The Firing Squad Synchronization Problem on Higher-dimensional CA with Multiple Updating Cycles

Luca Manzoni and Antonio E. Porreca
Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Viale Sarca 336, 20126 Milan, Italy
email: {luca.manzoni,porreca}@disco.unimib.it

Hiroshi Umeo
Division of Information and Computer Sciences,
Osaka Electro-Communication University,
Neyagawa-shi, Hastu-cho, 18-8, Osaka 572-8530, Japan
email: umeo@cyt.osakac.ac.jp

***Abstract*—Traditional cellular automata (CA) assume the presence of a single global clock regulating the update of all their cells. When this assumption is dropped, cells can update with different speeds, thus increasing the difficulty of solving synchronization problems. Here we solve the traditional and the generalized Firing Squad Synchronization Problem in dimension two and higher on multiple updating cycle CA.**

I. Introduction

The *firing squad synchronization problem* (FSSP) has been studied for more than five decades [1]. A solution to the FSSP consists in synchronizing a line of n identical automata, which all start in the same quiescent state except the first one (called *general*), by reaching all the same state (called the *firing state*) for the first time at the same moment. The first (non-optimal) solution for the one-dimensional case was found in the sixties [2] and was quickly followed by different algorithms all working in optimal time [3]–[7]. The problem was then generalized to multi-dimensional arrays [8]–[10], which is the focus of this paper.

In the construction of solutions to the FSSP one of the main assumptions is that all cells of the cellular automaton (CA) update synchronously. Recently, there has been a new interest in the study of asynchronicity in CA and how it influences the dynamical behaviour [11]–[13] and the ability to perform computations [14]–[16]. In the case of the FSSP the introduction of asynchronicity has focused on the presence of multiple *updating cycles* [17]. That is, each cell of the CA has a speed $1/k$ for some $k \in \mathbb{N}$ associated with it; that is, the cell updates its internal state only once every k steps. This is not the first time a non-uniform behaviour was added to a CA. For example, the addition of faulty cells [18]–[22] is a common theme, even if the definition of the behaviour of a faulty cell is not consistent in the literature: in some cases the faulty cells move all the signals at maximum speed, while in other cases the cells are always inactive.

Here we provide a new algorithm for the FSSP with multiple updating cycles for CA of dimension 2 and higher. Unlike the one-dimensional case, here the synchronization is subdivided into two phases. In the first phase information about the different speeds present in the CA is collected in order to allow each cell to slow down appropriately in the second phase, when an adapted traditional (i.e., for uniform speed) FSSP algorithm is executed. We also adapt our solution to the generalized FSSP, where the general can be in any position.

II. Multiple Updating Cycle CA

Multiple updating cycle CA were introduced in [17] to remove the assumption, that is present in classical CA, of a single clock shared among all cells. If we remove that assumption, we have a CA in which different cells have different clocks, therefore updating with different speeds.

Definition 1. Let $d \in \mathbb{N}$ and $n_1, \dots, n_d \in \mathbb{N}$. A d-dimensional multiple updating cycle CA (MUCCA) of size $n_1 \times n_2 \times \cdots \times n_d$ is a 5-tuple $(Q, \mathbf{b}, \lambda, S, s)$ where Q is a finite set of states that can be assumed by the cells of the CA, $\mathbf{b} \notin Q$ is a special state called *border state*, $\lambda\colon ((Q \times S) \cup \{\mathbf{b}\})^{2d+1} \to Q$ is the local rule of the CA using von Neumann neighbourhood, $S \subset \{1/k\colon k \in \mathbb{N}\}$ is a finite set of possible speeds, and $s\colon \mathbb{N}^d \to S$ is the *speed function*.

A configuration of the MUCCA is an element of $Q^{n_1 \times n_2 \times \cdots \times n_d}$. Given a configuration c of an MUCCA at time $t - 1 \in \mathbb{N}$, the next state configuration is given by a global function Λ_t defined for each cell position $i = (i_1, \dots, i_d)$ having neighbourhood N_i as:

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

where $s(N_i)$ are the speed of all the cells in the neighbourhood N_i. Notice that the local function depends on the speed of the cells being updated and thus there is not a single global function but a (finite) family of them indexed by the time t. From now on, when there is no ambiguity, we will use the notation Λ^t as a shorthand for the function composition $\Lambda_t \circ \Lambda_{t-1} \circ \cdots \Lambda_1$. To denote the configuration of the automaton at time $t \in \mathbb{N}$, starting with the initial configuration $c(0)$, we will use the notation $c(t)$ with $c_i(t) = \Lambda^t(c(0))_i$ denoting the i-th cell of the configuration of the automaton at time t.

