A Spatio-temporal Algorithmic Point of View on Firing Squad Synchronisation Problem

Luidnel Maignan and Jean-Baptiste Yunès

LIAFA, Université Paris-Diderot, France

Abstract. Firing Squad Synchronization Problems are well known to be solvable by voluminous transition tables describing signals traveling and colliding. In this paper, we show that it is possible to solve it by expressing directly the fact that we want a recursive division of the space into two parts of equal size, and a notification when no further division is possible. Using fields – objects associating a value to every point in space and time – as primitive objects, the solution is designed algorithmically by a semantically-intuitive decomposition of the global evolution into simpler evolutions.

The system we obtain has several interesting characteristics : it is understandable, time-optimal, tackles many initial configurations, and allows a new interpretation of the traditional signals and collisions point of view. We will quickly sketch how we can obtain a finite state automaton by reduction of the system using the Lipschitz-continuity of involved fields, and a kind of tail-recursivity property of the dependencies.

1 Introduction

Cellular automata are often described as signal machines, i.e. sets of signals or particles that move, bounce and collide in a continuous space. However, if this description is commonly sufficient to convince that a cellular automaton really computes what is wanted, it is far from clear how this continuous model helps to obtain a discrete realization in terms of a cellular automaton. But worse, it is much more unclear that a continuous idealization of a cellular automaton really helps to prove its correctness.

A typical example of this is the well-known firing squad synchronization problem (FSSP) [2,12,13], its great variety of solutions, [1] and the very poor number of proofs of their correctness [11,14,20]. It can be described as follows:

> Find a local evolution rule defined on a finite set such that, starting from arbitrary sized initial configurations where all but one cell (the general) are in a sleeping (quiescent) state, all cells synchronously enter, and for the first time, a given (fire) state.

Being quiescents at the beginning, cells will wake up one after the other from the non-quiescent one, which means that they are originally "desynchronized", and this explains why it is considered as a "synchronization" problem. This problem is as old as the cellular automaton model is and many solutions exist for the original problem or to variations or generalizations of it. For example, one can consider synchronizing with arbitrary general [21] at any arbitrary position [19], many generals synchronous or not [16], synchronizing 2D-spaces [3,6,18], 3D-spaces [17], graphs [15,5], and variants with different constraints on shape of the space. There are also solutions involving spaces that are dynamic to some extent [4]. As already said, the design and dynamics of these solutions are often explained in terms of signals. For example, in many papers one can read things like:

> [...] we launch two signals from the general: one at maximum speed 1 which bounces on the border and another one at speed $\frac{1}{3}$ that both collide right at the middle of the space. From this collision other signals may be generated [...] and as more signals are added to the process, quarters of the space, eighth of the space and so on are detected. At some point, a kind of saturation which corresponds to the fire state is obtained [...]

Such continuous intuition explains beautifully the fractal patterns that appears on the space time diagrams of the solutions. But starting from it, it is not clear how to discretize it.

In this paper, we focus only on cellular automata and discrete related objects and we will never use any continuous object or argument. Instead we propose to design a cellular automaton by decomposing the problem into easier sub-problems and compose their solutions into a global solution. To do so, we introduce the concept of dynamic fields which allow to express composable partial solutions. Informally speaking, fields are kinds of partial cellular automata. Their main interest is to have clear semantics relevant to the computation. To demonstrate their benefits, we use them to design a solution for the firing squad synchronization problem. Ultimately, we show that we recover the classical concepts of modularity, reusability, semantic decomposition, etc. All of this can be seen as an algorithmic methodology useful to synthesize cellular automata and to manage proofs of these algorithms.

But first, let us consider the simple example of Pascal's triangle modulo 2, in order to introduce and define the necessary concepts in a natural way.

1.1 Fields and Pascal's Triangle Modulo 2

Here, the goal is to synthesize a solution for the parity of the Pascal's triangle using fields. But first, let us make clear what fields are. They are defined using a local evolution rule uniformly dispatched over the space, and that has the same locality properties as CA rules (finite neighborhood, finite time dependency). In fact a cellular automaton is a particular field, but in the more general case a field may have infinite domains and may also depend on other fields.

