Cutting the Firing Squad Synchronization

Antonios Dimitriadis[1], Martin Kutrib[2(✉)], and Georgios Ch. Sirakoulis[1(✉)]

[1] Department of Electrical and Computer Engineering, Democritus University of Thrace, 67100 Xanthi, Greece
gsirak@ee.duth.gr

[2] Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany
kutrib@informatik.uni-giessen.de

Abstract. The firing squad synchronization problem on Cellular Automata (CA) has been studied extensively for many years, and a rich variety of synchronization algorithms have been proposed. From Mazoyer's paper it is known that a minimal-time solution with 6 states exists. The firing squad synchronization problem has also been studied for defective CA where a defective cell can still transmit information without processing it. In the present paper, we consider defective CA where the dynamic defects are such that a defective cell totally fails. The failures are permanent and may occur at any time in the computation. In this way the array is cut into two parts. The question addressed is how many cells in each part can still be synchronized and at which time steps. It is analyzed how many cells are synchronized, where and when this happens and how these three characteristics are connected with the position of the defective cell and the time at which the cell fails. Based on Mazoyer's 6-state algorithm, a solution for one-dimensional CA is proposed that synchronizes the maximal possible number of cells in each part.

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 [12] 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 [2] 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,

DOI: 10.1007/978-3-319-44365-2_12

for example, in [4,13] where encodings are established that allow the correction of so-called K-separated misoperations, in [5] where the studies are in terms of syntactical pattern recognition, in [6,7,15,17,20] where the firing squad synchronization problem is considered in defective cellular arrays, and in [1] where the early bird problem [14] 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. 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). It 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 [11]. 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. An overview can be found in [18].

From the perspective that a cell totally fails, the question addressed here is how many cells in each of the parts caused by the failure can still be synchronized and at which time steps. It is analyzed how many cells are synchronized, where and when this happens and how these three characteristics are connected with the position of the defective cell and the time at which the cell fails. Based on Mazoyer's 6-state algorithm, a solution for one-dimensional CA is proposed that synchronizes the maximal possible number of cells in each part. 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 be remain unsynchronized.

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.

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, *$\delta : S_\# \times S \times S_\# \to S$ 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 [5] 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.

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.

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* $0 \leq t < t_f$, $\Delta^t(c) = \#X_1X_2 \cdots X_n\#$ *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 [11], Goto [3] was the first who presented a minimal-time solution that uses several thousand states (see [16,21] for a reconstruction of this algorithm). The minimal solution time for the FSSP is $2n - 2$ [19].

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 [10], the minimal number of states is still an open problem. Currently, a 6-state solution is known [9]. In the same paper it is proved that there does not exist a time-optimal 4-state algorithm. It is a challenging open problem to prove or disprove that there exists a 5-state solution.

Since the algorithm to be presented here relies on Mazoyer's solution, we next sketch the basic idea from [9].

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. 1). 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 speed 1/2, that is, one cell every other time step. The fast signal is bounced at the right end and sent back to the left with speed. 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.

In order to send this number of signals with a finite state set, an approach shown in [19] 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

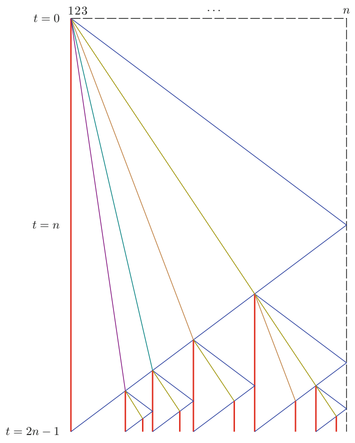

Fig. 1. Scheme of the time-optimal 6-state FSSP solution. The vertical solid lines are cells in the general state. For the sake of clarity not all signals are depicted.

signal reaches the leftmost cell, a new signal to be triggered is generated. Whenever a trigger signal reaches a triggered signal, the latter is moved. That way, the desired behavior is achieved, and a minimal-time solution for the FSSP is obtained. □

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.

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 $t_d+1+2(k-1)-2 = t_d+2k-3$.

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 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

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.

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

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.

Case 1: If $t_d \le k-1$, 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 $2(n-k-1)-2$ steps to synchronize all the $n-k$ cells of the right part. That is, the right part is synchronized at time $t_d + 1 + 2(n-k-1) - 2 = t_d + 2n - 2k - 3$.

Case 2: Let $k \le t_d \le n + k - 2$. 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 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 $2n-2 \ge t_d + n - k$ and, thus, $t_d \le n + k - 2$.

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 $2(n-k-1)-2$ steps after the signal has been emitted. That is, the synchronization takes place at time step $t_d+1+2(n-k-1)-2=t_d+2n-2k-3$.

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.

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.

Right part		
t_d	cells	t_f
$[1,\ldots,k-1]$	All	$t_d+2n-2k-3$
$[k,\ldots,n+k-2]$	All	$t_d+2n-2k-3$
$[n+k-1,\ldots,2n-2]$	$t_d-n-k+2$	$2n-2$

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 [9] is depicted in the left part of Fig. 2. The boundary cells are represented in yellow. The cells to the right of the failure are left unsynchronized and are depicted in red, while the cells to the left of the failure which are still synchronized are drawn in orange.

At the right hand side of Fig. 2 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).

In the second example a CA with 26 cells is presented (see Fig. 3) and is supposed that cell 12 fails at time step 14. Again, at the left hand side 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 right hand side of Fig. 3, 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.

