Reliable Node Placement in Wireless Sensor Networks Using Cellular Automata

Sami Torbey and Selim G. Akl

School of Computing, Queen's University, Kingston, Ontario, Canada

Abstract. Wireless sensor networks are often used to provide critical measurements in unattended harsh environments. They should be designed to adequately monitor their surroundings while being resilient to environmental changes. Appropriate sensor node placement greatly influences their capability to perform this task. Cellular automata have properties very similar to those of wireless sensor networks. In this paper, we present a sensor node placement algorithm that runs on a cellular automaton and achieves adequate coverage, connectivity and sparsity while being resilient to changing environmental conditions.

1 Background and Motivation

1.1 Wireless Sensor Networks

Wireless sensor networks are systems consisting of a large number of miniaturized sensor nodes deployed to operate autonomously in unattended (and frequently harsh) environments. They are often heterogenous, measuring different properties of their surroundings and sending the collected data to an access point either directly or through multi-hop paths. Wireless sensor networks have many applications including forest monitoring, disaster management, space exploration, factory automation, border protection and battlefield surveillance [1].

1.2 Cellular Automata

Generally speaking, a cellular automaton is a theoretical system consisting of a large number of simple processing elements (cells) locally interacting among themselves. The emphasis here is on the simplicity of individual elements, their connectivity and the absence of global control.

Cellular automata are commonly seen as consisting of a "cellular space" and a set of "transition rules" [2]. The cellular space is a set of cells, often shown in a given geometric configuration (usually a grid). Each one of these cells is a finite state machine in one of a constant number of possible states, evolving synchronously at discrete time units in parallel with all of the other cells in the system. The state of each cell at the next time unit is determined by a set of transition rules, which are functions of the current states of cells in its neighbourhood (a finite set of cells connected to it, usually in its geometric vicinity). This neighbourhood often also contains the cell itself.

J. Durand-Lose and N. Jonoska (Eds.): UCNC 2012, LNCS 7445, pp. 210–221, 2012.

In this paper, we use a finite grid configuration where the number of cells is finite but still large enough to clearly display complex behaviour. In such configurations, the Moore radius is often used as a short-hand method of specifying the neighbourhood. A Moore neighbourhood of radius r consists of the cell itself and every cell that is r or less cells apart from it in any direction: vertical, horizontal and diagonal (Figure 1).

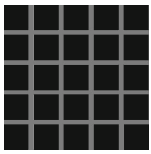

Fig. 1. Moore neighbourhood of radius 2

1.3 Node Placement in Wireless Sensor Networks

Node placement is an important wireless sensor network research area whose aim is to optimize one or more design variables such as coverage, connectivity and energy consumption through the appropriate positioning of sensor nodes. While the communication methods and protocols of the sensors can have an important impact on these variables, they are only considered after the node positions have been determined. We have opted to deal exclusively with the node placement problem in this paper, keeping in mind that any other constraints or protocols can be added to the resulting system later on. Therefore, we only consider wireless sensor networks where the owner has some control over the sensor positions, unlike, for example, the case of sensors strapped around animals or placed unanchored in the ocean. Specifically, we focus on sensors that have some ability to reposition themselves (e.g. sensors placed on robots). There are multiple problems related to the positioning of sensor nodes in wireless sensor networks:

- Coverage is a Quality of Service (QoS) problem whose goal is to minimize the part of the desired monitored area that is not covered by any sensor node. In other words, the coverage problem is optimally solved when every part of the area that needs to be monitored is covered by at least one sensor node.
- Connectivity is another QoS problem aiming to make sure that every sensor node is connected either directly or indirectly (through other sensor nodes) to an access point, since the information collected by the sensors is useless if it cannot be transmitted back
- Sensors are generally battery-operated; energy consumption is therefore a key performance metric because it determines the lifetime (and replacement cycle) of the sensors. Energy consumption should be minimized in order to maximize the lifetime of the system given a number of nodes.

Several positioning algorithms used to achieve desired node placement in wireless sensor networks are surveyed in [1,3]. They are divided into two categories: deterministic and random. Although deterministic methods provide better theoretical results, random positioning algorithms are used more often because the inaccessibility of the monitored location and the generally large number of sensors make them much more practical. Most of these algorithms aim to optimize only one of the three key performance metrics (coverage, connectivity and energy consumption), although some of them secondarily consider one other performance metric.