Now, we will decompose our problem into two sub-problems: the first one being the problem of building the Pascal's triangle, and the other one to compute the parity. The first is solved defining the field T of Pascal's triangle integers which given $N(x) = \{x-1, x+1\}$, the neighborhood of cell x, is simply defined as :

$$T_0(x) = \begin{cases} 1 & \text{if } x = 0, \\ 0 & \text{otherwise.} \end{cases} \qquad T_{t+1}(x) = \sum_{y \in N(x)} T_t(y)$$

The second is solved defining P as a filtering field, i.e. a field that only depends on other fields and not on its own values at $t-1$. P operates on any arbitrary given field X of integers, extracting the parity of the values of X:

$$P[X]_t(x) = X_t(x) \mod 2$$

Now we are able to compose the two fields P and T into a field F defined as:

$$F = P[T]$$

We obviously obtain the field of the parity of Pascal's triangle, but it remains to construct a cellular automaton. The problems are that the first field is not defined over a finite set, and it is not clear at first sight that the field F is computable by a cellular automata. But, if we remark that:

$$\begin{aligned} F_{t+1}(x) &= P[T]_{t+1}(x) \\ &= \sum_{y \in N(x)} T_t(y) \mod 2 \\ &= \sum_{y \in N(x)} (T_t(y) \mod 2) \mod 2 \\ &= \sum_{y \in N(x)} P[T]_t(y) \mod 2 \\ &= \sum_{y \in N(x)} F_t(y) \mod 2 \end{aligned}$$

we are now able to easily construct a cellular automaton which computes F.

Of course in the general case, many problems remain, but these considerations are out of scope of this paper. We would rather like to show how these tools can be successfully applied with benefits to some more complex problem such as the FSSP.

2 A Field-Based Description of the FSSP

In this section, we will first explain how the FSSP can be decomposed into simpler problems, how these elementary problems are represented as fields, how these fields are finally composed and then how we will be able to obtain a cellular automaton by overcoming the unbounded number of states and fields generated.

2.1 An Algorithmic Solution

We recall that the main idea is to divide the space into two equals regions and to proceed recursively until all regions have size 1.

Naturally, the first step is to *identify the middle of the physical space.* To do this we introduce three fields. The first field, R^0, represents the discovery of the region to be cut. The second field, D^0, will provide some distance information deduced from R^0, such that this distance will eventually allows us to detect the middle. The third one is a boolean field, F^0 that indicates the correctness of the values of R^0 and D^0. Indeed, R^0 and D^0 are dynamic fields, which means that R^0 discovers the space from time to time and so its derived field D^0 also updates accordingly. Eventually R^0 stabilizes when it corresponds to the whole physical space and leading in turn to the stabilization of D^0 (see Section 2.2).

Now that we have a region and its middle, we introduce another collection of three fields: R^1 that represents the discovery of the two regions induced by the previous cut of the space, D^1 the distance field deduced from R^1 such that the middles of the two regions will be detected, and F^1 the corresponding correctness field. As the reader might guess, this extends to a recursive schema which defines R^ℓ, D^ℓ and F^ℓ in terms of $R^{\ell-1}$, $D^{\ell-1}$ and $F^{\ell-1}$ (see Section 2.3).

This obviously implies that we need an unbounded number of fields. However, we will later explain how this can be reduced to a finite system (see Section 2.4).

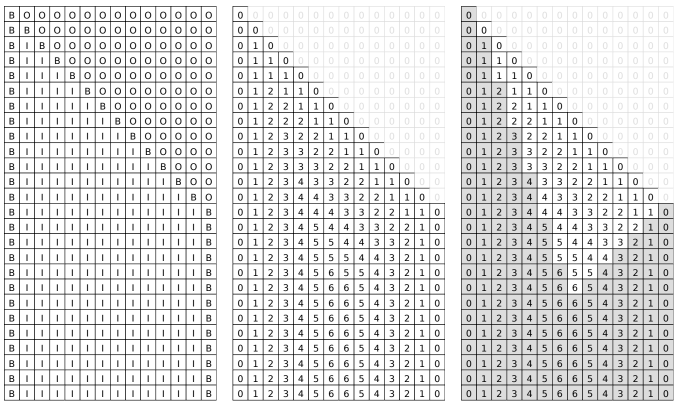
2.2 Initial Region and Its Middle

The *initial region field* R^0 is defined using three states O ("*outside*"), B ("*border*") and I ("*inside*"). O is the quiescent state of the field. A cell in state O will turn into B as soon as one of its neighbors is in state B. The "border" state is used to mark the border of the region currently discovered, which at this step must finally correspond the whole physical space. A cell in state B which does not coincide with a physical border of the space updates its state to I. With the help of a given static boolean field $Border^0(x)$ that states for each x if it is a physical border or not, the field R^0 is formally defined by:

$$R^0_t(x) = \begin{cases} B & \text{if } R^0_{t-1}(x) = O \wedge \exists y \in N(x); R^0_{t-1}(y) = B \\ I & \text{if } R^0_{t-1}(x) = B \wedge \neg Border^0(x) \\ R^0_{t-1}(x) & \text{otherwise.} \end{cases} \quad (1)$$

From any initial condition of the form $BO \dots O$, the evolution of R^0 produces a space-time diagram like the one depicted in Fig. 1(a).

