Nat Comput (2018) 17:441–453
https://doi.org/10.1007/s11047-017-9625-2

A design of generalized minimum-state-change FSSP algorithms and their implementations

Hiroshi Umeo[1] · **Keisuke Imai**[1] · **Akihiro Sousa**[1]

Published online: 22 June 2017

Abstract The firing squad synchronization problem (FSSP) on cellular automata has been studied extensively for more than 50 years, and a rich variety of FSSP algorithms has been proposed. Here we consider the FSSP from a view point of state-change complexity that models the energy consumption of SRAM-type storage with which cellular automata might be built. In the present paper, we propose minimum-state-change generalized FSSP (GFSSP) algorithms for synchronizing any one-dimensional (1D) cellular automaton, where the initial synchronization operation is started by any cell in the array. First, we construct two minimum-time, minimum-state-change GFSSP implementations on finite state automata: one is based on Goto's algorithm, known as the first minimum-time FSSP algorithm that was reconstructed again recently in Umeo et al. (A new reconstruction and the first implementation of Goto's FSSP algorithm, 2017), and the other is based on Gerken's (Diplomarbeit, Institut für Theoretische Informatik, Technische Universität Braunschweig, pp 1–50, 1987) one. These implementations are optimal not only in time but also in the state-change complexity. The implementations of the minimum-time GFSSP algorithms are the first ones having the minimum-state-change complexity. In addition, we also present a six-state 145-rule non-minimum-time, minimum-state-change GFSSP implementation. The implemented GFSSP algorithm is the smallest one, known at present, in number of states of the finite state automaton.

✉ Hiroshi Umeo
umeo@cyt.osakac.ac.jp; umeo@osakac.ac.jp

[1] University of Osaka Electro-Communication, Neyagawa-shi, Hastu-cho, 18-8, Osaka 572-8530, Japan

Keywords Cellular automaton · Firing squad synchronization problem · FSSP · State-change complexity

1 Introduction

We study a synchronization problem that gives a finite-state protocol for synchronizing large-scale cellular automata. The synchronization in cellular automata has been known as the firing squad synchronization problem (FSSP, for short) since its development, in which it was originally proposed by J. Myhill in Moore (1964) to synchronize some/all parts of self-reproducing cellular automata. The problem has been studied extensively for more than 50 years, and a rich variety of synchronization algorithms has been proposed. See Balzer (1967) - Yunès (2008). Here we consider the FSSP from a view point of state-change complexity that models the energy consumption of SRAM-type storage with which cellular automata might be built.

In the present paper, we propose three minimum-state-change generalized FSSP (GFSSP, for short) algorithms and their implementations for synchronizing any one-dimensional (1D) cellular automaton, where the initial synchronization operation is started not only from one end of the array but also from any cell in the array. Two minimum-time and one non-minimum-time GFSSP algorithm are proposed. Specifically, we attempt to answer the following questions:

- Can we synchronize 1D arrays in a generalized setting with a general at any position in the array, while keeping the minimum-state-change complexity?
  - Yes. The constructed GFSSP algorithm can synchronize any 1D cellular automaton of length n in $n-2+\max(k, n-k+1)$ minimum-time and it

has $\Theta(n \log n)$ minimum-state-change complexity, where the initial synchronization operation is started from any cell k $(1 \leq k \leq n)$ in the array.
- We construct two minimum-time, minimum-state-change GFSSP algorithms, one is based on Goto's algorithm with the general at one end, known as the first minimum-time FSSP algorithm that was reconstructed again recently in Umeo et al. (2017), and the other is based on Gerken's one (1987) also with the general at one end.
- These two algorithms are optimal not only in time but also in state-change complexity.
- The implemented minimum-time GFSSP algorithms are the first ones having the minimum-state-change complexity.

- How many states are required for its implementation on a finite state automaton?
  - The Goto-based GFSSP algorithm is realized on a cellular automaton with 434 internal states and 13328 state-transition rules.
  - The Gerken-based one is implemented on a cellular automaton with 213 internal states and 4077 state-transition rules.
- How can we reduce the number of states of an implementation of the GFSSP algorithms, while maintaining the minimum-state-change complexity?
  - We present a six-state 145-rule non-minimum-time, minimum-state-change GFSSP implementation.
  - The implemented GFSSP algorithm is the smallest one, known at present, in number of states of the finite state automaton.

In Sect. 2 we give a description of the 1D FSSP and review some basic results on FSSP and GFSSP algorithms. Section 3 gives new implementations and generalizations to the GFSSP algorithm having minimum-time, minimum-state-change complexities. Section 4 presents a six-state 145-rule implementation of the non-minimum-time, minimum-state-change GFSSP algorithm. Section 5 gives a summary of the paper.

2 Firing squad synchronization problem

2.1 Definition of FSSP

The FSSP is formalized in terms of a model of cellular automata. Consider a 1D array of finite state automata. All cells (except the end cells) are identical finite state automata. The array operates in lock-step mode such that the next state of each cell (except the end cells) is determined by both its own present state and the present states of its right and left nearest neighbors. All cells (*soldiers*), except one *general* cell, are initially in the *quiescent* state at time $t = 0$ and have the property that the next state of a quiescent cell having quiescent neighbors is the quiescent state. At time $t = 0$ the *general* cell is in the *fire-when-ready* state, which is an initiation signal to the array.

The FSSP is stated as follows: given an array of n identical cellular automata, including a *general* on the left end which is activated at time $t = 0$, we want to give the description (state set and next-state function) of the automata so that *at some future time* all of the cells will *simultaneously* and *for the first time* enter a special *firing* state. The initial general is on the left end of the array.

Figure 1 shows a finite 1D cellular array consisting of n cells, denoted by C_i, where $1 \leq i \leq n$. The set of states and the next-state transition function must be independent of n. Without loss of generality, we assume $n \geq 2$. The tricky part of the problem is that the same kind of soldiers having a fixed number of states must be synchronized, regardless of the length n of the array.

A formal definition of the FSSP is as follows: a cellular automaton $\mathcal{M}$ is a pair $\mathcal{M} = (\mathcal{Q}, \delta)$, where

1. $\mathcal{Q}$ is a finite set of states with three distinguished states G, Q, and F. G is an initial general state, Q is a quiescent state, and F is a firing state, respectively.
2. δ is a next-state function such that $\delta : \mathcal{Q} \cup \{*\} \times \mathcal{Q} \times \mathcal{Q} \cup \{*\} \rightarrow \mathcal{Q}$. The state $* \notin \mathcal{Q}$ is a pseudo state of the border of the array.
3. The quiescent state Q must satisfy the following conditions: $\delta(\mathtt{Q}, \mathtt{Q}, \mathtt{Q}) = \delta(*, \mathtt{Q}, \mathtt{Q}) = \delta(\mathtt{Q}, \mathtt{Q}, *) = \mathtt{Q}$.

A cellular automaton $\mathcal{M}_n$ of length n, consisting of n copies of $\mathcal{M}$, is a 1D array whose positions are numbered from 1 to n. We denote a state of C_i at time (step) t by $\mathtt{S}_i^t$, where $t \geq 0$ and $1 \leq i \leq n$. A *configuration* of $\mathcal{M}_n$ at time t is a function $\mathcal{C}^t : [1, n] \rightarrow \mathcal{Q}$ and denoted as $\mathtt{S}_1^t \mathtt{S}_2^t \ldots \mathtt{S}_n^t$. A *computation* of $\mathcal{M}_n$ is a sequence of configurations of $\mathcal{M}_n$, $\mathcal{C}^0$, $\mathcal{C}^1$, $\mathcal{C}^2$, ..., $\mathcal{C}^t$, ..., where $\mathcal{C}^0$ is a given initial configuration. The configuration at time $t+1$, $\mathcal{C}^{t+1}$, is computed by synchronous applications of the next-state function δ to each cell of $\mathcal{M}_n$ in $\mathcal{C}^t$ such that:

$\mathtt{S}_1^{t+1} = \delta(*, \mathtt{S}_1^t, \mathtt{S}_2^t)$, $\mathtt{S}_i^{t+1} = \delta(\mathtt{S}_{i-1}^t, \mathtt{S}_i^t, \mathtt{S}_{i+1}^t)$, and $\mathtt{S}_n^{t+1} = \delta(\mathtt{S}_{n-1}^t, \mathtt{S}_n^t, *)$.

