Smaller Solutions for the Firing Squad

Amber Settle[a] and Janos Simon[b]

[a]DePaul University, School of Computer Science, Telecommunications and Information Systems, 243 S. Wabash, Chicago, IL, 60604, asettle@cs.depaul.edu

[b]University of Chicago, Department of Computer Science, 1100 E. 58th Street, Chicago, IL 60637, simon@cs.uchicago.edu

In this paper we improve the bounds on the complexity of solutions to the ¯ring squad problem, also known as the ¯ring synchronization problem. In the ¯ring synchronization problem we consider a one-dimensional array of n identical ¯nite automata. Initially all automata are in the same state except for one automaton designated as the initiator for the synchronization. Our results hold for the original problem, where the initiator may be located at either endpoint, and for the variant where any one of the automata may be the initiator, called the generalized problem. In both cases, the goal is to de¯ne the set of states and transition rules for the automata so that all machines enter a special ¯re state simultaneously and for the ¯rst time during the ¯nal round of the computation. In our work we improve the construction for the best known minimal-time solution to the generalized problem by reducing the number of states needed and give non-minimal-time solutions to the original and generalized problem that use fewer states than the corresponding minimal-time solutions.

Keywords: cellular automata, ¯ring squad synchronization problem, ¯nite automata

1. Introduction

In the ¯ring synchronization problem we consider a one-dimensional array of n identical ¯nite automata. Initially all automata are in the same state except for one automaton designated as the initiator for the synchronization. The machines operate in lock-step, and the transitions of each automaton depend on the state of the automaton and the states of its neighbors. The goal is to de¯ne the set of states and transition rules for the automata so that all machines enter a special ¯re state for the ¯rst time and simultaneously during the ¯nal round of the computation.

Synchronizing a set of processes is an important problem in distributed algorithms, and the ¯ring synchronization problem is one of the simplest and oldest formalizations of this problem. By studying this fundamental and elegant question we hope to gain insight into other such problems and develop techniques and intuitions that will be useful for generalizations of the problem. For example, solutions to more general versions of the ¯ring synchronization problem, in which the underlying network is a ring, an undirected graph, or strongly-connected directed graph, work by reducing the graph to simpler structures that are synchronized by solutions to the one-dimensional problem [2,3,5,10,11].

Two obvious criteria for ranking solutions are the speed of the solution, namely the time needed to synchronize, and the complexity of the solution, measured by the number of states of the automaton.

It is easy to show that in the original problem an array of n automata cannot be synchronized before time step 2n ¡ 2 [9]. This is the minimal amount of time for the initiator to send a message to the far end automaton and get a message back. A minimal-time solution is a set of states and transition rules for which synchronization occurs after exactly 2n ¡ 2 time steps, whereas a non-minimal-time solution is one where synchronization takes more than 2n ¡ 2 time steps. The same notions can also be de¯ned for the generalized version of the problem, and there is a similar bound on the number of time steps necessary for synchronization.

In this paper we improve the bounds on the complexity of solutions to the ¯ring synchronization problem. We give a 9-state minimal-time automaton for a generalized version of the problem. This improves on the best previously known construction, an automaton using 10 states that appeared in a paper by Szwerinski [15]. We give a 6-state non-minimal-time automaton for the original problem where the initiator may be located at either endpoint. This automaton uses two fewer states than the best known minimal-time solution for the same problem [1]. We also present a 7-state non-minimal-time solution to the generalized problem that uses 2 fewer states than the best known minimal-time automaton mentioned above. Both of our non-minimal-time automata are based on a 6-state solution to a restricted version of the problem produced by Mazoyer [6]. We also give a proof of correctness for each of our non-minimal-time automata.

1.1. Previous work

The ¯ring synchronization problem has a long history, and many variants of the problem have been studied. We will focus on two of the earliest variants, the original ¯ring synchronization problem and the generalized ¯ring synchronization problem. In the original problem the goal is to synchronize a one-dimensional array of ¯nite automata, where the synchronization is initiated by one of the automata at the end of the array, appropriately named the initiator. A restricted version of the problem requires that the location of the initiator be ¯xed ahead of time. We call this the restricted problem. In the generalized problem we also consider a one-dimensional array, but the initiator for the synchronization is allowed to be located in any position of the array. A summary of results for these variants of the problem may be found in Section 1.4. We now describe the results.

The ¯ring synchronization problem was proposed by Myhill, and the ¯rst published solution to the problem is due to McCarthy and Minsky [7]. Their solution uses a divide and conquer algorithm and takes 3n steps to synchronize. The ¯rst minimal-time automaton for the original problem was produced by Goto, who gave a solution with over 1000 states in 1962 [4], and work in the area quickly focused on ¯nding minimal-time solutions using fewer states. In 1966 Waksman [17] gave a 16-state minimal-time solution, and Balzer [1] independently produced an 8-state solution using the same ideas. Balzer also showed, using a heuristic search algorithm, that there is no 4-state minimal-time solution to the original problem. (See Sanders' paper [12] for a recent result con¯rming the bound). In 1987 Mazoyer [6] produced a 6-state solution to the restricted version of the problem.

Mazoyer suggested that all solutions with few states must necessarily be minimal-time, a conjecture based on the idea that the simplest solution will naturally be the fastest. Yunès [18] contested the conjecture in 1994 by giving an implementation of McCarthy and Minsky's solution that requires only 13 states and time $t(n) = 3n - \epsilon_n \log n + C$, where $0 \le \epsilon_n < 1$ and by producing a 7-state solution that uses time $t(n) = 3n - 2\epsilon_n \log n + C$, where $0 \le \epsilon_n < 1$. Both automata solve the restricted version of the problem.

Moore and Langdon introduced the generalized firing synchronization problem described above in 1968 [8]. In their paper Moore and Langdon gave a 17-state minimal-time solution for the generalized problem. Varshavsky, Marakkovsky and Peschansky [16] improved this result, producing a 10-state minimal-time solution.

Further work on the generalized problem was done in 1982 by Szwerinski [15]. Szwerinski considered symmetric solutions. A symmetric solution is one in which an automaton cannot distinguish between its left and right neighbors. Szwerinski gave a 10-state, symmetric, minimal-time solution.

To the best of our knowledge, non-minimal-time solutions to the generalized problem had not been studied prior to our work [13].

1.2. Lower versus upper bounds

Despite its long history, many important open problems remain for the firing synchronization problem. One of the most fundamental is determining precisely how many states an automaton solving the problem requires.

Balzer has shown that no 4-state minimal-time automaton for the restricted version of the original problem exists. In each variant of the problem this leaves a gap between the lower bound and the best known minimal-time solutions. For the unrestricted original problem this gap is 4 states. Any lower bound for the original problem also applies to the generalized problem since the original problem must be solved as a subcase. Thus the gap for the generalized problem is 6 states.

Work on non-minimal-time solutions has been even more limited. The only known lower bound for non-minimal-time automata is a 3-state bound on solutions to the original firing synchronization problem [14]. This leaves a gap of 4 states between the best known solution to the original problem and the lower bound. As stated before there were no known non-minimal-time solutions to the generalized problem prior to this work.

1.3. Our contributions

In Section 3 we present a 9-state, minimal-time, symmetric solution to the generalized firing synchronization problem. The automaton contains within it an 8-state symmetric solution to the original problem. The 9-state automaton has the fewest states of any known minimal-time solution to the generalized problem.

We present in Section 4.1 a 6-state non-minimal-time solution to the original problem that allows the initiator to be located at either the left or the right endpoint of the array. This automaton has 2 fewer states than Balzer's 8-state minimal-time automaton [1]. The transition function for the automaton may be found in Table 8. We prove the solution correct by proving the following theorem:

Theorem 1.1 For any $n \in \mathbb{N}; n \ge 2$, a one-dimensional array of n automata with transition function given in Table 8 will synchronize, with all automata entering the fire state F at time t(n) = 2n ¡ 1 if the initiator is located at the left endpoint of the array or time t(n) = 3n + 1 if the initiator is located at the right endpoint.

