ORIGINAL ARTICLE

A faster algorithm for the Birthday Song Singers Synchronization Problem (FSSP) in one-dimensional CA with multiple speeds

Thomas Worsch[1]

Abstract

In cellular automata with multiple speeds for each cell i there is a positive integer p_i such that this cell updates its state still periodically but only at times which are a multiple of p_i. Additionally there is a finite upper bound on all p_i. Manzoni and Umeo have described an algorithm for these (one-dimensional) cellular automata which solves the Firing Squad Synchronization Problem. This algorithm needs linear time (in the number of cells to be synchronized) but for many problem instances it is slower than the optimum time by some positive constant factor. In the present paper we derive lower bounds on possible synchronization times and describe an algorithm which is never slower and in some cases faster than the one by Manzoni and Umeo and which is close to a lower bound (up to a constant summand) in more cases.

1 Introduction

The *Firing Squad Synchronization Problem (FSSP)* has a relatively long history in the field of cellular automata. The formulation of the problem dates back to the late fifties and first solutions were published in the early sixties. A general overview of different variants of the problem and solutions with many references can be found in [3]. Readers interested in more recent developments concerning several specialized problems and questions are referred to the survey [4].

In recent years asynchronous CA have received a lot of attention. In a "really" asynchronous setting (when nothing can be assumed about the relation between updates of different cells) it is of course impossible to achieve synchronization. As a middle ground the FSSP has been considered in what Manzoni and Umeo [2] have called *CA with multiple speeds*, abbreviated in the following as MS- CA. In these CA different cells may update their states at different times. But there is still enough regularity so that the problem setting of the FSSP makes sense: As in standard CA there is a global clock. For each cell i there is a positive integer p_i such that this cell only updates its state at times t which are a multiple of

---

✉ Thomas Worsch
  worsch@kit.edu

1   Department of Informatics , Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany

p_i. We will call p_i the *period* of cell i. Additionally there is a finite upper bound on all p_i, so that it can be assumed that each cell has stored p_i as part of its state. This also means that there are always times (namely the multiples of the least common multiple of all periods) when all cells update their states simultaneously.

The rest of this paper is organized as followed: In Sect. 2 we fix some notation, review the basics of standard CA in general and of the FSSP for them. In Sect. 3 cellular automata with multiple speeds (MS- CA) and the corresponding FSSP will be introduced. Since most algorithms for the FSSP make heavy use of signals, we have a closer look at what can happen with them in MS- CA. In Sect. 4 some lower bounds for the synchronization times will be derived. Finally, in Sect. 5 an algorithm for the FSSP in MS- CA will be described in detail.

2 Basics

$\mathbb{Z}$ denotes the set of integers, $\mathbb{N}_+$ the set of positive integers and $\mathbb{N}_0 = \mathbb{N}_+ \cup \{0\}$. For $k \in \mathbb{N}_0$ and $M \subset \mathbb{Z}$ we define $k \cdot M = \{km \mid m \in M\}$. For $k \in \mathbb{N}_+$ let $\mathbb{N}_k = \{i \in \mathbb{N}_+ \mid 1 \leq i \leq n\}$.

The greatest common divisor of a set M of numbers is abbreviated as gcd M and the least common multiple as lcm M.

We write B^A for the set of all functions $f : A \rightarrow B$. The cardinality of a set A is denoted $|A|$.

For a finite alphabet A and $k \in \mathbb{N}_0$ we write A^k for the set of all words over A having length k, $A^{\leq k}$ for $\bigcup_{i=0}^{k} A^i$, and $A^* = \bigcup_{i=0}^{\infty} A^i$. For a word $w \in A^*$ and some $k \in \mathbb{N}_0$ the longest prefix of w which has length at most k is denoted as $\mathrm{pfx}_k(w)$, i.e. $\mathrm{pfx}_k(w)$ is the prefix of length k of w or the whole word w if it is shorter than k. Analogously $\mathrm{sfx}_k(w)$ is used for suffixes of w.

Usually cellular automata are specified by a finite set S of states, a neighborhood N, and local transition function $f : S^N \rightarrow S$. In the present paper we will only consider one-dimensional CA with Moore neighborhood $N = \{-1, 0, 1\}$ with radius 1.

