Nat Comput (2018) 17:455–465
https://doi.org/10.1007/s11047-017-9628-z

Revisiting the cutting of the firing squad synchronization

Antonios Dimitriadis[1] · Martin Kutrib[2] · Georgios Ch. Sirakoulis[1]

Published online: 28 July 2017

Abstract Various synchronization algorithms have been introduced in literature during the last decades to deal with the firing squad synchronization problem on cellular automata (CA). Among others defective CA algorithms, where the CA cell is able to transmit information without previous processing, have been also presented. In our case, originating from the classical Mazoyer's paper, where a minimum-time solution is presented with 6 states, a one-dimensional CA where one cell may permanently fail is presented. In the proposed algorithm, the defective cell can neither process nor transmit information any longer, while it is considered that such dynamic defects may become apparent in any time step of computation. A thorough analysis of the synchronization, in terms of location and time at which cell fails, for the cells found in both sides of defective cell is delivered to decipher the corresponding maximal possible number of synchronized cells in each part of the cut, due to defect, CA array. The proposed algorithm is properly extended to consider more than one defective cells that may occur in the under study one-dimensional CA. Based on the aforementioned analysis, we provide the generalization of synchronization with multiple totally defective cells, while application examples of the generalized CA algorithm in case of two defective cells are also presented. Finally, another intriguing aspect refers to handling of states that could be tentatively characterized as unknown, in a confrontation similar to the previous defective state but also different, since now this(these) cell(s) are not stated as faulty but unknown. As a result, a new one-dimensional CA with less states, compared to the previous CA defective algorithms, able to synchronize the maximal possible number of cells in each part occurs.

Keywords Defective cellular automata · Firing squad synchronization problem · Fault tolerance · Synchronization algorithm · Multiple totally defective cells · Unknown states

✉ Georgios Ch. Sirakoulis
gsirak@ee.duth.gr

Antonios Dimitriadis
andimitr@ee.duth.gr

Martin Kutrib
kutrib@informatik.uni-giessen.de

1 Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece

2 Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany

1 Introduction

Nowadays it becomes possible to build massively parallel computing systems that consist of hundred thousands of processing elements. Each single component is subject to failure such that the probability of misoperations and loss of function of the whole system increases with the number of its elements. It was von Neumann (1956) who first stated the problem of building reliable systems out of unreliable components. Here we consider one-dimensional CA as a model for homogeneously structured parallel systems as are linear processor arrays. Such devices of interconnected parallel acting finite-state machines have been studied from the viewpoint of fault tolerance in several ways. In Gács (1986) reliable arrays are constructed under the assumption that a cell (and not its links) at each time step fails with a constant probability. Moreover, such a failure does not incapacitate the cell permanently, but only violates its rule of operation in the step when it occurs.

Under the same constraint that cells themselves (and not their links) fail (that is, they cannot process information but are still able to transmit it unchanged with unit speed) fault-tolerant computations have been investigated, for example, in Harao and Noguchi (1975), Nishio and Kobuchi (1975) where encodings are established that allow the correction of so-called K-separated misoperations, in Kutrib and Löwe (2002) where the studies are in terms of syntactical pattern recognition, in Kutrib and Vollmar (1991, 1995), Umeo (1994, 2004), Yunès (1996) where the firing squad synchronization problem is considered in defective cellular arrays, and in Fay and Kutrib (2004) where the early bird problem (Rosenstiehl et al. 1972) is investigated.

However, in the previous studies defective cells are considered such that cells still can transmit information without processing it. Here we consider defective CA where the dynamic defects are such that a defective cell totally fails (Dimitriadis et al. 2016). The failures are permanent and may occur at any time in the computation. In this way the array is cut into two parts. Our study is in terms of the famous Firing Squad Synchronization Problem (FSSP). This is done because in our opinion the FSSP is characteristic of hard parallel problems. Its solutions are myopic: from local properties a global behavior is achieved. Moreover, synchronization problems are of practical relevance. The FSSP was raised by Myhill in 1957 and emerged in connection with the problem to start several parts of a parallel machine at the same time. The first published reference appeared with a solution found by McCarthy and Minsky in Moore (1964). Roughly speaking, the problem is to set up a CA such that all cells change to a special state for the first time after the same number of steps. Many modifications and generalizations of the FSSP have been investigated. Just to mention a few of them, in Čulik (1989) Culik II has used some variations of the FSSP to design parallel algorithms for one-way automata networks. Waksman Waksman (1966) and Balzer Balzer (1967) have given minimal time solutions to the ordinary FSSP. Mazoyer Mazoyer (1987) has obtained a six-state algorithm. A variety of generalizations have been considered. Synchronization speed-ups in cellular automata with busses have been shown by Vollmar (1991). Solutions for higher dimensions can be found in Grasselli (1975), Kobayashi (1978b), Rosenstiehl et al. (1972), Shinahr (1974), Szwerinski (1982), Umeo et al. (2006). Herman et al. (1974) give solutions for cellular arrays, the lengths of which "grow". The problem of starting the synchronization from non-border automata has been solved by Moore and Langdon (1968). General networks have been considered in Čulik and Dube (1991), Jiang (1992), Kobayashi (1978a, b), Romani (1978), Rosenstiehl et al. (1972). Fault tolerant synchronizations are studied in Kutrib and Vollmar (1995), Umeo (2004), and in Imai and Morita (1996) the problem is solved for reversible cellular spaces. There are many more papers dealing with the FSSP. A valuable source of further references and topics is the survey by Umeo (2009).

In this paper, based on the aforementioned confrontation that the defective cell can neither process nor transmit information any longer, while it is considered that such dynamic defects may become apparent in any time step of computation, a novel CA algorithm originated from Mazoyer's 6-state algorithm is presented able to address the problem of more than one totally defective cells. For doing so, initially, a thorough analysis of the synchronization, in terms of location and time at which cell fails, for the cells found in both sides of defective cell is delivered to decipher the corresponding maximal possible number of synchronized cells in each part of the cut, due to defect, CA array. Moreover, application examples of the presented CA algorithm for randomly selected defective cells are also discussed. It has been found that Implementations of the proposed algorithm show that the algorithm has an average of 78% synchronization success, which means that in some cases a small number of cells could finally remain unsynchronized. In this context, another aspect of misbehaving cells is considered; namely handling of states that could be tentatively characterized as unknown, in a confrontation similar to the previous defective state but also different, since now this(these) cell(s) are not stated as faulty but unknown, is also investigated. As a result, a new one-dimensional CA with less states but also different synchronization time, compared to the previous CA defective algorithms, able to synchronize the maximal possible number of cells in each part occurs. Finally, the concept of more than one totally defective CA cells that totally fail during the synchronization process is also introduced. Based on the provided analysis, the generalization of synchronization with multiple totally defective cells is proposed. Application examples of the generalized CA algorithm in case of two defective cells are also presented.

