods.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases 2D Cellular automata, linear acceleration, language recognition

Digital Object Identifier 10.4230/LIPIcs.ICALP.2016.115

1 Introduction

Cellular automata (CA) were initially introduced by S. Ulam and J. von Neumann [14] in the 1960s to study self-reproduction in discrete dynamical systems. They are massively parallel systems consisting of an infinite array of cells. Cells evolve synchronously depending on the states of their neighbors according to a uniform deterministic rule. Although initially considered in two dimensions, the definition can be adapted to any dimensional cellular space and even more general uniform graphs [8].

Soon after their introduction, they were shown to be computationally universal [9, 1]. As a computation model, they have been extensively studied as one-dimensional language recognizers [11, 3] but are also very well suited to the study of two-dimensional "picture languages" [10, 12, 13].

The neighborhood of a cellular automaton defines the underlying communications graph of the cells. Although most of the existing work on two-dimensional cellular automata focuses on the von Neumann (4 closest neighbors) and Moore (8 closest neighbors) neighborhoods, understanding how the choice of the neighborhood affects the algorithmic capabilities of the model is a key to understanding parallel computation.

Linear acceleration theorems are well known for most of the commonly considered computation models. It was first proved for one-dimensional cellular automata working on the standard neighborhood and two-dimensional cellular automata on von Neumann's neighborhood by W. T. Beyer [2], inspired by similar constructions for sequential input cellular automata [3, 5, 6]. The one-dimensional case was later generalized by J. Mazoyer and N. Reimen for arbitrary neighborhoods [7]. As for two-dimensional neighborhoods, V. Terrier extended the construction to the Moore neighborhood [12] and then to the slightly more general class of neighborhoods whose convex hull has at most one vertex in the positive quarter plane [13].

43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016).
Editors: Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi;
Article No. 115; pp. 115:1–115:12


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we prove a general linear acceleration result for all complete neighborhoods on two-dimensional cellular automata.

The main theorem is stated in Section 3 and proved in Sections 4 to 7. Sections 4 and 6 describe the two main elements of the construction (compression of the input and accelerated simulation of the original automaton respectively). Section 5 presents a technique to perform a sequence of tasks on a cellular automaton without the need for synchronization at the start of each new task, used in the proof of the theorem to combine sections 4 and 6. Although this technique is elementary and has been used in previous publications (a special case was used by W. T. Beyer in 1969 [2]), reviews of previous articles seem to indicate that it is not common knowledge. It is therefore presented here in a separate section and stated generally in the hopes that it can be easily reused in future publications.

The construction presented in this article is similar in several ways to previously published constructions, most notably those of V. Terrier in [13]. Significant improvements include compression of the input in *almost* optimal time (Section 4, specifically Subsection 4.2) and a more general simulation technique (Section 6).

2 Definitions

2.1 Cellular Automata

▶ **Definition 1** (Cellular Automaton)**.** A *cellular automaton* (CA) is a quadruple $\mathcal{A} = (d, \mathcal{Q}, \mathcal{N}, \delta)$ where

- $d \in \mathbb{N}$ is the dimension of the automaton ;
- $\mathcal{Q}$ is a finite set whose elements are called *states* ;
- $\mathcal{N}$ is a finite subset of $\mathbb{Z}^d$ called *neighborhood* of the automaton ;
- $\delta : \mathcal{Q}^{\mathcal{N}} \to \mathcal{Q}$ is the *local transition function* of the automaton.

▶ **Definition 2** (Configuration)**.** A *d-dimensional configuration* $\mathfrak{C}$ over the set of states $\mathcal{Q}$ is a mapping from $\mathbb{Z}^d$ to $\mathcal{Q}$. The elements of $\mathbb{Z}^d$ will be referred to as *cells*.

Given a CA $\mathcal{A} = (d, \mathcal{Q}, \mathcal{N}, \delta)$, a configuration $\mathfrak{C} \in \mathcal{Q}^{\mathbb{Z}^d}$ and a cell $c \in \mathbb{Z}^d$, we denote by $\mathcal{N}_{\mathfrak{C}}(c)$ the neighborhood of c in $\mathfrak{C}$:

$$\mathcal{N}_{\mathfrak{C}}(c) : \left\{ \begin{array}{ccc} \mathcal{N} & \to & \mathcal{Q} \\ n & \mapsto & \mathfrak{C}(c+n) \end{array} \right.$$

From the local transition function δ of a CA $\mathcal{A} = (d, \mathcal{Q}, \mathcal{N}, \delta)$, we can define the *global transition function of the automaton* $\Delta : \mathcal{Q}^{\mathbb{Z}^d} \to \mathcal{Q}^{\mathbb{Z}^d}$ obtained by applying the local rule on all cells :

$$\Delta(\mathfrak{C}) = \left\{ \begin{array}{ccc} \mathbb{Z}^d & \to & \mathcal{Q} \\ c & \mapsto & \delta(\mathcal{N}_{\mathfrak{C}}(c)) \end{array} \right.$$

The action of the global transition rule makes $\mathcal{A}$ a dynamical system over the set $\mathcal{Q}^{\mathbb{Z}^d}$. Because of this dynamic, in the following we will identify the CA $\mathcal{A}$ with its global rule so that $\mathcal{A}(\mathfrak{C})$ is the image of a configuration $\mathfrak{C}$ by the action of the CA $\mathcal{A}$, and more generally $\mathcal{A}^t(\mathfrak{C})$ is the configuration resulting from applying t times the global rule of the automaton from the initial configuration $\mathfrak{C}$.