Therefore a (global) *configuration* of a CA is a function $c : \mathbb{Z} \rightarrow S$, i.e. $c \in S^{\mathbb{Z}}$. Given a configuration c and some cell $i \in \mathbb{Z}$ the so-called *local configuration observed by i in c* is the mapping $g : N \rightarrow S : n \mapsto c(i + n)$ and denoted by c_{i+N}. In the standard definition of CA the local transition function f induces a *global transition function* $F : S^{\mathbb{Z}} \rightarrow S^{\mathbb{Z}}$ describing one step of the CA from c to $F(c)$ by requiring that $F(c)(i) = f(c_{i+N})$ holds for each $i \in \mathbb{Z}$.

By contrast in MS- CA it is not possible to speak about *the* successor configuration. The relevant definitions will be given and discussed in the next section.

Before, we quickly recap the *Birthday Song Singers Synchronization Problem* (sometimes also called the *Firing Squad Synchronization Problem*, FSSP). On the occasion of Martin Kutrib's 60th birthday a lot of congratulators are meeting and want to sing "Happy birthday" for him. (Un)Fortunately there are many many singers and it is impossible for all of them to see the conductor. It is therefore not easy to have all singers start singing at the same time and one has to devise a protocol using only nearest neighbor communications. In addition, some guests have travelled long distances and are more or less tired. As a consequence they need different times between subsequent inputs from their neighbors, but fortunately at least these times are all positive integer multiples of the same unit.

A CA solving the FSSP has to have a set of states $S \supseteq \{\#, \mathrm{G}, \_, \mathrm{F}\}$. For $n \in \mathbb{N}_+$ the *problem instance of size n* is the initial configuration $I_n = c$ where

$$c(i) = \# \ \text{ if } i \leq 0 \quad c(1) = \mathrm{G} \quad c(i) = \_ \ \text{ if } 1 < i \leq n \quad c(i) = \# \ \text{ if } i > n$$

Cells outside the segment between 1 and n are in a state # which is supposed to be fixed by all transitions. State G marks the general conductor, state _ the singers, and state F indicates that the singers have been synchronized and start singing.

The goal is to find a local transition function which makes the CA transit from each I_n to configuration F_n in which all cells $1, \ldots, n$ are in state F (the other cells still all in state #) and no cell ever was in state F before. In addition the local transition function has to satisfy $f(\_, \_, \_) = \_$ and $f(\_, \_, \#) = \_$, which prohibits the trivial "solution" to have all cells enter state F in the first step and implies that "activities" have to start at the G-cell and spread to other cells from there.

Within the framework of synchronization let's call the set $\text{supp}(c) = \{i \in \mathbb{Z} \mid c(i) \neq \#\}$ the *support* of configuration c. As a consequence of all these requirements during a computation starting with some problem instance I_n all subsequent configurations have the same support $\mathbb{N}_n$.

It is well known that there are CA which achieve synchronization in time $2n - 2$ (for $n \geq 2$) and that no CA can be faster, not even for a single problem instance.

3 CA with multiple speeds

3.1 Definition of MS-CA

A cellular automaton with multiple speeds (MS- CA) is a specialization of standard CA. Its specifiation requires a *finite* set $P \subseteq \mathbb{N}_+$ of so-called possible *periods* (in [2] they are called lengths of update cycles). Before a computation starts a period has to be assigned to each cell which remains fixed throughout the computation. Requiring P to be finite is meaningful for several reasons:

- It can be shown [2, Prop. 3.1] that otherwise it is impossible to solve the FSSP for MS- CA with one fixed set of states.
- We want that each cell can make its computation depend on its own period and those of its neighbors to the left and right, but of course the analogue of the local transition function should still have a finite description. To this end we want to be able to assume that each has its own period stored in its state.

For the rest of the paper we assume that the set of states is always of the form $S = P \times S'$, and that the transition function *never changes* the first component. We will denote the period of a cell i as p_i. For $s = (p, s') \in S = L \times S'$ we write π_p and π_s for the projections on the first and second component and analogously for global configurations. For a global configuration $c \in S^{\mathbb{Z}}$ we write $P(c) = \{p_i \mid i \in \text{supp}(c)\}$ (or simply P if c is clear from the context) for the set of numbers that are periods of cells in the support of c; cells in state # can be ignored because they don't change their state by definition.