A *synchronized configuration* of $\mathcal{M}_n$ at time t is a configuration $\mathcal{C}^t$, $\mathtt{S}_i^t = \mathtt{F}$, for any $1 \leq i \leq n$.

The FSSP is to obtain an $\mathcal{M}$ such that, for all $n \geq 2$:

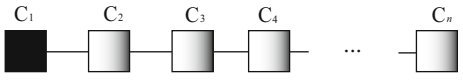


Fig. 1 A one-dimensional (1D) cellular automaton

1. A synchronized configuration at time $t = T(n)$, $\mathcal{C}^{T(n)} = \overbrace{\mathtt{F}, \ldots, \mathtt{F}}^{n}$ can be computed from an initial configuration $\mathcal{C}^0 = \mathtt{G}\overbrace{\mathtt{Q}, \ldots, \mathtt{Q}}^{n-1}$.
2. For every t and i such that $1 \leq t \leq T(n) - 1$, $1 \leq i \leq n, \mathrm{S}_i^t \neq \mathtt{F}$.

No cells fire before time $t = T(n)$. We say that $\mathcal{M}_n$ is synchronized at time $t = T(n)$ and the function $T(n)$ is a time complexity for the FSSP.

The GFSSP is to obtain an $\mathcal{M}$ such that, for every n and k such that $n \geq 2$, $1 \leq k \leq n$:

1. A synchronized configuration at time $t = T(k, n)$, $\mathcal{C}^{T(k,n)} = \overbrace{\mathtt{F}, \ldots, \mathtt{F}}^{n}$ can be computed from an initial configuration $\mathcal{C}^0 = \overbrace{\mathtt{Q}, \ldots, \mathtt{Q}}^{k-1}\mathtt{G}\overbrace{\mathtt{Q}, \ldots, \mathtt{Q}}^{n-k}$.
2. For every t and i such that $1 \leq t \leq T(k, n) - 1$, $1 \leq i \leq n, \mathrm{S}_i^t \neq \mathtt{F}$.

No cells fire before time $t = T(k, n)$. We say that $\mathcal{M}_n$ is synchronized at time $t = T(k, n)$ and the function $T(k, n)$ is a time complexity for the GFSSP.

2.2 Some related results on FSSP and GFSSP

Here we summarize some basic results on FSSP algorithms.

- *Minimum-time FSSP algorithms with a general at one end*

The FSSP problem was first solved by McCarthy and Minsky (1967) who presented a $3n$-step algorithm for n cells. In 1962, the first minimum-time, i.e. $(2n - 2)$ step, synchronization algorithm was presented by Goto (1962), with each cell having several thousands of states. Waksman (1966) presented a 16-state minimum-time synchronization algorithm. Afterward, Balzer (1967) and Gerken (1987) developed an eight-state algorithm and a seven-state synchronization algorithm, respectively, thus decreasing the number of states required for the synchronization. Mazoyer (1987) developed a six-state synchronization algorithm which, at present, is the algorithm having the fewest states.

Theorem 1 *There exists a cellular automaton that can synchronize any 1D array of length n in minimum $2n - 2$ steps, where the general is located at a left (or right) end of the array.*

- *Minimum-time GFSSP algorithms*

The GFSSP has also been studied, where an initial general can be located at any position in the array. The same kind of soldiers having a fixed number of states must be synchronized, regardless of the position k of the general and the length n of the array. Moore and Langdon (1968) first studied the problem and presented a 17-state minimum-time GFSSP algorithm, i.e. operating in $n - 2 + \max(k, n - k + 1)$ steps for n cells with the general on the kth cell from the left end of the array. See Umeo et al. (2010) for a survey on GFSSP algorithms and their implementations. Concerning the GFSSP, it has been shown impossible to synchronize any array of length n in less than $n - 2 + \max(k, n - k + 1)$ steps in Moore and Langdon (1968), where the general is located on C_k, $1 \leq k \leq n$. Umeo et al. (2010) gave an 8-state minimum-time GFSSP implementation.

Theorem 2 (Lower bounds) *The minimum-time in which the GFSSP could occur is no earlier than $n - 2 + \max(k, n - k + 1)$ steps, where the general is located on the kth cell from the left end.*

Theorem 3 *There exists an 8-state cellular automaton that can synchronize any 1D array of length n in minimum $n - 2 + \max(k, n - k + 1)$ steps, where the general is located on the kth cell from the left end.*

3 A class of minimum-time, minimum-state-change GFSSP algorithms

In this section we develop a general methodology for designing a minimum-time GFSSP algorithm based on freezing-thawing technique. We can construct a minimum-time GFSSP algorithm from any minimum-time FSSP algorithm with a general at one end. The *freezing-thawing technique* developed in Umeo (2004) enables us to have an FSSP algorithm with an arbitrary synchronization delay for 1D arrays.

3.1 Delaying synchronization steps

We introduce a *freezing-thawing technique* that yields a delayed synchronization for 1D arrays. The technique was developed by Umeo (2004) for designing several fault-tolerant FSSP algorithms for 1D arrays. The freezing-thawing technique can be employed efficiently for the design of minimum-time, minimum-state-change GFSSP algorithms in Sects. 3.4 and 3.5.

Theorem 4 *Let t_0, t_1, t_2 and Δt be any integer such that $t_0 \geq 0$, $t_1 = t_0 + n - 1$, $t_1 \leq t_2$ and $\Delta t = t_2 - t_1$. We assume that a usual synchronization operation is started at time $t = t_0$ by generating a special signal which acts as a general at the left end of 1D array of length n. We also assume that the right end cell of the array receives another*