Finally, in Section 4.4 we present a 7-state non-minimal-time solution to the generalized problem mentioned above, where the initiator can be anywhere in the array. The transition function for the automaton may be found in Table 12. This automaton has the fewest states of any known solution to this problem and requires 2 fewer states than our minimal-time automaton. We also have a proof of correctness for the 7-state solution which shows that a one-dimensional array of n automata with initiator located in position k will synchronize in 2n ¡ 2 + k time steps for any n 2 N ; n ¸ 2. The details of the proof are omitted but may be found elsewhere [14].

Few researchers in this area have provided correctness proofs for their automata. Indeed, as far as we know, prior to this work only Balzer [1] and Mazoyer [6] have published proofs of correctness.

Our work provides additional evidence that Mazoyer's conjecture does not hold by giving non-minimal-time solutions to both the original and generalized versions of the ¯ring synchronization problem which require fewer states than the best known minimal-time solutions. Indeed optimal non-minimal-time solutions may use even fewer states than our constructions, as our automata are based on minimal-time solutions to restricted versions of the problem.

1.4. Summary of results

Results for both the original ¯ring synchronization problem and the generalized ¯ring synchronization problem are summarized in Tables 1 and 2. Table 1 gives the minimal-time solutions and Table 2 lists non-minimal-time results.

2. The ¯ring synchronization problem

The ¯ring synchronization problem, sometimes also called the ¯ring squad problem, is a classical problem of synchronization. Consider a one-dimensional array of n ¯nite automata in which all automata are identical except the ones on either end of the array. The end machines di®er from the interior machines in that each is aware that it is at the end of the array. It is possible to use a single transition function for all machines if we allow an end marker next to each end machine and use this marker in de¯ning the transition function. This end marker does not count as a state in the solution.

The automata in the array work synchronously, and the state of an automaton at time t only depends on its and its neighbors' states at time t ¡ 1. At time step 0 of the computation all automata are in a special quiescent state except for one automaton that is designated as the initiator for the synchronization. In the original ¯ring synchronization problem this machine must be located at the end of the array. The problem is to de¯ne the set of states and transition rules for the automata so that all machines enter a special ¯re state simultaneously and for the ¯rst time at some time t(n).

The transition function for each automaton can be given as a set of 4-tuples. The 4-tuple (X,Y,Z,W) represents the rule that an automaton currently in state Y, with left neighbor in state X and right neighbor in state Z will enter state W at the next time step. We will denote this by XYZ ! W. By de¯nition automata solving the ¯ring synchronization

Table 1
Summary of results: minimal-time solutions

Author	**Year**	**# States**	**Notes**	**Reference**
Original problem – upper bounds				
Goto	1962	> 1000		[4]
Waksman	1966	16	See also [1]	[17]
Balzer	1967	8	See also [17]	[1]
Mazoyer	1987	6	Restricted problem	[6]
Original problem – lower bound				
Balzer/Sanders	1967/1994	4	Also applicable to general problem	[1,12]
Generalized problem – upper bounds				
Moore, Langdon	1968	17		[8]
Varshavsky, et al.	1970	10		[16]
Szwerinski	1982	10	Symmetric	[15]
Settle	1998	9	Symmetric	[13]

problem are deterministic so that there is at most one tuple (X,Y,Z,W) for any triple of states X,Y,Z. We assume that end markers are used to allow a single transition function for all automata. When giving transitions for automata at the end of the array, we will use a star to indicate the end marker. For example, the transition $XY? \rightarrow W$ indicates that an automaton on the right end of the array in state Y with neighbor in state X will enter state W at the next time step.

It is easy to show that $t(n) \geq 2n - 2$ [9]. This is the minimal amount of time for the initiator to send a message to the automaton at the opposite end of the array and get a message back.

A minimal-time solution is a set of states and transition rules for which $t(n) = 2n - 2$, whereas a non-minimal time solution is one for which $t(n) > 2n - 2$. An N-state solution of the problem is one in which each automaton has N states, including the quiescent and fire states.

A symmetric automaton is one which has a symmetric transition function, that is, whenever a transition $XYZ \rightarrow W$ is defined, the transition $ZYX \rightarrow W$ must also be defined. This means that the automata cannot distinguish their left and right neighbors.

In the generalization of the original problem introduced by Moore and Langdon [8], the initiator may be located anywhere in the one-dimensional array of finite automata. Let k denote the position of the initiator in the array, where $1 \leq k \leq n$, and let $m = \min\{k - 1; n - k\}$. Moore and Langdon showed that $2n - m - 2$ is the minimal firing time for the generalized problem.

In the proof of Theorem 1.1 given in Section 4.1, we require some additional notation. Let $(p_{i;m}; t) = S$ indicate that the state of the automaton in position i at time t of a

Table 2
Summary of results: non-minimal-time solutions

Author	**Year**	**# States**	**Time**	**Notes**	**Reference**
Original problem { upper bounds					
Yunes	1994	13	$3n \S \pounds_n \log n + C$	$0 \cdot \pounds_n < 1$, Restricted.	[18]
Yunes	1994	7	$3n \S 2\pounds_n \log n + C$	$0 \cdot \pounds_n < 1$, Restricted.	[18]
Settle, Simon	1998	6	$2n$ ¡ 1 or $3n + 1$	Unrestricted	[13]
Original problem { lower bound					
Settle, Simon	1999	3			[14]
Generalized problem { upper bound					
Settle, Simon	1998	7	$2n$ ¡ $2 + k$		[13]

synchronization for an array of length m is S. Also, let $(p_{[i;j];m}; t) = S_i S_{i+1} ::: S_j$ denote that 8l $i \cdot l \cdot j$ $(p_{l;m}; t) = S_l$. Finally, $(p_{[i;j];m}; t) = S$ means that 8l $i \cdot l \cdot j$ $(p_{l;m}; t) = S$. If the length of the array under consideration has been ¯xed, then the second subscript on the above notation will be omitted.

3. The minimal-time solution

In this section we describe the 9-state minimal-time solution to the generalized ¯ring synchronization problem. The 9-state automaton is a modi¯cation of Szwerinski's 10-state solution.

The strategy for Szwerinski's automaton, like all other known solutions to the generalized problem [8,16], is to reduce the synchronization of the generalized problem to the original problem. Once this has been completed, the synchronization is ¯nished by a solution to the original problem contained within the transition function for the generalized solution. For this reason Szwerinski's solution works in two phases, the ¯rst of which accomplishes the reduction to the original problem, and the second which completes the synchronization using the underlying original solution.

Szwerinski's 10-state automaton contains an 8-state solution to the original problem and uses two additional states for the ¯rst phase of the synchronization. The 9-state automaton also contains the 8-state original solution, but uses only one additional state for the ¯rst phase.

In the remainder of the section, we describe at a high level how the 9-state automaton works. We then give a detailed explanation of the underlying 8-state solution to the original problem and explain how the two phases work in the 9-state automaton. We also describe the changes made to Szwerinski's automaton to reduce the number of states. Finally, we give the transition function for the 9-state automaton and discuss the consequences of the work for solutions to d-dimensional arrays.

Table 3
Nine-state solution

State	State's purpose
F	¯ring state
G	initiator state; produces new A-signals
A	wake-up signal; produces new initiators in conjunction with the B-marker
B	marker for the placement of new initiators
D	alternate marker for initiator placement; also the state for the initiator that starts the simulation; not used in the 8-state solution
R	signal to advance and produce B-markers; also a parity marker
Z	quiescent state; also used as a ¯ller state and a parity marker
Q	parity marker
P	parity marker

3.1. A high-level description

In the 9-state minimal-time solution, the line is repeatedly divided into halves as new initiators are placed in the center of each of the intervals. The simulation ends when all automata become initiators and ¯re at the next time step.

The transition function for the 9-state automaton is given in Table 6. It can be seen from the transition function that there are several states that propagate toward neighboring automata. We call these states signals, since their purpose is to carry information from one part of the array to another. Other states remain stationary until they come into contact with certain signals. We call these states markers. They act as placeholders indicating signi¯cant positions in the array, such as the center of the line. The states of the 9-state solution and each state's purpose is summarized in Table 3.

In order to understand how the division of the array is performed, consider what happens when the initiator is located at either end. A run for this case can be found in Table 4. The time steps given below refer to the run in that table.