For MS- CA it is not possible to speak about *the* successor configuration. Instead it is necessary to know how many time steps have already happened since the CA started. Borrowing some notation from asynchronous CA, for any subset $A \subseteq \mathbb{Z}$ of cells and any $c \in S^{\mathbb{Z}}$ denote by $F_A(c)$ the configuration reached from c if exactly the cells in A update their states according to f and all other cells do not change their state:

$$\forall i \in \mathbb{Z}: F_A(c)(i) = \begin{cases} f(c_{i+N}) & \text{if } i \in A \\ c(i) & \text{if } i \notin A \end{cases}$$

(Thus, the global transition function of a standard CA is $F_{\mathbb{Z}}$.) Given some MS-CA C and some time t denote by $A_t = \{i \in \mathbb{Z} \mid 0 = t \bmod p_i\}$ the set of so-called *active* cells at time t. Then, for each initial configuration c the computation resulting from it is the sequence $(c_0, c_1, c_2, \dots)$ where $c_0 = c$ and $c_{t+1} = F_{A_t}(c_t)$ for each $t \in \mathbb{N}_0$. In particular this means, that at time $t = 0$ *all* cells will update their states according to f. More generally this is true for all $t \in p_c \cdot \mathbb{N}_0$, where p_c is the least common multiple of all $p \in P$, i.e. all t that are a multiple of all elements in P. We will speak of a *common update* when $t = p_c$.

The observations collected in the following lemma are very simple and don't need an explicit proof.

Lemma 1 *Let $g = \gcd P$ be the greatest common divisor of all $p \in P$ and let $P' = \{p/g \mid p \in P\}$. Let $A_t = \{i \in \mathbb{Z} \mid 0 = t \bmod p_i\}$ as before.*

1. *For each $t \notin g \cdot \mathbb{N}_+$ the set A_t is empty.*
2. *For each computation $C = (c_0, c_1, \dots)$ for each $k \in \mathbb{N}_0$ $c_{kg+1} = c_{kg+2} = \cdots = c_{kg+g}$.*
3. *The computation $C' = (c_0, c_g, c_{2g}, \dots)$ results when using P' instead of P and exactly the same local transition function.*
4. *When all cells involved in a computation C have the same period p, C is simply a p times slower "copy" of the computation in a standard CA.*

Therefore the interesting cases are whenever $|P| \geq 2$ and $\gcd P = 1$. We will assume this for the rest of the paper without always explicitly mentioning it again.

Fact 2 If $|P| \geq 2$ and $\gcd P = 1$ then there is at least one *odd* number p that can be used as a period.

3.2 Signals in MS-CA

Since almost all CA algorithms for the synchronization problem make extensive use of signals, they are also our first example for some MS-CA. Figure 1 shows a sketch of a space-time diagram. Time is increasing in the downward direction (throughout this paper). When a cell is active at time t a triangle between the old state in row t and the new state in row $t + 1$ indicates the state transition. The numbers in the top row are the periods of the cells.

At this point it is not important to understand how an appropriate local transition function could be designed to realize the depicted signal. But, assuming that this can be done, the example has been chosen such that the signal is present in a cell i for the first time exactly p_i steps after it was first present in the left neighbor $i - 1$.

One possibility to construct such computations is the following. Putting aside an appropriate number of cells at either end, the configuration consists of blocks of cells. In each block all cells have some common period p and there are $k = p_c/p$ cells in the block. For example, in Fig. 1 $p_c = 6$ (at least one can assume that, since only periods 1, 2, and 3 are used) and there are 2 cells with period 3 and 3 cells with period 2. Let's number the cells in such a block from 1 to k.

Assume that a signal should move to the right as fast as possible. For each such block the following holds: If the signal appears in the left neighbor of such a block for the first time after a common update at some time t, then it can only enter cell 1 of the block at time $t + p$, cell 2 of the block at time $t + 2p$ etc., and hence the last cell k of the block at time $t + kp = t + p_c$. That cell is the left neighbor of the next block. Hence by induction the same is true for every block and the passage of the signal through each block will take p_c steps. This happens to be the sum of all periods of cells in one block.