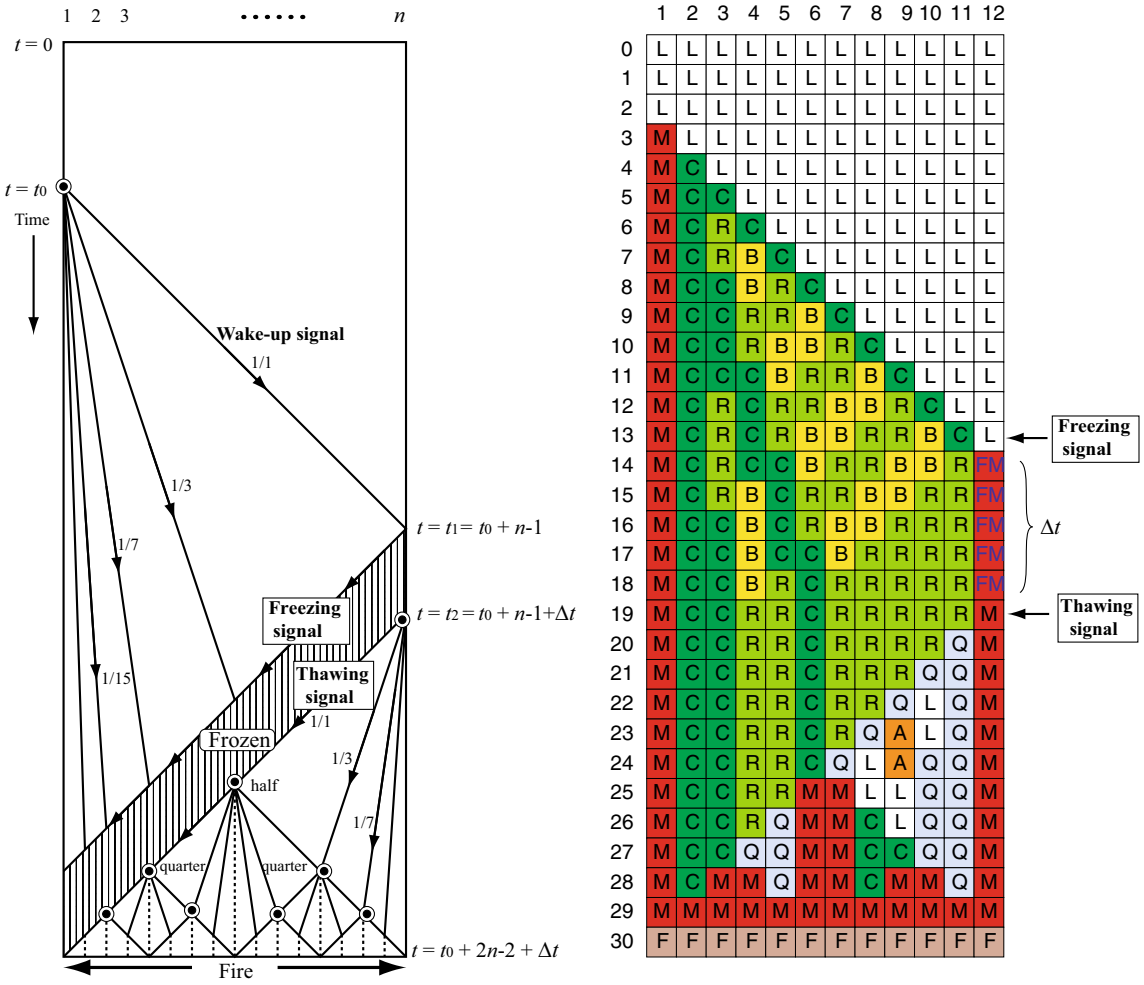


Fig. 2 Space-time diagram for delayed FSSP schema based on the *freezing-thawing* technique (*left*) and delayed synchronized configurations on $n = 12$ cells for $\Delta t = 5$ steps (*right*)

special signals from the outside at time $t_1 = t_0 + n - 1$ *and* $t_2 = t_1 + \Delta t$, *respectively. Then, there exists a 1D cellular automaton that can synchronize the array of length n at time* $t = t_0 + 2n - 2 + \Delta t$.

The array operates as follows:

1. Start a minimum-time FSSP algorithm at time $t = t_0$ at the left end of the array. A 1/1 speed, i.e., 1 cell per 1 step, signal is propagated towards the right direction to wake-up cells in quiescent state. We refer the signal as *wake-up signal*. A *freezing signal* is given from outside at time $t_1 = t_0 + n - 1$ at the right end of the array. The signal is propagated in the left direction at its maximum speed, that is, 1 cell per 1 step, and freezes the configuration progressively. Any cell that receives the freezing signal from its right neighbor has to stop its state-change and transmit the freezing signal to its left neighbor. The frozen cell keeps its state as long as no thawing signal will arrive.
2. A special signal supplied from the outside at the right end at time $t_2 = t_1 + \Delta t$ is used as a *thawing signal* that thaws the frozen configuration progressively. The thawing signal forces the frozen cell to resume its state-change procedures immediately. See Fig. 2 (left). The signal is also transmitted toward the left end at speed 1/1.

The readers can see how those three special signals work. We can freeze the entire configuration during Δt steps and delay the synchronization on the array for Δt steps. Figure 2 (right) shows some snapshots of delayed (for $\Delta t = 5$) configurations for minimum-time synchronization algorithm on 12 cells. In this example, note that the wake-up signal is supplied with the array at time $t_0 = 3$. We refer the scheme as *freezing-thawing technique*.

3.2 Designing minimum-time GFSSP algorithms

Consider a cellular array $C_1, C_2, \ldots, C_n$ of length n with an initial general on C_k, where $1 \leq k \leq n$. At time $t = 0$ the general sends a unit speed (1 cell/1 step) signal to both

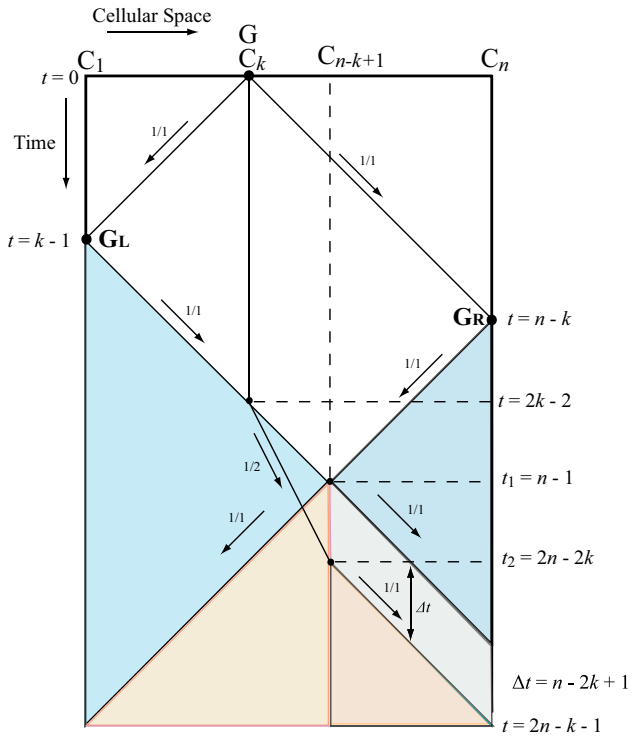


Fig. 3 Space-time diagram for the construction of minimum-time GFSSP algorithm

ends. The cell C_k keeps its state to indicate its initial position on the array. The signal reaches at the left and right ends at time $t = k - 1$ and $t = n - k$, respectively, and generates a new general, denoted as G_L and G_R at each end. In Fig. 3, we illustrate a space-time diagram for the GFSSP construction. Each general, G_L and G_R, starts minimum-time synchronization operations for the cellular space where the general is at its end by sending out a wake-up signal. At time $t = n - 1$ the two signals collide with each other on the cell C_{n-k+1} and the cellular space is divided into two parts by the collision.

First, we consider the case where the initial general is in the left half of the given cellular space, i.e. $k \leq n - k + 1$. The wake-up signal generated by G_L reaches C_k at time $t = 2k - 2$, then collides with the wake-up signal generated by G_R. The larger part (left one in this case) is synchronized in the usual way, while the small one is synchronized with time delay $\Delta t = n - 2k + 1$. The wake-up signal for the larger part splits into two signals on C_k, one is an original wake-up signal and the other is a new slow signal which follows the wake-up signal at 1/2-speed. Note that the wake-up signal for the smaller part (right one in this case) never reaches C_k. As for the synchronization for the smaller part, a freezing-signal is generated at time $t_1 = n - 1$ on C_{n-k+1} and the configuration in the smaller part is frozen by the propagation of the 1/1-speed right-going freezing signal. At time $t_2 = 2n - 2k$, the split slow signal reaches C_{n-k+1} and there a thawing signal is generated. The thawing signal thaws the frozen configuration progressively. Theorem 4 shows that the smaller part of length k is synchronized at time $t = 2n - k - 1$. The larger part is also synchronized at time $t = 2n - k - 1$. Thus, the whole space can be synchronized at time $t = 2n - k - 1 = n - 2 + \max(k, n - k + 1)$.