Now that we have the field that, after some time, represents the whole initial region, we want to determine its middle point. To do so, we use the fact that the middle of a region is the inner point farthest from both "borders". So, we build the *distance field* D^0, which associates to each cell its distance to latest observed nearest "borders" of the region. As the middle is necessarily an inner cell, the value of D^0 of any cell that is not inside is defined as 0. One can note that this is consistent with the fact that a "border" is obviously at distance 0



(a) Region field (b) Distance field (c) Fixity field

Fig. 1. The three fields describing the initial region and its middle

from a "border". For an inner cell its distance to the latest observed nearest "border" is obviously 1 plus the smallest distance to the latest observed nearest "border" of its neighbors. This is formally defined by:

$$D^0_t(x) = \begin{cases} 0 & \text{if } R^0_t(x) \neq I \\ \min_{y \in N(x)} 1 + D^0_{t-1}(y) & \text{otherwise} \end{cases} \tag{2}$$

This rule has been extensively studied as a generic building block in cellular automata that solve different geometric problems (see [7,8,9,10]). It is sufficient to detect non-strict local maxima from the distance field to obtain the middle cell(s) of the region. This is illustrated in Fig. 1(b) which shows how region and distance fields evolve.

A final piece is required with respect to synchronization: we need to know when values provided by the region and distance fields are finals. Hence, we define a boolean field F^0 which associates to each cell a boolean indicating if its respective region and distance values are definitely correct. It is possible to determine the appropriate value by a simple case analysis. First, no field's value of an "outside" cell is considered correct (it has not been discovered yet). "Border" cell field's values are correct only if they coincide with a physical border. For "inside" cells, we know by construction that they are really inside so this value is always correct, but we still need to ensure that the distance value is also correct. To determine the distance value correctness, we use the fact that the region only grows, which implies that distance values only increase. And, from the point of view of a cell x, this means that once a neighbor is both correct and minimaly-valued in its neighborhood, it will remain such forever. This ensures that the distance value of x will not evolve anymore since it correspond to this fixed value + 1 as specified in Eq. (2). Altogether, this leads to the following formal definition, whose evolution is illustrated in Fig. 1(c).

$$F_t^0(x) = \bigvee \begin{cases} R_t^0(x) = B \wedge Border^0(x) \\ R_t^0(x) = I \wedge \exists y \in N(x);\ D_t^0(x) = 1 + D_{t-1}^0(y)\ \wedge\ F_{t-1}^0(y) \end{cases} \tag{3}$$

Remark 1. It is interesting to note that, although our description is very different from the classical one, we obtain very similar spatio-temporal patterns. Indeed, they appear because they are directly related to the locality of communications, and the induced delay. Suppose that at time t, the borders are respectively located in cells a and b. It is clear that the middle in between a and b can only be identified at time $t + \frac{d(a,b)}{2}$. This is an invariant due to the locality of system, and if the speed of a is 0 and the speed of b is 1, simple calculations indicate that the middle moves at speed $\frac{1}{3}$. This is the reason why one side of the triangle has slope 3 (classical signal speed $\frac{1}{3}$).

2.3 Subsequent Regions and Divisions

Now that the initial region is identified and that enough information has been built to divide it, let us proceed by adding new fields to obtain the division and provide sufficient information to recurse.

First, let us clearly identify what we want to build. From Fig. 1, it should be clear that we are going to build one region starting from the left and another starting from the right. However, we shall prevent ourselves to trust our eyes too much, but try to describe what we want by definition.