This case is simply the original problem and is handled by the underlying 8-state automaton. The initiator sends out a signal that produces a second initiator when it reaches the opposite end of the line. In the sample run this occurs between time steps 0 and 16. When this wake-up signal is re°ected back by the new initiator, it intersects with markers created in the wake of the ¯rst signal and produces a third initiator (or pair of initiators depending on the parity of the original line) located at the center of the array. This occurs at time step 24 of the sample run. This division of the line continues until every other automaton is an initiator, which occurs at time step 30. At the next time step in the run every automaton becomes an initiator, and at the following time step all automata ¯re.

In the case where the initiator is located somewhere in the middle of the array, the goal is to reduce the problem to the original problem. To achieve this reduction, a new initiator is produced at the center of the array at time n ¡ m + b$\frac{n}{2}$c. The run then continues from that point as if the ¯rst initiator had been located at one of the endpoints. An extra state is used to achieve this ¯rst subdivision, but after the central initiator is created the remainder of the run is handled by the subset of states corresponding to the 8-state solution and the extra state does not appear.

3.2. The underlying 8-state solution to the original problem

In order to explain in more detail how the 9-state solution works, we ¯rst present the 8-state solution to the original problem. Recall that the original problem requires that the ¯rst initiator be located at one of the endpoints of the array of automata. The 8-state solution is derived from the 9-state solution to the generalized problem by deleting all occurrences of the state D. The time steps given below refer to the run found in Table 4. The solution works as follows.

The ¯rst initiator in state G sends out an A-signal to the other end of the line. In the run the A-signal is produced at time step 1. The A-signal moves at a rate of one automaton per time step. As the A-signal advances away from an automaton, it leaves the automaton in one of two states, either R or P. An R is produced at all even time steps and a P at all odd time steps. Thus the parity of the line segment the A-signal crosses can be determined by the state appearing behind the A-signal once it reaches the end of the line.

The R moves back in the direction from which the A-signal came at the rate of one automaton per time step. The ¯rst R-signal is produced at time step 2 of the sample run. When the R-signal collides with the initiator at the other end it produces a B-marker. This occurs in the run at time step 3. The new B-marker moves away from the initiator one position each time it encounters a new R-signal. For example, the ¯rst B-marker advances at time step 6 of the run. This ¯rst marker will be the one that will mark the center(s) where the next initiator(s) should be produced. In order to mark the $\frac{1}{4}$, $\frac{1}{8}$, :::, positions in the array, where the next initiator(s) need to be placed, additional B-markers need to be produced. This is done by allowing the R-signal to continue past a B-marker every other time a B-marker advances. The R-signal can then produce and/or advance other B-marker(s). The state of the automaton behind the B-marker determines whether the R-signal advances. If the state of the automaton behind the B-marker is a P, then the R-signal will regenerate behind the B-marker after advancing it. An example of this can be seen at time steps 11, 12 and 13 of the run. On the other hand, if the B-marker is followed by a quiescent automaton, the R-signal will vanish after moving the B-marker forward. This case can be seen at time steps 14 and 15.

As the A-signal hits the end of the array, it is re°ected back. In the sample run this occurs at time steps 15, 16 and 17. Depending on the parity of the array, one of two state con¯gurations will be produced behind the A-signal as it advances. If the line is of odd length, the A-signal will be followed at alternating time steps by an R or P, as with the ¯rst A-signal. If the array has even length, then the A will be followed by a Z which is alternately followed by an R or P. The sample run has 17 automata so that the former case holds.

By the time the A-signal is re°ected back, it has sent enough R-signals to bring the

B-marker to the middle of the line. This is because the A-signal produces an R-signal every other time step, creating half as many R-signals as the length of the array. Because the leading B-marker moves one position each time it encounters one of these R-signals, it will have moved to the center of the array once all of the R-signals reach it. This happens before the re°ected A-signal can reach the B-marker.

When the A-signal reaches the B-marker it produces the new initiator(s). In the run this occurs at time step 24. A single new initiator is produced if the line has odd length, since in that case there is a middle point of the array. This is true if the A-signal reaches the B-marker with the automaton behind it in state P. If the line length has even parity then two new initiators need to be produced since there are two central positions in the array. This occurs when the A-signal reaches the B-marker with a quiescent automaton behind it. Again, because there are 17 automata in the run, the former case holds.

The new initiator(s) now begins to recursively subdivide the array. A-signals are sent out toward each end of the array. These will intersect the remaining B-markers produced in the wake of the ¯rst A-signal and the re°ected A-signal to produce the initiators at the quarter positions. This process continues until every other automaton is an initiator. At that point all automata become initiators and then ¯re at the next time step.

3.3. The 9-state automaton

The 9-state solution to the generalized problem works in a manner similar to the 8-state automaton. The additional state D is used as the state for the ¯rst initiator. If this initiator is at the end of the line, it sends an A-signal toward the other end of the array and enters state G. The rest of the simulation is then the same as that produced by the 8-state automaton described above.

In the case where the ¯rst initiator is located somewhere in the middle of the array the goal is to reduce the synchronization task to the original problem. The ¯rst initiator begins this process by sending out A-signals in both directions. The R-signals produced in the wake of the A-signals meet at the initiator and disappear. The A-signals will create new initiators as they reach the end of the line and are re°ected back toward the middle. If the array is of odd length and the initiator is located at the center point, then the A-signals will meet back at the original initiator, putting it into state G. If the initiator is not located at the center point of the array, then the A-signal sent to the closer end returns to the initiator ¯rst. If the length of the shorter segment is even, a D-marker is produced when the A-signal reaches the initiator. If the shorter segment has odd parity, then the A-signal creates a B-marker when it reaches the initiator. These markers now advance in response to the R-signals sent by the A-signal on the opposite side and move to the center of the array. There they are met by the re°ected A-signal and create initiator(s) in state G. Whether a single initiator or two initiators are produced is determined by the parity of both the short and long line segments, which is encoded by the marker state and the state of the automaton behind the A-signal.

Once the central initiator(s) are created, the remainder of the run is performed by the 8-state automaton, as described previously. The state D does not appear after this point.

3.4. Changes necessary to reduce the number of states

As was shown above, the 9-state solution uses the same ideas as Szwerinski's automaton and yet achieves the same results with one fewer state. This section will describe the changes made to Szwerinski's solution to allow 9 states to su±ce for the synchronization. Note that both the automata in question are symmetric so that when a transition XYZ ! W is mentioned below, the transition ZYX ! W is also de¯ned.

The 9-state automaton is produced by merging the states M and D in Szwerinski's solution. These are the additional states used in conjunction with the 8-state solution to the original problem that allow the initiator to be located at an arbitrary position in the array. In Szwerinski's solution M is used as the ¯rst initiator and D is used as the marker produced when the A-signal is re°ected back from the shorter side of the line. In the 9-state automaton, D is used for both of these purposes.

In order to merge two states it needs to be the case that the two states do not have any con°icts, either direct or indirect. A direct con°ict between two distinct states X and Y occurs when there are states R, S, T and U such that the transitions RXS ! T and RYS ! U are de¯ned where T and U are di®erent states. An indirect con°ict exists between distinct states X and Y if there are states R, S, T and U such that the transitions XRS ! T and YRS ! U are de¯ned, where T and U are di®erent states. Both con°icts prohibit the merging of the states since if the states were merged one transition would be incorrectly de¯ned.

In Szwerinski's automaton there are three con°icts between the states D and M, two direct con°icts and one indirect con°ict. The direct con°icts are ADQ ! G and AMQ ! B and DDP ! Q and DMP ! G, and the indirect con°ict is BDD ! P and BDM ! G.

The ¯rst step in eliminating these con°icts in order to merge the states D and M is to determine which transitions are unused and eliminate them. A transition XYZ ! W is unused if the con¯guration XYZ does not occur. The con¯gurations AZD, DZP, ZBG, BDD, BDR, DDP, PRQ, QRG, QPQ and PGQ do not occur, and this means that the transitions BDD ! P and DDP ! Q are unused. This eliminates one of the direct con°icts and the only indirect con°ict between the states D and M. The remaining direct con°ict cannot be resolved this way since both the transitions ADQ ! G and AMQ ! B are used.

Resolving the remaining direct con°ict requires changing the way in which the D-markers work. The goal of this change is to ensure that the con¯guration ADQ never occurs. This would in turn render the transition ADQ ! G unused and allow the states D and M to be merged.