Similar discussions can be made in the case where the initial general is in the right half of the cellular space. It is seen that any minimum-time FSSP algorithm with a general at one end can be embedded as a sub-algorithm for the synchronization of divided parts. A similar technique was used for solving an FSSP with many generals in Schmid and Worsch (2004). Thus, we have:

Theorem 5 *The schema given above can realize a minimum-time GFSSP algorithm by implementing two minimum-time FSSP algorithms with a general at one end.*

3.3 State-change complexity

Vollmar (1982) introduced *state-change complexity* in order to measure the efficiency of cellular automata, motivated by energy consumption in certain SRAM-type memory systems. The state-change complexity is defined as the sum of *proper* state changes of the cellular space during the computations. A formal definition is as follows: consider an FSSP (GFSSP) algorithm operating on n cells. Let $T(n)$ (resp., $T(k, n)$) be synchronization steps of the FSSP (GFSSP) algorithm. We define a matrix C of size $T(n) \times n$ ($T(n)$ rows, n columns) [resp., $T(k, n) \times n$ ($T(k, n)$ rows, n columns)] over $\{0, 1\}$, where each element $c_{i,j}$ on ith row, jth column of the matrix C is define:

$$c_{i,j} = \begin{cases} 1 & \mathrm{S}_i^j \neq \mathrm{S}_i^{j-1} \\ 0 & otherwise. \end{cases} \tag{1}$$

The state-change complexity $SC(n)$(resp., $SC_g(n)$) of the FSSP (GFSSP) algorithm is the sum of 1's elements in C defined as:

$$SC(n) = \sum_{j=1}^{T(n)} \sum_{i=1}^{n} c_{i,j}, \tag{2}$$

$$SC_g(n) = 1/n \sum_{k=1}^{n} \sum_{j=1}^{T(k,n)} \sum_{i=1}^{n} c_{i,j}. \tag{3}$$

Vollmar (1982) showed that $\Omega(n \log n)$ state-change is required for synchronizing n cells in $(2n - 2)$ steps.

Theorem 6 $\Omega(n \log n)$ *state-change is necessary for synchronizing n cells in minimum-steps.*

Gerken (1987) presented a minimum-time, $\Theta(n \log n)$ minimum-state-change FSSP algorithm with a general at one end.

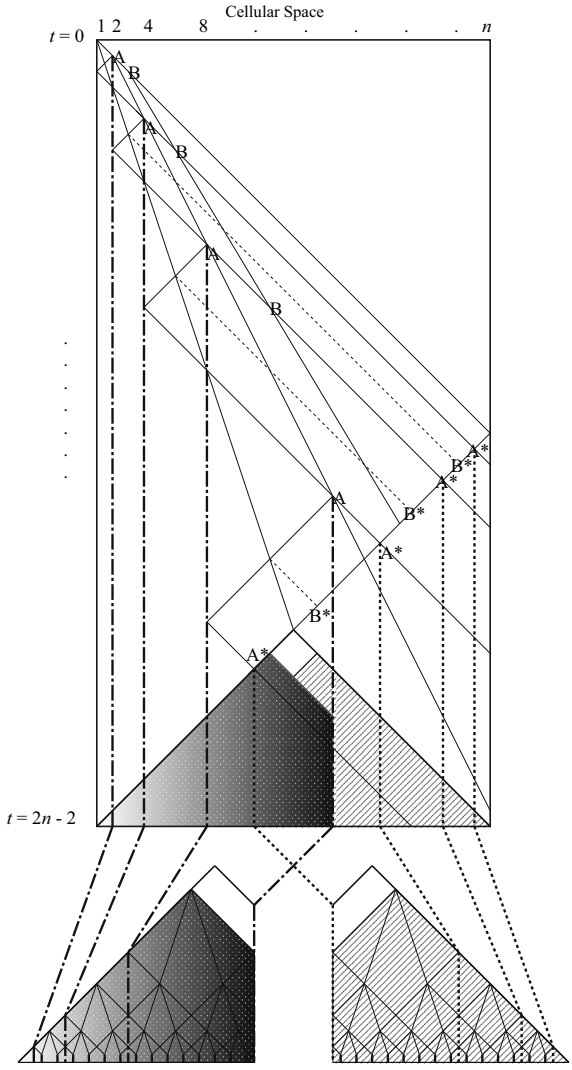
Theorem 7 *$\Theta(n \log n)$ state-change is sufficient for synchronizing n cells in $2n - 2$ steps.*

Goto's algorithm (Goto 1962) has been known as the first minimum-time FSSP algorithm, however the paper itself has been a mysterious one for a long time due to its hard accessibility. Umeo (1996) reconstructed the Goto's algorithm and it is noted in Umeo (2009) that the algorithm has $\Theta(n \log n)$ minimum-state-change complexity. Recently, Umeo et al. (2017) reconstructed the Goto's algorithm again and realized it on a cellular automaton having 165-state and 4378 transition rules.

Fig. 5 Snapshots of configurations for Goto-based minimum-time, ▶ minimum-state-change GFSSP algorithm developed on $n = 32$ cells with a general on C_7 (*top left*), C_{14} (*top right*), C_{20} (*bottom left*), and C_{28} (*bottom right*), respectively

Theorem 8 *The reconstructed Goto's algorithm has $\Theta(n \log n)$ state-change complexity for synchronizing n cells in $2n - 2$ steps.*

In order to get a minimum-time, minimum-state-change GFSSP algorithm, we embed the reconstructed Goto's algorithm and Gerken's one with a general at one end. The state-change complexity in the right and left parts in Fig. 3 is $O((n-k+1)\log(n-k+1))$ and $O(k \log k)$, respectively, thus the total state-change complexity of the constructed GFSSP algorithm is $O((n-k+1)\log(n-k+1)) + O(k \log k) \leq O(n \log n)$.