2 Preliminaries

Let A denote a finite set of letters. Then we write A^* for the set of all finite words (strings) built with letters from A and A^+ for the set of all non-empty words. We use $\subseteq$ for set inclusion and $\subset$ for strict set inclusion. For a set S and a symbol a we abbreviatory write S_a for $S \cup \{a\}$.

A one-dimensional CA is a linear array of identical deterministic finite-state machines, called cells. Except for the leftmost cell and rightmost cell each one is connected to its both nearest neighbors. We identify the cells by positive integers. The state transition depends on the current state of a cell itself and the current states of its two neighbors, where the outermost cells receive a permanent boundary symbol on their free input lines (Fig. 1).

Definition 1 A *cellular automaton* (*CA*) is a system $M = \langle S, A, \#, \delta \rangle$, where S is the finite, nonempty set of *cell states*, $A \subseteq S$ is the nonempty set of *input symbols*, $\# \notin S$ is the permanent *boundary symbol*, $\delta : S_\# \times S \times S_\# \longrightarrow S$ is the *local transition function*.

A *configuration* c_t of M at time $t \geq 0$ is a string of the form $\#S^*\#$, that reflects the cell states from left to right. The computation starts at time 0 in a so-called *initial configuration*, which is defined by the input $w = a_1 a_2 \cdots a_n \in A^+$. We set $c_0 = \#a_1 a_2 \cdots a_n\#$. During the course of its computation a *CA* steps through a sequence of configurations, whereby successor configurations are computed according to the global transition function Δ: Let c_t be a configuration reached at time $t \geq 0$ in some computation. Then its successor configuration $c_{t+1} = \Delta(c_t)$ is as follows. For $2 \leq i \leq n-1$, $c_{t+1}(i) = \delta(c_t(i-1)), c_t(i), c_t(i+1))$, and for the leftmost and rightmost cell we set $c_{t+1}(1) = \delta(\#, c_t(1), c_t(2))$ and $c_{t+1}(n) = \delta(c_t(n-1), c_t(n), \#)$, for $t \geq 0$. Thus, the global transition function Δ is induced by δ.

Next, we consider CA with dynamic defects. In Kutrib and Löwe (2002) dynamic defects have been studied so that a defective cell can still transmit information without processing it. In this way the array is not cut into pieces. Here we investigate dynamic defects so that a defective cell totally fails, such failures are permanent and may occur at any time in the computation. In order to define CA with this type of defects more formally, a possible failure is seen as a weak kind of nondeterminism for the local transition function.

Definition 2 A cellular automaton $M = \langle S, A, \#, \delta \rangle$ is a *cellular automaton with (totally) dynamic defects* (*TDCA*), if δ is extended so that it may map any triple from $S_\# \times S \times S_\#$ to $S_\#$, that is, either to a state from S or alternatively to the boundary symbol #.

If a cell works fine the local transition function maps to a state from S. Otherwise it maps to #. In the latter case, for the remaining computation the cell behaves as the boundary to its both neighbors. Since the transition function is not defined for #, the failure is permanent and the cell can be seen as totally defective. The time step at which a cell enters the boundary symbol from a non-boundary symbol is said to be the time step at which the cell *fails* or its *failure time*. We assume that initially all cells are intact and, thus, no cell fails at time 0.

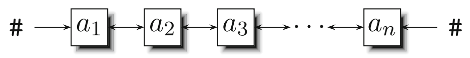

Fig. 1 A one-dimensional cellular automaton

3 The firing squad synchronization problem

Roughly speaking, the problem is to set up a CA such that all cells change to a special state for the first time after the same number of steps. Originally, the problem has been stated as follows: Consider a finite but arbitrary long chain of finite automata that are all identical except for the automata at the ends. The automata are called soldiers, and the automaton at the left end is the general. The automata work synchronously, and the state of each automaton at time step $t+1$ depends on its own state and on the states of its both immediate neighbors at time step t. The problem is to find states and state transitions such that the general may initiate a synchronization in such a way that all soldiers enter a distinguished state, the *firing state*, for the first time at the same time step. At the beginning all non-general soldiers are in the quiescent state. More formally, the FSSP is defined as follows.

Definition 3 Let C be the set of all configurations of the form $\#GQQ \cdots Q\#$. The *Firing Squad Synchronization Problem* is to specify a CA $\langle S, A, \#, \delta \rangle$ so that for all $c \in C$,

1. there is a *synchronization time* $t_f \geq 1$ such that $\Delta^{t_f}(c) = \#FF \cdots F\#$,
2. for all $0 \leq t < t_f$, $\Delta^t(c) = \#X_1 X_2 \cdots X_n\#$ with $X_i \neq F$, $1 \leq i \leq n$, and
3. $\delta(Q, Q, Q) = \delta(\#, Q, Q) = \delta(Q, Q, \#) = Q$.

While the first solution of the problem takes $3n$ time steps to synchronize n cells (Moore 1964), Goto (1962) was the first who presented a minimal-time solution that uses several thousand states (see Umeo (1996), Yunès (2009) for a reconstruction of this algorithm). The minimal solution time for the FSSP is $2n-2$ (Waksman 1966).

Apart from time optimality there is a natural interest in efficient solutions with respect to the number of states or the number of bits to be communicated to neighbors. While there exists a time optimal solution where just one bit of information is communicated (Mazoyer 1989), the minimal number of states is still an open problem. The first time optimal solution (Goto 1962) uses several thousand states. The algorithm from Waksman (1966) takes 16 states. About one year later, an eight state time optimal solution was published (Balzer 1967). Currently, a six-state solution is known (Mazoyer 1987). In the same paper it is proved that there does not exist a time optimal four-state algorithm. It is a challenging open problem to prove or disprove that there exists a five-state solution.

Since the algorithm to be presented here relies on Mazoyer's solution, we next sketch the basic idea from Mazoyer (1987).

Algorithm 4 The FSSP is solved by iteratively dividing the array of length n into parts on which the same algorithm is applied recursively (see Fig. 2). First the array is divided into two parts. Then the process is applied to both parts in parallel, etc. The cut-points are chosen so that one of the parts is twice as long as the other (up to the remainder of n modulo 3). Exactly when all cells are cut-points they enter the firing state synchronously.

In order to divide the array into two parts, the general sends two signals to the right. One signal moves with speed 1, that is, one cell per time step, and the other signal with speed 1/2, that is, one cell every other time step. The fast signal is bounced at the right end and is sent back to the left with speed 1. Both signals meet at position $2/3 \cdot n$ (up to the remainder of n modulo 3), where the cell becomes a general. Now the right part is treated as an array of length $(n+i)/3$, where $i \in \{0,1,2\}$ so that the synchronization starts with 0, 1, or 2 steps delay.

The next cut-point in the left part, which is at total position $(2/3)^2$, can be determined by another signal sent at initial time by the general at the left end. This signal moves with speed 2/7. It meets the bounced signal from above at the required position. In order to determine the cut-point at total position $(2/3)^3$ in this way, the general has to send a further signal with speed 4/23, and so on. Altogether, for a solution the general has to send a number of signals that depends on the length of the array.