Fig. 1 Sketch of how a basic signal could move right in a MS- CAas fast as possible. The numbers in the top row are the periods of the cells. Time is going down. Triangles indicate active state transitions, i. e. when they are missing the state has to stay the same
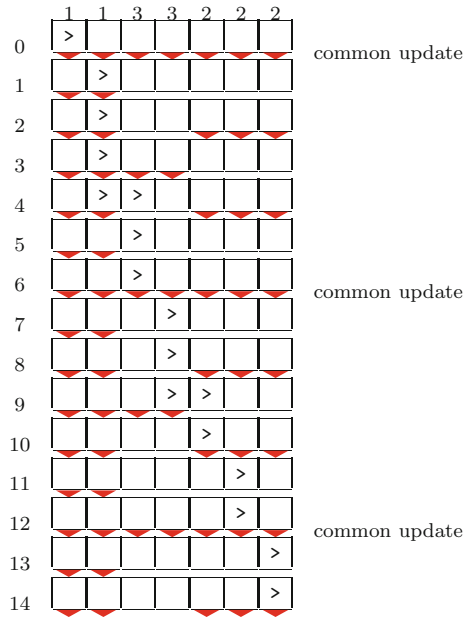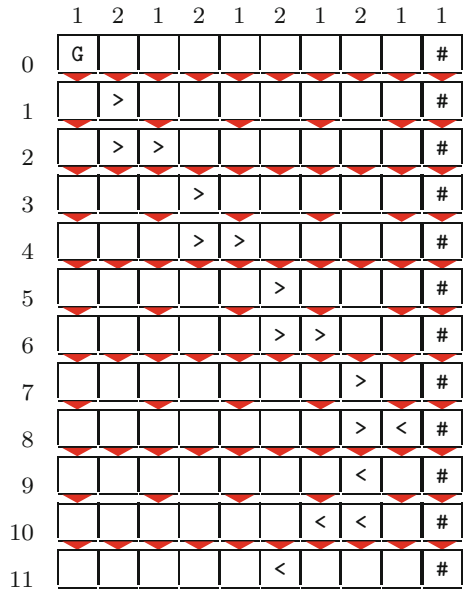
Fig. 2 Basic signal first moving right and bouncing back at the border with speed 1 although half of the cells has only period 2

Unfortunately there are also cases in which signals are *not* delayed by the increase of the periods of some cells. Figure 2 shows a situation where periods 1 and 2 are assigned to subsequent cells alternatingly. As can be seen, a signal can move from each cell to next one in every step, including a change of direction at the right border.

3.3 The FSSP in MS-CA

In the standard setting for each n there is exactly one problem instance of the FSSP of length n.

In MS- CA we will assume that the set of states is always of the form $S = P \times S'$. We will call a configuration $c \in S^{\mathbb{Z}}$ a *problem instance for the* MS- FSSP if two conditions are satisfied:

– $\pi_s(c)$ is a problem instance for the FSSP in standard CA.
– The period of all border cells is the same as that of G-cell 1.

By definition border cells never change their state, no matter what their period is. The second condition just makes sure that formally a period is assigned even to border cells, but this does not change the set of periods that are present in the cells of the support that do the real work.

Now for each size n there are $|P|^n$ problem instances of the MS- FSSP.

It should be clear that the minimum synchronization time will at least in some cases depend on the periods. Assume that there are two different $p, q \in P$, say $p < q$. Then, when all cells have $p_i = p$ synchronization can be achieved more quickly than when all cells have $p_i = q$. A straightforward transfer of a (time optimal) FSSP algorithm for standard CA (needing $2n + O(1)$ steps) yields a MS- CA running in time $(2n - 2)p$. This is faster than any MS- CA with uniform period q can be which needs $(2n - 2)q$ (see Sect. 4).

4 On lower bounds for the synchronization time on MS-CA

In the case of standard CA the argument used for deriving lower bounds for the synchronization time uses the following observation. Whenever an algorithm makes the leftmost cell 1 fire at some time t, it can only be correct if changing the border state # in cell $n + 1$ to state _ (i. e. increasing the size of the initial configuration by 1) can possibly (and in fact will) have an influence on the state of cell 1 at time t. If $t \leq 2n - 3$ changing the state at the right end cannot have an influence on cell 1. But then adding n cells in state _ to the right will still make cell 1 enter state F at time t, while the now rightmost cell $2n$ will not have had any chance to leave its state.