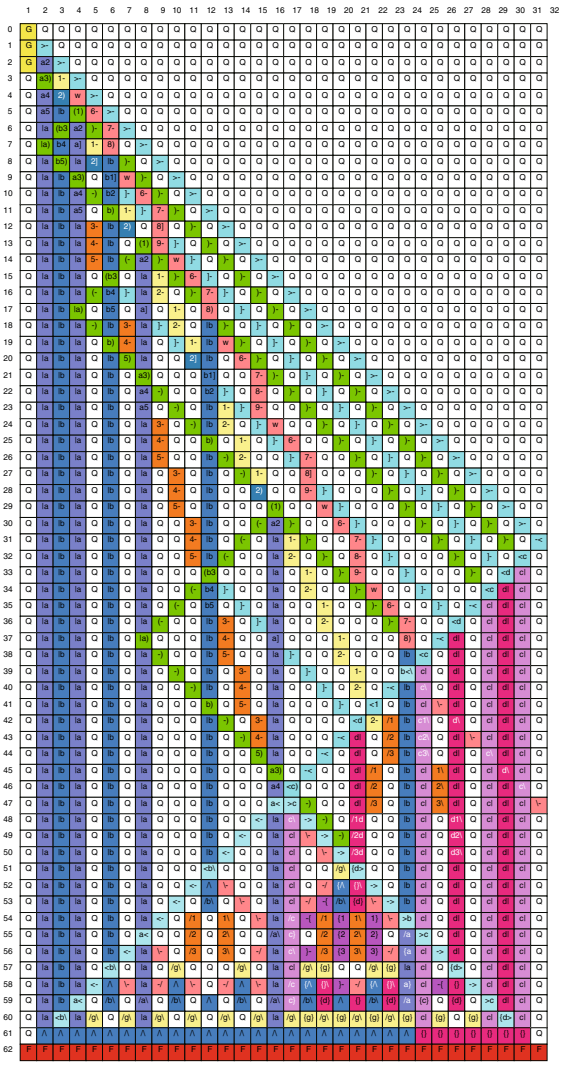
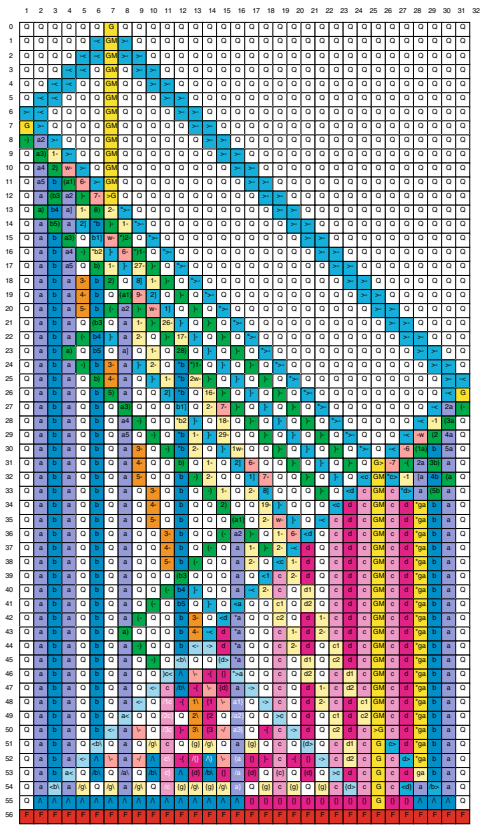
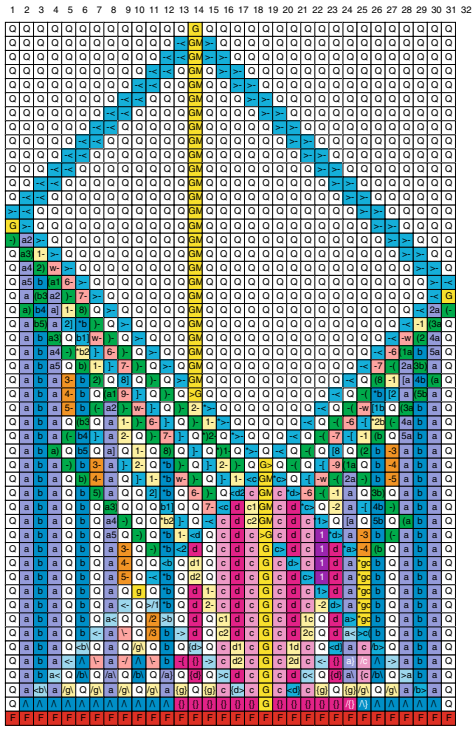
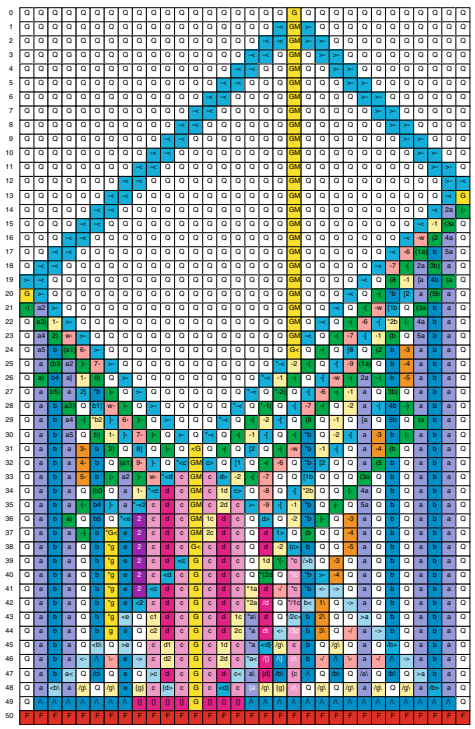
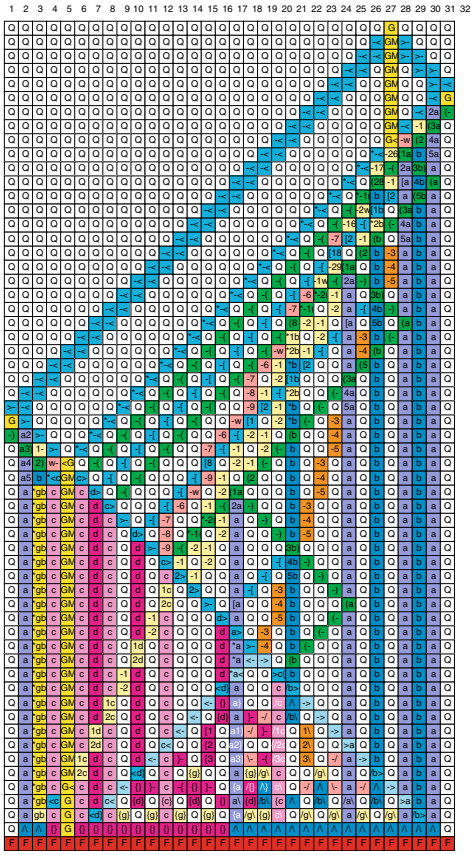


Fig. 4 An overview of the reconstructed Goto's FSSP algorithm (*left*) and its snapshots on 32 cells of the 165-state, 4378-transition-rule implementation in Umeo et al. (2017)









Thus, we have:

Theorem 9 *There exists a minimum-time, minimum-state-change GFSSP algorithm.*

3.4 An implementation of Goto-based minimum-time, minimum-state-change GFSSP algorithm

The algorithm that Umeo et al. (2017) reconstructed is a non-recursive algorithm consisting of a marking phase and a $3n$-step synchronization phase. In the first phase, by printing a special marker in the cellular space, the entire cellular space is divided into many smaller subspaces whose length increases exponentially with a common ratio of two, that is 2^j, for every integer $j \geq 1$. The exponential marking is made by counting cells from both left and right ends of a given cellular space. In the second phase, each subspace is synchronized by starting a well-known conventional $3n$-step synchronization algorithm from center point of each divided subspace. Figure 4 illustrates an overview of the reconstructed Goto's algorithm. It can be seen that the overall algorithm is a non-recursive one and it does not call itself. Based on the reconstructed Goto's algorithm in Umeo et al. (2017), we realize a minimum-time, minimum-state-change GFSSP algorithm on a cellular automaton with 434 internal states and 13328 state-transition rules. Figure 5 shows some snapshots for the constructed GFSSP algorithm on 32 cells with a general on C_7, C_{14}, C_{20}, and C_{28}, respectively.

Fig. 7 Snapshots of configurations of Gerken-based minimum-time, minimum-state-change GFSSP algorithm developed on $n = 34$ cells with a general on C_{10} (*top left*), C_{16} (*top right*), C_{22} (*bottom left*), and C_{29} (*bottom right*), respectively ▶

3.5 An implementation of Gerken-based minimum-time, minimum-state-change GFSSP algorithm

Gerken (1987) constructed a minimum-time 154-state 2371-rule FSSP algorithm and showed that the algorithm has a minimum-state-change complexity. The algorithm is the first one having the $\Theta(n \log n)$ minimum-state-change complexity for synchronizing n cells with a general at one end. Figure 6 gives a space-time diagram for the algorithm (left) and some snapshots for the synchronization processes on 37 cells. Based on the algorithm, we realize a minimum-time, minimum-state-change GFSSP algorithm on a cellular automaton with 213 states and 4077 rules. Figure 7 shows some snapshots for the constructed GFSSP algorithm on 34 cells with a general on C_{10}, C_{16}, C_{22}, and C_{29}, respectively.

We also show a comparison of state-change complexities among several GFSSP algorithms in Fig. 8. It includes the $O(n^2)$ state-change complexity in Moore and Langdon (1968), Szwerinski (1982), Umeo et al. (2010) and $O(n \log n)$ ones in Goto-based and Gerken-based GFSSP algorithms constructed in this paper.