It is already known that there is no solution to the FSSP independent from the set of speeds of the automaton [17]. However, a family of FSSP algorithms, indexed by the possible set of speeds, can be devised.

III. FSSP Solution for Two Dimensions

The main idea of the algorithm is to send a signal from the position of the general in the top left cell of the automaton to the cell in the bottom right corner. This signal will collect the *least common multiple* ℓ of all the denominators of the speeds encountered (i.e., if the speeds 1, $\frac{1}{2}$, and $\frac{1}{3}$ are encountered then 6 is collected). This allows to start an adapted algorithm for the FSSP from the bottom right corner. If q is an original state of the FSSP solution then the adapted solution will have a state (q, t) with t ranging from 0 to $\ell \times s(i) - 1$. While t is incremented (modulo $\ell \times s(i)$) each time the cell i is updated, the (original) state q is updated only when $t = 0$. In this way the FSSP algorithm is slowed down to a speed "compatible" with all the speeds present in the automaton. For more details on the slow down process we refer the reader to the original work for the FSSP on one-dimensional MUCCA [17].

The main idea of the algorithm is to update a cell only when both the cell on the top and on the left are not quiescent, i.e., they are either the border, the general, or they have previously updated. With this transition we obtain that the cell in position (i, j) contains the least common multiple of the denominators of all the speeds of the cells in the "cone"

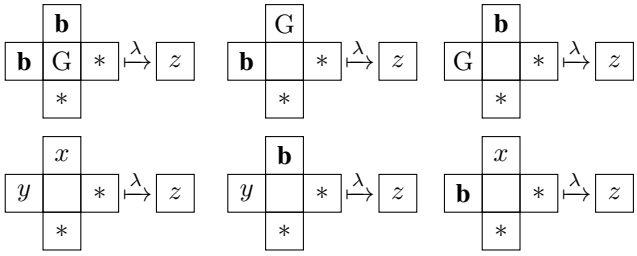
$$\{(i', j') \colon 0 \le i' \le i \text{ and } 0 \le j' \le j\}.$$

As an example, Fig. 1 shows how this idea can be applied in the one dimensional case.

Formally, the states *for this initialization phase* of an automaton with speeds $S = \left\{\frac{1}{k_1}, \frac{1}{k_2}, \dots, \frac{1}{k_m}\right\}$ are

$$Q = \{\mathrm{G}, \mathbf{q}\} \cup \left((K \cup \{\mathrm{G}'_k \colon k \in K\}) \times \{0, \dots, \ell - 1\}\right)$$

where $K = \left\{\operatorname{lcm} L \colon L \subseteq \{k_1, \dots, k_m\}\right\}$ and ℓ is the least common multiple of all the elements in $\{k_1, \dots, k_m\}$. That is, the set of states contains, apart from the general G and the quiescent state $\mathbf{q}$, the least common multiples of the denominators of all combinations of speeds that can appear in the automaton, and "new generals" G'_k having as subscript the possible slow down factors for the second phase. As a second component, all new states also maintain the current time step modulo $\operatorname{lcm}\{k_1, \dots, k_m\}$ in order to have the new general G'_k start the slowed down FSSP algorithm at a time step multiple of k. This second component will not be shown in the rest of the paper. The local rule is formalized as