▶ **Definition 3** (Quiescent and Permanent States)**.** For a given CA $\mathcal{A}$, we say that a state q is *quiescent* if a cell in state q remains in this state if all its neighbors are also in q. We say that q is *permanent* if a cell in state q remains in that state regardless of the state of its neighbors.

In this article we will only consider 2-dimensional cellular automata (2DCA). From now on the set of cells will always be $\mathbb{Z}^2$.

2.2 Neighborhoods

Throughout the article, we use the additive notation for vector sums, the power notation for neighborhood composition and the product notation for scalar product:

▶ **Definition 4** (Vector Sum). Given two neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$ and a cell $c \in \mathbb{Z}^2$, we define the vector sums $\mathcal{N}_1 + \mathcal{N}_2 = \{x + y \mid x \in \mathcal{N}_1, y \in \mathcal{N}_2\}$ and $c + \mathcal{N}_1 = \{c + x \mid x \in \mathcal{N}_1\}$.

▶ **Definition 5** (Neighborhood Powers). Given a neighborhood $\mathcal{N}$, we define

$$\mathcal{N}^0 = \{0\} \tag{1}$$

$$\forall k \in \mathbb{N}, \quad \mathcal{N}^{k+1} = \mathcal{N} + \mathcal{N}^k \tag{2}$$

▶ **Definition 6** (Scalar product). Given a neighborhood $\mathcal{N}$ and an integer $k \in \mathbb{Z}$, we define the scalar product $k\mathcal{N} = \{kx \mid x \in \mathcal{N}\}$.

▶ **Definition 7** (Complete Neighborhood). A neighborhood $\mathcal{N}$ is said to be *complete* if

$$\bigcup_{k \in \mathbb{N}} \mathcal{N}^k = \mathbb{Z}^2$$

▶ **Definition 8** (Convex Hull and Convex Neighborhood). The *convex hull* of a neighborhood $\mathcal{N}$ is the smallest convex polygon $\mathrm{CH}(\mathcal{N}) \subset \mathbb{R}^2$ such that $\mathcal{N} \subseteq \mathrm{CH}(\mathcal{N})$. Moreover a neighborhood $\mathcal{N}$ is said to be *convex* if it contains all points of integer coordinates in its convex hull: $\mathcal{N} = \mathrm{CH}(\mathcal{N}) \cap \mathbb{Z}^2$.

▶ Remark. If $\mathcal{N}$ is a convex neighborhood, $\mathcal{N}^p$ is also convex for any $p \in \mathbb{N}$.

2.3 Two-Dimensional Language Recognition

▶ **Definition 9** (Picture). For $n, m \in \mathbb{N}$ and Σ a finite alphabet, an (n, m)-*picture* (picture of width n and height m) over Σ is a mapping

$$p : [\![0, n-1]\!] \times [\![0, m-1]\!] \to \Sigma$$

$\Sigma^{n,m}$ denotes the set of all (n, m)-pictures over Σ and $\Sigma^{*,*} = \bigcup_{n,m \in \mathbb{N}} \Sigma^{n,m}$ the set of all pictures over Σ. A *picture language* over Σ is a set of pictures over Σ.

▶ **Definition 10** (Picture Configuration). Given an (n, m)-picture p over Σ, we define the *picture configuration* associated to p with quiescent state $q_0 \notin \Sigma$ as

$$\mathfrak{C}_{p,q_0} : \begin{cases} \mathbb{Z}^2 & \to & \Sigma \cup \{q_0\} \\ x, y & \mapsto & \begin{cases} p(x, y) & \text{if } (x, y) \in [\![0, n-1]\!] \times [\![0, m-1]\!] \\ q_0 & \text{otherwise} \end{cases} \end{cases}$$

▶ **Definition 11** (Picture Recognizer). Given a picture language L over an alphabet Σ, we say that a 2DCA $\mathcal{A} = (2, \mathcal{Q}, \mathcal{N}, \delta)$ such that $\Sigma \subseteq \mathcal{Q}$ recognizes L with quiescent state $q_0 \in \mathcal{Q} \setminus \Sigma$, accepting state $q_a \in \mathcal{Q}$ and rejecting state $q_r \in \mathcal{Q}$ in time $\tau : \mathbb{N}^2 \to \mathbb{N}$ if q_a and q_r are permanent states and for any picture p (of size $n \times m$), starting from the picture configuration $\mathfrak{C}_{p,q_0}$ at time 0, the origin cell of the automaton at time $\tau(n, m)$ is in state q_a if $p \in L$ and state q_r if $p \notin L$.