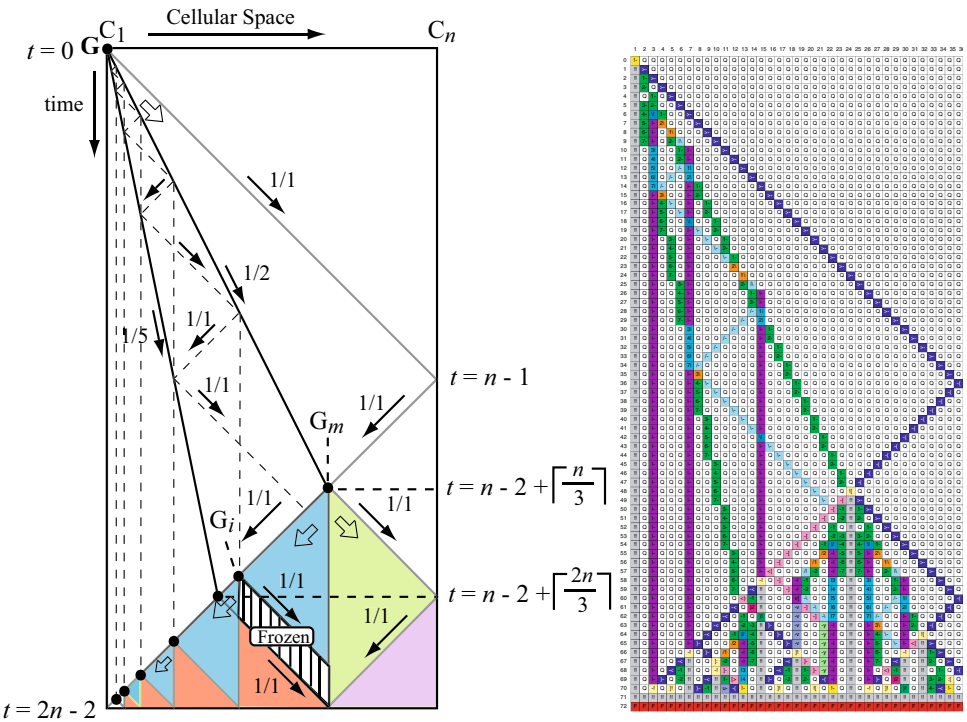
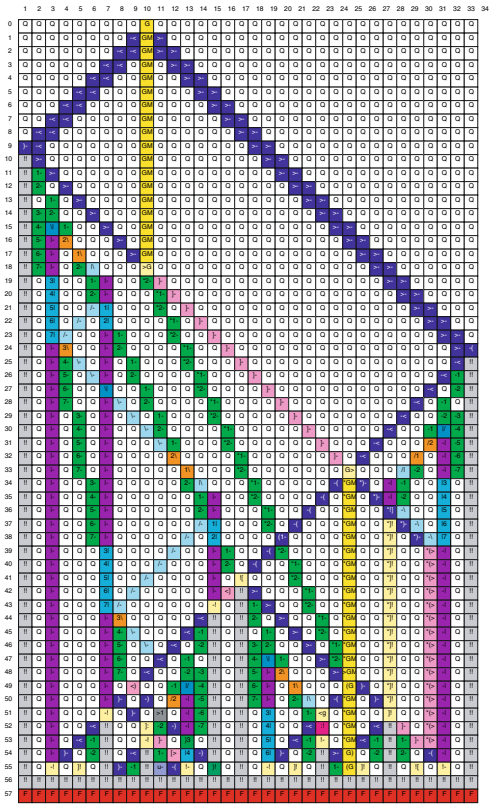
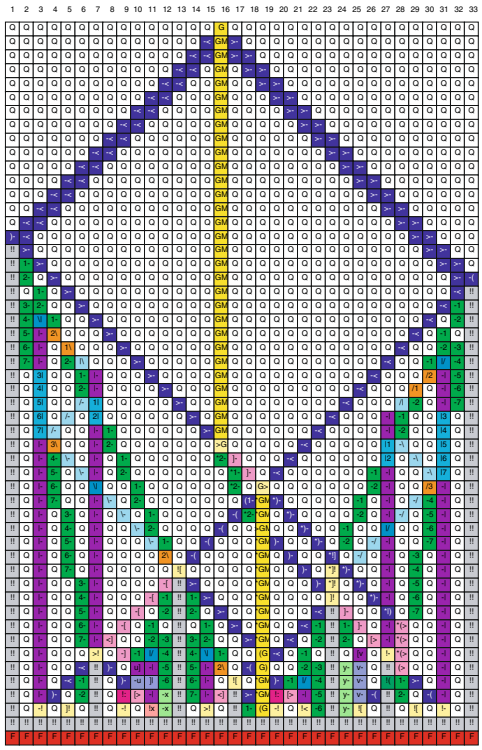
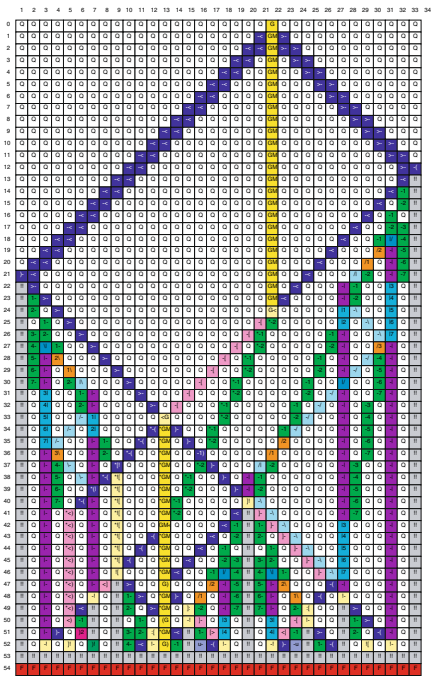
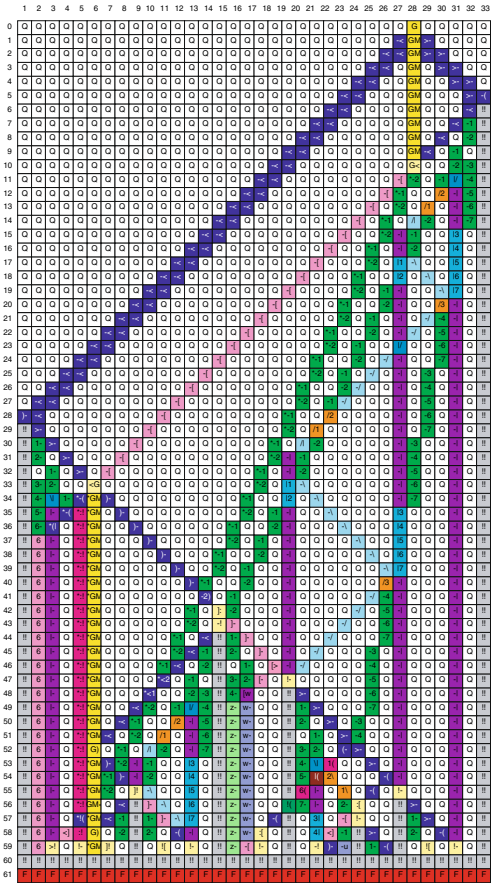


Fig. 6 Space-time diagram of Gerken's FSSP algorithm (*left*) and its snapshots on 37 cells









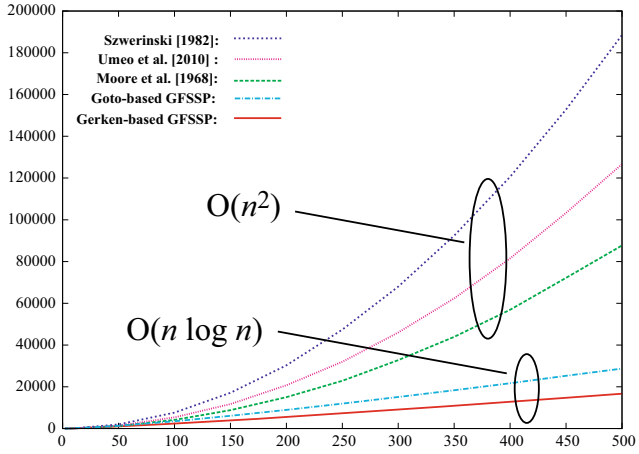


Fig. 8 A comparison of state-change complexities among several minimum-time GFSSP algorithms

4 Six-state non-minimum-time, minimum-state-change GFSSP algorithm

4.1 3*n*-Step FSSP algorithms

A class of 3*n*-step algorithms is an interesting class of synchronization algorithms among many variants of FSSP algorithms due to its simplicity and straightforwardness and it is important in its own right in the design of cellular algorithms. The first 3*n*-step non-minimum-time algorithm was developed by McCarthy and Minsky (1967). Figure 9 (left) shows a space-time diagram for the well-known 3*n*-step FSSP algorithm with a general at the left end of the array. The synchronization process can be viewed as a typical *divide-and-conquer strategy* that operates in parallel in the cellular space. An initial *general* G, located at left end of the array of size *n*, generates two signals, referred to as *a-signal* and *b-signal*, which propagate in the right direction at a speed of 1/1 and 1/3, respectively. The a-signal arrives at the right end at time $t = n - 1$, reflects there immediately, and continues to move at the same speed in the left direction. The reflected signal is referred to as *r-signal*. The b- and the r-signals meet at one or two center cells of the arry, depending on the parity of *n*. In the case where *n* is odd, the cell $C_{\lceil n/2 \rceil}$ becomes a *general* at time $t = 3\lceil n/2 \rceil - 2$. The *general* is responsible for synchronizing both its left and right halves of the cellular space. Note that the *general* is shared by the two halves. In the case where *n* is even, two cells $C_{\lceil n/2 \rceil}$ and $C_{\lceil n/2 \rceil + 1}$ become the next *general* at time $t = 3\lceil n/2 \rceil$. Each *general* is responsible for synchronizing its left and right halves of the cellular space, respectively.