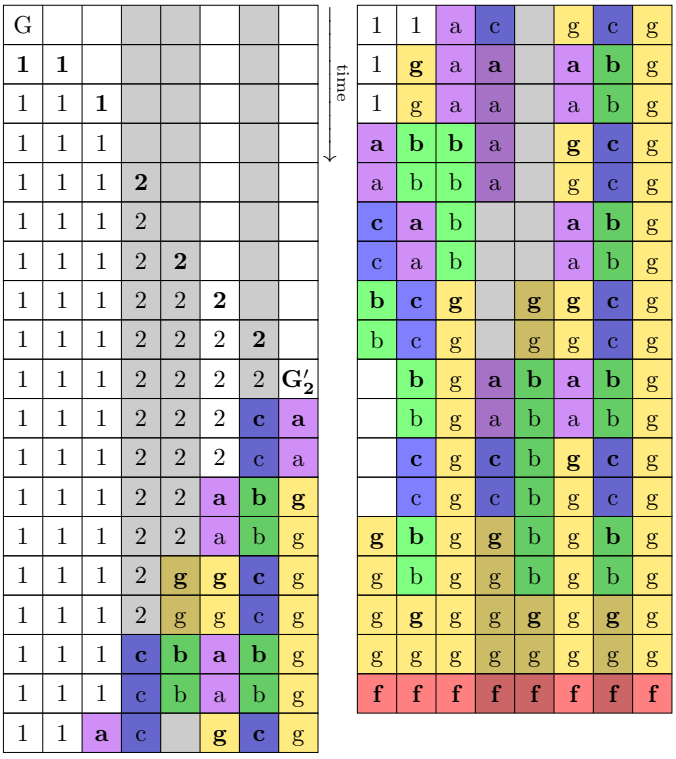
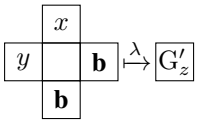


Figure 1. A complete execution of the proposed algorithm in the one-dimensional case. Grayed-out cells have speed $1/2$ while all the others have speed 1. In the first phase we can see how the "new general" G'_2 is produced. Once it appears, it triggers the execution of a slowed-down adaptation of the FSSP solution by Mazoyer [7]. The component of the local state used to slow down the algorithm is not shown here.



where $*$ is a "don't care" symbol, with the restriction that the two states in those positions must not be simultaneously **b**. Furthermore $x, y \in K$, while z is the least common multiple of $1/s(i)$ and of all adjacent states belonging to $K \cup \{\mathrm{G}\}$, where G is here interpreted as the denominator of the speed of the cell containing G. When the bottom right cell is reached a new general G'_z is obtained and a traditional FSSP algorithm can start and be executed by each cell i with a slow down of $z \times s(i)$. An example of computation is presented in Fig. 2.

The time required for the first phase is linear with respect to $m + n$ for an MUCCA of size $m \times n$. There is however a multiplicative slow down factor that can be up to $\operatorname{lcm} K$, but the actual slow down depends on the distribution of slow cells. The second phase is the time required for a traditional 2D FSSP algorithm, i.e., linear in $m + n$ in the optimal case, slowed down by the least common multiple of speeds appearing in the automaton.

A. Extension to Higher Dimensions

The algorithm presented for dimension two can be extended without difficulty to any higher number of dimensions. While

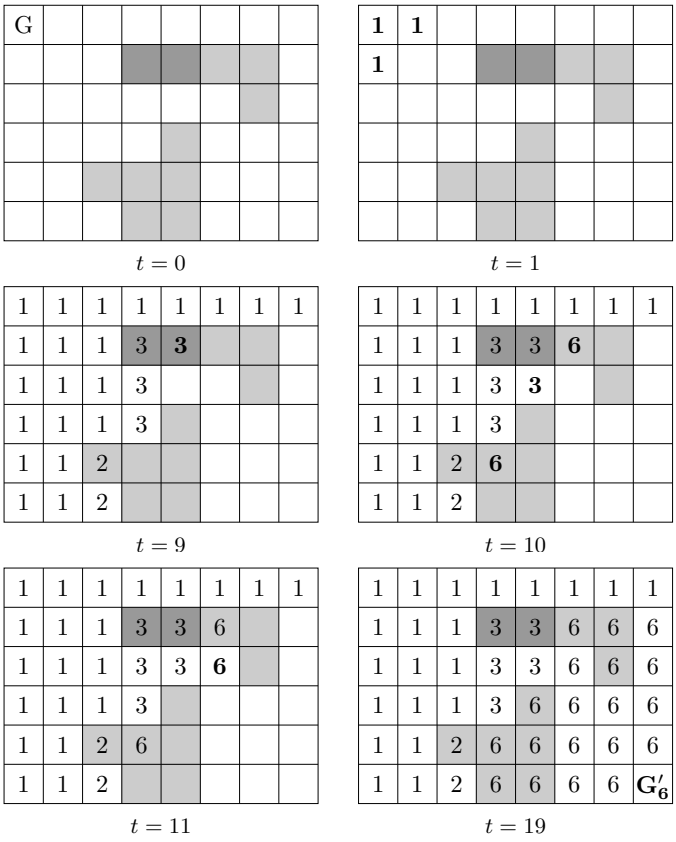
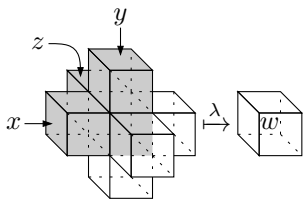