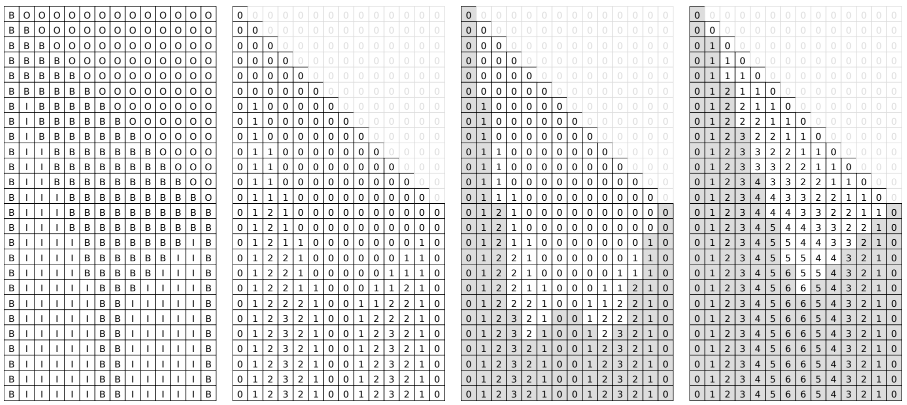
Let us come back on what we have done for the initial region and do nearly the same here. Given the predicate $Border^0(x)$, what we built is a region field whose values are, after some time, $R^0(x) = B$ for x's that are physical borders, and $R^0(x) = I$ for x's that are not physical borders and so inner cells.

In the region field R^1, we want to obtain as borders all the "borders" obtained at the previous level and new ones corresponding to the middle(s) cell(s) finally obtained at the previous level. Thus, we consider as borders of the two regions all correct x's such that $R^0(x) = B$, and all x's that correspond to correct non-strict local maxima of D^0. We also want to have $R^1(x) = I$ everywhere x is correct and is neither a "border" nor a maximum among its neighbors in D^0. This naturally leads to the following recursive formal definition of the two predicates *Border* and *Inside* for any level $l > 0$:

$$Border_t^{\ell+1}(x) = \bigvee \begin{cases} R_t^\ell(x) = B \wedge F_t^\ell(x) \\ \forall y \in \{x\} \cup N(x);\ D_{t-1}^\ell(x) \geq D_{t-1}^\ell(y)\ \wedge\ F_{t-1}^\ell(y) \end{cases} \tag{4}$$

$$Inside_t^{\ell+1}(x) = F_t^\ell(x) \wedge R_t^\ell(x) \neq B \wedge \exists y \in N(x);\ D_{t-1}^\ell(y) > D_t^\ell(x) \tag{5}$$

Given these two boolean fields, we can apply the same reasoning as before and, obtain nearly the same evolution rule as the initial region. We only need to change the use of $\neg Border^0(x)$ in Eq. 1 into $Inside_t^1(x)$ and the use of $Border^0(x)$ in Eq. 3 into $Border_t^1(x)$. Thus, we obtain the three additional fields



(a) Region field (b) Distance field (c) Fixity field (d) Level 0

Fig. 2. The three fields describing the level 1

describing the first level of division, and iterating this construction, for any level $l > 0$ we obtain the following recursive definition:

$$R_t^\ell(x) = \begin{cases} B & \text{if } R_{t-1}^\ell(x) = O \wedge \exists y \in N(x); R_{t-1}^\ell(y) = B \\ I & \text{if } R_{t-1}^\ell(x) = B \wedge Inside_t^\ell(x) \\ R_{t-1}^\ell(x) & \text{otherwise.} \end{cases} \tag{6}$$

$$D_t^\ell(x) = \begin{cases} 0 & \text{if } R_t^\ell(x) \neq I \\ \min_{y \in N(x)} 1 + D_{t-1}^\ell(y) & \text{otherwise.} \end{cases} \tag{7}$$

$$F_t^\ell(x) = \bigvee \begin{cases} R_t^\ell(x) = B \wedge Border_t^\ell(x) \\ R_t^\ell(x) = I \wedge \exists y \in N(x);\ D_t^\ell(x) = 1 + D_{t-1}^\ell(y)\ \wedge\ F_{t-1}^\ell(y) \end{cases} \tag{8}$$

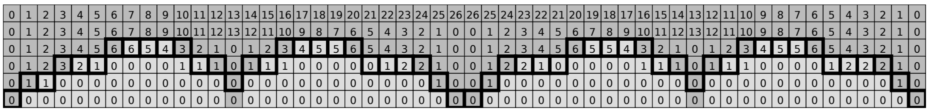
Fig. 2 shows how the three fields evolve at level 1 of the algorithm. In Fig. 2(a) we have one region that grows from the left and starts at the initial time, and another one that grows from the right and starts at time $n-1$ (n is the number of cells). The distance field D^1 evolves inside each region described by R^1.