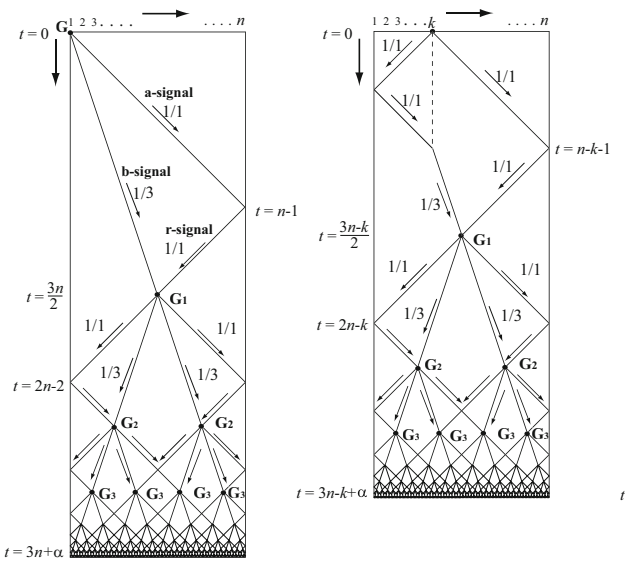


Fig. 9 A space-time diagram for 3*n*-step thread-like FSSP algorithm

Thus, at time $t = t_{\text{center}}$:

$$t_{\text{left}} = \begin{cases} 3\lceil n/2 \rceil - 2 & \text{n : odd} \\ 3\lceil n/2 \rceil & \text{n : even,} \end{cases} \quad (4)$$

the array knows its center point and generates one or two new *general*(*s*) G_1. The new *general*(*s*) G_1 generates the same 1/1- and 1/3-speed signals in both left and right directions simultaneously and repeat the same procedures as above. Thus, the original synchronization problem of size *n* is divided into two sub-problems of size $\lceil n/2 \rceil$. In this way, the original array is split into equal two, four, eight, …, subspaces synchronously. Note that the first general G_1 itself generated at the center is synchronized at time $t = t_{\text{center}}$, and the second general G_2 are also synchronized, and the generals generated after that time on are also synchronized. In the last, the original problem of size *n* can be split into many small sub-problems of size 2. In this way, by increasing the synchronized generals step by step, the initial given array is synchronized. Most of the 3*n*-step synchronization algorithms developed in the past are, more or less, based on the similar schema. It can be seen that, from the path of the b-signal with or without 1 step delay at the center points at each halving iteration, the time complexity $T(n)$ for synchronization scheme above is $T(n) = 3n \pm O(\log n)$.

Figure 9 (right) illustrates a space-time diagram for a generalized 3*n*-step FSSP algorithm that can synchronize the array from any position in the array. The initial general generates two unit-speed signals, each propagating to the left and right ends. Each signal reflects at each end and continues to move towards the center of the array. Depending on the position of the initial general in the array, the right or left reflected signal arrives at the cell where the initial general was. Then, the reflected signal reduces its propagating speed to 1/3 and continues to move

in the same direction at 1/3 speed. The 1/3-speed signal meets the other reflected signal at a center point of the array at time $t = t_{\text{center}}$:

$$t_{\text{center}} = \begin{cases} 3\lceil n/2 \rceil - \min(k, n-k+1) - 1 & \text{n : odd} \\ 3\lceil n/2 \rceil - \min(k, n-k+1) + 1 & \text{n : even,} \end{cases} \quad (5)$$

A new general G_1 is generated at time $t = t_{\text{center}}$. The synchronization operations afterwards are the same as in the case of the initial general at left end. The time complexity $T(k, n)$ for the generalized synchronization scheme above is $T(k,n) = 3n - \min(k, n-k+1) \pm O(\log n)$.

Let $SC(n)$ be state-change complexity for synchronizing n cells with a general at the left end using the $3n$-step thread-like FSSP algorithm illustrated in Fig. 9 (left). One can see that all the synchronization processes are described by thread-like signals and no area-filled-in (paint) type control mechanisms are used in any zone of the space-time diagram. It is noted that the state-changes occur in cells along the a-, b-, and r-signals of length $O(n)$ propagating in the space-time domain. No other cells change their states. Thus we have:

$$SC(n) = \alpha n + 2SC(\lceil n/2 \rceil) \quad (6)$$

Here α is a constant which depends on the width of the threads used in the construction of the a-, b-, and r-signals. Now we have $SC(n) = O(n \log n)$. Intuitively, the state-change complexity $O(n \log n)$ derives form the space-time diagram of the algorithm, where it is the thread-like one, but not filled-in (paint) type.

It can be easily seen that the state-change complexity $SC_g(n)$ for the GFSSP algorithm shown in Fig. 9 (right) is also $SC_g(n) = O(n \log n)$. A survey on the class of $3n$-step FSSP algorithms can be seen in Umeo et al. (2015).

4.2 Six-state non-minimum-time, minimum-state-change GFSSP algorithm

Yunès (2008) presented a 6-state 125-rule implementation for the $3n$-step FSSP algorithm with a general at one end. Our 6-state GFSSP implementation is based on Yunès (2008). The set $\mathcal{Q}$ of the internal states for the constructed GFSSP algorithm is $\mathcal{Q}$={A, Q, B, C, D, E}, where the state A is the initial *general* state, Q is the *quiescent* state, and E is the *firing* state, respectively.

Figure 10, consisting of 145 rules, shows the transition table for the constructed GFSSP. The time complexity for synchronizing any array of length n is $2n + \max(k, n-k) + \mathrm{O}(\log n)$. Figure 11 shows some snapshots of the synchronization process of the algorithm on 16 cells with a general on C_1, C_4, C_7, C_{11}, C_{14}, and C_{16}, respectively.

Thus, we have:

Theorem 10 *There exists a 6-state, non-minimum-time, minimum-state-change GFSSP algorithm.*

5 Summary

We studied the FSSP from a view point of state-change complexity that models the energy consumption of SRAM-type storage with which cellular automata might be built. We have constructed two minimum-time, minimum-state-change GFSSP implementations: one is based on Goto's algorithm, known as the first minimum-time FSSP algorithm, and the other is based on Gerken's one. The Goto-based GFSSP algorithm is realized on a cellular automaton with 434 states and 13328 rules. The Gerken-based one is implemented on a cellular automaton with 213 states and 4077 rules. These algorithms are optimal not only in time

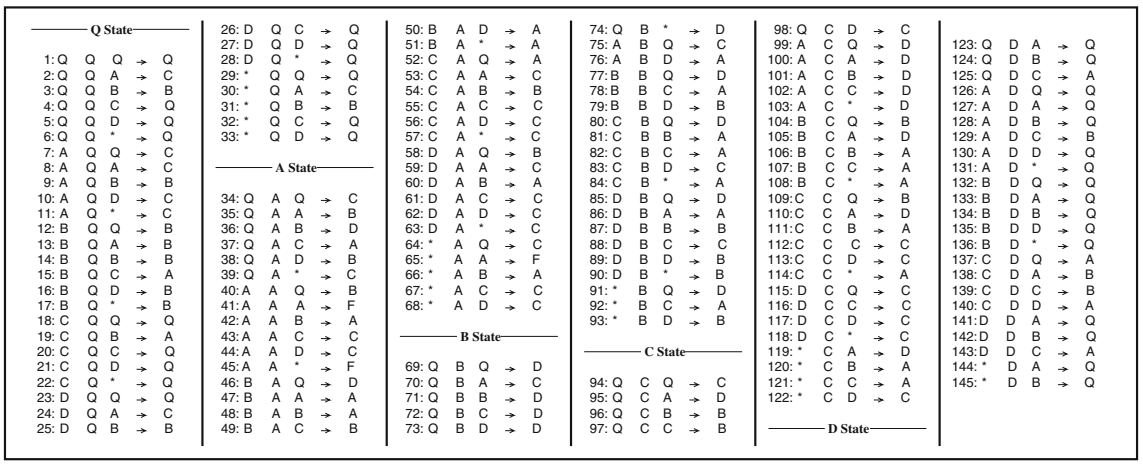


Fig. 10 A 6-state transition table of the non-minimum-time, minimum-state-change GFSSP algorithm