Figure 2. A computation of the first phase of synchronization of an MUCCA with white, light gray, and dark gray cells having speed 1, $1/2$, and $1/3$ respectively. The cells with a bold state are those that have changed their state in the last step. Notice how all cells with speed $1/2$ do not update in step 11, and some cells (e.g., the one in the fourth row and fifth column at time $t = 10$) do no change state even if they can activate, since not all information about their cone is available in the neighbourhood.

for dimension two the cell was waiting for the cells at the top and on the left, in higher dimensions the same effect can be obtained by having the cell in position $i = (i_1, \ldots, i_d)$ wait for all the cells $(i_1 - 1, i_2, \ldots, i_d)$, $(i_1, i_2 - 1, i_3, \ldots, i_d)$, up to $(i_1, i_2, \ldots, i_{d-1}, i_d - 1)$. For example, in dimension three the central cell will only look at itself and the cells depicted in gray in the following diagram, having states x, y, and z, in order to compute its next state w:

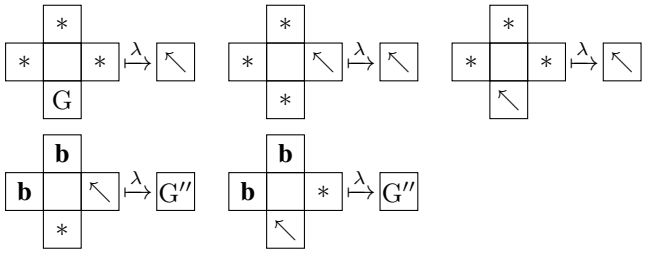


As with dimension two, in an MUCCA of size $n_1 \times \cdots \times n_d$ the time required remains linear in the sum of the dimensions $n_1 + \cdots + n_d$, with a slow down factor that depends by the cells present in the automaton.

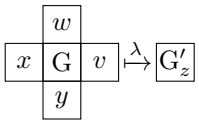
IV. Generalized FSSP Solution

The generalization of the two-dimensional solution to any position of the general can be obtained by partitioning the two-dimensional space in four quadrants. In each quadrant, the corner of the automaton opposite to the general is found. Then, the first phase of the solution with the general in a corner is applied independently in order to find the least common multiple of the denominators of the speeds in each quadrant. The results are then collected by the general, that can start a slowed down version of a generalized FSSP solution for traditional CA.

The state of the automaton that are used in the initial phase are the same as the solution for the non-generalized case (i.e., the set Q in the previous section) together with the extra states $\{\nwarrow, \nearrow, \swarrow, \searrow, \mathrm{G}''\}$. The arrows will propagate as fast as possible toward the corners they point to. When a corner is reached a "pseudo-general" G'' triggers the execution of the first phase the algorithm for the non-generalized FSSP described in the previous section, restricted to the corresponding quadrant. The corresponding local rules for the propagation of the arrow in the north-western quadrant (the others can be obtained by rotation) are



where the "don't care" symbol $*$ represents any possible state with at most one them a border. The general G remains inactive until the four quadrant have completed the backward propagation of their own slow down factors. When the four values are adjacent to the general it perform the following transition



where z is the least common multiple v, w, x, y and $1/s(i)$, with i the position of the cell with the general.

Now, as in Section III, the new general G'_z triggers the execution of a traditional FSSP algorithm which is perform by each cell i with a slowdown $z \times s(i)$. A sample computation is shown in Fig. 3.

