Theoretical Computer Science 559 (2014) 108–117

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

The firing squad synchronization problem on CA with multiple updating cycles

Luca Manzoni [a,*], Hiroshi Umeo [b]

[a] Laboratoire i3S, Université Nice Sophia Antipolis, CS 40121 06903 Sophia Antipolis Cedex, France
[b] Division of Information and Computer Sciences, Osaka Electro-Communication University, Neyagawa-shi, Hastu-cho, 18-8, Osaka 572-8530, Japan

ARTICLE INFO

Article history:
Received 15 August 2013
Received in revised form 22 June 2014
Accepted 7 August 2014
Available online 23 August 2014

Keywords:
Firing squad synchronization problem
Cellular automata
Asynchronicity

ABSTRACT

Classical cellular automata have a single, global clock that allows all the cells to update their states simultaneously at every clock cycle. Here, we overturn this assumption and we study cellular automata where different cells can use different clocks cycles and, as a consequence, they update at different times. We show how it is possible to solve the firing squad synchronization problem in this new setting and propose a synchronization algorithm operating in time linear with respect to the automata size.

1. Introduction

It has been believed that synchronous updating under a single global clock is a key condition for constructing synchronization algorithms in cellular automata (CA). A question is how we can relax the restriction of synchronous updating in the firing squad synchronization problem (FSSP).

One of the approaches to introduce asynchronicity in the FSSP is to replace the single global clock by multiple clocks, each one responsible for synchronizing the update of only a part of the cells. In fact, the presence of a single global clock is based on the strong assumption that it is possible to propagate the clock among arbitrarily far cells. The presence of multiple clocks allows us to remove this assumption while introducing additional challenges that were not present in the single clock model.

Hence, here we tackle the problem of solving the classical FSSP using CA where cells update with different speeds (i.e., the update is not performed at every time step for all the cells). This new kind of CA is part of a larger trend whose main aim is to study the effect of asynchronicity in CA.

1.1. Asynchronous CA

The work presented here is a part of the study of asynchronous CA [19,22,29]. While a broadly accepted definition of "what is asynchronicity in CA" is missing, a common theme to all approaches is that the cells of the automata do not update themselves all at the same time (i.e., there could be active and inactive cells in a particular time step). The way of introducing this behavior varies depending on the approaches taken. We recall, for example, α-asynchronous CA [6,21], fully-asynchronous CA [5,13], probabilistic CA [7], and some other approaches to asynchronicity [4,3].

* Corresponding author.
E-mail addresses: luca.manzoni@i3s.unice.fr, luca.manzoni@disco.unimib.it (L. Manzoni), umeo@cyt.osakac.ac.jp (H. Umeo).

http://dx.doi.org/10.1016/j.tcs.2014.08.011
0304-3975/

Here we deal with cells that have different clock cycles. That is, a cell updates only every $k \in \mathbb{N}$ time steps (i.e., with speed $\frac{1}{k}$) where k is not uniform across all the cells of the lattice. We show that, when there is a known lower bound on the possible updating speeds, we can solve the FSSP. Furthermore, we show that a solution is impossible if the set of possible speeds is unknown and the considered CA has at least two cells. Intuitively, when a lower bound is present then it is possible to "slow down" the fast cells to allow a synchronization with the slow ones. When no bound is present it is always possible to insert a cell slow enough to make this synchronization process impossible.

1.2. Firing squad synchronization problem

The FSSP was introduced more than fifty years ago [17] and then extensively studied. Its goal is to have a line of n identical automata, all in the same state except the first (the *general*), reach for the first time a particular state (the *firing state*) all at the same time. The first solution for the classical one-dimensional (1D) case is due to McCarthy and Minsky [16]. Subsequently, optimal time solutions were found [9,28,1,8], with the one requiring the smallest number of states (6 states) due to Mazoyer [15]. In the years the research interest was also on the study of variations of the FSSP. From the generalization to 2D arrays [2,10,23] to solutions that work for generals in a position that is not the first [18,24,27] or that limit the communication to one bit between cells [20,26].

Here we study a generalization in which the behavior of the cells of the automaton is not uniform. We can identify at least two other approaches in this direction. The first one is the introduction of *faulty cells* in the CA [25,11,12,30]. In this case there are some damaged cells and every signal entering them moves at speed 1/1. Umeo proved that a synchronization for the non-faulty cells is possible [25] and gave bounds on the synchronization time. The other one is due to Mazoyer [14], that introduced cells in which there is a non-uniform delay in the communication between cells.

The paper is organized as follows. In Section 2 the asynchronous CA with multiple updating speeds is formally defined. In Section 3 we present a way to solve the FSSP, first when only two speed are present and then when multiple speeds appear. The paper ends with some further remarks in Section 4.

2. Multiple speeds CA

In this section we introduce the concept of *multiple speeds CA*. This kind of automata allows the cells to update with different speeds.

Definition 2.1. A 1D multiple speeds CA (MSCA) *with known speeds* is a 5-tuple $\mathcal{C} = (Q, \mathbf{b}, \lambda, S, s)$, where Q is a finite set of states, $\mathbf{b} \notin Q$ is a special state called the border state, $\lambda : ((Q \cup \{\mathbf{b}\}) \times S) \times (Q \times S) \times ((Q \cup \{\mathbf{b}\}) \times S) \to Q$ where $S = \{1/k \mid k \in \mathbb{N}_+\}$ is the local rule, and $s : \{1, \ldots, n\} \to S$ is the speed function.

An MSCA of length $n \in \mathbb{N}_+$ is a lattice of n cells, indexed from 1 to n, having states in Q and with two border cells in position 0 and $n+1$ having state $\mathbf{b}$. Recall that a configuration is a word in Q^n and a state $\mathbf{q} \in Q$ is *quiescent* iff $\lambda(\mathbf{q}, \mathbf{q}, \mathbf{q}) = \lambda(\mathbf{b}, \mathbf{q}, \mathbf{q}) = \lambda(\mathbf{q}, \mathbf{q}, \mathbf{b}) = \lambda(\mathbf{b}, \mathbf{q}, \mathbf{b}) = \mathbf{q}$ (i.e., the quiescent state is preserved if all neighbors are in the quiescent state or the border state).