One can observe that while in D^0 the non-strict local maxima spanned two cells, then in D^1 there is two non-strict local maxima that both span only one cell. This depends on whether the region's length is odd or even (Fig. 2(b)).

2.4 Reduction to a Finite Number of States

Now we face two problems. The first one is that distance fields are defined over integers and the other one that we obtained an unbounded number of fields. Altough a complete and detailed explanation of the reducability in finite state is out of the scope of this paper, let us sketch the more important steps.

The first problem can be solved using a special property. If an integer field is Lipschitz-continuous, i.e. the difference of values between two neighbors is



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	2	3	4	5	6	6	5	4	3	2	1	0	1	2	3	4	5	5	6	5	4	3	2	1	0	0	1	2	3	4	5	6	5	5	4	3	2	1	0	1	2	3	4	5	5	6	5	4	3	2	1	0
0	1	2	3	2	1	0	0	0	0	1	1	1	0	1	1	1	0	0	0	0	0	1	2	2	1	0	0	1	2	2	1	0	0	0	0	0	1	1	1	0	1	1	1	0	0	0	0	1	2	2	2	1	0
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 3. Stack (level 0 on top) of all field values computed at time 96. Line length is 54.

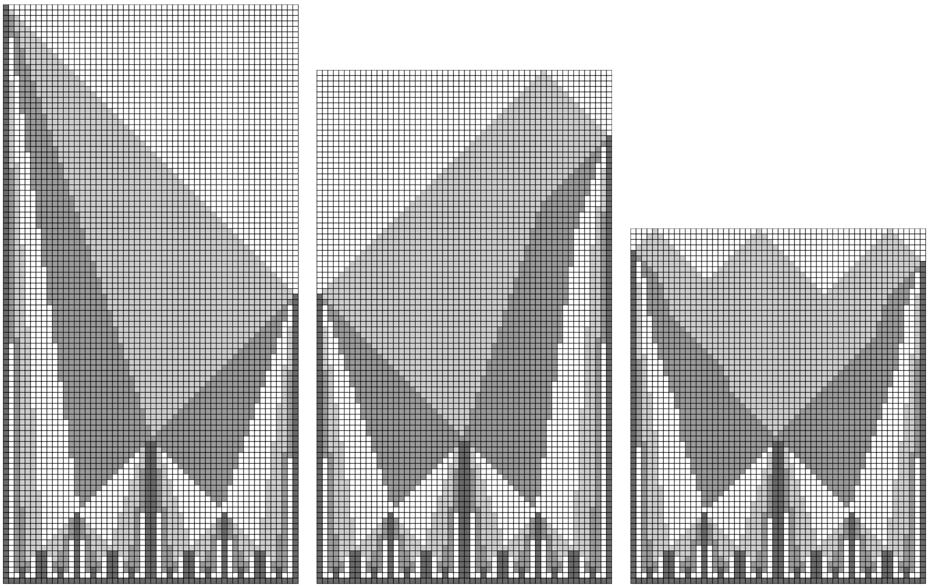


Fig. 4. Evolution of the complete system with different set of generals

bounded, and the information used in the system only depends on this difference, then it can be transformed into a finite-state field (refer to [8] for all the details). An application of this result is that when the difference is at most 1, then only 3 states are required. With definitions given in Eq. 2 and Eq. 7, it's easy to remark that all the distance fields D^ℓ can therefore be represented with only 3 states each.

To solve the second problem we remark that *in some sense* the recursive schema is "tail-recursive". Indeed, tail-recursiveness is about conserving only the information that are required but the subsequent recursive calls. From the point of view of a cell x, if its field values at given level ℓ are correct, this means that they do not evolve anymore. If furthermore its field values at $\ell + 1$ are also correct and so are the values of its neighbors, then its values at level ℓ are no more useful and can be discarded. This is observable in Fig. 3 where fields values are represented for all cells at a given time. Values (x, ℓ) in darker gray are correct ($F_t^\ell(x)$ is true), and if the whole neighborhood at the next level is also gray, then (x, ℓ) can be "forgotten". By discarding all these gray values (and a little bit more with a much finer analysis), we obtain for each cell a *lowest useful level* represented by a bold surround in the figure. In fact, these are the only necessary values that need to be stored, along with their associated lowest level number (which can be represented with only three state thanks to the Lipschitz-continuous argument).

Altogether, this shows that field values are uniformly bounded, and that only a finite number of fields is required. This implies that we finally describe the behavior of a cellular automaton.