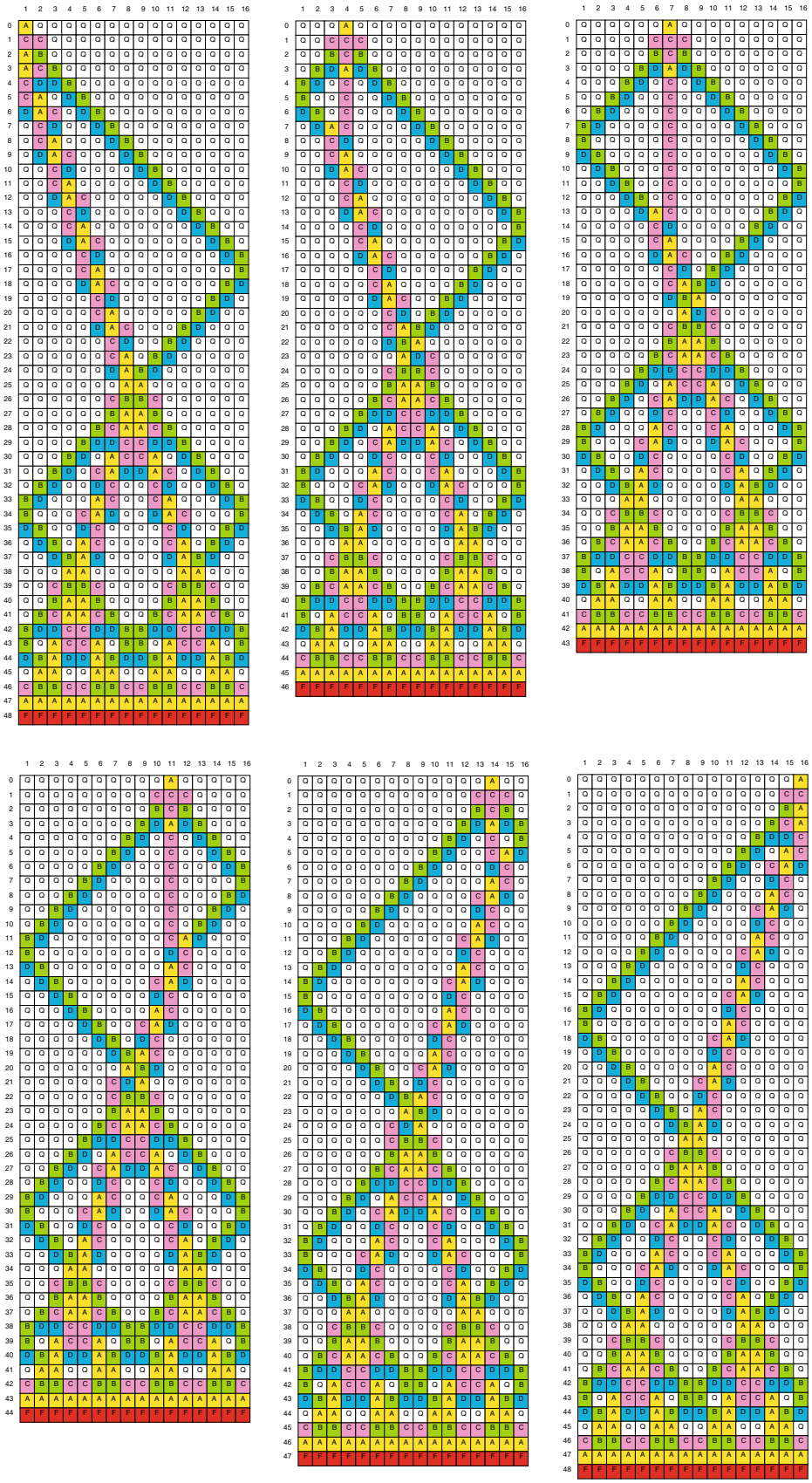


Fig. 11 Snapshots of the synchronization process of the 6-state algorithm for 16 cells with a general on the 1st (*top left*), 4th (*top middle*), 7th (*top right*), 11th (*bottom left*), 14th (*bottom middle*), and 16th (*bottom right*) cells, respectively

but also in state-change complexity. The implemented minimum-time GFSSP algorithms are the first ones having the minimum-state-change complexity. We have also constructed a six-state 145-rule non-minimum-time, minimum-state-change GFSSP implementation. The implemented GFSSP algorithm is the smallest one, known at present, in number of states of the finite state automaton.

Acknowledgements A part of this work is supported by JSPS 16K00026.

References

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

Gerken H-D (1987) Über Synchronisationsprobleme bei Zellularautomaten. Diplomarbeit, Institut für Theoretische Informatik. Technische Universität Braunschweig

Goto E (1962) A minimal time solution of the firing squad problem. Dittoed course notes for Applied Mathematics 298 (with an illustration in color). Harvard University, Harvard

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

Minsky M (1967) Computation: finite and infinite machines. Prentice-Hall, Upper Saddle River

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 GG (1968) A generalized firing squad problem. Inf Control 12:212–220

Schmid H, Worsch T (2004) The firing squad synchronization problem with many generals for one-dimensional CA. In: Proceedings of IFIP World Congress, pp 111–124

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

Umeo H (1996) A note on firing squad synchronization algorithms–a reconstruction of Goto's first-in-the-world optimum-time firing squad synchronization algorithm. In: Kutrib M, Worsch T (eds) Proceedings of IFIP cellular automata workshop 1996. Schloss Rauischholzhausen, Giessen, p 65

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 RA (ed) Encyclopedia of complexity and system science, vol 4. Springer, New York, pp 3537–3574

Umeo H, Imai K (2016) A class of minimum-time, minimum-state-change generalized FSSP algorithms. In: El Yacoubi S et al (eds) Proceedings of ACRI 2016, LNCS, vol 9863. Springer, Heidelberg, pp 1–11

Umeo H, Kamikawa N, Nishioka K, Akiguchi S (2010) Generalized firing squad synchronization protocols for one-dimensional cellular automata—a survey. Acta Phys Pol B Proc Suppl 3:267–289

Umeo H, Maeda M, Sousa A, Taguchi K (2015a) A class of non-optimum-time 3*n*-step FSSP algorithms—a survey. In: Proceedings of the 13th international conference on parallel computing technologies, PaCT 2015, LNCS, vol 9251. Springer, Heidelberg, pp 231–245

Umeo H, Imai K, Sousa A (2015b) A generalized minimum-time, minimum-state-change FSSP algorithm. In: Proceedings of the 4th international conference on the theory and practise of natural computing, TPNC 2015, LNCS, vol 9477. Springer, Heidelberg, pp 161–173

Umeo H, Hirota M, Nozaki Y, Imai K, Sogabe T (2017) A new reconstruction and the first implementation of Goto's FSSP algorithm. Appl Math Comput. doi:10.1016/j.amc.2017.05.015

Vollmar R (1982) Some remarks about the efficiency of polyautomata. Int J Theore Phys 21(12):1007–1015

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

Yunès J-B (2008) An intrinsically non minimal-time Minsky-like 6-states solution to the firing squad synchronization problem. Theor Inf Appl 42(1):55–68