Younis and Akkaya also describe a few repositioning algorithms that determine where to move one or more sensors after the initial deployment in order to achieve better coverage, connectivity or energy consumption given some changes in the environmental conditions [1]. However, all of the described algorithms, whether for positioning or repositioning, are global (the decisions are made centrally after information from every sensor is received) and rigid (they assume strict definitions of performance and provide one optimal position given the current conditions without regard to the fragility of these conditions).

Unlike most of the algorithms described in [1,3], our aim is to optimize all three key performance metrics: coverage, connectivity and energy consumption. Energy consumption can be greatly reduced by having simple components with exclusively local decisions. It can also be seen as inversely proportional to node sparsity (if the overall energy of the system is considered) [3]; this means that our goal should be to achieve the best balance between maximum coverage and connectivity and a minimal number of nodes.

We also aim to provide a flexible probabilistic approach that operates based on local conditions, provides redundancy and speedy recovery after changes in the system conditions, and does not differentiate between initial deployment and repositioning. As such, the algorithm presented here leads to simpler, more resilient and more autonomous networks. On the flip side, it requires more nodes than optimal deterministic placement (in order to achieve the desired redundancy) and more sensor movement than what is strictly necessary, which, if not properly managed, can significantly reduce the lifetime of the network by depleting the nodes' energy.

1.4 Wireless Sensor Network Simulation Using Cellular Automata

Wireless sensor networks are often deployed in harsh environments to perform critical monitoring tasks. Yet most of the node positioning algorithms in the literature do not take that into consideration, and assume that each of the sensors will live for its prescribed lifetime and maintain its original position and neighbourhood throughout.

The simplicity and locality of nodes in sensor networks bears a striking resemblance to that of cells in cellular automata. Our implementation runs on a cellular automaton, simulating a wireless sensor network node placement algorithm that remains active throughout the network's lifetime, constantly adjusting the positioning in real-time to meet the changing requirements. We believe that cellular

automata are ideally suited for this task given the locality of their interaction (any global positioning issue or change would be difficult to communicate in time) and the simplicity of their rules (sensor nodes do not have enormous processing power). The main goal of the proposed algorithm is to improve node coverage and connectivity while maintaining sparsity. It is clear that coverage and connectivity can be optimized by deploying a large number of sensor nodes; however, we aim to achieve full (or almost full) coverage and connectivity throughout the monitored area with as few sensors as possible. Therefore, our algorithm optimizes all three performance metrics, which is uncommon among wireless sensor network node placement algorithms. Moreover, the proposed algorithm tackles three of the four open problems in the area [3], as shown in Section 4.4.

2 System Description

R_S and R_C are two widely-used characteristics of nodes in wireless sensor networks. R_S is the sensing radius; it defines the maximum distance that a point can be from a sensor while still being covered by that sensor. R_C is the communication radius, which is the maximum distance two sensors can be from each other while still being able to communicate. We are only considering $R_S \leq R_C \leq 2R_S$. This makes sense according to our objectives: if $R_C < R_S$ then R_S does not need to be as large as it is since the necessity of connectivity guarantees that there are several sensors covering the same area (although some redundancy is desirable, too much of it contradicts the sparsity requirement we set earlier). On the other hand, if $R_C > 2R_S$ then R_C could be reduced because the need for coverage ensures that sensors are within $2R_S$ of each other.

We simulate the wireless sensor network on a two-dimensional cellular automaton: space is therefore discretized. This is not perceived as a limitation since many of the existing mathemcatical models of sensor networks also assume discretized space. In our model, a cell in state 0 (white) does not contain a sensor - but still needs to be monitored by a sensor. A cell in state 1 (grey) is an access point and a cell in state 2 (black) is a sensor node.

For our purposes, both access points and sensor nodes can monitor their environment and they have the same sensing and communication radii. The difference between them is that access points are capable of communicating directly with the external observer (through wired or powerful long-range wireless connections); this means that access points need to be wired somehow whether for connectivity or power. Therefore, access point positions are fixed while sensor nodes are mobile.