As was described in the previous section, the D-markers are produced when the ¯rst initiator is not located at the center of the array nor at either end of the array. In this case, the A-signal sent to the shorter end of the array will reach the initiator's position ¯rst. If the parity of the segment crossed is even then a D-marker will be produced. In order to understand the change that eliminated the last con°ict, we need to consider how the D-markers advance.

The D-markers serve the same purpose as the B-markers, that is, they are used to mark the position where a new initiator needs to be produced. Like the B-markers, the D-markers advance when they intersect with an R-signal, but the way that they move is slightly di®erent from the B-markers. When an R-signal hits a D-marker, the D-marker does not immediately advance. Instead it produces a B-marker in front of it at the next time step. One step later the B-marker vanishes and the D-marker moves into its position.

The R-signal that caused the D-marker to advance is allowed to pass through the D-marker every other time that the marker advances, as with the B-markers. When the D-marker has an automaton in state P behind it, the R-signal is allowed to pass through. In this case, the D-marker leaves state Q behind as it advances into the B-marker's position. This is the state that allows the R-signal to be reproduced one position away from the D-marker at the next time step. An example of this is given in the following diagram. In this ¯gure the R-signal intersects the D-marker at time i, the D-marker produces the B-marker at time i + 1, the D-marker advances at time i + 2, and the R-signal is reproduced at time i + 3.

i	:	...	P	**D**	R	...
i + 1	:	...	P	**D**	B	...
i + 2	:	...	P	Q	**D**	...
i + 3	:	...	R	Q	**D**	...

When the D-marker has an automaton in state Q behind it, the R-signal vanishes after it advances the D-marker. This can be seen in the following example. In this diagram the R-signal intersects the D-marker at time i, the D-marker produces the B-marker at time i + 1, the D-marker advances at time i + 2 and the R-signal does not appear after time i.

i	:	...	Q	**D**	R	...
i + 1	:	...	Z	**D**	B	...
i + 2	:	...	Z	P	**D**	...

Thus the Q-marker provides parity information for the D-marker, determining that the R-signal should vanish if the Q appears behind the D when the R-signal hits it, as well as allowing the R-signal to be re-created. The con¯guration ADQ arises when the re°ected A-signal hits the D-marker when it has a Q located behind it. A way to eliminate the con¯guration ADQ is to replace the Q-marker with another state that serves the same purpose. This is precisely what the 9-state solution does.

The 9-state solution uses the state R as a replacement for the Q-marker. This means that the states R and P now provide parity information for the D-marker. An R-signal will be reproduced behind the advanced D-marker if it intersects the D-marker when a P appears behind the D-marker. This can be seen in the following example. As before the R-signal intersects the D-marker at time i, the D-marker produces the B-marker at time i + 1, the D-marker advances at time i + 2 and the R-signal is reproduced at time i + 3.

i	:	...	P	**D**	R	...
i + 1	:	...	P	**D**	B	...
i + 2	:	...	P	R	**D**	...
i + 3	:	...	R	R	**D**	...

On the other hand, if the R-signal hits the D-marker when the neighbor behind it is in state R, the R-signal will disappear after the D-marker advances. This can be seen below. Again, the R-signal intersects the D-marker at time i, the D-marker produces the B-marker at time i + 1, the D-marker advances at time i + 2 and the R-signal does not appear after time i.

i	:	...	R	**D**	R	...
i + 1	:	...	Z	**D**	B	...
i + 2	:	...	Z	P	**D**	...

This substitution forces one additional change to the way the automaton works. The 10-state solution contained the transition PRM ! Q, but the changes made above require that the transition PRD ! R be de¯ned. This is a concern since we now have an indirect con°ict between D and M.

The con°ict does not cause a problem, and in order to see why we need to consider where the transition PRM ! Q is used in the 10-state solution. It appears immediately before the central initiator(s) are produced and is used to create a new A-signal after the initiator(s) are produced. An example of this situation appears below. The transition is used at time i, the new initiators are produced at time i + 1, and the A-signal appears at time i + 2.

i	:	...	D	M	R	P	B	...
i + 1	:	...	G	G	Q	R	B	...
i + 2	:	...	G	G	A	B	B	...

Since the transitions GGR ! G, GRR ! A and RRB ! B are already de¯ned in the automaton, substituting the state R for the state Q here causes no problems. Thus we can simply de¯ne PRM ! R and produce the correct con¯gurations.

Once these changes are made, the con¯guration ADQ never appears, so that the transition ADQ ! G becomes unused. There are now no con°icts, either direct or indirect, between the states D and M, and the 9-state solution can be produced by merging these two states.

3.5. The nine-state transition function

Table 6 shows the transition function for the 9-state automaton. The state of an automaton at the next time step can be found by looking at the entry in the column corresponding to the automaton's present state and the row corresponding to the states of its neighbors. Since the automaton is symmetric, the orientation of the neighbors is irrelevant. A star is used to indicate the end of the array. In order to obtain the 8-state automaton that solves the original problem, the column corresponding to D must be removed and all occurrences of D deleted in the remaining table.

3.6. Consequences for d-dimensional solutions

In his paper, Szwerinski not only gave a solution to the one-dimensional ¯ring synchronization problem, but he also provided a symmetric, time-optimal solution to the problem for d-dimensional arrays [15].

In a 2-dimensional array, each automaton is connected to 4 other machines, that is, the machines above, below, and to each side of it. End markers are used to simulate the 4th (and, as necessary, 3rd) neighbors for the automata on the ends of rows and columns. Note that the machines along the diagonals from an automaton are not considered to be adjacent to the automaton. A d-dimensional array is simply the generalization of this con¯guration. The initiator for the d-dimensional problem may be located anywhere in the array.

Szwerinski solved the d-dimensional problem using a cross-product of the states for the one-dimensional solution. The state for the machines in the d-dimensional case is an element of A £ A £ C £ C £ S where A is the set of states for his one-dimensional solution, that is, jAj = 10, jCj = 8 and jSj = 4. Thus his solution requires that the state space have size 25; 600.

By improving the one-dimensional solution to use only 9 states, we improve the size of the state space for the d-dimensional problem to 20; 736.

4. The non-minimal-time automata

In this section we discuss the non-minimal-time automaton. In Section 4.1 we present the six-state solution to the original problem, and in Section 4.4 we give the seven-state solution to the generalized problem.

4.1. A 6-state automaton for the original problem

The 6-state automaton is based on Mazoyer's 6-state solution to the restricted version of the original ¯ring synchronization problem. Recall that Mazoyer's minimal-time automaton requires the initiator to be located at the left endpoint of the array. Mazoyer's solution works by dividing the line of automata into unequal parts, one of length $\frac{2}{3}$n and the other of length $\frac{1}{3}$n. An initiator is placed at the left endpoint of the shorter segment, and each segment is then recursively subdivided. After every automaton becomes an initiator, the automata ¯re and the synchronization ends. For a detailed description of that solution see Mazoyer's paper [6].

Unlike Mazoyer's solution, the initial con¯guration for our 6-state non-minimal-time automaton allows the initiator to be located at either the left or right endpoint of the array. In either case the goal of the non-minimal-time automaton is to produce the initial con¯guration necessary for Mazoyer's solution. The synchronization of the array is then completed by the minimal-time automaton.

4.1.1. The description of the solution

The behavior of our 6-state automaton is as follows. The state B is used as the state for the ¯rst initiator. If the initiator is located at the left endpoint in the initial con¯guration, the automaton simply enters the state G at the next time step. This puts the array in the con¯guration necessary for the minimal-time automaton, which then synchronizes the line. The entire process takes one additional step beyond the time for the minimal-time synchronization, and the line is synchronized in time 2n ¡ 1.

If the initiator is located at the right endpoint when the synchronization begins, a signal is sent toward the left endpoint. The purpose of this signal is to produce an initiator in state G at the left end of the array, leaving the rest of the automata in the quiescent state L as the signal passes. This puts the array in the con¯guration necessary for the minimal-time solution, which then completes the synchronization.