As in Section III-A, the algorithm is easily generalized to higher dimensional CA. For dimension d, instead of 4 quadrants the space is partitioned in 2^d higher dimensional equivalents.

The analysis of the time required is similar to the one of Section III-A; the time required for the initial phase is at most doubled, but the exact slowdown depends on the distribution of speeds. The time, however, remains linear in $n_1 + n_2 + \cdots + n_d$ for an automaton of size $n_1 \times n_2 \times \cdots \times n_d$.

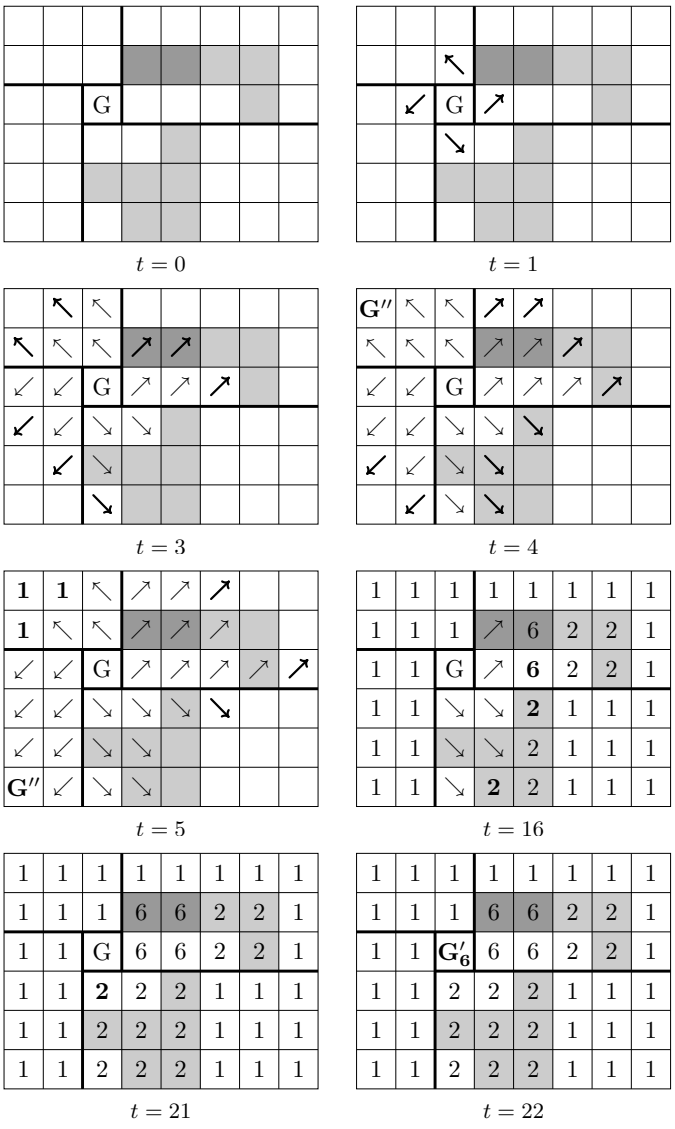


Figure 3. A computation of a generalized FSSP solution on an MUCCA. The bold lines represent the four quadrants in which the algorithm subdivides the automaton in the first phase. Notice that when the arrows reach a corner (e.g., at time $t = 4$), a new general G' is created and the information about the speeds in the quadrant is then sent to the original general G (e.g., at time $t = 21$). Only when the information from all quadrants reaches G the cell changes its state in order to start the final phase (at time $t = 22$).

V. Conclusions

We have presented the first solution to the FSSP on MUCCA for dimension two and higher and for arbitrary general positions. The time needed for the synchronization is linear with respect to the sum of the sides of the automaton.

Recall that the original one-dimensional solution for the FSSP on MUCCA does not need an initial exploration phase [17] and it can be as fast as the optimal solutions in standard CA [7] when no slow cell is present. It is still open if similar results can be obtained for higher dimensions or for arbitrary positions of the general, thus speeding up the solution presented here. Furthermore, it is still unknown what are the optimal time and optimal number of states necessary to have a solution to the FSSP on MUCCA. It would also be interesting to investigate if it is possible to improve the run time of an FSSP solution when the distribution of the cell speeds is partially or even completely known.