The speed function s assigns to every cell in position $i \in \{1, \ldots, n\}$ a rational number $s(i)$, which is called the *cell speed*, whose meaning is that the cell i updates only every $1/s(i)$ steps.

Given a configuration $c \in Q^n$ at time $t - 1 \in \mathbb{N}$, the next state configuration is given by a global function Λ_t defined for $i \in \{1, \ldots, n\}$ as:

$$\Lambda_t(c)_i = \begin{cases} \lambda(c_{i-1}, s(i-1), c_i, s(i), c_{i+1}, s(i+1)) & \text{if } t \equiv 0 \pmod{1/s(i)} \\ c_i & \text{otherwise} \end{cases}$$

Note that in this case the local rule can depend on the speed of the cells. Since, by the definition, the border cells have no assigned speed and do not update, we can give any values to $s(i) = 0$ when $i \notin \{1, \ldots, n\}$. Furthermore, notice that there is more than one global transition function because the set of cells that update depends on the current time step. In fact, if $s(\{1, \ldots, n\}) = \{\frac{1}{k_1}, \ldots, \frac{1}{k_h}\}$ is the image of the speed function on an automaton having n cells, then the number of different distinct global functions is equal to the least common multiple of $k_1, \ldots, k_h$. From now on, when there is no ambiguity, we will use the notation Λ^t as a shorthand for $\Lambda_t \circ \Lambda_{t-1} \circ \cdots \circ \Lambda_1$. To denote the configuration of the automata at time $t \in \mathbb{N}$ we will use the notation $c(t)$, with $c_i(t)$ denoting the i-th cell of the configuration of the automaton at time t.

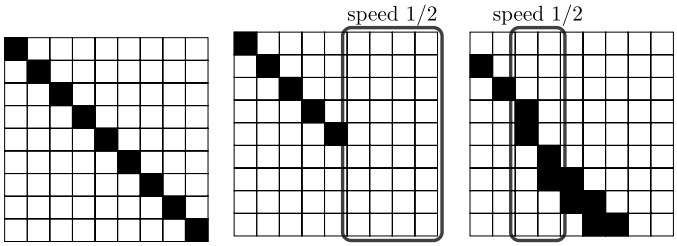
Remark 2.1. Notice how the definition of the MSCA preserves some of the interesting properties of classical CA. In particular the computational power of the single cells of the CA remains the same, since they are still only finite state machines, and the local rule remains uniform across all the cells. The removed characteristic is the global clock allowing all the cells to update at the same time, that is now substituted by multiple, independent clocks. Another way to see this modification is that multiple global clock exist, which is an assumption that does not appear stronger that stating the existence of a single global clock.

Now fix three state, $\mathbf{g}, \mathbf{f}$ and $\mathbf{q}$ and a finite set $S \subset \mathbb{N}$. A solution to the FSSP for an MSCA with speeds S is a set of states Q, with $\mathbf{g}, \mathbf{f}, \mathbf{q} \in Q$, and a local rule λ with the quiescent state $\mathbf{q}$ such that for all $n \in \mathbb{N}$ and for all speed functions with range S, starting from the configuration $\mathbf{g}\mathbf{q}^{n-1}$ there exists a time $t \in \mathbb{N}$ in which the configuration of the MSCA is $\mathbf{f}^n$ and $\mathbf{f}$ has not appeared as a state of any cell before time t.

2.1. Comparison with automata with non-uniform delay

The class of automata studied here share some characteristics with automata with non-uniform delay [14]. However, there are some differences. First of all, in automata with delay a signal never disappears, it is only received with some delay. However, in CA with different speeds, a signal can disappear because of an inactive cell that does not receive it when it arrives (or it can be duplicated because a cells reads it two times), as it can be seen in Example 2.1. Therefore, the issues that we need to address in the two settings are different. In automata with delay the problem is the non-uniformity of the delay. In CA with speeds the problems is the correct preservation of the signals. Furthermore, note that a cell at speed 1 has no delay in receiving a signal from a cell at a different speed. However, a cell with speed $1/2$ can have either 0 or 1 time steps of delay. This means that we have delays that are neither symmetric nor constant.

Example 2.1. In this example we show how a signal can disappear or be duplicated in an MSCA. Let $\mathcal{C} = (\{0, 1\}, \mathbf{b}, \lambda, S, s)$ be an MSCA where λ is a shift local rule. In the leftmost figure (a), all cells have speed 1 and a signal moves from left to right. In the central figure (b), the cells with index 6 or more have speed $\frac{1}{2}$, and the signal disappear when reaching those cells. Finally, in the leftmost figure (c), the signal correctly enters a speed $\frac{1}{2}$ zone, but, when exiting it, the signal appears duplicated.



3. Solving the FSSP

Here we present a solution for the synchronization of an MSCA with only two possible speeds, 1 and $1/k$ for some fixed $k \in \mathbb{N}_+$. We prove that the value of k must be either known or at least bounded for the synchronization to be possible.

3.1. Impossibility of synchronization

However, we claim that a local rule that is able to synchronize even two cells for all possible speeds cannot exist. Therefore, in the remaining part of the paper we will describe how to solve the FSSP when there is a lower bound on the possible speeds.