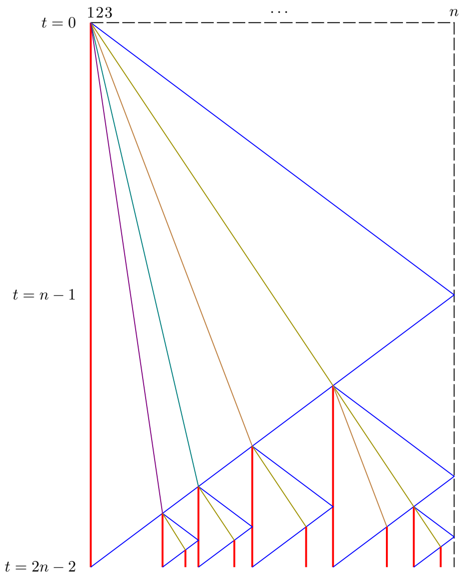

Fig. 2 Scheme of the time-optimal 6-state FSSP solution. The *vertical solid lines* are cells in the general state. For the sake of clarity not all signals are depicted

In order to send this number of signals with a finite state set, an approach shown in Waksman (1966) can be adopted. The idea is rather simple, the additional signals are generated and moved by trigger signals. The left-moving trigger signals themselves are emitted by the initial right-moving signal. Whenever a trigger signal reaches a triggered signal, the latter is moved. A triggered signal deletes every other trigger signal that arrives at it. Moreover, whenever a trigger signal reaches the leftmost cell, a new signal to be triggered is generated. That way, the desired behavior is achieved in Waksman (1966), and a minimal-time solution for the FSSP is obtained. Since in Mazoyer's solution, the array is not split in the middle but at position $2/3 \cdot n$ the implementation of the trigger signals is more involved but follows the same underlying idea. These signals are referred to as *family of slow signals* in Mazoyer (1987). □

4 Synchronization with a totally defective cell

In this section we turn to consider the effect of a cell becoming totally defective on Mazoyer's algorithm that is extended. For non-defective CA, the algorithm runs in optimal time, that is, in time $2n-2$ where n is the number of cells to be synchronized. For the case that a cell k with $1<k<n$ fails the array is cut into two independent parts, that is, into the cells $1,2,\ldots,k-1$ on the left and the cells $k+1,k+2,\ldots,n$ on the right. The problem is now to synchronize these two parts independently of each other, if possible at all. However, this may yield extended synchronization times. The main goal in the sequel is to determine for a given situation how many cells can still be synchronized, and how many time steps are needed. The algorithm depends naturally on the time step at which a cell fails (recall that this is the time step at which it enters the boundary symbol from a non-boundary state) and its position in the array. For our notation, in the following we assume that at most one cell k with $1<k<n$ fails at time step t_d with $0<t_d\leq 2n-2$.

In general, cell $k-1$ has to detect that the failure occurred. This means, it has to distinguish between a boundary symbol to its right that is due to a failure and a boundary symbol that is initial. On the other hand, this distinction is irrelevant if the failure occurs when cell $k-1$ has not received the initial signal. So, it is sufficient that each cell remembers the information whether or not its right neighbor is the boundary symbol when the cell receives the initial signal. To this end, no further copies of the states are used, but the remembering is successfully encapsulated in the transition function.

In general, since the leftmost cell is cell 1, the running time of the initial signal to cell k is $k-1$ time steps.

4.1 Analysis for the left part

When cell k fails at time t_d, its left neighbor enters a state that may depend on the fact that a failure occurs at the earliest at time $t_d + 1$. The actual behavior of cell $k-1$ at time $t_d + 1$ depends on the position of the defective cell, that is, on k and on t_d. See Figs. 3 and 4 for examples.

Case 1 If $t_d \leq k-3$, then the initial signal sent by the leftmost cell of the FSSP still did not arrive at cell $k-1$ when its neighbor failed. The running time of this signal to cell $k-1$ is $k-2$ time steps. So, cell k acts a boundary cell for the FSSP that synchronizes all the $k-1$ cells of the left part in $2(k-1)-2 = 2k-4$ steps. This behavior does not apply to the case $k-2 \leq t_d$. The reason is that the state of cell $k-1$ at time $k-2$ is given by the transition function that sees the quiescent state in cell k at time $k-3$. So, if cell k fails at time $k-2$, then the state of cell $k-1$ does not reflect the bounced signal.

Case 2 Let $k-2 \leq t_d \leq 2n-k-1$. Then cell $k-1$ was already reached by the initial signal and the synchronization is in progress. Therefore, the algorithm we consider is set up so that cell $k-1$ informs the cells of the left part about the failure and to stop the running FSSP. To this end, it sends a signal to the left. This signal is started at time $t_d + 1$ and arrives at time $t_d + 1 + k - 2 = t_d + k - 1$. If the synchronization time $2n-2$ of the running FSSP is after the arrival time of the signal in cell 1, none of the cells in the left part will fire according to the running FSSP. This happens if $2n-2 \geq t_d + k - 1$ and, thus, $t_d \leq 2n-k-1$.

Now the algorithm is further extended such that the signal that stops the running synchronization is additionally the initial signal of a new (mirrored) FSSP instance where the synchronization is initiated by the rightmost cell of the array. In particular, this implies that the left part is synchronized in $2(k-1)-2$ steps after the signal has been emitted. That is, the synchronization takes place at time step $t_d + 1 + 2(k-1) - 2 = t_d + 2k - 3$.

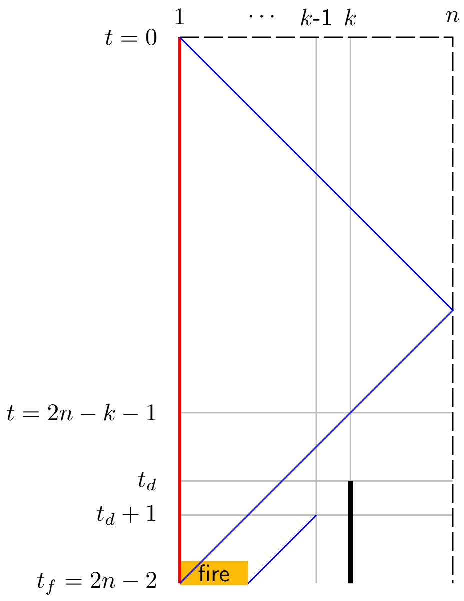

Fig. 4 Example for Case 3 of the analysis for the *left part*. The *vertical fat solid line* shows the defect of cell k. For the sake of clarity only the initial right-moving signal and its returns are depicted

For example, if $k=2$ and $t_d=2$ then Case 1 cannot apply, but Case 2 may. Thus, the sole cell 1 is synchronized at time step 2.