▶ **Definition 12** (Real Time). Given a complete neighborhood $\mathcal{N}$, the real time function $\mathrm{RT}_{\mathcal{N}} : \mathbb{N}^2 \to \mathbb{N}$ associated to $\mathcal{N}$ is defined as

$$\mathrm{RT}_{\mathcal{N}}(n, m) = \min\{t \mid [\![0, n-1]\!] \times [\![0, m-1]\!] \subseteq \mathcal{N}^t\}$$

3 The Main Theorem

Most of the article will be dedicated to the proof of the following theorem

▶ **Theorem 13** (Linear Acceleration). *For any complete neighborhood $\mathcal{N}$, any real number $\epsilon > 0$, any finite alphabet Σ and any language $L \subseteq \Sigma^{*,*}$, if L is recognized by a 2DCA working on $\mathcal{N}$ in time*

$$(n, m) \mapsto \mathrm{RT}_{\mathcal{N}}(n, m) + f(n, m)$$

for some function $f : \mathbb{N}^2 \to \mathbb{N}$ then L can be recognized in time

$$(n, m) \mapsto \lceil (1 + \epsilon) \mathrm{RT}_{\mathcal{N}}(n, m) + \epsilon f(n, m) \rceil$$

by a 2DCA with neighborhood $\mathcal{N}$.

▶ **Corollary 14.** *For any complete neighborhood $\mathcal{N}$, any language recognized in time $(n, m) \mapsto k \mathrm{RT}_{\mathcal{N}}(n, m)$ for some $k > 1$ can be recognized in time $(n, m) \mapsto (1 + \epsilon) \mathrm{RT}_{\mathcal{N}}(n, m)$ for any real number $\epsilon > 0$.*

To prove Theorem 13, we consider a 2DCA $\mathcal{A}$ working on a complete neighborhood $\mathcal{N}$ and describe the construction of a 2DCA $\mathcal{A}'$ working on the same neighborhood that simulates the behavior of $\mathcal{A}$ in a way that enables it to recognize the same language as $\mathcal{A}$ in a linearly shorter time.

3.1 Preliminary Remarks

The following observations will greatly simplify the proof of Theorem 13.

▶ **Claim 15.** *It is sufficient to prove Theorem 13 up to an additive constant, meaning that we only need to prove that L can be recognized in time*

$$(n, m) \mapsto (1 + \epsilon) \mathrm{RT}_{\mathcal{N}}(n, m) + \epsilon f(n, m) + O(1)$$

Proof. Consider that we have this weaker result. To get rid of the $O(1)$ term simply choose $\epsilon' < \epsilon$. For any $C > 0$, for (n, m) large enough we have

$$(1 + \epsilon) \mathrm{RT}_{\mathcal{N}}(n, m) + \epsilon f(n, m) > (1 + \epsilon') \mathrm{RT}_{\mathcal{N}}(n, m) + \epsilon' f(n, m) + C$$

The automaton can handle all the finitely many inputs of small size in real time. ◀

▶ **Claim 16.** *It is sufficient to prove Theorem 13 for all complete convex neighborhoods.*

Proof. Consider a complete neighborhood $\mathcal{N}$ and let $\mathcal{N}'$ be the convex neighborhood having same convex hull as $\mathcal{N}$. The real time functions $\mathrm{RT}_{\mathcal{N}}$ and $\mathrm{RT}_{\mathcal{N}'}$ differ by at most a constant. Moreover a CA on $\mathcal{N}$ can simulate the behavior of a CA on $\mathcal{N}'$ with a loss of at most a constant number of steps and conversely (see [4] for more details).

The property from the theorem therefore translates directly from one neighborhood to the other with at most a constant difference that can be ignored according to Claim 15. ◀

From now on we will consider that $\mathcal{N}$ is a convex neighborhood.

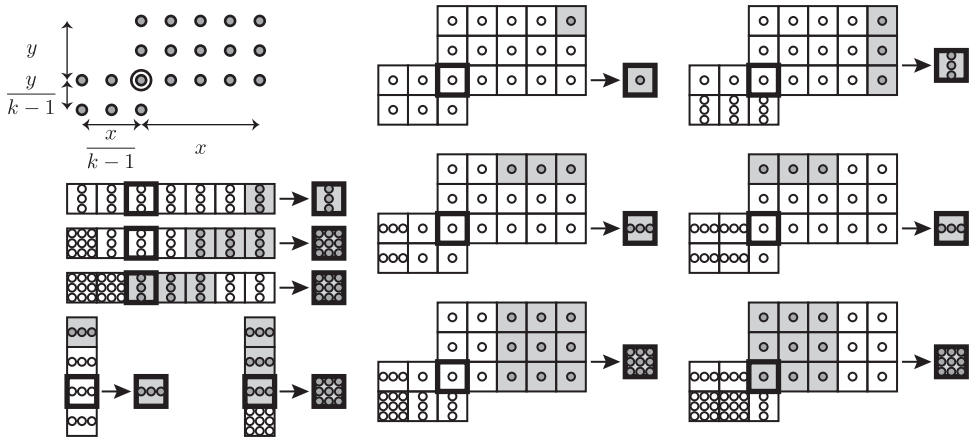

Figure 1 Rules for the compression by a factor 3 of inputs of ratio $\frac{n}{m} = 2$ with the neighborhood represented in the top left. The information that the cell takes as its new state is represented in grey. Information travels towards the bottom left. By looking down and left a cell determines if the information from the top right should simply pass through (first case) or if some of its neighbors are already full in which case it should start packing information. Left column shows simplified rules for which the cell has already packed information in one of the directions and therefore only the neighbors in the remaining direction are significant.