Proposition 3.1. *Let $\mathcal{C} = (Q, \mathbf{b}, \lambda, S, s)$ be any MSCA with known speeds with two cells and such that $s(1) = 1$ and $s(2) < \frac{1}{|Q|}$. Then λ cannot solve the FSSP for the MSCA.*

Proof. By contradiction, assume that λ is able to synchronize the array. Then, consider the last $|Q|+1$ elements of the orbit ending with $\mathbf{ff}$, that is:

$$\Lambda^k(\mathbf{gq}), \Lambda^{k+1}(\mathbf{gq}), \Lambda^{k+2}(\mathbf{gq}), \ldots, \Lambda^{k+|Q|}(\mathbf{gq}) = \mathbf{ff} \tag{1}$$

for some $k \in \mathbb{N}$. Note that synchronization requires at least $|Q|$ time steps, hence this construction is always possible. Since $\frac{1}{s(2)} > |Q|$, we can rewrite (1) as:

$$\mathbf{rp}, \lambda(\mathbf{b}, \mathbf{r}, \mathbf{p})\mathbf{p}, \ldots, \lambda^{|Q|-1}(\mathbf{b}, \mathbf{r}, \mathbf{p})\mathbf{p}, \lambda^{|Q|}(\mathbf{b}, \mathbf{r}, \mathbf{p})\mathbf{f} = \mathbf{ff}$$

for some $\mathbf{r}, \mathbf{p} \in Q$. To ease the notation, call $c_1(i)$ the state $\lambda^i(\mathbf{b}, \mathbf{r}, \mathbf{p})$. Note that the sequence $c_1(0), \ldots, c_1(|Q|-1)$ must respect the condition that $\forall i \in \{0, \ldots, |Q|-1\}\ c_1(i) \neq \mathbf{f}$, that implies that $\forall i, j \in \{0, \ldots, |Q|-1\}$, $i \neq j \Rightarrow c_1(i) \neq c_1(j)$. This condition is necessary, otherwise $c_1(|Q|-1)\mathbf{p}$ is a configuration that has appeared before as $c_1(j)\mathbf{p}$ (with $j < |Q|-1$) and, as a consequence, we would have $c_1(j+1) = \mathbf{f}$. However, since there are only $|Q|-1$ possible states and $|Q|$ elements in

the sequence $c_1(0), \ldots, c_1(|Q|-1)$, this condition is unattainable. Thus the initial assumption was inconsistent (i.e., λ is unable to synchronize the array). □

Since not knowing the minimal possible speed of the cells does not allow us to achieve the synchronization, it is reasonable to fix a *minimum speed* and design not a single local rule but a family of local rules indexed by the smallest minimum speed for which they can successfully synchronize the array. Formally, instead of having a single MSCA $\mathcal{C}$ able to solve the FSSP for all possible speeds, we need to define a mapping that, to each $k \in \mathbb{N}$, associates an MSCA $\mathcal{C}_k$ that solves the FSSP for all speed functions that does not assign to any cell a speed smaller than $\frac{1}{k}$. Proposition 3.1 assures us that such a mapping cannot simply return the same MSCA for all $n \in \mathbb{N}$ and, as a consequence, that using a family of MSCA instead of a single MSCA is necessary in order to solve the FSSP. In the rest of the paper we will show that, for each classical solution of the FSSP, we can build such a family, thus providing a possibly non-exhaustive classification of such families (i.e., there exists at least one family of solution for the FSSP in the MSCA setting for each solution of the FSSP for classical CA).

3.2. Solution for two speeds

The main idea is to modify an existing rule for synchronization in order to use the 1/1 signal sent from the general as a "probe" that slow down the cells of speed 1 when it reaches a cell of speed $\frac{1}{k}$. To accomplish this goal we provide a general method to build from a solution for the classical FSSP a solution of the FSSP for the MSCA.

Let $(Q, \mathbf{b}, \lambda)$ be a synchronous CA that solves the FSSP, we will refer to it as the *original CA*, since it will be used as a base to define a solution for a two-speeds MSCA. Similarly, we will refer to its alphabet (resp., local rule) as the original alphabet (resp., original local rule). Then we define a new MSCA $\mathcal{C} = (Q', \mathbf{b}, \lambda', S, s)$ that solves the FSSP in the multiple speeds case when the image of s is $\{1, \frac{1}{k}\}$ for some $k \in \mathbb{N}$. The alphabet Q' is defined as follows:

$$Q' = \left(Q \setminus \{\mathbf{f}\}\right)^2 \times \{0,1\} \times \{0, \ldots, k-1\} \times \{1, k\} \cup \{\mathbf{q}, \mathbf{f}\}.$$

The state of a cell c_i is one of the followings: **q**, **f**, or a state containing the following information:

- **Original state**: the state of the original solution. This will be denoted by $\mathfrak{o}(c_i)$.
- **Previous state**: the state at the previous time step. This is used in the unfreezing process. The previous state will be denoted by $\mathfrak{p}(c_i)$.
- **Frozen**: each cell has one bit of information that is 1 when the cell is frozen and 0 otherwise. This bit will be denoted by $\mathfrak{f}(c_i)$.
- **Timer**: a timer that, for cell of speed 1, keeps track of the current time modulo k. The timer will be denoted by $\mathfrak{t}(c_i)$.
- **Cell Speed**: a value that cells of speed 1 use to check if they must update the value $\mathfrak{o}(c_i)$ of their state as normal or if they must update it only every k steps (i.e., they "simulate" a cell of speed $\frac{1}{k}$). This information will be denoted by $\mathfrak{s}(c_i)$.

Remark 3.1. Notice that if we start from a 6 states solution, the solution for the multiple speed has $100 \cdot k + 2$ states. That is, in the simplest non-trivial case (i.e., $k = 2$) there are 202 states. Since the construction that we are going to present is not dependent from the solution for the FSSP that we take as the starting point, we have that, in general, if we start from an m states solution, and we have the slowest cell with speed $\frac{1}{k}$, then the number of states of the resulting solution for the MSCA is $4k(m-1)^2 + 2$. Hence, the local rule is still representable as a finite state automaton when k is fixed (and, when it is not, Proposition 3.1 assure us that a finite state automaton is not sufficient).

We will say that a cell c_i updates its original state when the original state of a cell is updated according to the original local rule λ. That is, given a configuration $c \in Q'^n$, $b = \lambda'(c_{i-1}, c_i, c_{i+1})$ is such that $\mathfrak{o}(b) = \lambda(\mathfrak{o}(c_{i-1}), \mathfrak{o}(c_i), \mathfrak{o}(c_{i+1}))$. Notice that we can have a cell that updates with speed 1 but, when only the original state is considered, it cannot be distinguished by one with speed $\frac{1}{k}$ if, by using a timer modulo k, it updates its original state only every k steps.