The signal which produces the initiator at the left endpoint consists of four states, AACB. The B initiator enters state A at time step 1, and the A then advances left, producing the rest of the signal behind it during time steps 2 through 4. The signal then moves at a rate of one automaton per time step toward the left. As the signal moves, the automata behind it are once again put into the quiescent state L. When the signal reaches the left end of the array, the signal collapses, leaving only the last two states in the signal. At the next time step the G initiator is produced and the minimal-time synchronization takes place.

One step is necessary to produce the lead state A in the signal. Another n time steps are required for the A to reach the left endpoint. It then takes an additional two time steps for the ¯rst two states of the signal to vanish and create the G initiator. The minimal-time automaton then ¯nishes the synchronization. Thus the whole process takes time $1 + n + 2 + (2n ¡ 2) = 3n + 1$. A run of the six-state solution may be found in Table 7.

4.2. The 6-state transition function

Table 8 gives the transition function for the 6-state non-minimal-time automata. The state of an automaton at the next time step can be found by looking at the table corresponding to the automaton's present state. The state that the automaton should enter at the next time step is the one in the row and column corresponding to the states of its left and right neighbors respectively. A star is used to indicate the end of the array.

4.3. A proof of correctness

We now give the proof of correctness for the 6-state non-minimal-time automaton by proving Theorem 1.1. There are two parts of the proof, one for each of the possible positions of the ¯rst initiator. In each part it su±ces to show that at some time step t for any simulation of a length n array, $(p_1; t) = G$ and $(p_{[2;n]}; t) = L$. This is because that con¯guration is the initial con¯guration for Mazoyer's 6-state minimal-time automaton, which completes the synchronization.

4.3.1. Part I: Initiator at the left endpoint

Proving the following lemma proves that if the initiator is located at the left endpoint, then the initial con¯guration for Mazoyer's automaton is produced at time step 1.

Lemma 4.1 Suppose $(p_1; 0) = B$ and $(p_{[2;n]}; 0) = L$. Then $(p_1; 1) = G$ and $(p_{[2;n]}; 1) = L$.

Proof: There are three cases to be considered, depending on the value of n.

Case 1: $(n = 2)$ By de¯nition of the problem we know that $(p_0; 0) = ?$ and $(p_3; 0) = ?$. The transition ?BL ! G implies $(p_1; 1) = G$ and BL? ! L means that $(p_2; 1) = L$. This is the conclusion of the lemma.

Case 2: $(n = 3)$ Again we have that $(p_0; 0) = (p_4; 0) = ?$, so that ?BL ! G implies $(p_1; 1) = G$, BLL ! L means $(p_2; 1) = L$ and LL? ! L implies $(p_3; 1) = L$.

Case 3: $(n ¸ 4)$ Similar to the previous cases, ?BL ! G means $(p_1; 1) = G$, BLL ! L implies $(p_2; 1) = L$, LLL ! L shows $(p_{[3;n ¡ 1]}; 1) = L$ and ?LL ! L means $(p_n; 1) = L$. This is the con¯guration claimed in the lemma. }

4.3.2. Part II: Initiator at the right endpoint

The following lemma shows that if the initiator is located at the right endpoint, then the initial con¯guration for Mazoyer's 6-state minimal-time automaton is produced at time step n + 3.

Lemma 4.2 Suppose $(p_{[1;n ¡ 1]}; 0) = L$ and $(p_n; 0) = B$. Then $(p_1; n + 3) = G$ and $(p_{[2;n]}; n + 3) = L$.

The remainder of this section will consider arrays for which n ¸ 5, since the proof for Lemma 4.2 for the cases n < 5 is by inspection. In section 4.3.4 we consider runs for these short arrays.

There are three stages to the process of producing the initial con¯guration for the 6-state minimal-time automaton. Since the signal that will produce the minimal-time initiator is 4 states long, it takes several steps to produce the signal. We will refer to this stage as the launching of the signal, and the result is expressed in the following lemma.

Lemma 4.3 (The Launching Lemma) If $(p_{n;n}; 0)$ = B and $(p_{[1;n ¡ 1]}; 0)$ = L then $(p_{[n ¡ 2;n]}; 3)$ = AAC and $(p_{[1;n ¡ 3]}; 3)$ = L for n ¸ 5.

Proof: The transition LB? ! A implies $(p_n; 1)$ = A, and the transitions ?LL ! L, LLL ! L and LLB ! L mean $(p_{[1;n ¡ 1]}; 1)$ = L.

At the next step of the synchronization, the transition LA? ! G means $(p_n; 2)$ = G, LLA ! A implies $(p_{n ¡ 1}; 2)$ = A and ?LL ! L and LLL ! L mean $(p_{[1;n ¡ 2]}; 2)$ = L.

Finally, AG? ! C implies $(p_n; 3)$ = C, LAG ! A means $(p_{n ¡ 1}; 3)$ = A, LLA ! A means $(p_{n ¡ 2}; 3)$ = A and ?LL ! L and LLL ! L imply $(p_{[1;n ¡ 3]}; 3)$ = L. }

In order to prove that the remaining two stages of the synchronization work as required we need a technical lemma. This lemma formalizes the notion that the AACB signal leaves the automata behind it in the quiescent state L as it advances left.

Lemma 4.4 (The Quiescent Lemma) If $(p_{[j;j+1]}; t)$ = CB, $(p_{[j+2;n]}; t)$ = L and $(p_{n+1}; t)$ = ? then $(p_{[j+1;n]}; t + 1)$ = L for t; j; n 2 N; j < n, and n ¸ 5.

Proof: If j = n ¡ 1 then CB? ! L implies $(p_{j+1}; t + 1)$ = L which is the conclusion of the lemma.

If j = n ¡ 2 then the transition CBL ! L implies $(p_{j+1}; t + 1)$ = L and $(p_{j+2}; t + 1)$ = L because BL? ! L is de¯ned. This is what we needed to show.

If j = n ¡ 3 then CBL ! L means $(p_{j+1}; t + 1)$ = L, BLL ! L implies $(p_{j+2}; t + 1)$ = L and $(p_{j+3}; t + 1)$ = L because LL? ! L is de¯ned. Since j + 3 = n this is what the lemma requires.

Finally, if j < n ¡ 3 then CBL ! L implies $(p_{j+1}; t + 1)$ = L, $(p_{j+2}; t + 1)$ = L because BLL ! L, LLL ! L means $(p_{[j+3;n ¡ 1]}; t + 1)$ = L and $(p_n; t + 1)$ = L since LL? ! L is de¯ned. Again, this is what is required by the lemma. }

The next stage of the synchronization involves the migration of the signal from the left side of the array to the right. It is summarized in the following lemma.

Lemma 4.5 (The Migration Lemma) If $(p_{[1;n ¡ 3]}; 3)$ = L and $(p_{[n ¡ 2;n]}; 3)$ = AAC then $(p_{[n ¡ j+1;n ¡ j+4]}; j)$ = AACB, $(p_{[1;n ¡ j]}; j)$ = L and $(p_{[n ¡ j+5;n]}; j)$ = L for 4 · j · n ¡ 1 and n ¸ 5.

Proof: The proof is by induction on the time step j.

Base case: (j = 4) The transition LLA ! A implies $(p_{n ¡ 3}; 4)$ = $(p_{n ¡ j+1}; 4)$ = A, $(p_{n ¡ 2}; 4)$ = $(p_{n ¡ j+2}; 4)$ = A since LAA ! A is de¯ned, AAC ! C means $(p_{n ¡ 1}; 4)$ = $(p_{n ¡ j+3}; 4)$ = C and $(p_n; 4)$ = $(p_{n ¡ j+4}; 4)$ = B since AC? ! B.

It remains to show that the rest of the automata are quiescent. LLL ! L implies $(p_{[2;n ¡ 4]}; 4) = L$ and $(p_1; 4) = L$ because ?LL ! L.

Inductive hypothesis: Now assume that the lemma is true for some $j; 5 · j · n ¡ 2$. This means $(p_{[n ¡ j+1;n ¡ j+4]}; j) = AACB$, $(p_{[1;n ¡ j+1]}; j) = L$ and $(p_{[n ¡ j+5;n]}; j) = L$. Consider the time step $j + 1$.