4 Compression of the Input

The first phase of the construction is to compress the input by a factor $k > \frac{1}{\epsilon}$. We want to move the states of the initial configuration towards the origin, packing them in groups of $k \times k$ as illustrated by Figure 2.

Although such compressions are relatively simple to perform on the von Neumann and Moore neighborhoods, on a more general neighborhood it is not possible to know in which direction the information should travel to move towards the origin at optimal speed. In general, the optimal travel direction depends on the proportion $\frac{n}{m}$ of the input.

We first show that if the proportion of the input is fixed, compression can be done in optimal time on any complete neighborhood. Then, by performing a finite number of compressions in parallel, each assuming a different proportion, we show that any input is close enough to one of these assumed proportions to be compressed in "nearly optimal" time, which will be sufficient for the proof of Theorem 13.

4.1 Compression of an Input of Constrained Proportion

If the size of the input is known to be of proportion $\frac{n}{m} = \alpha$ for some fixed rational α, we can perform a compression by a factor k with a neighborhood such as the one illustrated on the top left of Figure 1 with $\frac{x}{y} = \alpha$. On such a neighborhood, compressing the input is simply a matter of transferring the states from the top right to the bottom left, packing them in groups of $(1 \times k)$, $(k \times 1)$ or $(k \times k)$ when they cannot go any further in one or both directions (see Figures 1 and 2). The compression is completed in time $\lceil \frac{k-1}{k} \mathrm{RT} \rceil$.

Note that in order to compress by a factor k, the cell must be able to see $\frac{1}{k-1}$ times as far towards the left and bottom as it sees towards the right and top in order to properly determine when it should start packing states. Because the convex hull of a complete neighborhood $\mathcal{N}$ contains an open set around the origin (otherwise $(\mathcal{N}^p)_{p\in\mathbb{N}}$ would not expand in all directions), it contains its homothetic image of ratio $-\frac{1}{k_0-1}$ for some k_0 (see Figure 3). On such a neighborhood compression by any factor $k \geq k_0$ is possible.

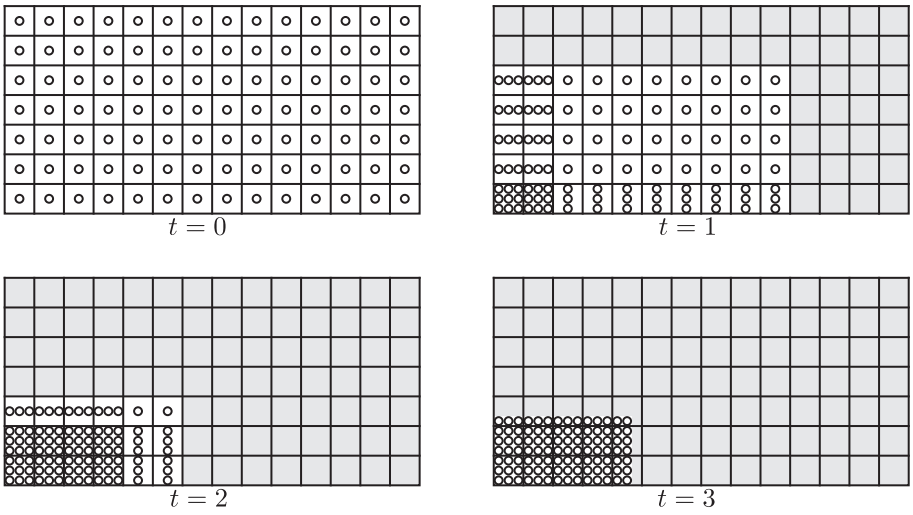

Figure 2 Compression of an input of size (14×7) with the neighborhood and rules from Figure 1.

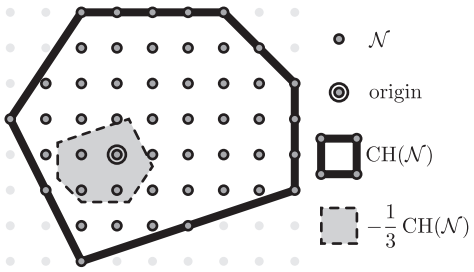

Figure 3 An example neighborhood on which a compression by a factor $k = 4$ is possible.

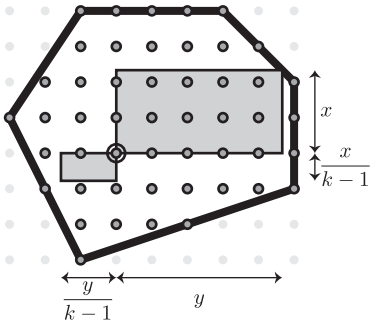

Figure 4 The area of the neighborhood that will be used for the compression of inputs of proportion $\frac{n}{m} = 2$ by a factor $k = 4$.

To compress inputs of proportion $\frac{n}{m} = \alpha$ on some complete neighborhood $\mathcal{N}$, we consider the largest rectangle $[0, x] \times [0, y]$ with $x, y \in \mathbb{Q}$ and $\frac{x}{y} = \alpha$ included in the convex hull of $\mathcal{N}$ (see Figure 4). For all $k \geq k_0$, the rectangle $[-\frac{x}{k-1}, 0] \times [-\frac{y}{k-1}, 0]$ is also in the convex hull of $\mathcal{N}$.