The mobility of the sensor nodes classifies this system as a dynamic positioning system, as opposed to a static positioning system where the sensors are assigned fixed positions upon deployment. Sensing and communication radii are assimilated to the Moore neighbourhood radius of the cellular automaton; a direct implication of this fact is that sensing and communication radii are of constant size relatively to the size of the system. Since $R_C \leq 2R_S$, the transition

rules need only focus on the communication radius; this is the case because under this restriction, the fact that two nodes can communicate means that they are collectively fully monitoring the area between them.

2.1 Transition Rules

The inherent design of cellular automata makes the direct simulation of particle movement difficult, particularly when probabilistic rules are employed and it is not possible for one cell to guess what the intentions of the neighbouring cells are; such movement thus needs to be described as the disappearance of a particle from one location and its appearance in another. We simulate this requirement with even-odd rules (as previously described in [4]), where the transition rules for even cycles are different from those of odd cycles. Therefore, the automaton can be seen as running according to cycles of two steps: nodes decide where they want to move in the first step, while they actually carry out the movement in the second step. As mentioned earlier, the goal of that movement is to maximize coverage, communication and sparsity.

Even Cycles. The first step is when the nodes announce their intention to move. The decision to move is taken probabilistically based on the number of other nodes in a sensor's neighbourhood weighed by the distance of these nodes from the sensor. For example, for a neighbourhood of size 4 every sensor node calculates a number k as follows:

$$k = 8N_1 + 4N_2 + 2N_3 + 1N_4$$

In this formula, N_1 is the number of nodes within a distance of 1 cell from the sensor in question, N_2 the number of nodes within a distance of 2 cells, etc.

k is then used to determine the probability of movement:

- For $k = 0$ or $k \geq 8$ the node has a 50% chance of moving
- For $5 \leq k \leq 7$ the node has a 20% chance of moving
- For $1 \leq k \leq 4$ the node has a 5% chance of moving

These numbers are not cast in stone; they are only meant to give a greater incentive for a node to move when it has too few or too many neighbours. The assumption is that a node that has no neighbours is isolated (incapable of reaching an access point either directly or indirectly) and is therefore forced to move for the sake of connectivity. On the other hand, a node that has too many neighbours is not needed at its current location (while being probably needed somewhere else) and is hence encouraged to move for the sake of sparsity. Note that chances of moving are kept at or below 50% to provide some stability to the system, and above 0% to maintain some fault tolerance allowing the system to correct itself (for example, $k = 4$ may mean that a node only has one neighbour that is two cells away from it, which is often not an ideal scenario because it needs four neighbours to ensure that its entire surrounding area is being covered).

Our initial design suggested that a node should be strongly encouraged to move even if it has some neighbours, as long as it does not have enough of them. However, we changed the rules after realizing that such movement can cause it to lose its connectivity, which is more important than coverage or sparsity (a node that cannot transmit its results back is completely useless). In the current design, a node only moves if it has no neighbours or too many neighbours. The constants in the formula worked very well in our testing, both when we had enough nodes to cover the entire area and when we had less than that. Testing with less nodes than needed is important because it really underlines the difference between a cell having too few neighbours and a cell having too many neighbours. This is not the case when there are enough nodes because a cell having too few neighbours implies that another cell has too many neighbours, and therefore, movement is created when a cell has too few neighbours regardless of whether this is expressly stipulated or not. The constants were particularly chosen to discourage movement when a cell has only one neighbour in order to maintain connectivity (unless that one neighbour is right next to the cell, in which case connectivity is maintained even in the extreme case where both the cell and its neighbour move in opposite directions). They were also chosen to encourage a cell to have many neighbours (ideally four) that are far from it, rather than a few neighbours close to it. In short, the constants' goal is to encourage sparsity whenever possible as long as connectivity is maintained.

Note that the probabilistic element in the rules described above can be embedded within the cellular automaton as shown in [4] by adding to each cell a few separate state bits implementing transition rules from a Wolfram Class 3 automaton such as Rule 30 [5].

The question that remains to be answered is: "where does a node move?" Once it has taken the decision to move, a sensor chooses at random one of its eight immediate neighbouring cells while following two conditions:

- The chosen neighbouring cell must be empty
- The chosen neighbouring cell must also be outside the reach of all other nodes (conflicts are resolved by simply preventing them from occuring in the first place)

The sensor then points to the cell it has randomly chosen by changing its state to a number from 3 to 10 reflecting one of the eight possible directions. If on the other hand it decides not to move, it remains in state 2.