The initial state of the automaton is $\mathbf{g}'\mathbf{q}^{n-1}$ where the general $\mathbf{g}'$ is the 5-tuple $(\mathbf{g}, \mathbf{g}, 0, 0, 1)$, with $\mathbf{g}$ the general of the original solution. We define the *frontier* for a certain time t as the two adjacent cells of which the one to the right is the leftmost one in the state **q**. This is, in some sense, a frontier between the part of the automaton that has started the synchronization and the currently inactive part. Hence, we can recognize three different areas in which a configuration can be divided:

- The area from the initial cell to the frontier. In this area the cell may be frozen or unfrozen. However, neither freezing nor unfreezing signal is ever generated in this area.
- The frontier. Here the freezing and unfreezing signals must be generated.
- The remaining part of the CA contains only quiescent cells.

Every cell between the beginning of the array and the frontier when it is active increments its internal timer modulo k. That is, every cell of speed 1 has the necessary information to know when a cell of speed $\frac{1}{k}$ will activate.

We present a sketch of the algorithm that represents the local rule λ' (in the pseudocode *lcm* is the least common multiple) in Algorithm 1.

```
c_i(t+1) ← c_i(t)
if c_i(t) = q and c_{i−1}(t) = q then
    return
end
if c_i(t) = q and (c_{i−1}(t) = q or o(c_{i−1}(t)) = q) then
    time ← (t(c_{i−1}(t)) + 1) mod (k · s(i))
    if s(i − 1) = 1/k then
        time ← 2 mod (k · s(i))
    end
    speed ← s(i)lcm(1/s(i), s(c_{i−1}(t))/s(i−1))
    c_i(t+1) = (q, q, 0, time, speed)
    if time ≡ 0 mod speed then
        o(c_i(t+1)) ← λ(o(c_{i−1}(t)), o(c_i(t)), o(c_{i+1}(t)))
        if speed/s(i) = 1 and s(i+1) = 1/k then
            sc_i(t+1) ← k
            fc_i(t+1) ← 1
        end
    end
    return
end
t(c_i(t+1)) ← (t(c_i(t)) + 1) mod (k · s(i))
if t(c_i(t)) ≢ 0 mod s(c_i(t)) then
    return
end
if f(c_i(t)) = 1 then
    if f(c_{i+1}(t)) = 0 then
        f(c_i(t+1)) ← 0
        if f(c_{i−1}(t)) = 0 then
            o(c_i(t+1)) ← λ(p(c_{i−1}(t)), o(c_i(t)), o(c_{i+1}(t)))
        else
            o(c_i(t+1)) ← λ(o(c_{i−1}(t)), o(c_i(t)), o(c_{i+1}(t)))
        end
    end
    return
end
o(c_i(t+1)) ← λ(o(c_{i−1}(t)), o(c_i(t)), o(c_{i+1}(t)))
if f(c_{i+1}(t)) = 1 then
    f(c_i(t+1)) ← 1
    s(c_i(t+1)) ← s(i)lcm(1/s(i), s(c_{i+1}(t))/s(i+1))
end
```

Algorithm 1: The pseudocode of a solution for the FSSP with two speeds.

Notice that, for the sake of simplicity, the pseudocode presented is not shown to save the previous state at every update and to treat the case of the general being in a cell of speed 1 with the second cell at speed $\frac{1}{k}$ (that must be considered separately since the code assumes the cell generating the freezing signal to be originally in a quiescent state).

Now we are going to describe the different signals that are generated and the details of the local rule λ'. In particular, when the freezing and unfreezing signals are generated, what happens when the cells freeze and unfreeze, and how the cells of speed 1 recognize when they must start updating their original state every k steps instead of at every step. An example of the signals generated by the updating rule and a synchronization of an array is presented in Fig. 1 and the space time diagram on Fig. 2.

Generation of the freezing signal

The freezing signal is generated in the frontier when the right cell $c_i(t)$ of the frontier has speed 1 but $c_{i+1}(t)$ has speed $\frac{1}{k}$. The cell $c_i(t+1)$ is frozen and has a *cell speed* equal to k (i.e., $\mathfrak{s}(c_i(t+1)) = k$). The unfreezing signal is generated when the cell in position $i+1$ updates its state (i.e., the first $t' > t$ when $\mathfrak{t}(c_i(t')) = 0$). Hence, up to $k-1$ time steps could pass after the generation of the freezing signal and before the generation of the unfreezing signal.

Freezing and unfreezing

When a cell $c_i(t)$ has its right neighbor that is in the frozen state, it becomes frozen (i.e., when $\mathfrak{f}(c_{i+1}(t)) = 1$). If its speed is 1, then $\mathfrak{s}(c_i(t+1)) = k$. When a cell that is in its frozen state has its right neighbors that is not frozen, it exits the frozen state. Notice that when a cell updates its original state after exiting from the frozen state at time t, if $\mathfrak{f}(c_{i-1}(t)) = 1$ it needs to use the state $\mathfrak{p}(c_{i-1}(t))$ instead of $\mathfrak{o}(c_{i-1}(t))$ to update its state.