This argument can of course be transferred to MS- CA, and it means that one has to find out the minimum time to send a signal to the rightmost cell of the support and back to cell 1.

Theorem 3 *For every* MS- CA *solving the* MS- FSSP *there are constants $a > 1$ and d such that for infinitely many $n \geq 2$ there are at least a^n problem instances c of size n such that C needs at least*

$$\left(2 \sum_{i=1}^{n} p_i \right) - d$$

steps for synchronization of c.

Proof The example in Fig. 1 can be generalized.

We first define a set M of periods we will make use of. According to Fact 2 the set $P' = \{p \in P \mid p \text{ is odd}\}$ is not empty. If $|P'| \geq 2$ then let $M = P'$ and $q = \max M$. If $|P'| = 1$ then let $M = P' \cup \{q\}$ where $q = \max(P \setminus P')$ (choosing the maximum is not important; we just want to be concrete). Let $m_c = \operatorname{lcm} M$.

We will use blocks of length $b_p = m_c/p$ of sucessive cells with the same period p (as in Fig. 1). As will be seen it is useful to have at least one odd b_p. Indeed, $b_q = m_c/q$ is odd (because q is the only possibly even number in M).

Since there are at least 2 different numbers $p \in M$, there are also at least 2 different block lengths in $B = \{b_p \mid p \in M\}$, denoted as b and b'.

Consider now all problem instances similar to Fig. 1. Let $m \in \mathbb{N}_+$

- The periods of the first 2 cells are the same but otherwise arbitrary.
- The rest of the cells is partitioned into $2m$ blocks. There are m blocks consisting of b cells and the period of all cells in that block is m_c/b and there are m blocks consisting of b' cells and the period of all cells in that block is m_c/b'.
- At the right end of the instance there is a segment consisting of $1 + h$ cells of period q where $h = \lfloor b_q/2 \rfloor$. Since b_q is odd, one has $2h + 1 = b_q$.

For each m the total size of the problem instances is $2 + m(b + b') + 1 + h$ which is linear in m, and there are $\binom{2m}{m}$ different arrangements of the blocks. This number is known to be larger than $4^m/(2m + 1)$ (proof by induction). Formulated the other way around for these problem sizes n there is a number of problem instances which exponential in m and hence also in n (for some appropriately chosen base $a > 1$).

It remains to estimate the synchronization time for these problem instances. As already described in Sect. 3.2 a signal that is supposed to first move to the right border as fast as possible and then back to cell 1 will arrive in cell 2 after the first (common) update. From that time on for each block it will take exactly m_c steps to "traverse" each block which is also the sum of all periods of the cells in the block.

For the passage through the last $h + 1$ cells forth and back have a look at Fig. 3. Each cell is passed twice, once when the signal moves right and once when it moves back to the left, each time for q steps. The only exception is the rightmost cell, where the signal stays only for the duration of 1 period, i.e. q steps. Altogether these are $(2h + 1)q = m_c$ steps which is exactly the same number of steps as for each full block to the left. Consequently the position of the signal moving back to the left is for the first time in the cell to the right of a full block immediatly after a common update.

Summing up all terms for the movement of a signal from the very first cell to the right border and back results in a time of

$$t = 1 + m \cdot m_c + m \cdot m_c + h \cdot q + q + h \cdot q + m \cdot m_c + m \cdot m_c + p_2 + 1$$

This contains twice the term

$$t' = m \cdot m_c + m \cdot m_c + h \cdot q = \sum_{i=3}^{n-1} p_i = \left(\sum_{i=1}^{n} p_i \right) - p_1 - p_2 - p_n$$

hence

$$t = 1 + t' + p_n + t' + p_2 + 1 = 1 + 2 \left(\sum_{i=1}^{n} p_i \right) - 2p_1 - 2p_2 - 2p_n + p_n + p_2 + 1$$

$$= 2 \left(\sum_{i=1}^{n} p_i \right) - 2p_1 - p_2 - p_n + 2$$

as claimed.