It is worth mentioning that the mirrored FSSP costs extra states. Here we can trade states for a slowdown as follows. Cell $k-1$ still sends the signal to the left in order to stop the running FSSP. If the signal arrives in cell 1 a new (non-mirrored) FSSP is initiated that synchronizes the left part in further $2(k-1)-2$ steps, that is, at time $t_d + 1 + k - 2 + 2(k-1) - 2 = t_d + 3k - 5$. Since the new signal requires just one additional state, we trade one state for $k-2$ additional time steps.

Case 3 Let $2n-k \leq t_d \leq 2n-2$. In this case, the signal emitted by cell $k-1$ to stop the running FSSP does not

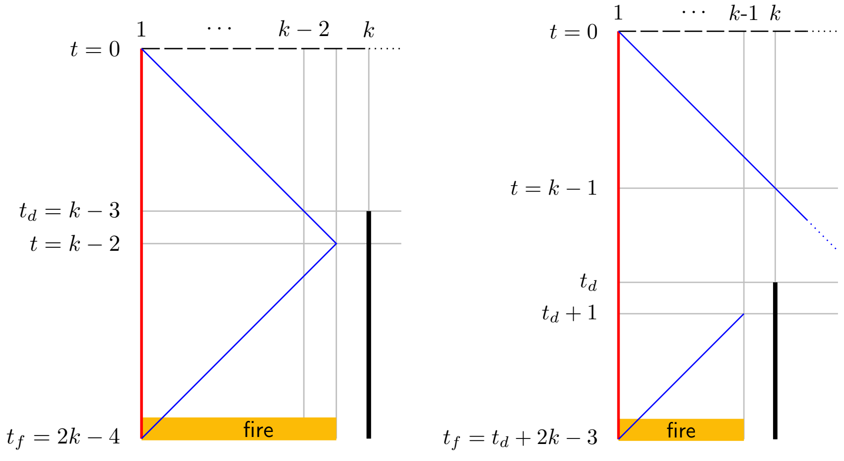

Fig. 3 Examples for the analysis for the *left part*; Case 1 (*left*) and Case 2 (*right*). The *vertical fat solid lines* show the defect of cell k. For the sake of clarity only the initial right-moving signal and its returns are depicted

reach all cells of the left part before time step $2n-2$ at which the running synchronization takes place. However, at time $t_d + x$ the signal has affected x cells, where $x \geq 1$. Setting $2n-2 = t_d + x$ implies $x = 2n-2-t_d$. So, $2n-2-t_d$ cells are affected and, thus, not synchronized by the running FSSP. Conversely, this means that $k-1-(2n-2-t_d) = t_d - 2n + k + 1$ cells are synchronized at time step $2n-2$.

For example, if $t_d = 2n-k$ then just one cell is synchronized. This is the leftmost cell that cannot be reached by the signal in due time. Setting $t_d = 2n-2$ gives $k-1$ synchronized cells. These are all cells in the left part since the synchronization takes place at the time cell k fails.

Table 1 summarizes the results for the left part.

4.2 Analysis for the right part

As for the left part, when cell k fails at time t_d, its right neighbor enters a state that may depend on the fact that a failure occurs at the earliest at time $t_d + 1$. The actual behavior of cell $k+1$ at time t_d+1 depends on the position of the defective cell, that is, on k and on t_d. See Figs. 5 and 6 for examples.

Case 1 If $t_d \leq k-1$, then the initial signal sent by the leftmost cell of the FSSP still did not arrive at cell $k+1$ when its left neighbor failed. The running time of this signal to cell $k+1$ is k time steps. So, when cell $k+1$ is still in the quiescent state with a boundary to its left, it can start a new instance of a FSSP that synchronizes the right part. Here we note that a quiescent cell next to the left boundary does not occur without a failure, since initially the leftmost cell is in the general state. The new instance is set up when cell $k+1$ enters the general state at time t_d+1. Then it takes another $2(n-k)-2$ steps to synchronize all the $n-k$ cells of the right part. That is, the right part is synchronized at time $t_d + 1 + 2(n-k) - 2 = t_d + 2n - 2k - 1$.

Case 2 Let $k \leq t_d \leq n+k-2$. Then cell $k+1$ was already reached by the initial signal and the synchronization is in progress. Therefore, the algorithm we consider is set up so that cell $k+1$ informs all cells of the right part

Table 1 Summary of synchronization times and cells in the left part, where t_d denotes the time of failure, the columns with head *Cells* show the number of cells synchronized, and t_f denotes the time step at which the cells are synchronized

t_d	Cells	t_f
Left part		
$[1,\ldots,k-3]$	All	$2k-4$
$[k-2,\ldots,2n-k-1]$	All	t_d+2k-3
$[2n-k,\ldots,2n-2]$	$t_d-2n+k+1$	$2n-2$

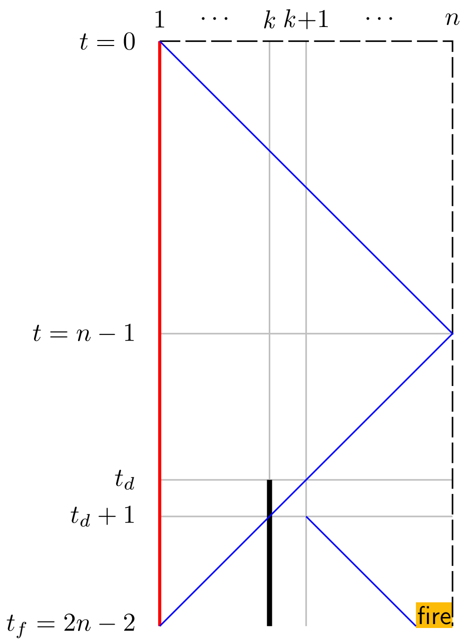

Fig. 6 Example for Case 3 of the analysis for the *right part*. The *vertical fat solid line* shows the defect of cell k. For the sake of clarity only the initial right-moving signal and its returns are depicted

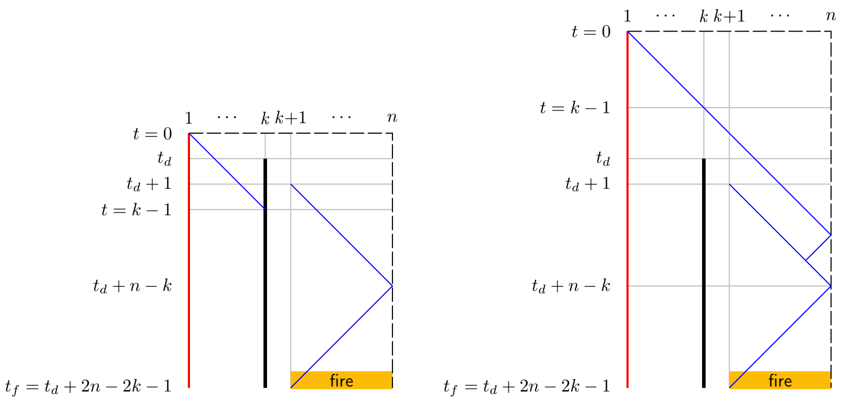

Fig. 5 Examples for the analysis for the *right part*; Case 1 (*left*) and Case 2 (*right*). The *vertical fat solid lines* show the defect of cell k. For the sake of clarity only the initial right-moving signal and its returns are depicted

about the failure and to stop the running FSSP. To this end, it sends a signal to the right. This signal is started at time $t_d + 1$ and arrives at time $t_d + 1 + n - k - 1 = t_d + n - k$. If the synchronization time $2n - 2$ of the running FSSP is after the arrival time of the signal in cell n, none of the cells in the right part will fire according to the running FSSP. This happens if $2n - 2 \geq t_d + n - k$ and, thus, $t_d \leq n + k - 2$.