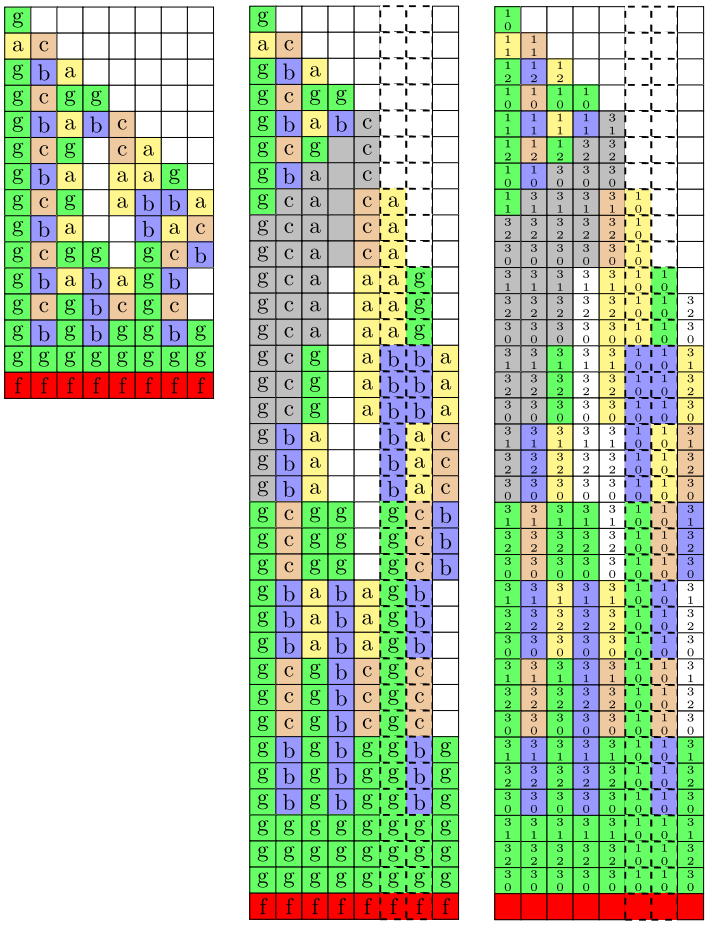


Fig. 1. The leftmost figure is an example of the original Mazoyer's rule [15] while the other two are an example in which the adapted rule is used. The dashed cells have speed $\frac{1}{3}$ and the frozen cells are represented by a gray color. On the central automata every cell is labeled with its original state. On the right every cell contains – when they exist – the value of the *cell speed* and of the *timer*.

Adaptation of the speed

When the quiescent cell $c_i(t)$ in the frontier updates, if it has speed 1 it must check if the cell $c_{i-1}(t)$ has either speed $\frac{1}{k}$ or if $\mathfrak{s}(c_{i-1}(t)) = k$. In that case the cell must also change its *cell speed* to k (i.e., $c_i(t+1) = k$).

Normal update

When no other condition is present a cell simply updates its original state in one of the following way:

- $\mathfrak{o}(c_i(t+1)) = \lambda(\mathfrak{o}(c_{i-1}(t)), \mathfrak{o}(c_i(t)), \mathfrak{o}(c_{i+1}(t)))$ only when $s(i) = \frac{1}{k}$ or $\mathfrak{s}(c_i(t)) = 1$ or $\mathfrak{t}(c_i(t)) \equiv 0 \bmod k$.
- $\mathfrak{o}(c_i(t+1)) = \mathfrak{o}(c_i(t))$ otherwise.

In all cases $\mathfrak{t}(c_i(t+1)) = (\mathfrak{t}(c_i(t)) + 1) \bmod \frac{1}{s(i)}$.

It is important to note that after the freezing and unfreezing processes all the cells that participated to the process has updated the original state in the same number of time. That is, after the unfreezing process we are still "simulating" the original local rule, only with speed $\frac{1}{k}$.

For a configuration $c \in Q'^n$, we can define the projection $\pi(c)$ of c to Q as follows: for all $i \in \{1, \ldots, n\}$,

$$\pi(c)_i = \begin{cases} c_i & \text{if } c_i = \mathbf{q} \text{ or } c_i = \mathbf{f} \\ \mathfrak{o}(c_i) & \text{otherwise} \end{cases}$$

We are now going to show how the proposed solution simulates the original solution in the sense that, when considering a solution to the classical FSSP and its adaptation according to the previously described construction, they behave in the same way (i.e., the projection to Q of the solution for the MSCA is the same as the solution for the CA) when all cells of speed less than one are quiescent and have quiescent cells as neighbors. Intuitively, this means that there is no penalty (with respect to time needed to attain synchronization) when no slow cells are present. Furthermore, we require that when