These rectangles have rational but not necessarily integer dimensions. If we consider the neighborhood $\mathcal{N}^p$ for some large p, all is scaled up by a factor p and the corresponding rectangles can be made of integer dimensions. The real time function on inputs of proportion α for $\mathcal{N}^p$ is equal to the real time function of the neighborhood containing only the two rectangles (of integer coordinates). The compression algorithm described by Figures 1 and 2 therefore finishes in time $\frac{k-1}{k}\,\mathrm{RT}_{\mathcal{N}^p} + O(1)$ on $\mathcal{N}^p$. An automaton working on $\mathcal{N}$ can simulate one step of an automaton working on $\mathcal{N}^p$ in p time steps, and since $\mathrm{RT}_{\mathcal{N}^p} = \lceil \frac{1}{p}\,\mathrm{RT}_{\mathcal{N}} \rceil$, the compression can be completed on $\mathcal{N}$ in time $\frac{k-1}{k}\,\mathrm{RT}_{\mathcal{N}} + O(1)$.

4.2 Compression of General Input

Let us now consider inputs of arbitrary proportions. As discussed in the previous subsection, for inputs of proportion α the optimal direction in which the information should travel for

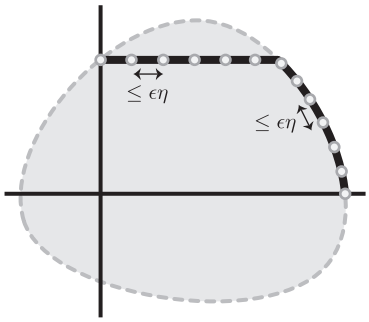

Figure 5 Choosing the set S of proportions that will be used by $\mathcal{A}'$ for compressions. The thick line is the set of corners of maximal rectangles for all possible proportions. Along this line, we pick a finite set of points at intervals of at most $\epsilon\eta$.

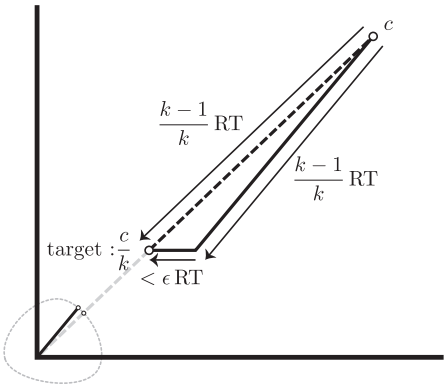

Figure 6 Compression of an input of arbitrary proportion. The optimal direction for the compression is represented by a dashed line. The closest chosen direction is represented by a solid line inside the neighborhood. It is at a distance at most $\epsilon\eta$ of the optimal vector. The travel path of the farthest cell of the input is represented as a solid black line, made of an initial segment along the almost-optimal direction and an extra segment to compensate for the deviation.

a compression is defined by the diagonal of the largest rectangle $[0, x] \times [0, y]$ with $\frac{x}{y} = \alpha$ included in the convex hull of $\mathcal{N}$. The first thing to note is that since $\mathcal{N}$ is complete, there exists $\eta > 0$ such that $[0, \eta] \times [0, \eta] \subseteq \mathrm{CH}(\mathcal{N})$ and hence all maximal rectangles in $\mathrm{CH}(\mathcal{N})$ have at least one dimension greater than η. The corners (x, y) of such maximal rectangles all lie on a line. Let us pick a finite set S of rational points on this line from one extremity to the other with distance at most $\epsilon\eta$ between two consecutive points (see Figure 5).

For each proportion $\alpha = \frac{x}{y}$ with $(x, y) \in S$, $\mathcal{A}'$ performs a compression of the input as described in the previous subsection. All compressions take place at the same time in parallel. Note that even if the proportion of the input is not exactly that for which the compression is optimized, the input is still compressed properly although not as quickly.

Let us prove that one of the compressions that are run by the automaton compresses the input in time at most $(\frac{k-1}{k} + \epsilon)\,\mathrm{RT}$. A compression along the vector corresponding exactly to the proportion of the input would take a time $\frac{k-1}{k}\,\mathrm{RT}$. A compression along one of the vectors in S that is closest to the optimal vector (at distance at most $\epsilon\eta$) puts all states from the input within a distance at most $\epsilon\eta\,\mathrm{RT}$ from their destination in time $\frac{k-1}{k}\,\mathrm{RT} + O(1)$. By choosing the closest vector properly amongst the two choices, the remaining distance can be travelled in time at most $\epsilon\,\mathrm{RT} + O(1)$ as illustrated by Figure 6 (information travels at speed at least η in one of the dimensions).

For any possible input, at least one of the compressions completes in time at most $(\frac{k-1}{k} + \epsilon)\,\mathrm{RT} + O(1)$.