$(p_{n ¡ (j+1)+1}; j + 1) = (p_{n ¡ j}; j + 1) = A$ because the transition LLA ! A is de¯ned. LLA ! A means $(p_{n ¡ (j+1)+2}; j + 1) = (p_{n ¡ j+1}; j + 1) = A$. The transition AAC ! C implies $(p_{n ¡ (j+1)+3}; j + 1) = (p_{n ¡ j+2}; j + 1) = C$. $(p_{n ¡ (j+1)+4}; j + 1) = (p_{n ¡ j+3}; j + 1) = B$ because ACB ! B is de¯ned.

Since $(p_{[n ¡ j+3;n ¡ j+4]}; j) = CB$ and $(p_{[n ¡ j+5;n]}; j) = L$, we can apply the Quiescent Lemma to conclude that $(p_{[n ¡ j+4;n]}; j + 1) = L$.

Finally, ?LL ! L implies $(p_1; j + 1) = L$ and $(p_{[2;n ¡ j ¡ 1]}; j + 1) = L$.

We have now shown that $(p_{[n ¡ j;n ¡ j+3]}; j + 1) = AACB$, $(p_{[1;n ¡ j ¡ 1]}; j + 1) = L$ and $(p_{[n ¡ j+4;n]}; j + 1) = L$, which is the conclusion of the lemma. }

The ¯nal stage of the synchronization is the collapse of the AACB signal, which leaves the array in the initial con¯guration necessary for Mazoyer's minimal-time automaton. This stage is expressed in the next lemma.

Lemma 4.6 (The Collapsing lemma) If $(p_{[1;5]}; n ¡ 1) = LAACB$ and $(p_{[6;n]}; n ¡ 1) = L$ for $n ¸ 5$ then $(p_1; n + 3) = G$ and $(p_{[2;n]}; n + 3) = L$.

Proof: ?LA ! B implies $(p_1; n) = B$, $(p_2; n) = A$ because LAA ! A is de¯ned, $(p_3; n) = C$ since AAC ! C and ACB ! B means $(p_4; n) = B$. Further, the Quiescent Lemma with $j = 4$ and $t = n ¡ 1$ applies here which shows that $(p_{[5;n]}; n) = L$.

Now consider the next time step. ?BA ! B gives $(p_1; n + 1) = B$, BAC ! C implies $(p_2; n + 1) = C$ and $(p_3; n + 1) = B$ because ACB ! B. The Quiescent Lemma with $j = 3$ and $t = n$ shows $(p_{[4;n]}; n + 1) = L$.

At the next time step of the simulation ?BC ! C implies $(p_1; n + 2) = C$, $(p_2; n + 2) = B$ because the transition BCB ! B is de¯ned and an application of the Quiescent Lemma with $j = 2$ and $t = n + 1$ shows $(p_{[3;n]}; n + 2) = L$.

Then ?CB ! G implies $(p_1; n+3) = G$ and another application of the Quiescent Lemma with $j = 1$ and $t = n + 2$ gives $(p_{[2;n]}; n + 3) = L$. This is the desired con¯guration. }

We can now put all the pieces together and prove Lemma 4.2.

Proof: The proof for $n < 5$ is by inspection. The simulations for these cases may be found at the end of this section.

Consider the case for $n ¸ 5$. By the Launching Lemma we know that $(p_{[n ¡ 2;n]}; 3) = AAC$ and $(p_{[1;n ¡ 3]}; 3) = L$. We next apply the Migration Lemma to get $(p_{[1;5]}; n ¡ 1) = LAACB$ $(p_2; n ¡ 1) = A$ and $(p_{[6;n]}; n ¡ 1) = L$. We then apply the Collapsing Lemma to show $(p_1; n + 3) = G$ and $(p_{[2;n]}; n + 3) = L$. This is precisely the conclusion of the lemma. }

4.3.3. The proof of the main theorem

We can now prove Theorem 1.1.

Proof: Consider the case where the initiator is located at the left endpoint. This means $(p_1; 0) = B$ and $(p_{[2;n]}; 0) = L$. By Lemma 4.1 we can conclude $(p_1; 1) = G$ and $(p_{[2;n]}; 1) = L$. This is the initial con¯guration for Mazoyer's 6-state minimal-time solution. Because the transitions for Mazoyer's automaton are a strict subset of the transitions for our automaton, Mazoyer's automaton begins a minimal-time synchronization at time step 1. This process requires an additional 2n ¡ 2 time steps. Thus, by the proof of correctness for the 6-state minimal-time automaton which may be found in Mazoyer's paper [6], the automata simultaneously, and for the ¯rst time, enter state F at time 2n ¡ 2 + 1 = 2n ¡ 1 as claimed.

If the initiator is located at the right endpoint, then the initial con¯guration for the array is $(p_{[1;n ¡ 1]}; 0)$ = L and $(p_n; 0)$ = B. We can apply Lemma 4.2 to get $(p_1; n + 3)$ = G and $(p_{[2;n]}; n + 3)$ = L. Again, as was argued above, this su±ces since Mazoyer's automaton then works exactly as it would have if the synchronization had begun at time n + 3. Thus ¯ring occurs at time n + 3 + 2n ¡ 2 = 3n + 1 as required. }

4.3.4. Runs for short arrays

As stated before, Lemma 4.2 is proved for the case of $n < 5$ by inspection. Tables 9, 10, and 11 give the runs for each of these possibilities.

4.4. The 7-state solution to the generalized problem

The 7-state automaton, like the 6-state solution to the original problem, is based on Mazoyer's 6-state minimal-time solution described in the previous section. It allows the ¯rst initiator to be located anywhere in the array and works by sending a signal from the ¯rst initiator back toward the left endpoint. When the signal reaches the end of the line, it transforms into the initiator for Mazoyer's 6-state solution, and the minimal-time synchronization begins.

In order to allow the ¯rst initiator to be located anywhere in the array, a new state D is added to Mazoyer's automaton. D is used both for the ¯rst initiator state and as the state for the signal that moves left. This results in the D migrating across the line of automata until it reaches the end. Once the D signal reaches the left endpoint, it puts the leftmost automaton in state G, the initiator state for Mazoyer's automaton. The synchronization is then completed by the 6-state minimal-time solution.

If the ¯rst initiator is located in position k of the array, it takes k ¡ 1 steps for the D signal to reach the left endpoint. At the next time step the D transforms into a G and the minimal-time synchronization begins. This means that the entire synchronization takes time 2n ¡ 2 + k time steps.

A sample run of the 7-state solution is given in Table 13. The proof of correctness for the 7-state automaton has been omitted but may be found elsewhere [14].

4.5. The 7-state transition function

Table 12 gives the transition function for the 7-state non-minimal-time automata. The state of an automaton at the next time step can be found by looking at the table corresponding to the automaton's present state. The state that the automaton should enter at the next time step is the one in the row and column corresponding to the states of its left and right neighbors respectively. A star is used to indicate the end of the array.

5. Conclusion

In this paper we presented improved bounds on the complexity of one-dimensional variants of the ¯ring synchronization problem. We gave a 9-state minimal-time automaton for a generalized version of the problem that has the fewest states used by any minimal-time generalized solution. We gave a 6-state non-minimal-time automaton for the original problem that allows the initiator to be located at either endpoint. We also presented a 7-state non-minimal-time solution to the generalized problem, the only known non-minimal-time solution for the generalized problem. We also have a proof that each of the non-minimal-time automata correctly solve the ¯ring synchronization problem.

This work narrows the gap between the upper and lower bounds on the number of states required for an automaton solving the ¯ring synchronization problem. The 6-state non-minimal-time automaton for the unrestricted original problem presented here uses 2 fewer states than the best known minimal-time automaton solving the same problem and uses only 3 states more than the lower bound on non-minimal-time solutions to the problem.

Progress is also made in the generalized case. The lower bound for minimal-time solutions to the original problem applies to the generalized problem. We give a 9-state minimal-time solution and a 7-state non-minimal time automaton for the generalized problem. In this case, the minimal-time solution uses 5 states more than the lower bound and the non-minimal-time solution uses only 4 states more than the lower bound on non-minimal-time automata.

6. Open problems

It is unknown whether allowing automata to take more time to synchronize, that is, producing a non-minimal-time solution, requires fewer states than producing a minimal-time solution. Evidence that this may be the case can be seen in our work, as our non-minimal-time solutions require fewer states than the best-known minimal-time solutions for the same problems. One might expect optimal non-minimal-time solutions to use even fewer states than our constructions, as our automata are built on top of minimal-time solutions to restricted versions of the problem. It would be interesting to formally investigate this idea.