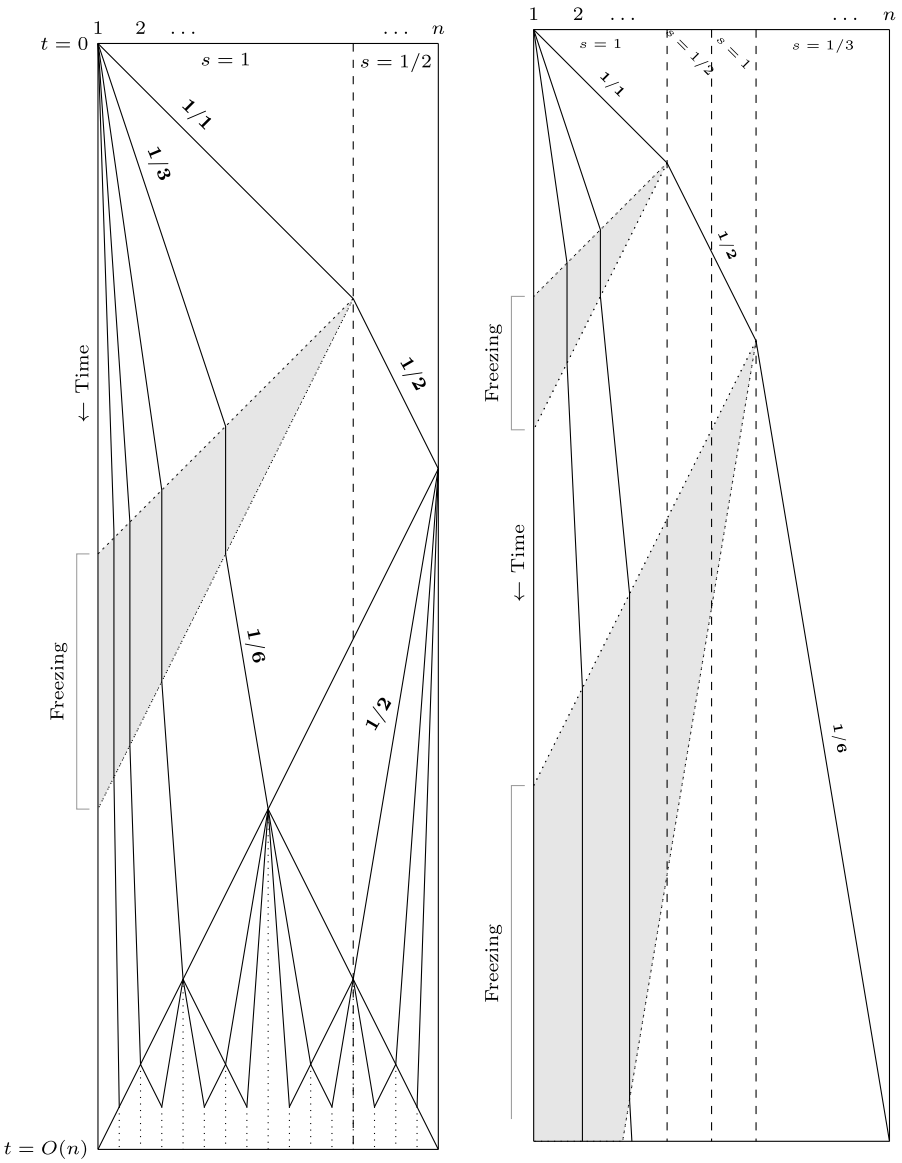


Fig. 2. The space–time diagrams of the solution of FSSP with two speeds (on the left) and with three speed speeds (on the right), both using the Waksman algorithm [28].

slow speed cells are present, there exists a time step t after which the MSCA behaves as the classical CA, in the same sense used above, but with cells updating every k time steps. The following proposition formalizes this notion of simulation.

Proposition 3.2. *Let $\mathcal{C} = (Q, \mathbf{b}, \lambda)$ be a CA solving the FSSP for every size $n \in \mathbb{N}$. Let $\mathcal{C}' = (Q', \mathbf{b}, \lambda', S, s)$ be an MSCA defined starting from $\mathcal{C}$ with the procedure of Section 3.2, with minimal speed $\frac{1}{k}$ for some $k > 0$. Finally, let ℓ be the rightmost cell of speed 1 not preceded by a cell of speed $\frac{1}{k}$. Then the following properties hold:*

(1) $\forall t < \ell$, $\pi(c'(t)) = c(t)$.
(2) $\forall t > 2\ell - 2$, $\pi(c'(t' + u)) = c(t)$ *for all* $u \in \{1, \ldots, k\}$ *and for*

$$t' = (k+1)(\ell - 1) + h + k\big(t - 2(\ell - 1) - 1\big),$$

where $h = \min\{j \in \mathbb{N} \mid \ell - 1 + j \equiv 0 \bmod k\}$.

Proof. Property (1) holds by construction, since the local rule λ' behaves as λ when no cell of speed less than 1 has been met.

To show that property (2) holds, we will distinguish three cases in which a cell of $\mathcal{C}'$ can be between time ℓ and $2\ell-2$ to show that the property holds for $t=2\ell-1$ and, as a consequence, for each successive time step. The cases are:

- **Updating at speed** 1. This situation only happens for cells between position 1 and ℓ that have not been reached by the freezing signal. The freezing signal is generated immediately after cell ℓ changes, for the first time, from a quiescent to a non-quiescent state, that is, at time step ℓ. The signal travels from position ℓ to 1 with speed 1. Hence, a cell in position $i \in \{0,\dots,\ell\}$ performs $\ell-i+1$ updates before being reached by the freezing signal.
- **Frozen**. While frozen, a cell does not update its state before the unfreezing signal reaches it. The unfreezing signal is generated at the first time step greater than $\ell-1$ that is congruent to 0 modulo k, i.e., h time steps after the generation of the freezing signal. Since the unfreezing signal travels with speed $\frac{1}{k}$, a cell in position $i \in \{0,\dots,\ell\}$ remains frozen for $h+k(\ell-i)$ time steps.
- **Updating at speed** $\frac{1}{k}$. Finally, the cells with index not greater then ℓ, after being unfrozen, update their original state every k time steps. Before the unfreezing signal disappears, each of those cells updates $i-1$ times, where i is the position of the cell. Also the cells in position greater than ℓ updates every k steps and they update $\ell-1$ times.

Hence, before the generation of the freezing signal $\ell-1$ updates of the original state are performed in $\ell-1$ time steps of the MSCA. Between the generation of the freezing signal and the disappearance of the unfreezing signal, the number of updates to the original state performed were $(\ell-i+1)+(i-1)=\ell$ (independently of the cell position i) in a total of $h+k(\ell-1)$ time steps of the MSCA. Hence, at time $(\ell-1)+k(\ell-1)+h+1$ the state of the MSCA is, considering only the first component, equal to the one of the original FSSP solution after $2\ell-1$ time steps. By noticing that, by construction, after the generation of the disappearance of the freezing signal, all the cells of the MSCA will update the original state every k time steps (i.e., at speed $\frac{1}{k}$, either because it is the speed of the cell or because of the use of timers hidden in the local state), the statement of the proposition follows. □

Corollary 3.3. *The synchronization time of an MSCA of n cells with two speeds,* $\{1, \frac{1}{k}\}$*, is the following, depending of the position* ℓ *of the first cell such that* $s(\ell+1)=\frac{1}{k}$*:*

- $2n-2$ *if no cell of speed* $\frac{1}{k}$ *exists.*
- $(k+1)(\ell-1)+h+k(2n-2\ell-1)+1$ *if* $1 \le \ell \le n$*, where h defined as in Proposition 3.2.*

Proof. Since the time needed to synchronize a classical CA with an optimal solution to the FSSP, is $2n-2$, it is possible to substitute this value into the bounds for the equality of configurations given in Proposition 3.2 in order to obtain the bounds of the statement of this corollary. □

Remark 3.2. In this work we have considered only the case in which the cells know their speed; that is, the local function depends on the speed of the cells. However, other situations are possible. For example, it is possible to have a local rule that depends only on the state of the cells (and not of their speed) but, by saving in the state the speed of the cell, this additional restriction can be easily overcome. Another possible restriction is that the cells may not have saved in the state any information about their speed. Even in this case it is possible to attain a solution to the FSSP by first providing a solution to tag each cell with its speed (Example 2.1 shows that regions of cells having different speeds propagate signals in different ways and, hence, can be distinguished) and then start the solution for two speeds presented here.