Now, as for the left part, the algorithm is extended such that the signal that stops the running synchronization is additionally the initial signal of a new FSSP instance. This implies that the right part is synchronized in $2(n - k) - 2$ steps after the signal has been emitted. That is, the synchronization takes place at time step $t_d + 1 + 2(n - k) - 2 = t_d + 2n - 2k - 1$.

Case 3 Let $n + k - 1 \leq t_d \leq 2n - 2$. In this case, the signal emitted by cell $k + 1$ to stop the running FSSP does not reach all the cells of the right part before time step $2n - 2$ at which the running synchronization takes place. However, at time $t_d + x$ the signal has affected x cells, where $x \geq 1$. Setting $2n - 2 = t_d + x$ implies $x = 2n - 2 - t_d$. So, $2n - 2 - t_d$ cells are affected and, thus, not synchronized by the running FSSP. Conversely, this means that $n - k - (2n - 2 - t_d) = t_d - n - k + 2$ cells are synchronized at time step $2n - 2$.

For example, if $t_d = n + k - 1$ then just one cell is synchronized. This is the rightmost cell that cannot be reached by the signal in due time. Setting $t_d = 2n - 2$ gives $n - k$ synchronized cells. These are all cells in the right part since the synchronization takes place at the the time cell k fails.

Table 2 summarizes the results for the right part.

Furthermore, we have dealt with the situation where a CA cell is found in an unknown state, not including in any of the previous states as mentioned before. In order to handle this situation a new CA algorithm, based on the aforementioned CA algorithm, is proposed minimizing the total number of requested states to 11. The analysis and the functionality of this novel algorithm is similar with the one presented and their main difference occurs at the confrontation of the extra state, characterized as unknown and results to different time for synchronization, namely $2n - 1$ in case no unknown states occurs. In case of an unknown CA cell state, the previously presented analysis occurs with similar time frames, i.e. t_f for the left part and $t_f + 1$ for the right part. It should be mentioned that these changes in the provided synchronization times result from the new 11-state algorithm due to different number of transition rules.

Table 2 Summary of synchronization times and cells in the right part, where t_d denotes the time of failure, the columns with head *Cells* show the number of cells synchronized, and t_f denotes the time step at which the cells are synchronized

t_d	Cells	t_f
Right part		
$[1, \ldots, k - 1]$	All	$t_d + 2n - 2k - 1$
$[k, \ldots, n + k - 2]$	All	$t_d + 2n - 2k - 1$
$[n + k - 1, \ldots, 2n - 2]$	$t_d - n - k + 2$	$2n - 2$

5 Graphical representation of two examples

In the first example a CA with 17 cells is considered. Let cell 9 fail at time step 15. A simulation of the original algorithm from Mazoyer (1987) is depicted in the left part of Fig. 7. The boundary cells are represented in yellow. The cells to the right of the failure are left unsynchronized and are depicted in orange, while the cells to the left of the failure which are still synchronized are drawn in brown.

At the middle part of Fig. 7 the extended algorithm is simulated. In particular, all non-defective cells are synchronized (though the left and the right part fire independently at different time steps).

Moreover, for exactly the same example, namely a CA with 17 cells and cell 9 found in an unknown state at time step 15 and remaining at the same state for the rest of the CA time evolution, the results of the proposed algorithm are shown in Fig. 7 at the bottom part. Please consider that boundary cells are depicted in lime color, while the finally synchronized cells are depicted in brown color.

In the second example a CA with 26 cells is presented (see Fig. 8) and is supposed that cell 12 fails at time step 14. Again, at the upper part of the figure a simulation of the original algorithm is shown. The colors are as before. Note, that none of the cells fires. At the middle part of Fig. 8, a simulation based on the extended algorithm is presented. As in the first example, now all non-defective cells are synchronized, where the firing times for the left and right part necessarily differ. Finally, like before, for the same example, i.e. CA with 26 cells and cell 12 found in an unknown state at time step 14 and remaining at the same state for the rest of the CA time evolution, the results of the proposed algorithm for handling unknown state cell are shown in Fig. 8 at the bottom part. Please consider that the same colors apply as before.

6 Generalization of synchronization with multiple totally defective cell

As before we consider one-dimensional cellular automata of n cells, and we will present the application of our proposed algorithm in the case of more than one totally defective cell. In particular, as depicted in Fig. 9, we consider that the coloured red cell found in place n_1 fails at

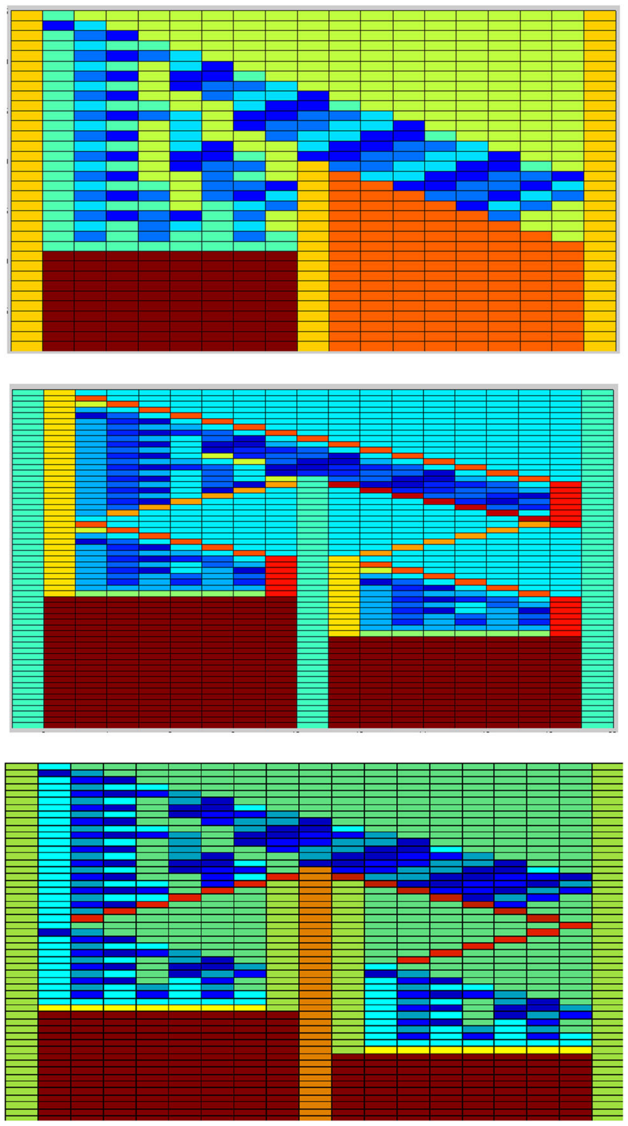
Fig. 7 Simulations of the first example. The original algorithm (*upper*) and the extended algorithm for totally defective cell (*middle*) and the new algorithm (*below*) for unknown state cell. Boundary cells are depicted in *yellow* (for original algorithm), *turquoise* (extended defective algorithm) and *lime* (unknown state algorithm), respectively, finally synchronized cells in *orange* and *brown*, and non-synchronized cells in *orange*. For all the algorithms, time $t = 0$ corresponds to the *upper row* as provided in the previous theoretical examples. (Color figure online)