5 Transition

After the input has been compressed, the automaton $\mathcal{A}'$ should immediately start simulating the behavior of $\mathcal{A}$, k steps at a time. However the cells of $\mathcal{A}'$ receive the compressed input at different times. If we wanted all the cells to start the next phase at the same time, we would

require some synchronization scheme such as a firing-squad synchronization algorithm but this would take a linear time. Instead, we show that synchronization is not required to start the accelerated simulation as each cell of the automaton proceeds with the next phase as soon as the relevant information is available.

This technique is very general and can be used in numerous situations where a cellular automaton performs a computation by executing a series of separate tasks one after the other without having to spend time synchronizing all cells. In its general form, it can be stated in the following way:

▶ **Proposition 17** (Passive Synchronization). *Given a CA $\mathcal{A}$ of any dimension working on a complete neighborhood $\mathcal{N}$, there exists a CA $\mathcal{A}'$ working on the same neighborhood $\mathcal{N}$ that can simulate the behavior of $\mathcal{A}$ on any input even if the configuration is given asynchronously in such a way that each cell of $\mathcal{A}'$ computes states of the simulated automaton at least as fast as if the computation had started synchronously when the last cell receives its input.*

Formally, if we denote by $\mathcal{Q}$ the states of $\mathcal{A}$, $\mathcal{A}'$ has states $\{\perp\} \cup (\mathcal{Q} \times \mathcal{Q}')$ where $\perp$ is a permanent state (cannot be changed by the transition rule of the automaton) and $\mathcal{Q}'$ is a set of extra working states containing a default state ν. The cells of $\mathcal{A}'$ are initially in state $\perp$ and considered inactive*. Before each transition of the automaton, any number of inactive cells of $\mathcal{A}'$ might be* activated *by some external action over which $\mathcal{A}'$ has no control. Activating a cell c changes its state to $(\mathfrak{C}(c), \nu)$ where $\mathfrak{C}$ is the input of the simulated automaton $\mathcal{A}$.*

If there exists a time t_0 at which all cells have been activated then for any cell c the projection on $\mathcal{Q}$ of the state of c in $\mathcal{A}'$ at time $(t_0 + t)$ is the state of c in the evolution of $\mathcal{A}$ from the configuration $\mathfrak{C}$ at time t' for some $t' \geq t$.

Proof. The idea is to make all cells of $\mathcal{A}'$ compute one step of $\mathcal{A}$ whenever they have enough information to do so, while remembering their past states that other cells might need at a later time.

When a cell c is activated, it receives the initial state $\mathfrak{C}(c)$ and we say that its *simulated time* is 0. From that point on, it looks at its neighbors and waits for all of them to be activated. When this happens, it sees all initial states in its neighborhood and can compute the next state in the evolution of $\mathcal{A}$, increasing its simulated time to 1. As time passes it keeps watching its neighbors until all of them are also at a simulated time at least equal to its own, which means that it has all the information necessary to compute the next step and increase its simulated time further.

Let us prove that this process can be carried out with finitely many states. First, notice that since $\mathcal{N}$ is complete, there exists $\tau \in \mathbb{N}$ such that $-\mathcal{N} \subseteq \mathcal{N}^\tau$. In order to compute its state for a simulated time $(\tau + t)$ a cell c needs to have had access to the state at the simulated time t of all cells in $(c + \mathcal{N}^\tau)$ which includes the set $(c - \mathcal{N})$ of cells that have c in their neighborhood. This means that a cell cannot be more than τ steps ahead in its simulation than the cells that have it in their neighborhood, which implies that the difference of simulated times between two neighbor cells is at most τ. If each cell stores the value of its simulated time modulo $(2\tau + 1)$, it is possible for a cell to know the relative difference in simulated time with all cells in its neighborhood. Furthermore, it is sufficient that a cell remembers its last τ simulated states to be sure that when a cell c at simulated time t looks at its neighbors that are more advanced in the simulation it can see their simulated state at time t.

Finally, we prove by induction that at time $(t_0 + t)$ all cells have a simulated time at least t. This is obviously true at time t_0 since all cells have been activated. By induction, at time

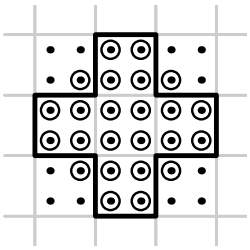

Figure 7 Compression of factor $k = 2$ with the von Neumann neighborhood. The thick line represents $\rho(c+\mathcal{N})$, the states that are visible to the central cell in one step. The circles represent $\rho(c) + \mathcal{N}^k$, the states that should be known to compute two steps of the original automaton on the states that the central cell holds.

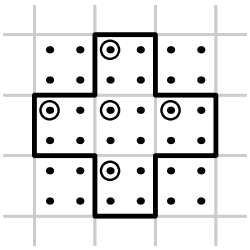

Figure 8 The thick line represents $\rho(c + \mathcal{N})$. The circles represent the cells in $(c' + k\mathcal{N})$ for some c' in $\rho(c)$ (the one in the top left).

$(t_0 + t)$ for any cell at simulated time t all its neighbors are at least at simulated time t so it can compute a new step of the simulation, which proves that at time $(t_0 + t + 1)$ all cells have a simulated time of at least $(t + 1)$.