3.3. Solution with multiple speeds

Here we present a direct extension of the procedure presented in the previous section to the case when more than two speeds are present in an MSCA.

The idea of synchronization when more than two speeds are present is similar to the one used for two speeds. That is, the simulation proceeds at a certain speed and, when a cell of different speed is firstly encountered, a combination of a freezing and an unfreezing signal is generated to adapt the speed of the CA. Since there are more than two speeds this process can happen more than one time. For example if firstly a cell of speed $\frac{1}{2}$ is encountered then a pair of signal is generated to have the update of the original state performed only every 2 time steps. If a cell of speed $\frac{1}{4}$ is then encountered the process needs to be performed again to have all the cells of the CA update their original state only every 4 time steps. Another example is presented in Fig. 3 (and an example highlighting only the signals on Fig. 2). Notice how when there are two speeds $\frac{1}{k_1}$ and $\frac{1}{k_2}$ with neither k_1 dividing k_2 or k_2 dividing k_2 then to obtain an update of the original state that can be performed by the cells in both speeds at the same time we need to perform the update at least every $lcm(k_1,k_2)$, i.e., the least common multiple of k_1 and k_2.

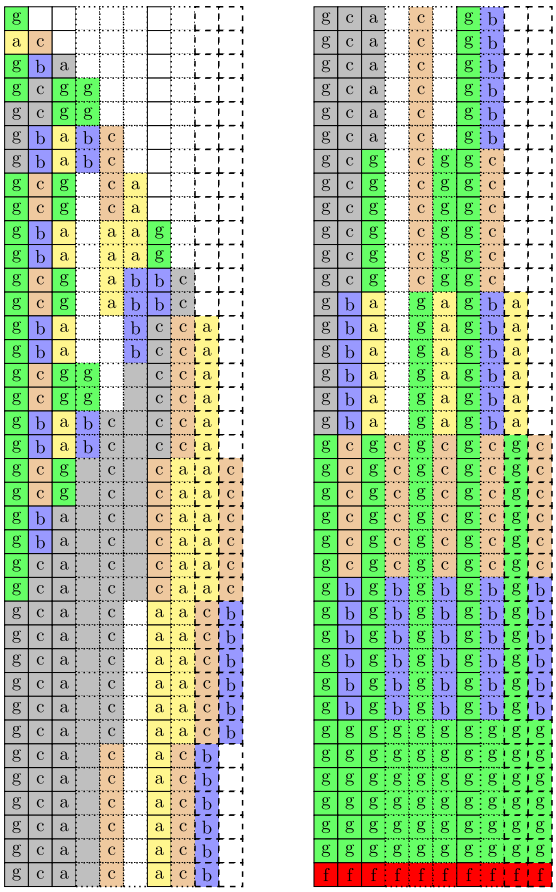


Fig. 3. An example of a synchronization with three speeds. The dotted cells have speed $\frac{1}{2}$ and the dashed ones $\frac{1}{3}$. Frozen cells are colored in gray.

Let $K = \{\frac{1}{k_1}, \frac{1}{k_2}, \ldots, \frac{1}{k_m}\}$ be the set of possible speeds that can be present in the MSCA. To correctly perform the update we can use *timer* and the *cell speed* component of the state of every cell. The only modification of the solution for two speeds are the following:

- The timer $\mathfrak{t}(c_i)$ of a cell needs to count modulo $lcm(k_1, \ldots, k_m)$ since, when all speeds are present in the MSCA, this is the number of steps that must pass between two updates of the original state.
- The cell speed $\mathfrak{s}(c_i)$ can have more values than simply 1 and k. In fact, for every subset $H = \{\frac{1}{k_{i_1}}, \ldots, \frac{1}{k_{i_p}}\}$ of K, $lcm(k_{i_1}, \ldots, k_{i_p})$ is a possible value for the cell speed since it is the number of steps that a cell having speed 1 must wait before updating when cells of speed $k_{i_1}, \ldots, k_{i_p}$ have been encountered.

With those differences in the states, the only remaining variation is that when a cell of speed different from the ones already met is reached, the new "common speed" of the algorithm must be the least common multiple of the reciprocal of all currently encountered speeds. This was also true for the two speeds case, where it was trivial since the least common multiple of 1 and k is k. The synchronization time for multiple speeds remains linear in the size of the CA, but the exact time depends not only on the set of speed but also on their distribution in the array.

Remark 3.3. The solution presented for the multiple speeds case can also be used to deal with speeds that are any positive rational number not greater than one by means of a rescaling. For example, suppose that an MSCA has cells with speeds 1 and $\frac{2}{3}$. It can also be considered as an MSCA with two speeds $\frac{1}{2}$ and $\frac{1}{3}$, obtained by multiplying the original speeds by $\frac{1}{2}$. In general, it is possible to return to a case in which all the speeds have one at the numerator by multiplying each speed by the least common multiple of all the numerators of all the speeds that occur in the automaton. Since the multiple speeds solutions can be applied to such "rescaled" automaton, there is no loss of generality in considering only speeds of the form $\frac{1}{k}$.

4. Further remarks

In this work we have introduced a new kind of CA in which there is no single global clock but multiple local ones. This means that the cells require a number of steps to update that can be non-uniform. We have studied two conditions. The first one is when only two clocks are present and the second one allows multiple clocks to exist. We have shown that under mild conditions – mainly that there exists a lower bound on the possible speeds – it is possible to solve the FSSP.

It is important to notice that the proposed procedure to slow down cells as needed can also be carried in all the cases in which the initial configuration has all but one extremal cells in a quiescent configuration. All the problems with such a characteristic can also be solved with the MSCA model by slightly adapting the solution for the FSSP that we have proposed.

An FSSP on 2D MSCA model would be an interesting future extension. The main difficulty to obtain an efficient solution of the FSSP for 2D MSCA is that, differently from the one-dimensional case, a single signal traveling from the general to the right may not encounter all the speeds that are present in the automaton. Therefore, it seems necessary to coordinate multiple signals in order to "discover" all the speeds that are present in a 2D automaton.