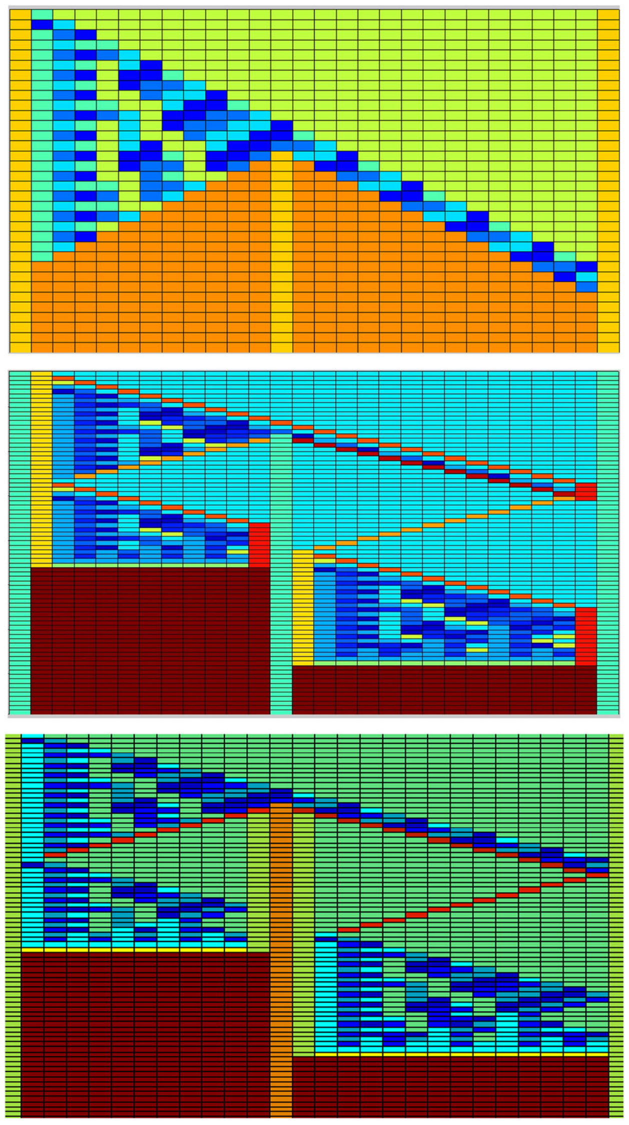
Fig. 8 Simulations of the second example. The original algorithm (*upper*) and the extended algorithm for totally defective cell (*middle*) and the new algorithm (*below*) for unknown state cell. Boundary cells are depicted in *yellow* (for original algorithm), *turquoise* (extended defective algorithm) and *lime* (unknown state algorithm), respectively, finally synchronized cells in *orange* and *brown*, and non-synchronized cells in *orange*. For all the algorithms, time $t = 0$ corresponds to the *upper row* as provided in the previous theoretical examples. (Color figure online)

time step t_1. However, in the examined case, a second cell located at cells $[1,\ldots,n_1-1]$ or $[n_1+1,\ldots,n]$ stops to function and it is characterized as defective (Fig. 9). Moreover, it is understood that the second cell, coloured orange, is defective at time t_2, where $t_2 \geq t_1$ and t_2 is the exact time step when n_2 is no more functional, in correspondence to time t_1 of cell n_1. As a result, we end up with two possible cases referring to the location of n_2 in correspondence to n_1.

Case 1 $n_2 \in [1,\ldots,n_1-1]$. This case applies when $n_1 > n_2$ (Fig. 9a). Consequently, three different areas are formed, namely $[1,\ldots,n_2-1], [n_2+1,\ldots,n_1-1]$ and $[n_1+1,\ldots,n]$. As already shown earlier, our proposed scheme aims to synchronize the areas $[1,\ldots,n_2-1]$ and $[n_2+1,\ldots,n_1-1]$ placed at the left and right side of cell n_2, while for the right part $[n_1+1,\ldots,n]$, it will synchronized according to the synchronization scheme of the first, that is n_1, defective cell, described in Sect. 4.2.

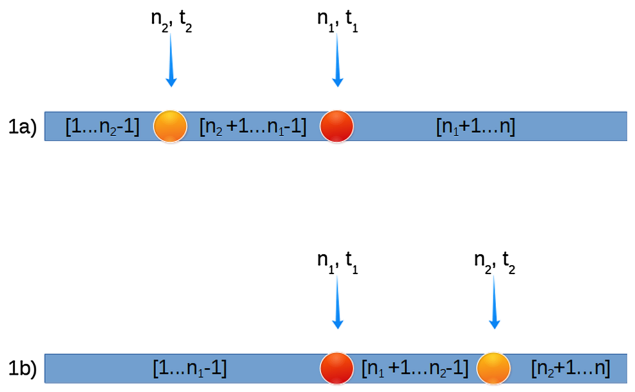

Fig. 9 The possible locations of two defective cells, n_1, *red coloured*, and n_2, *orange coloured*. It is assumed that cell n_1 failed earlier at time step t_1 than n_2, that is, $t_2 \geq t_1$. **a** The case where $n_2 \in [1, \ldots, n_1 - 1]$, **b** The case where $n_2 \in [n_1 + 1, \ldots, n]$

Case 2 $n_2 \in [n_1 + 1, \ldots, n]$. In this case, $n_1 < n_2$ applies (Fig. 9b). As a result we have the following three parts, $[1, \ldots, n_1 - 1], [n_1 + 1, \ldots, n_2 - 1]$ and $[n_2 + 1, \ldots, n]$. Based on the proposed algorithm, each defective cell is efficient enough to synchronize left and right parts beyond it. As a result, for n_2, the $[n_1 + 1, \ldots, n_2 - 1]$ and $[n_2 + 1, \ldots, n]$ parts found in both sides of n_2 will be synchronized accordingly. The missing part, that is $[1, \ldots, n_1 - 1]$, is synchronized as shown in Sect. 4.1.