Note that this process is such that the cells who are behind in their simulation can compute new states without delay, whereas the ones ahead wait for their neighbors to catch up. ◀

In the following section we describe how $\mathcal{A}'$ can simulate k steps of $\mathcal{A}$ at a time starting from a compressed input. We assume that all cells complete the compression and start the simulation synchronously when the last cell receives its compressed information. Using Proposition 17, we can connect the two constructions (cells are *activated* for the simulation when they receive their compressed input) and ensure that the origin is always at least as advanced in its computation as if the simulation had started synchronously.

6 Simulation on a Compressed Input

Let us denote by ρ the function that maps a cell of $\mathcal{A}'$ to the set of cells of $\mathcal{A}$ whose states it receives after the compression of a factor k :

$$\forall c \in \mathbb{Z}^2, \quad \rho(c) = \{kc + (x, y) \mid x, y \in [\![0, k-1]\!]\}$$

and extend the notation to sets of cells by $\rho(S) = \bigcup_{c \in S} \rho(c)$.

The states of $\mathcal{A}$ that are held in the neighborhood of a cell c in $\mathcal{A}'$ are the ones corresponding to the cells of $\mathcal{A}$ in $\rho(c + \mathcal{N})$. To be able to compute k steps of the original automaton, the cell c in $\mathcal{A}'$ needs to be able to see in its neighborhood the states corresponding to the cells $(\rho(c) + \mathcal{N}^k)$ of $\mathcal{A}$. Although this is the case for simple rectangular neighborhoods, it is not true for some neighborhoods (see Figure 7).

What is true however is that $\rho(c) + k\mathcal{N} \subseteq \rho(c + \mathcal{N})$ since for any $v \in \mathcal{N}$ if the state of a cell c' in $\mathcal{A}$ is held by a cell c in $\mathcal{A}'$ after compression of the input, the cell $(c + v)$ in $\mathcal{A}'$ holds the state of $(c' + kv)$ in $\mathcal{A}$ (see Figure 8).

▶ **Lemma 18.** $\forall k \in \mathbb{N}, \exists \alpha \in \mathbb{N}, \quad \mathcal{N}^\alpha + k\mathcal{N} = \mathcal{N}^\alpha + \mathcal{N}^k$

Proof. The inclusion $\mathcal{N}^\alpha + k\mathcal{N} \subseteq \mathcal{N}^\alpha + \mathcal{N}^k$ is obvious for any α. As for the converse, choose α such that $\alpha + k > |\mathcal{N}|(k-1)$. Any $x \in \mathcal{N}^\alpha + \mathcal{N}^k$ can be written as the sum of $(\alpha + k)$

elements of $\mathcal{N}$ and therefore at least one of these elements appears at least k times, which proves that $x \in \mathcal{N}^\alpha + k\mathcal{N}$. ◀

By Lemma 18, we can choose α such that $\mathcal{N}^\alpha + k\mathcal{N} = \mathcal{N}^\alpha + \mathcal{N}^k$. We now modify the behavior of $\mathcal{A}'$ so that during the first α steps of the computation, before starting the compression, all cells gather the initial states contained in their $\mathcal{N}^\alpha$ neighborhood. From here onwards, each cell performs all of the computation as described earlier on all the states it holds : a cell c holds at time $(\alpha + t)$ the states that would have been on all the cells in $(c + \mathcal{N}^\alpha)$ at time t on the automaton $\mathcal{A}'$ as described until now. At time $(\alpha + t)$ the cells in $(c + \mathcal{N})$ as a whole hold all the states that would have been on the cells in $(c + \mathcal{N}^{\alpha+1})$ at time t, which is exactly what is needed to compute the states of all cells in $(c + \mathcal{N}^\alpha)$ at time $(t + 1)$. This extra step adds a constant time α to the computation of the automaton[1].

After the initial gathering and compression of the input, the cell c in $\mathcal{A}'$ holds the initial states in $\mathcal{A}$ for the cells in $(\rho(c) + \mathcal{N}^\alpha)$. Let us show by induction that this is enough to simulate the behavior of $\mathcal{A}'$ with a linear speed-up of factor k. Assume that at time $(t_0 + t)$, any cell c of $\mathcal{A}'$ holds the states at time t for the cells of $\mathcal{A}$ in $(\rho(c) + \mathcal{N}^\alpha)$.

This means that the cells in the neighborhood $(c + \mathcal{N})$ of c in $\mathcal{A}'$ at time $(t_0 + t)$ hold the states at time t in $\mathcal{A}$ of the cells in $\rho(c + \mathcal{N}) + \mathcal{N}^\alpha$. By Lemma 18, we have

$$\rho(c + \mathcal{N}) + \mathcal{N}^\alpha \quad \supseteq \quad \rho(c) + k\mathcal{N} + \mathcal{N}^\alpha \quad \supseteq \quad \rho(c) + \mathcal{N}^{\alpha+k}$$

which shows that cell c in $\mathcal{A}'$ at time $(t_0 + t)$ sees enough information to compute the states in $\mathcal{A}$ for the cells in $(\rho(c) + \mathcal{N}^\alpha)$ at time $(t + k)$.

7 Total time

We have completed the description of the behavior of the automaton $\mathcal{A}'$. Let us now evaluate the total time taken to recognize the language L recognized by $\mathcal{A}$ in time $\mathrm{RT}_\mathcal{N} + f$.