Future work will involve a search for the optimal synchronization time in this setting and if a reduction of the synchronization time is possible. Furthermore, it remains interesting to study a reduction in the number of states needed for the synchronization, that is currently much higher than the number found in the most recent solutions in the synchronous case. A study of the MSCA and their effect in our ability to solve other problems in this new setting is also an interesting research question.

Acknowledgements

The first author was supported by the Japanese Society for the Promotion of Science (JSPS) under the JSPS Postdoctoral Fellowship Program (Short-term) for North American and European Researchers (Gaikokujin Tokubetsu Kenkyuin (Obei-Tanki)), fellowship number PE12067. This work was partially supported by the by the French National Research Agency project EMC (ANR-09-BLAN-0164) and by the Italian MIUR PRIN 2010-2011 grant "Automata and Formal Languages: Mathematical and Applicative Aspects" H41J12000190001.

References

[1] R. Balzer, An 8-state minimal time solution to the firing squad synchronization problem, Inform. Control 10 (1) (1967) 22–42.
[2] W.T. Beyer, Recognition of topological invariants by iterative arrays, Ph.D. thesis, Massachusetts Institute of Technology, 1969.
[3] A. Dennunzio, E. Formenti, L. Manzoni, G. Mauri, m-Asynchronous cellular automata: from fairness to quasi-fairness, Nat. Comput. 12 (4) (2013) 561–572.
[4] N. Fatès, M. Morvan, An experimental study of robustness to asynchronism for elementary cellular automata, Complex Systems 16 (1) (2005) 1–27.
[5] N. Fatès, M. Morvan, N. Schabanel, E. Thierry, Fully asynchronous behaviour of double-quiescent elementary cellular automata, Theoret. Comput. Sci. 362 (2006) 1–16.
[6] N. Fatès, D. Regnault, N. Schabanel, E. Thierry, Asynchronous behaviour of double-quiescent elementary cellular automata, in: Proceedings of LATIN'2006, in: Lecture Notes in Computer Science, vol. 3887, Springer, 2006, pp. 455–466.
[7] H. Fukś, Probabilistic cellular automata with conserved quantities, Nonlinearity 17 (1) (2004) 159–173.
[8] H.D. Gerken, Über Synchronisationsprobleme bei Zellularautomaten, Ph.D. thesis, Diplomarbeit, Technische Universität Braunschweig, 1987.
[9] E. Goto, A minimal time solution of the firing squad problem, Dittoed Course Notes Appl. Math. 298 (1962) 52–59.
[10] A. Grasselli, Synchronization of cellular arrays: the firing squad problem in two dimensions, Inform. Control 28 (2) (1975) 113–124.
[11] M. Kutrib, R. Vollmar, Minimal time synchronization in restricted defective cellular automata, J. Inform. Process. Cybern. 27 (3) (1991) 179–196.
[12] M. Kutrib, R. Vollmar, The firing squad synchronization problem in defective cellular automata, IEICE Trans. Inf. Syst. 78 (7) (1995) 895–900.
[13] L. Manzoni, Asynchronous cellular automata and dynamical properties, Nat. Comput. 11 (2) (2012) 269–276.
[14] J. Mazoyer, Synchronization of a line of finite automata with nonuniform delays, Research report TR 94-49, Ecole Normale Supérieure de Lyon, 1995.
[15] J. Mazoyer, On optimal solutions to the firing squad synchronization problem, Theoret. Comput. Sci. 168 (2) (1996) 367–404.
[16] M. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, Englewood Cliffs, 1967.
[17] E.F. Moore, The firing squad synchronization problem, in: Sequential Machines: Selected Papers, 1964, p. 213.
[18] F.R. Moore, G.G. Langdon, A generalized firing squad problem, Inform. Control 12 (3) (1968) 212–220.
[19] K. Nakamura, Asynchronous cellular automata and their computational ability, Syst. Comput. Controls 5 (1974) 58–66.
[20] J. Nishimura, H. Umeo, An optimum-time synchronization protocol for 1-bit-communication cellular automaton, in: Proc. of the 9th World Multi-Conference on Systemics, Cybernetics and Informatics, vol. X, 2005, pp. 232–237.
[21] D. Regnault, Abrupt behaviour changes in cellular automata under asynchronous dynamics, in: Electronic Proc. of 2nd European Conference on Complex Systems, ECCS, Oxford, UK, 2006, 6 pages.
[22] B. Schönfisch, A. de Roos, Synchronous and asynchronous updating in cellular automata, BioSystems 51 (1999) 123–143.
[23] I. Shinahr, Two- and three-dimensional firing-squad synchronization problems, Inform. Control 24 (2) (1974) 163–180.
[24] H. Szwerinski, Time-optimal solution of the firing-squad-synchronization-problem for n-dimensional rectangles with the general at an arbitrary position, Theoret. Comput. Sci. 19 (3) (1982) 305–320.
[25] H. Umeo, A simple design of time-efficient firing squad synchronization algorithms with fault-tolerance, IEICE Trans. Inf. Syst. 87 (3) (2004) 733–739.
[26] H. Umeo, K. Michisaka, N. Kamikawa, A synchronization problem on 1-bit communication cellular automata, in: Computational Science, ICCS 2003, in: Lecture Notes in Computer Science, vol. 2657, Springer, 2003, pp. 492–500.
[27] V.I. Varshavsky, V.B. Marakhovsky, V.A. Peschansky, Synchronization of interacting automata, Theory Comput. Syst. 4 (2) (1970) 212–230.
[28] A. Waksman, An optimum solution to the firing squad synchronization problem, Inform. Control 9 (1) (1966) 66–78.
[29] T. Worsch, A note on (intrinsically?) universal asynchronous cellular automata, in: Proceedings of Automata 2010, 14–16 June 2010, Nancy, France, 2010, pp. 339–350.
[30] J.B. Yunes, Fault tolerant solutions to the firing squad synchronization problem in linear cellular automata, J. Cell. Autom. 1 (3) (2006) 253–268.