It can be observed that in the case that all $p_i = 1$ the formula becomes the well-known lower bound of $2n - 2$. □

In the following section we will describe an algorithm which achieves a running time which is slower than the lower bound in Theorem 3 by only a constant summand.
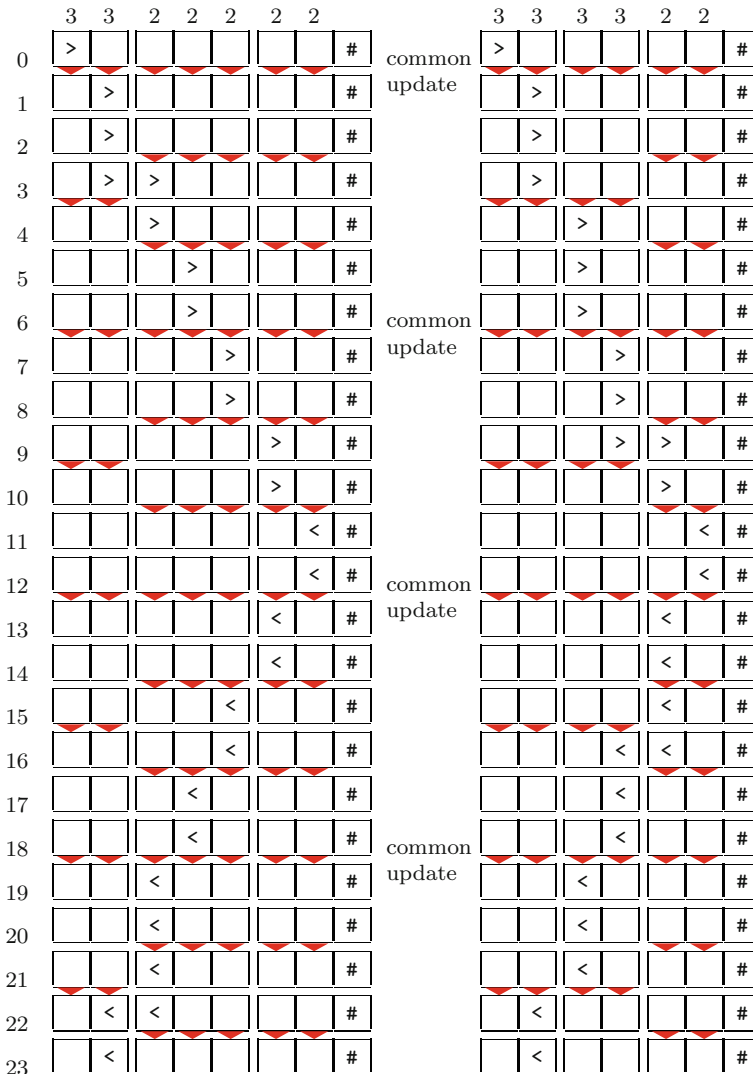
Fig. 3 Reflection of a "fast" signal at the right border. We use $m_c = 6$, $q = 2$, hence $b_q = 3$, $h = 1$ and $h + 1 = 2$. Therefore at the right end there are always $h + 1 = 2$ cells with period 2. We have introduced small spaces to make the boundaries of the last full block of cells with period 2 better visible. On the left hand side the last full block has 3 cells of period 2 and on the right hand side the last full block has 2 cells of period 3. For more details see the proof of Theorem 3

5 Detailed description of the synchronization algorithm for MS-CA

To the best of our knowledge the paper by Manzoni and Umeo [2] is the only work on the FSSP in one-dimensional MS- CA until now. They describe an algorithm which achieves synchronization in time $n \cdot p_{\max}$ where $p_{\max} = \max\{\pi_p(c(i)) \mid 1 \leq i \leq n\}$ is maximum period used by some cell in the initial configuration c.

Below we will describe an algorithm which proofs the following:

Theorem 4 *For each P there is a constant d and an algorithm which synchronizes each* MS- FSSP *instance c of size n with periods $p_1, \ldots, p_n \in P$ in time*

$$\left(2 \sum_{i=1}^{n} p_i\right) + d \tag{1}$$

In the case of standard CA all $p_i = 1$ and formula (1) becomes $2n + d$ which is only a constant number of $d + 2$ steps slower than the fastest algorithms possible.

5.1 Core idea for synchronization

In the proof of a lower bound above we have constructed problem instances consisting of blocks consisting of cells with identical period. The arrangement was chosen in such a way that a signal, even if it were to move as fast as possible, would have to spend p steps in a cell with period p before moving on. In a standard CA this is the time a signal with speed 1 needs to move across p cells. Which leads to the idea to have each cell with period p of the MS- CA simulate p cells of a standard CA (solving the FSSP). We'll call the simulated cells *virtual cells* or *v-cells* for short, and where disambiguation seems important call the cells of the MS- CA *host cells*. States of v-cells will be called v-states.