Another open problem is to further improve Szwerinski's solution for d-dimensional arrays. Our 9-state automaton improves Szwerinski's result by reducing the number of states for the one-dimensional solution on which it is based. It is unknown whether the number of states his algorithm uses can be reduced further.

7. Acknowledgments

We are indebted to Sophie Laplante for many productive discussions about these results and for her suggestions on improvements to drafts of the work. We would also like to thank Andr¶ Berthiaume and Marcus SchÄfer for their comments on earlier drafts of this paper.

REFERENCES

1. R. Balzer. An 8-state minimal time solution to the ¯ring squad synchronization problem. Information and Control, 10:22{42, 1967.
2. K. Culik. Variations of the Firing Squad Problem and Applications. Information Processing Letters, 30:153{157, 1989.
3. S. Even, A. Litman, and P. Winkler. Computing with snakes in directed networks of automata. Journal of Algorithms, 24(1):158{170, 1997.
4. E. Goto. A minimum time solution of the ¯ring squad problem. Course Notes for Applied Mathematics 298, Harvard University, 1962.
5. K. Kobayashi. The ¯ring squad synchronization problem for a class of polyautomata networks. Journal of Computer and System Sciences, 17(3):300{318, 1978.
6. J. Mazoyer. A six-state minimal time solution to the ¯ring synchronization problem. Theoretical Computer Science, 50:183{238, 1987.
7. M. Minsky. Computation: Finite and In¯nite Machines. Prentice Hall, 1972.
8. F. R. Moore and G. G. Langdon. A generalized ¯ring squad problem. Information and Control, 12:212{220, 1968.
9. E. F. Moore. The ¯ring squad synchronization problem. In Sequential Machines - Selected Papers, pages 213{214. Addison-Wesley, 1964.
10. Y. Nishitani and N. Honda. The ¯ring squad synchronization problem for graphs. Theoretical Computer Science, 14(1):39{61, 1981.
11. R. Ostrovsky and D. Wilkerson. Faster computation on directed networks of automata. In Proceedings of 14th Annual ACM Symposium on Principles of Distributed Computing, pages 38{46, 1995.
12. P. Sanders. Massively parallel search for transition-tables of polyautomata. In C. Jesshope, V. Jossifov, and W. Wilhelmi, editors, Proceedings of the VI International Workshop on Parallel Processing by Cellular Automata and Arrays, pages 99{108. Akademie, 1994.
13. A. Settle and J. Simon. Improved bounds for the distributed ¯ring synchronization problem. In SIROCCO 5: Proceedings of the 5th International Colloquium on Structural Information and Communication Complexity, Amal¯, Italy, pp. 66-81, Carleton Scienti¯c, 1998.
14. A. Settle. New bounds for the distributed ¯ring synchronization problem. Ph.D. thesis, Department of Computer Science, University of Chicago, 1999.
15. H. Szwerinski. Time-optimal solution of the ¯ring-squad-synchronization-problem for n-dimensional rectangles with the general at an arbitrary position. Theoretical Computer Science, 19:305{320, 1982.
16. V. I. Varshavsky, V. B. Marakhovsky, and V. A. Peschansky. Synchronization of interacting automata. Mathematical Systems Theory, 4:212{230, 1970.
17. A. Waksman. An optimum solution to the ¯ring squad synchronization problem. Information and Control, 9:66{78, 1966.
18. J. B. Yunes. Seven-state solutions to the ¯ring squad synchronization problem. Theoretical Computer Science, 127:313{332, 1994.

Table 4
A run of the 8-state original automaton contained within the 9-state solution. In the simulation n = 17 and the initiator is at the left. To better illustrate the mechanics of the solution, quiescent automata (automata in state Z) are not shown.

0 :	G																
1 :	G	A															
2 :	G	R	A														
3 :	G	**B**	P	A													
4 :	G	**B**	P	R	A												
5 :	G	**B**	R	Q	P	A											
6 :	G	**B**	**B**		P	R	A										
7 :	G	**B**	**B**		R	Q	P	A									
8 :	G	**B**	**B**	R	Q		P	R	A								
9 :	G	**B**	P	**B**			R	Q	P	A							
10 :	G	**B**	P	**B**		R	Q		P	R	A						
11 :	G	**B**	P	**B**	R	Q			R	Q	P	A					
12 :	G	**B**	P	Q	**B**			R	Q		P	R	A				
13 :	G	**B**	R	Q	**B**		R	Q			R	Q	P	A			
14 :	G	**B**	**B**		**B**	R	Q			R	Q		P	R	A		
15 :	G	**B**	**B**		P	**B**			R	Q			R	Q	P	A	
16 :	G	**B**	**B**		P	**B**		R	Q			R	Q		P	R	G
17 :	G	**B**	**B**		P	**B**	R	Q			R	Q			R	A	G
18 :	G	**B**	**B**		P	Q	**B**			R	Q			R	A	R	G
19 :	G	**B**	**B**		R	Q	**B**		R	Q			R	A	P	**B**	G
20 :	G	**B**	**B**	R	Q		**B**	R	Q			R	A	R	P	**B**	G
21 :	G	**B**	P	**B**			P	**B**			R	A	P	Q	R	**B**	G
22 :	G	**B**	P	**B**			P	**B**		R	A	R	P		**B**	**B**	G
23 :	G	**B**	P	**B**			P	**B**	R	A	P	Q	R		**B**	**B**	G
24 :	G	**B**	P	**B**			P	Q	G	R	P		Q	R	**B**	**B**	G
25 :	G	**B**	P	**B**			R	A	G	A	R			**B**	P	**B**	G
26 :	G	**B**	P	**B**		R	A	R	G	R	A	R		**B**	P	**B**	G
27 :	G	**B**	P	**B**	R	A	P	**B**	G	**B**	P	A	R	**B**	P	**B**	G
28 :	G	**B**	P	Q	G	R	P	**B**	G	**B**	P	R	G	Q	P	**B**	G
29 :	G	**B**	R	A	G	A	R	**B**	G	**B**	R	A	G	A	R	**B**	G
30 :	G	**B**	G	R	G	R	G	**B**	G	**B**	G	R	G	R	G	**B**	G
31 :	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G
32 :	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

Table 5
A run of the 9-state automaton for n = 18 with initiator in position 5. As in Table 4, quiescent automata are not shown.

0 :					**D**													
1 :				A	R	A												
2 :			A	P	Q	P	A											
3 :		A	R	P	Q	P	R	A										
4 :	G	P	Q	R	Q	R	Q	P	A									
5 :	A	A		Q	**D**	Q		P	R	A								
6 :	G		A		Q			R	Q	P	A							
7 :	G	R		A	Q		R	Q		P	R	A						
8 :	G	**B**	P	R	**D**	R	Q			R	Q	P	A					
9 :	G	**B**	R	R	D	**B**			R	Q		P	R	A				
10 :	G	**B**	**B**		P	**D**		R	Q			R	Q	P	A			
11 :	G	**B**	**B**		P	**D**	R	Q			R	Q		P	R	A		
12 :	G	**B**	**B**		P	**D**	**B**			R	Q			R	Q	P	A	
13 :	G	**B**	**B**		P	R	**D**		R	Q			R	Q		P	R	G
14 :	G	**B**	**B**		R	R	**D**	R	Q			R	Q			R	A	G
15 :	G	**B**	**B**	R	Q		**D**	**B**			R	Q			R	A	R	G
16 :	G	**B**	P	**B**			P	**D**		R	Q			R	A	P	**B**	G
17 :	G	**B**	P	**B**			P	**D**	R	Q			R	A	R	P	**B**	G
18 :	G	**B**	P	**B**			P	**D**	**B**			R	A	P	Q	R	**B**	G
19 :	G	**B**	P	**B**			P	R	**D**		R	A	R	P		**B**	**B**	G
20 :	G	**B**	P	**B**			R	R	**D**	R	A	P	Q	R		**B**	**B**	G
21 :	G	**B**	P	**B**		R	Q		**D**	**D**	R	P		Q	R	**B**	**B**	G
22 :	G	**B**	P	**B**	R	Q			G	G	R	R			**B**	P	**B**	G
23 :	G	**B**	P	Q	**B**			A	G	G	A	Q	R		**B**	P	**B**	G
24 :	G	**B**	R	Q	**B**		A	R	G	G	R	A	Q	R	**B**	P	**B**	G
25 :	G	**B**	**B**		**B**	A	P	**B**	G	G	**B**	P	A	**B**	Q	P	**B**	G
26 :	G	**B**	**B**		G	R	P	**B**	G	G	**B**	P	R	G	Q	R	**B**	G
27 :	G	**B**	**B**	A	G	A	R	**B**	G	G	**B**	R	A	G	A	**B**	**B**	G
28 :	G	**B**	G	R	G	R	G	**B**	G	G	**B**	G	R	G	R	G	**B**	G
29 :	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G
30 :	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