The compression of the input takes a time $(\frac{k-1}{k} + \epsilon)\,\mathrm{RT}_\mathcal{N} + O(1)$. The simulation of $\mathcal{A}$ from a fully compressed input takes a time $\frac{1}{k}(\mathrm{RT}_\mathcal{N} + f) + O(1)$ for some $k \geq \frac{1}{\epsilon}$, and Proposition 17 shows that no time is lost by completing the compression asynchronously (the time of the compression is the time at which the last cell is correctly compressed).

The total time for the simulation of $\mathcal{A}$ is therefore

$$(\frac{k-1}{k} + \epsilon)\,\mathrm{RT}_\mathcal{N} + \frac{1}{k}(\mathrm{RT}_\mathcal{N} + f) + O(1) \leq (1 + \epsilon)\,\mathrm{RT}_\mathcal{N} + \epsilon f + O(1)$$

By Claim 15, the $O(1)$ term can be eliminated, which concludes the proof of Theorem 13.

8 Conclusion

The linear acceleration presented in this article is slightly weaker than the previously known results on a limited class of neighborhoods (which contains the von Neumann and Moore neighborhoods). On these neighborhoods, as well as all one-dimensional complete neighborhoods, any language that can be recognized in time $(\mathrm{RT} + f)$ can be recognized in time $(\mathrm{RT} + \epsilon f)$ for any $\epsilon > 0$.

[1] The constant time α is actually not lost since the origin holds the states that would be on the cells in $\mathcal{N}^\alpha$, which enables it to compute its own state α time steps ahead. However, for the purpose of proving Theorem 13, adding a constant time to the computation is irrelevant.

Although the difference is only significant if $f = o(\mathrm{RT})$, it would be interesting to know whether this stronger statement can be proved for general two-dimensional complete neighborhoods. This would either require an optimal-time compression of the input or a completely different construction skipping the compression altogether.

As we currently understand it, optimal-time compression seems unlikely on general neighborhoods. The problem is that states from the initial configuration should move towards the origin in the optimal direction permitted by the neighborhood. Before receiving any information from the axes, a cell has no way of knowing the precise direction to the origin. If the neighborhood's convex hull has more than one vertex in the positive quarter of the plane, moving along any of the directions permitted by the neighborhood might be sub-optimal, as oppposed to the case of the Moore neighborhood in which going diagonally at first is never sub-optimal and by the time it is necessary to change direction to go either horizontally or vertically information is received from the axes.

If only one cell needs to send its information towards the origin, the problem can be solved by spreading the information in all directions and spreading symmetric signals from the origin. It is however not possible to implement this for all cells at the same time with finitely many states.

Acknowledgments. The authors would like to thank Jacques Mazoyer for his helpful conversations and inspiring ideas at the start of the work that led to this article.

References

1 J. Albert and K. Čulik II. A simple universal cellular automaton and its one-way and totalistic version. *Complex Systems*, 1:1–16, 1987.

2 W.T. Beyer. *Recognition of topological invariants by iterative arrays*. Massachusetts Institute of Technology, Project MAC, 1969.

3 Stephen N. Cole. Real-time computation by n-dimensional iterative arrays of finite-state machines. *IEEE Transactions on Computers*, C-18(4):349–365, 1969.

4 Martin Delacourt and Victor Poupet. Real time language recognition on 2d cellular automata: Dealing with non-convex neighborhoods. In Ludek Kucera and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, volume 4708 of *Lecture Notes in Computer Science*, pages 298–309. Springer, 2007. doi:10.1007/978-3-540-74456-6_28.

5 P. C. Fischer. Generation of primes by one-dimensional real-time iterative array. *Journal of the Assoc. Comput. Mach.*, 12:388–394, 1965.

6 F.C. Hennie. *Iterative Arrays of Logical Circuits*. MIT Press Classics. MIT Press, 1961.

7 Jacques Mazoyer and Nicolas Reimen. A linear speed-up theorem for cellular automata. *Theor. Comput. Sci.*, 101(1):59–98, 1992. doi:10.1016/0304-3975(92)90150-E.

8 Zsuzsanna Róka. Simulations between cellular automata on Cayley graphs. *Theoretical Computer Science*, 225(1-2):81–111, 1999.

9 Alvy R. Smith III. Simple computation-universal cellular spaces. *J. ACM*, 18(3):339–353, 1971. doi:10.1145/321650.321652.

10 Alvy R. Smith III. Two-dimensional formal languages and pattern recognition by cellular automata. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (Swat 1971)*, SWAT'71, pages 144–152, Washington, DC, USA, 1971. IEEE Computer Society. doi:10.1109/SWAT.1971.29.

11 Alvy R. Smith III. Real-time language recognition by one-dimensional cellular automata. *Journal of the Assoc. Comput. Mach.*, 6:233–253, 1972.

12 Véronique Terrier. Two-dimensional cellular automata recognizer. *Theor. Comput. Sci.*, 218(2):325–346, 1999. doi:10.1016/S0304-3975(98)00329-6.

13 Véronique Terrier. Two-dimensional cellular automata and their neighborhoods. *Theor. Comput. Sci.*, 312(2-3):203–222, 2004. doi:10.1016/j.tcs.2003.08.011.

14 John von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, IL, USA, 1966.