5.2 Details of the synchronization algorithm

From now on, assume that we are given some standard CA for the standard FSSP. Its set of states will be denoted as Q.

Algorithm 5 As a first step we will sketch the components of the set S of states of the MS- CA.
  We already mentioned in Sect. 3.3 that we assume S to be of the form $S = P \times S_1$. Let $t_c = \mathrm{lcm}\, P$ denote the least common multiple of all $p \in P$. Since in the algorithm below host cells will have to count from 0 up to $t_c - 1$, we require that the set of states always contains a component $T = \{i \in \mathbb{Z} \mid 0 \leq i < t_c\}$. Hence $S = P \times T \times S_2$, and we assume that the transition function will in each step update the T-component of a cell by incrementing it by its period, modulo t_c. Imagine that this is always "the last part" of a transition whenever a cell is active. Thus an active cell can identify the common updates by the fact that its T-component is 0. But of course it is equally easy for an active cell to identify an activation that is *the last before* a common update.
  Next, each host cell will have to store the states of some v-cells. As will be seen this will not only comprise the states of the p v-cells it is going to simulate, but also the states of v-cells simulated by neighboring host cells; we will call these *neighboring v-cells*. To this end we choose $S = P \times T \times Q^{\leq t_c} \times Q^{\leq P_{\max}} \times Q^{\leq t_c} \times S_3$. We will denote the newly introduced components of a cell as x, y, and z. In x a host cell will accumulate the states of more and more neighboring v-cells from the left. Analogously, in z a host cell will accumulate the states of more and more neighboring v-cells from the right. In the middle component y a host cell will always store the states of the p v-cells it has to simulate itself.
  The simulation will run in cycles each of which is t_c steps long and begins with a common update. During one cycle a cell with period p will be active t_c/p times. Whenever a host cell is active it collects as many neighboring v-states as possible, but at most t_c from either side. More precisely this is done as depicted in the following

(x_{-1}, y_{-1}, z_{-1})	(x_0, y_0, z_0)	(x_1, y_1, z_1)

$(\mathrm{sfx}_{t_c}(x_{-1}y_{-1}), y_0, \mathrm{pfx}_{t_c}(y_1 z_1))$

table: In other words, the states of the own v-cells are not changed, but more and more neighboring v-states are being collected. We will show in Lemma 6 below that during the last activation of a cycle, i.e. the last activation before a common update, after having collected neighboring v-states, the length of the x and z components are in fact t_c and not shorter. It is therefore now possible for each host cell to replace the v-states of its p v-cells by the v-states those v-cells would be in after t_c steps. The x and z components are reset to the empty word.

It is during the last activation of a cycle that a host will compute state F for each of its v-cells. The immediately following activation is a common update for all host cells. They will simultaneously detect that their v-cells reached the "virtual F" and all enter the "real firing state".

For a proof of Theorem 4 only the following two aspects remain to be considered.

Lemma 6 *After one cycle of algorithm 5 each host cell will have collected the states of t_c neighboring v-cells, to the left and to the right.*

Proof Without loss of generality we only consider the case to the left. We will prove by induction on the global time that for all $\bar{t} \in \mathbb{N}_0$ the following holds:

For $t = \bar{t} \bmod t_c$ for each cell with components (x, y, z) as above and with period p and for all $j \in \mathbb{N}_0$ with $0 \le j \le t$: If $jp = t$ then the cell is active and after the transition $|x| \ge \min(jp, t_c)$.

If $\bar{t} = 0$ then $t = 0$, $j = 0$, and obviously $|x| \ge 0$ holds.

Now assume that the statement is true for all times less or equal some $\bar{t} - 1$. Again, nothing has to be done if $t = 0$; assume therefore that $t > 0$.