Table 6
The transition function for the 9-state automaton

neighbors'	present state								neighbors'	present state							
states	Z	A	B	D	R	P	Q	G	states	Z	A	B	D	R	P	Q	G
Z{Z	Z	Z	B	R			Q	G	B{Q	Z			P	B	R	P	
Z{A	A	Z	G		A		D		B{G	A	R	B			A	A	G
Z{B	Z	G	B	P		P			B{?								G
Z{D	A	R	D	G		P	Z		D{R	R	Q		G	Z		Z	
Z{R	R	P	P	D	Q	R	Z	G	D{P				G	R		Q	
Z{P	Z	R	B	D	Q				D{Q	Z			G	B	P		
Z{Q	Z	R	B	D	Q	R			D{G	B		B		B			
Z{G	A	R			B	A		G	D{?	G							
Z{?	Z			G				G	R{R		P		D	Z		D	G
A{A		G		G	Q		G	G	R{P		R	Q	D	Q		Z	A
A{B	A	G	G		G	P			R{Q	R	P		D		R	Z	G
A{D					D				R{G		R	B		A		A	G
A{R	P			G			A		R{?								G
A{P	R		G	G	Q		D		P{P							Q	A
A{Q	A	Z	G	B	A	P			P{Q	Z	R					Z	
A{G	R	G	G		B			G	P{G			B		A		A	A
A{?	G	G						G	P{?								A
B{B	Z			P		P	P	G	Q{Q				Q	Q			A
B{D	Z			G		P			Q{G	A	R	B			A	A	G
B{R	R	P	P	P	B	R	Z	G	G{G	G		G		G			F
B{P	Z	R		R			Q		G{?	G							F

Table 7
A run of the 6-state automaton for n = 10, initiator at the left endpoint. Quiescent automata (automata in state L) are not shown.

```
0  :                   B
1  :                   A
2  :                 A G
3  :               A A C
4  :             A A C B
5  :           A A C B
6  :         A A C B
7  :       A A C B
8  :     A A C B
9  :   A A C B
10 : B A C B
11 : B C B
12 : C B
13 : G
14 : A C
15 : G B A
16 : G C G G
17 : G B A B C
18 : G C G   C A
19 : G B A   A A G
20 : G C G   A B B C
21 : G B A     B C C A
22 : G C G G     C A A C
23 : G B A B C   A A C B
24 : G C G   C   A C B
25 : G B A   C   G B
26 : G C G   C G G C
27 : G B A   G A G B A
28 : G C G C G C G C G C
29 : G B G B G B G B G B
30 : G G G G G G G G G G
31 : F F F F F F F F F F
```

Table 8
The transition function for the 6-state automaton

A	A	B	C	G	L	?
A	A	B	C		A	F
B		G	C	C	G	C
C	A				A	
G			C	C		C
L	A	L	G	A		G
?	F		G			

B	A	B	C	G	L	?
A	B	B	L		G	
B	A	B	C	B	G	
C	A			L	L	L
G	C		B	G	C	G
L	G	B	L	B		A
?	B		C	L	G	

C	A	B	C	G	L	?
A		B		B	B	B
B		B	C	G	C	G
C	A	B	C	B	C	
G		B		B	B	B
L	A	G	C	G	C	
?		G				

L	A	B	C	G	L	?
A	L	L	L	C	G	C
B	L	L	L	L	L	L
C	L	L	L	G	A	G
G	L	L	L	A	C	A
L	A	L	L	L	L	L
?	B	L		B	L	

G	A	B	C	G	L	?
A		G	G		B	C
B		G	G	G	B	G
C		G	G	A	A	A
G		G	G	F	B	F
L	G	G	G			L
?		G	G	F	A	

Table 9
A run of 6-state automaton when n = 2. The initial con¯guration for Mazoyer's automaton is produced at time step n + 3 = 5 and ¯ring occurs at time 3n + 1 = 7. For ease of understanding, quiescent automata are not shown.

```
0 :   B
1 :   A
2 : B G
3 :   G
4 : B
5 : G
6 : A A
7 : F F
```

Table 10
A run of the 6-state automaton when n = 3. The initial con¯guration is produced at time step n + 3 = 6 and ¯ring occurs at time 3n + 1 = 10. Quiescent automata are not shown.

```
0  :     B
1  :     A
2  :   A G
3  : B A C
4  : B C B
5  : C B
6  : G
7  : A C
8  : G B G
9  : G G G
10 : F F F
```

Table 11
A run of the 6-state automaton when n = 4. The initial con¯guration is produced at time n + 3 = 7 and ¯ring occurs at time 3n + 1 = 13. Quiescent automata are not shown.

```
0  :       B
1  :       A
2  :     A G
3  :   A A C
4  : B A C B
5  : B C B
6  : C B
7  : G
8  : A C
9  : G B A
10 : G C G C
11 : G B G B
12 : G G G G
13 : F F F F
```

Table 12
The transition function for the 7-state automaton

A	A	B	C	G	L	?
A	A	B	C	B	A	F
B		G	C	C	G	C
C	A				A	
G			C	C		C
L	A	L	G			
?	F		G			

B	A	B	C	G	L	?
A	B	B	L		G	
B	A	B	C	B	G	
C	A			L	L	L
G	C		B	G	C	G
L	G	B	L	B		

C	A	B	C	G	L	?
A		B		B	B	B
B			C	G	C	G
C	A	B	C	B	C	
G		B		B	B	B
L	A	G	C	G	C	

L	A	B	C	D	G	L	?
A	L	L	L		C	G	C
B	L	L	L		L	L	L
C	L	L	L		G	A	G
D						L	L
G	L	L	L		A	C	A
L		L	L	D	L	L	L
?				D		L	

G	A	B	C	G	L	?
A		G	G		B	
B		G	G	G	B	G
C		G	G	A		
G		G	G	F	B	F
L	G	G	G			
?		G	G	F	A	

D	L	?
L	L	L
?	G	

Table 13
A run of the 7-state automaton for n = 12, the initiator in position 9. Quiescent automata (those in state L) are not shown.

0 :									D			
1 :								D				
2 :							D					
3 :						D						
4 :					D							
5 :				D								
6 :			D									
7 :		D										
8 :	D											
9 :	G											
10 :	A	C										
11 :	G	B	A									
12 :	G	C	G	G								
13 :	G	B	A	B	C							
14 :	G	C	G		C	A						
15 :	G	B	A		A	A	G					
16 :	G	C	G		A	B	B	C				
17 :	G	B	A			B	C	C	A			
18 :	G	C	G	G			C	A	A	G		
19 :	G	B	A	B	C		A	A	B	B	C	
20 :	G	C	G		C		A	B	B	C	C	G
21 :	G	B	A		C			B	C	C	B	A
22 :	G	C	G		C	A			C	B	A	C
23 :	G	B	A		A	A	G		G	A	C	B
24 :	G	C	G		A	B	B	A	G	C	B	
25 :	G	B	A			B	A	C	G	B		
26 :	G	C	G	G		G	C	B	G	C		
27 :	G	B	A	B	A	G	B		G	B	A	
28 :	G	C	G	B	C	G	C		G	C	G	C
29 :	G	B	G	B	G	G	B	G	G	B	G	B
30 :	G	G	G	G	G	G	G	G	G	G	G	G
31 :	F	F	F	F	F	F	F	F	F	F	F	F