3 Conclusion

Without any modification, the system described in this paper is much more general than one can think. Indeed, in our whole description we never use the property that there is only one general on the left. Since nothing such has been assumed in the design of the solution, we can naturally expect that it is agnostic to such particularities, and this is exactly the case as one can observe in Fig. 4. We also never assumed that the wake-up of the cells happens one after the other from the general, so that removing the corresponding sub-system, one obtains a solution for arbitrary initial desynchronized configurations.

It seems also possible to compose the same fields in slightly different ways to obtain different kind of solutions or to extend this solution to higher dimensions. We can also expect that a proof of correctness of the solution for all sizes and all initial desynchronized configurations seems to be much easier than for classical solutions, each field is simple and almost correct by construction, and so is their composition.

Finally, what we propose is a semantic-oriented framework which let anyone describe cellular machines in very natural modular way, which is a very common point of view in classical algorithmic and computer programming. Each of these composable fields, expressed in their original form, can be reused in many other different contexts. We think that this method is really helpful and so powerful that many things previously considered as hard to manage and to understand now appear to be clearer and easier.

References

1. Culik, K.: Variations of the firing squad problem and applications. Information Processing Letters 30, 153–157 (1989)
2. Goto, E.: A minimum time solution of the firing squad synchronization problem. Courses Notes for Applied Mathematics, vol. 298. Harvard University (1962)
3. Grasselli, A.: Synchronization of cellular arrays: The firing squad problem in two dimensions. Information and Control 28, 113–124 (1975)
4. Herman, G.T., Liu, W., Rowland, S., Walker, A.: Synchronization of growing cellular automata. Information and Control 25, 103–122 (1974)
5. Jiang, T.: The synchronization of nonuniform networks of finite automata. Information and Control 97, 234–261 (1992)
6. Kobayashi, K.: The firing squad synchronisation problem for two-dimensional arrays. Information and Control 34, 177–197 (1977)

7. Maignan, L., Gruau, F.: Integer gradient for cellular automata: Principle and examples. In: Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, pp. 321–325. IEEE Computer Society, Washington, DC (2008)
8. Maignan, L., Gruau, F.: A 1D cellular automaton that moves particles until regular spatial placement. Parallel Processing Letters 19(2), 315–331 (2009)
9. Maignan, L., Gruau, F.: Convex Hulls on Cellular Automata. In: Bandini, S., Manzoni, S., Umeo, H., Vizzari, G. (eds.) ACRI 2010. LNCS, vol. 6350, pp. 69–78. Springer, Heidelberg (2010)
10. Maignan, L., Gruau, F.: Gabriel graphs in arbitrary metric space and their cellular automaton for many grids. ACM Trans. Auton. Adapt. Syst. 6, 12:1–12:14 (June 2011)
11. Mazoyer, J.: A six states minimal time solution to the firing squad synchronization problem. Theoretical Computer Science 50, 183–238 (1987)
12. Minsky, M.: Computation: Finite and Infinite Machines. Prentice-Hall (1967)
13. Moore, E.E.: Sequential machines, Selected papers. Addison-Wesley (1964)
14. Noguchi, K.: Simple 8-state minimal time solution to the firing squad synchronization problem. TCS 314, 303–334 (2004)
15. Rosenstiehl, P., Fiskel, J.R., Holliger, A.: Intelligent Graphs: Networks of Finite Automata capable of Solving Graph Problems. In: Read, R.C. (ed.) Graph Theory and Computing. Academic Press (1972)
16. Schmid, H., Worsch, T.: The firing squad synchronization problem with many generals for one-dimensional CA. In: Lévy, J.-J., Mayr, E.W., Mitchell, J.C. (eds.) IFIP TCS, pp. 111–124. Kluwer (2004)
17. Shinahr, I.: Two and three dimensional firing squad synchronization problems. Information and Control 24, 163–180 (1974)
18. Szwerinski, H.: Time-optimal solution of the firing-squad synchronization problem for n-dimensional rectangles with the general at an arbitrary position. Theoretical Computer Science 19, 305–320 (1982)
19. Varshavsky, V.I., Marakhovsky, V.B., Peshansky, V.A.: Synchronization of interacting automata. Mathematical System Theory 4(3), 212–230 (1969)
20. Yunès, J.-B.: An intrinsically non minimal-time Minsky-like 6-states solution to the firing squad synchronization problem. RAIRO ITA/TIA 42(1), 55–66 (2008)
21. Yunès, J.-B.: Known CA synchronizers made insensitive to the initial state of the initiator. JCA 4(2), 147–158 (2009)