6.1 Generalization in case of almost all cells

Initially, let us assume that beyond the above mentioned two defective cells, a third one, that is cell n_3, fails. In the provided analysis the algorithm categorizes hierarchically the defective cells based on their failure times. So, if we consider cell n_3 as the latest cell that failed, it should be located either between parts $[1, \ldots, n_2 - 1]$, or $[n_2 + 1, \ldots, n_1 - 1]$ or $[n_1 + 1, \ldots, n]$ as described in *Case 1*, or between parts $[1, \ldots, n_1 - 1]$ or $[n_1 + 1, \ldots, n_2 - 1]$ or $[n_2 + 1, \ldots, n]$. In any of the above parts where cell n_3 might be located in, it will synchronize the right and left parts found at its both sides, as analyzed earlier in Sect. 4. The other two cells, that is n_2 and n_1, will be synchronized as found in the previous analysis. In other words, every cell that fails will affect only the part that is placed in correspondence to the previously failed cells. This confrontation will gradually result to a tree structure able to deal with defective cells across CA grid and time.

6.2 Graphical representations for the n_1 and n_2 example cases

In the case of two n_1 and n_2 defective cells example, a CA with 25 cells is considered. The n_1 located at cell 11 fails at time step 14, while the n_2 cell is located left of n_1 at cell 5 and fails at time step 31 (upper part of Fig. 10). For sake of the different examined cases, n_2 is placed alternatively on the right of n_1, at cell 19 and fails at time 21 as shown in the upper middle part of Fig. 10. The presented Figures correspond to the application of the generalized CA algorithm able to handle multiple totally defective cells. The boundary cells are represented in green color and it is clear that the cells in both failure places of n_1 and after in n_2 are still synchronized and are drawn in brown.

We also present application examples in case of more than one cells found in unknown state. For reasons of readability we use exactly the same conditions mentioned above, i.e. a CA with 25 cells is considered and two n_1 and n_2 cells are found in unknown state. The n_1 located at cell 11 is found in unknown state at time step 14 and remains for the rest of CA evolution, while the n_2 cell is located left of n_1 at cell 5 and is found in unknown state at time step 31 (lower middle part of Fig. 10). For sake of the different examined cases, once again, n_2 is placed alternatively on the right of n_1, at cell 19 and turns to unknown state at time 21 as shown in the lower part of Fig. 10. Please consider that this is the outcome of the new 11 state synchronization algorithm responsible for handling more than one cells found at some moment in unknown state and remaining for the rest of the CA evolution. The boundary cells are represented in green color and it is clear that the cells in both failure places of n_1 and accordingly n_2 are still synchronized and are drawn in brown.

7 Conclusions

The time-optimal solution of the FSSP by Mazoyer has been considered for one-dimensional CA where at most one cell or more than one cells may totally fail, that is, it (they) can neither process nor transmit information any longer. It is worth mentioning that other time-optimal solutions of the FSSP which obey the same recursive principle of splitting the line could have been used for our investigations as well. For example, this includes the solutions given in Balzer (1967), Goto (1962), Waksman (1966) and others. However, here we used the most state efficient optimal-time solution known. In order to synchronize as many cells as possible, the algorithm has been extended by several features. The proposed algorithm divides the initial array into two separated parts, which are treated independently. When more than one cells are found defective, e.g. two cells, the rest of the areas located in both sides of the last defective cell are also divided in two separated parts and also treated independently; the same applies when more cells fail and the aforementioned cutting of the synchronization scheme applies as many times as needed. It has been shown that the number of the cells that still can be synchronized in these cases and the

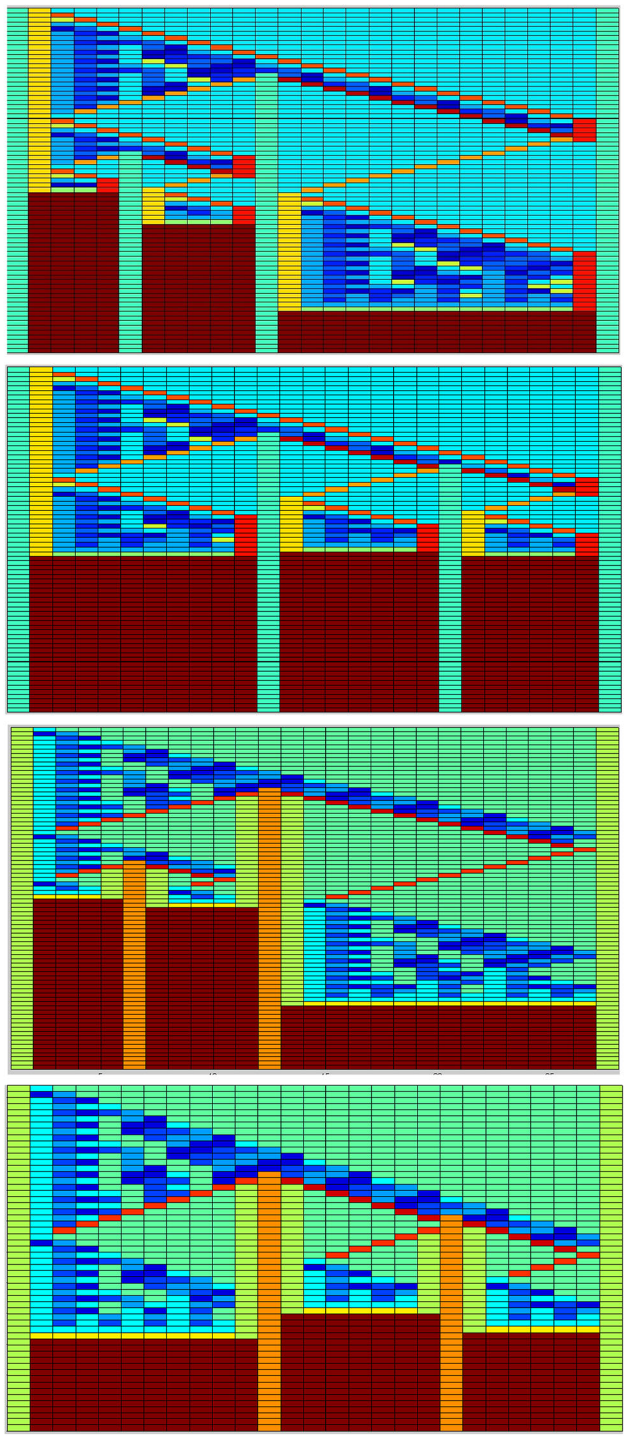

Fig. 10 Simulations of the two n_1 and n_2 defective cells case are depicted in the *upper part* of figure. The n_2 cell can be located either *left* to n_1 originally failed cell (*upper*) or its *right side* (*upper middle*). Boundary cells are depicted in *green* and finally synchronized cells in *brown*. More simulation results of two n_1 and n_2 cells found in unknown state are depicted in the *lower part* of figure. The n_2 cell can be located either left to n_1 cell found originally in unknown state (*lower middle*) or its *right side* (*lower*). Once again, boundary cells are depicted in *green* and finally synchronized cells in *brown*. For all the algorithms, time $t = 0$ corresponds to the *upper row* as provided in the previous theoretical examples

corresponding synchronization times naturally depend on the position of the defective cell and the time at which this(ese) cell(s) fails.