Consider a cell with components (x, y, z) and period p and its left neighbor with components (x', y', z') and period q, and therefore $|y'| = q$. Let $\bar{t}'$ be the time when the left neighbor was active for the last time before $\bar{t}$, and let $t' = \bar{t}' \bmod t_c$. Then $t' < t$, $t' = kq$ for some k, and since it was the *last* activation before t, $t' + q = (k + 1)q \ge t$. By induction hypothesis the left neighbor already had $|x'| \ge \min(kq, t_c)$. The new x of the cell under consideration is $x = \mathrm{sfx}_{t_c}(x'y')$ which then has length at least $\min(t_c, |x'y'|) \ge \min(t_c, \min(kq, t_c) + q) = \min(t_c, (k + 1)q)$. Since $(k + 1)q \ge jp$ the proof is almost complete.

Strictly speaking the above argument does not hold when the left neighbor is a border cell. But in that case a cell can treated by its neighbor as if that has already x filled with t_c states #. □

Lemma 7 *For a problem instance c of size n with periods $p_1, \ldots, p_n$ the time needed by Algorithm 5 for synchronization can be bounded by*

$$\left(2 \sum_{i=1}^{n} p_i \right) + d . \tag{2}$$

Proof The total number of v-cells simulated is $k = \sum_{i=1}^{n} p_i$. A time optimal FSSP algorithm for standard CA needs $2k - 2$ steps for the synchronization of that many cells. During each cycle of length t_c exactly t_c steps of each v-cell are simulated, except possibly for the last cycle. During that, the F v-state may be reached in less than t_c steps.

Hence the total number of steps is $t_c \cdot \lfloor (2k - 2)/t_c \rfloor + t_c + 1 \le 2k + d = \left(2 \sum_{i=1}^{n} p_i \right) + d$ for some appropriately chosen constant d. □

To sum up taking together Theorem 3 and Theorem 4 one obtains

Corollary 8 *For each P there is a constant d such that there is an* MS- CA *for the* MS- FSSP *which needs synchronization time* $\left(2\sum_{i=1}^{n} p_i\right) + d$ *and for infinitely many sizes n there are* a^n *problem instances (a > 1) for which there is a lower bound on the synchronization time of* $\left(2\sum_{i=1}^{n} p_i\right) - d$.

6 Outlook

In this paper we have described a MS- CA for the synchronization problem which is sometimes faster and never slower than the one by Manzoni and Umeo. For a number of problem instances which is exponential in the number of cells to be synchronized the time needed is close to some lower bound derived in Sect. 4. The corrections pointed out by an anonymous referee are greatfully acknowledged.

While higher-dimensional MS- CA have been considered [1], in the present paper we have restricted ourselves to the *one-dimensional* case. In fact it is not completely clear how to generalize the algorithm described above to two-dimensional CA. The MS- CA described in this paper it is essential that

– from one cell to another one there is only one shortest path
– and it is clear how many v-cells a cell should simulate.

The generalization of this approach to 2-dimensional CA is not obvious for us. In addition the derivation of reasonably good lower bounds on the synchronization times seem to be more difficult, but if one succeeds that might give a hint as to how to devise an algorithm. As a matter of fact, the same happened in the one-dimensional setting.

Similarly it is not clear how to apply the ideas in the case of CA solving some *other* problem, not the FSSP, because only (?) for the FSSP it is obvious which state(s) to choose for the v-cells in the initial configuration.

Both aspects, algorithms and lower bounds, are interesting research topics but need much more attention. Even in the 1-dimensional case there is still room for improvement as has been seen in Fig. 2.

It remains an open problem how singers with different degrees of alertness should be ordered to ensure that they can start singing "Happy Birthday" as soon as possible. But no matter how long it takes:

Happy birthday, Martin!

References

1. Manzoni, L., Porreca, A.E., Umeo, H.: The firing squad synchronization problem on higher-dimensional CA with multiple updating cycles. In: Fourth International Symposium on Computing and Networking, CANDAR 2016, Hiroshima, Japan, November 22–25, 2016, pp. 258–261 (2016)
2. Manzoni, L., Umeo, H.: The firing squad synchronization problem on CA with multiple updating cycles. Theor. Comput. Sci. **559**, 108–117 (2014)
3. Umeo, H.: Firing squad synchronization problem in cellular automata. In: Encyclopedia of Complexity and Systems Science. Springer, pp. 3537–3574 (2009)
4. Umeo, H.: How to synchronize cellular automata—recent developments. Fundam. Inf. **171**(1–4), 393–419 (2020)