References

[1] E. Moore, "The Firing Squad Synchronization Problem," *Sequential machines: selected papers*, p. 213, 1964.

[2] M. Minsky, "Computation: finite and infinite machines," *Englewood Cliffs, N.J.: Prentice-Hall*, 1967.

[3] E. Goto, "A minimal time solution of the firing squad problem," *Dittoed course notes for Applied Mathematics*, vol. 298, pp. 52–59, 1962.

[4] A. Waksman, "An optimum solution to the firing squad synchronization problem," *Information and Control*, vol. 9, no. 1, pp. 66–78, 1966.

[5] R. Balzer, "An 8-state minimal time solution to the firing squad synchronization problem," *Information and Control*, vol. 10, no. 1, pp. 22–42, 1967.

[6] H. Gerken, "Über Synchronisationsprobleme bei Zellularautomaten," Ph.D. dissertation, Technische Universität Braunschweig, 1987.

[7] J. Mazoyer, "On optimal solutions to the firing squad synchronization problem," *Theoretical Computer Science*, vol. 168, no. 2, pp. 367–404, 1996.

[8] W. Beyer, "Recognition of topological invariants by iterative arrays," Ph.D. dissertation, Massachusetts Institute of Technology, 1969.

[9] A. Grasselli, "Synchronization of cellular arrays: The firing squad problem in two dimensions," *Information and Control*, vol. 28, no. 2, pp. 113–124, 1975.

[10] I. Shinahr, "Two-and three-dimensional firing-squad synchronization problems," *Information and Control*, vol. 24, no. 2, pp. 163–180, 1974.

[11] N. Fatès, M. Morvan, N. Schabanel, and E. Thierry, "Fully asynchronous behaviour of double-quiescent elementary cellular automata," *Theoretical Computer Science*, vol. 362, pp. 1–16, 2006.

[12] L. Manzoni, "Asynchronous cellular automata and dynamical properties," *Natural Computing*, vol. 11, no. 2, pp. 269–276, 2012.

[13] A. Dennunzio, E. Formenti, L. Manzoni, and G. Mauri, "m-Asynchronous Cellular Automata: from fairness to quasi-fairness," *Natural Computing*, vol. 12, no. 4, pp. 561–572, 2013.

[14] 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.

[15] ——, "(Intrinsically?) Universal Asynchronous Cellular Automata II," in *Proceedings of Automata 2012*, 19-21 September 2012, Bastia (France), 2012, pp. 222–235.

[16] A. Dennunzio, E. Formenti, and L. Manzoni, "Computing Issues of Asynchronous CA," *Fundamenta Informaticae*, vol. 120, no. 2, pp. 165–180, 2012.

[17] L. Manzoni and H. Umeo, "The firing squad synchronization problem on CA with multiple updating cycles," *Theoretical Computer Science*, vol. 559, pp. 108–117, 2014.

[18] H. Umeo, M. Maeda, and N. Fujiwara, "An Efficient Mapping Scheme for Embedding Any One-Dimensional Firing Squad Synchronization Algorithm onto Two-Dimensional Arrays," in *Cellular Automata: 5th International Conference on Cellular Automata for Research and Industry, ACRI 2002 Geneva, Switzerland, October 9–11, 2002 Proceedings*, S. Bandini, B. Chopard, and M. Tomassini, Eds. Springer, 2002, pp. 69–81.

[19] H. Umeo, "A simple design of time-efficient firing squad synchronization algorithms with fault-tolerance," *IEICE Transactions on Information and Systems*, vol. 87, no. 3, pp. 733–739, 2004.

[20] M. Kutrib and R. Vollmar, "Minimal time synchronization in restricted defective cellular automata," *Journal of Information Processing and Cybernetics*, vol. 27, no. 3, pp. 179–196, 1991.

[21] ——, "The firing squad synchronization problem in defective cellular automata," *IEICE Transactions on Information and Systems*, vol. 78, no. 7, pp. 895–900, 1995.

[22] J. Yunès, "Fault tolerant solutions to the firing squad synchronization problem in linear cellular automata," *Journal of Cellular Automata*, vol. 1, no. 3, pp. 253–268, 2006.