The proposed generalized algorithm able to deal with more than one defective cells has been implemented with 14 states. It has been tested in experiments with all array lengths between 4 and 500 and for all possible failure times and positions. The tests were run on a commercially available Windows PC and took several days running time. It turned out that the algorithm has an average of 78% synchronization success. Moreover, by a case-by-case analysis the number of synchronized cells as well as their synchronization times were derived. A definition of the minimal time to solve the problem is not that obvious as it depends on the number of cells that are synchronized.

Furthermore, the case of a new algorithm dealing sufficiently with the case of just one and more than one cells found in unknown state, is also introduced. This algorithm is implemented with less states, compared to the defective CA algorithm, namely 11 states, and different number of transition rules and results in $2n - 1$ synchronization time in case no unknown states occurs. In case of an unknown CA cell state, the previously presented analysis occurs with similar time frames, i.e. t_f for the left part and $t_f + 1$ for the right part. Like before, application examples for both cases, i.e. one cell with unknown state and two cells found in unknown state, respectively, are given.

Finally, for all the introduced algorithms it clear that a formal proof would require that the precise configuration of an array at failure time is involved in the argumentation. However, it is not hard to see that both algorithms proposed in the context of this paper all work in minimal time for the number of cells that each one synchronizes.

References

Balzer RM (1967) An 8-state minimal time solution to the firing squad synchronization problem. Inf Control 10:22–42

Čulik K II (1989) Variations of the firing squad problem and applications. Inf Process Lett 30:153–157

Čulik K II, Dube S (1991) An efficient solution of the firing mob problem. Theor Comput Sci 91:57–69

Dimitriadis A, Kutrib M, Sirakoulis G Ch (2016) Cutting the firing squad synchronization. In: El Yacoubi S, Was J, Bandini S (eds) Cellular automata—12th international conference on cellular automata for research and industry, ACRI 2016. Lecture notes in computer science, vol 9863. Springer, pp 123–133

Fay B, Kutrib M (2004) The fault-tolerant early bird problem. IEICE Trans Inf Syst E87–D:687–693

Gács P (1986) Reliable computation with cellular automata. J Comput Syst Sci 32(1):15–78

Goto E (1962) A minimal time solution of the firing squad problem. In: Course notes for applied mathematics, vol 298. Harvard University, Cambridge, MA

Grasselli A (1975) Synchronization of cellular arrays: the firing squad problem in two dimensions. Inf Control 28:113–124

Harao M, Noguchi S (1975) Fault tolerant cellular automata. J Comput Syst Sci 11:171–185

Herman GT, Liu WH, Rowland S, Walker A (1974) Synchronization of growing cellular arrays. Inf Control 25:103–122

Imai K, Morita K (1996) Firing squad synchronization problem in reversible cellular automata. Theor Comput Sci 165(2):475–482

Jiang T (1992) The synchronization of nonuniform networks of finite automata. Inf Comput 97:234–261

Kobayashi K (1978a) The firing squad synchronization problem for a class of polyautomata networks. J Comput Syst Sci 17:300–318

Kobayashi K (1978b) On the minimal firing time of the firing squad synchronization problem for polyautomata networks. Theor Comput Sci 7:149–167

Kutrib M, Löwe JT (2002) Massively parallel fault tolerant computations on syntactical patterns. Future Gener Comput Syst 18:905–919

Kutrib M, Vollmar R (1991) Minimal time synchronization in restricted defective cellular automata. J Inf Process Cybern EIK 27:179–196

Kutrib M, Vollmar R (1995) The firing squad synchronization problem in defective cellular automata. IEICE Trans Inf Syst E78–D(7):895–900

Mazoyer J (1987) A six-state minimal time solution to the firing squad synchronization problem. Theor Comput Sci 50:183–238

Mazoyer J (1989) A minimal time solution to the firing squad synchronization problem with only one bit of information exchanged. Technical Report TR 89-03, Ecole Normale Supérieure de Lyon, Lyon

Moore EF (1964) The firing squad synchronization problem. In: Moore EF (ed) Sequential machines—selected papers. Addison-Wesley, Reading, pp 213–214

Moore FR, Langdon GC (1968) A generalized firing squad problem. Inf Control 12:17–33

Nishio H, Kobuchi Y (1975) Fault tolerant cellular spaces. J Comput Syst Sci 11:150–170

Romani F (1978) The parallelism principle: speeding up the cellular automata synchronization. Inf Control 36:245–255

Rosenstiehl P, Fiksel JR, Holliger A (1972) Intelligent graphs: networks of finite automata capable of solving graph problems. In: Read RC (ed) Graph theory and computing. Academic Press, New York, pp 219–265

Shinahr I (1974) Two- and three-dimensional firing-squad synchronization problems. Inf Control 24:163–180

Szwerinski H (1982) Time optimal solution of the firing squad synchronization problem for n-dimensional rectangles with the general at an arbitrary position. Theor Comput Sci 19:305–320

Umeo H (1994) A fault-tolerant scheme for optimum-time firing squad synchronization. In: Joubert GR, Trystram D, Peters FJ, Evans DJ (eds) Parallel computing: trends and applications. North-Holland, Amsterdam, pp 223–230

Umeo H (1996) A note on firing squad synchronization algorithms. In: Kutrib M, Thomas W (eds) IFIP cellular automata workshop 1996. Universität Giessen, Giessen, p 95

Umeo H (2004) A simple design of time-efficient firing squad synchronization algorithms with fault-tolerance. IEICE Trans Inf Syst E87–D(3):733–739

Umeo H (2009) Firing squad synchronization problem in cellular automata. In: Meyers R (ed) Encyclopedia of complexity and systems science. Springer, Berlin, pp 3537–3574

Umeo H, Maeda M, Hisaoka M, Teraoka M (2006) A state-efficient mapping scheme for designing two-dimensional firing squad synchronization algorithms. Fundam. Inform. 74(4):603–623

von Neumann J (1956) Probabilistic logics and the synthesis of reliable organisms from unreliable components. In: Shannon CE, McCarthy J (eds) Automata studies. Princeton University Press, Princeton, pp 43–98

Vollmar R (1991) FSSP for cellular automata with busses. Trans IEICE E 74(9):2965–2968

Waksman A (1966) An optimum solution to the firing squad synchronization problem. Inf Control 9:66–78

Yunès JB (1996) Fault tolerant solutions to the firing squad synchronization problem. Technical Report LITP 96/06, Institut Blaise Pascal, Paris

Yunès JB (2009) Goto's construction and Pascal's triangle: new insights into cellular automata synchronization. In: Durand B (ed) Symposium on cellular automata—journées automates cellulaires (JAC 2008). MCCME Publishing House, Moscow, pp 195–203