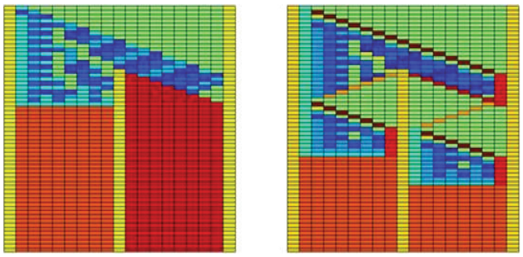

Fig. 2. Simulations of the first example. The original algorithm (left) and the extended algorithm (right). Boundary cells are depicted in yellow, finally synchronized cells in orange, and non-synchronized cells in red. (Color figure online)

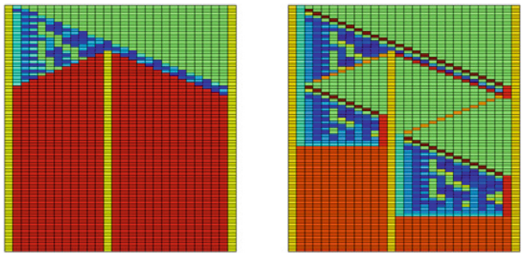

Fig. 3. Simulations of the second example. The original algorithm (left) and the extended algorithm (right). Boundary cells are depicted in yellow, finally synchronized cells in orange, and non-synchronized cells in red. (Color figure online)

6 Conclusions

The time-optimal solution of the FSSP by Mazoyer has been considered for one-dimensional CA where at most one cell may totally fail, that is, it can neither process nor transmit information any longer. 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. The number of cells that still can be synchronized and the synchronization times naturally depend on the position of the defective cell and the time at which it fails.

The new algorithm 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. Finally, 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. Moreover, 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 the algorithm proposed here works in minimal time for the number of cells that it synchronizes.

References

1. Fay, B., Kutrib, M.: The fault-tolerant early bird problem. IEICE Trans. Inf. Syst. **E87–D**, 687–693 (2004)
2. Gács, P.: Reliable computation with cellular automata. J. Comput. Syst. Sci. **32**(1), 15–78 (1986)
3. Goto, E.: A minimal time solution of the firing squad problem. Course Notes for Applied Mathematics 298, Harvard University (1962)
4. Harao, M., Noguchi, S.: Fault tolerant cellular automata. J. Comput. Syst. Sci. **11**, 171–185 (1975)
5. Kutrib, M., Löwe, J.T.: Massively parallel fault tolerant computations on syntactical patterns. Future Gener. Comput. Syst. **18**, 905–919 (2002)
6. Kutrib, M., Vollmar, R.: Minimal time synchronization in restricted defective cellular automata. J. Inform. Process. Cybern. **EIK 27**, 179–196 (1991)
7. Kutrib, M., Vollmar, R.: The firing squad synchronization problem in defective cellular automata. IEICE Trans. Inf. Syst. **E78–D**, 895–900 (1995)
8. Maignan, L., Yunès, J.-B.: Experimental finitization of infinite field-based generalized FSSP solution. In: Wąs, J., Sirakoulis, G.C., Bandini, S. (eds.) ACRI 2014. LNCS, vol. 8751, pp. 136–145. Springer, Heidelberg (2014)
9. Mazoyer, J.: A six-state minimal time solution to the firing squad synchronization problem. Theoret. Comput. Sci. **50**, 183–238 (1987)
10. Mazoyer, J.: 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 (1989)
11. Moore, E.F.: The firing squad synchronization problem. In: Sequential Machines - Selected Papers, pp. 213–214. Addison-Wesley (1964)
12. von Neumann, J.: Probabilistic logics and the synthesis of reliable organisms from unreliable components. In: Automata Studies, pp. 43–98. Princeton University Press (1956)
13. Nishio, H., Kobuchi, Y.: Fault tolerant cellular spaces. J. Comput. Syst. Sci. **11**, 150–170 (1975)
14. Rosenstiehl, P., Fiksel, J.R., Holliger, A.: Intelligent graphs: networks of finite automata capable of solving graph problems. In: Graph Theory and Computing, pp. 219–265. Academic Press (1972)
15. Umeo, H.: A fault-tolerant scheme for optimum-time firing squad synchronization. In: Parallel Computing: Trends and Applications, North-Holland, pp. 223–230 (1994)
16. Umeo, H.: A note on firing squad synchronization algorithms. In: IFIP Cellular Automata Workshop 1996, p. 95. Universität Giessen (1996)
17. Umeo, H.: A simple design of time-efficient firing squad synchronization algorithms with fault-tolerance. IEICE Trans. Inf. Syst. **E87–D**, 733–739 (2004)
18. Umeo, H.: Firing squad synchronization problem in cellular automata. In: Meyers, R.A. (ed.) Encyclopedia of Complexity and System Science, pp. 3537–3574. Springer, New York (2009)
19. Waksman, A.: An optimum solution to the firing squad synchronization problem. Inform. Control **9**, 66–78 (1966)

20. Yunès, J.B.: Fault tolerant solutions to the firing squad synchronization problem. Techical report LITP 96/06, Institut Blaise Pascal (1996)
21. Yunès, J.B.: Goto's construction and Pascal's triangle: new insights into cellular automata synchronization. In: Symposium on Cellular Automata - Journées Automates Cellulaires, JAC 2008, pp. 195–203. MCCME Publishing House (2009)