Odd Cycles. The rules for odd cycles are very simple:

- A cell in state 1 or 2 does not change its state
- A cell in state 0 changes its state to 2 only if there is a cell in its immediate neighbourhood pointing in its direction (having the right state number greater than or equal to 3)
- A cell in any of the states 3 to 10 changes its state to 0

Thus, the odd cycle executes the moves designated in the previous even cycle. Then the next even cycle designates new moves, which are executed by its subsequent odd cycle, and so on.

3 Testing

Ideal placement (with a minimal number of sensors) is possible given R_C and R_S. The problem with such placement is that it is static and extremely vulnerable to any minor position change or sensor failure. However, we can use the ideal placement as a benchmark against which to compare our placement algorithm. Taking $R_C = R_S = 3$, the ideal placement (shown in Figure 2) for a cellular automaton with periodic boundaries requires one sensor for every 18 cells.

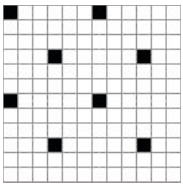

Fig. 2. Ideal static placement of sensors for a small cellular automaton with periodic boundaries and $R_C = R_S = 3$

Now that we have this benchmark, testing is straightforward: all we need to do is compare the number of sensors needed to achieve different rates of coverage and connectivity (on average, since the system is constantly moving) using our algorithm to the number of sensors needed in the ideal placement. Then for each application, one can choose the desired trade-off between coverage and connectivity on one hand, and sparsity on the other, based on the minimum acceptable coverage and connectivity ratios.

Starting with $\frac{10{,}000}{18} \simeq 556$ sensors for an ideal placement, we performed several tests with varying parameters on a cellular automaton with 10,000 cells. The average results of these tests are given in Table 1.

Table 1. Performance of the proposed algorithm given various parameters

number of sensors ($\frac{actual}{minimum}$)	R_S	R_C	uncovered area	disconnected sensors
1.1	3	3	2%	3.1%
1.1	3	4	2%	0.2%
1.3	3	4	1%	0%
1.5	3	4	0.3%	0%
2.0	3	4	0.05%	0%

From these tests, we see that deploying 50% more sensors than the minimum yields excellent results with no disconnections and almost complete coverage (Figure 3). Note that in our system the area covered changes between cycles (the numbers displayed in Table 1 are average values). Thus, the small areas missed by sensors in one cycle are covered in subsequent cycles, unlike with static placement algorithms. We also notice that a communication radius slightly larger than the sensing radius dramatically reduces disconnection rates. However, a significantly larger communication radius is not necessary since the large number of deployed sensors (for coverage purposes) would prevent it from having any effect.

4 Enhancements and Conclusions

Although our algorithm as presented achieves the design objectives we set earlier (coverage, connectivity and sparsity), we still have concerns regarding its energy consumption, and consequently the resulting wireless sensor network's lifetime. In this section, we attempt to mitigate these concerns and treat some of the open problems presented earlier.

4.1 Constant Movement

In our scheme, every node in the system moves at least 5% of the time, and up to 50% of the time. This depletes the nodes' energy very quickly given that movement uses significant power (usually more than sensing and connectivity). However, we cannot let the nodes remain in their positions once they have found a good balance between coverage, connectivity and sparsity because the environment around them (moving neighbours, dying neighbours, etc.) often disturbs this balance. Moreover the balance assumption may not even be correct to begin with because of the limitations of the movement formula discussed earlier. On the other hand, we also cannot encourage the nodes to move indefinitely while looking for that elusive balance.

Therefore, we propose using simulated annealing to regulate the probability of movement. Simulated annealing is an approach inspired by metallurgy and designed to find good approximations of global solutions for optimization problems. In our case, simulated annealing involves slowly decreasing the probability of movement from the up to 50% probabilities given above to much lower values (but never nil in order to maintain some reliability and fault tolerance in the system). This process allows the system to look for a desirable position while the probability is high and settle in it as it gets lower.

However, we cannot allow the system to remain indefinitely in a low probability of movement state, given that there could be major environmental changes that require substantial movement to be overcome. Therefore, we briskly raise the probabilities to their original levels at constant time intervals, only to slowly decrease them again and allow the system to settle in its new state. This process is shown in Figure 4.

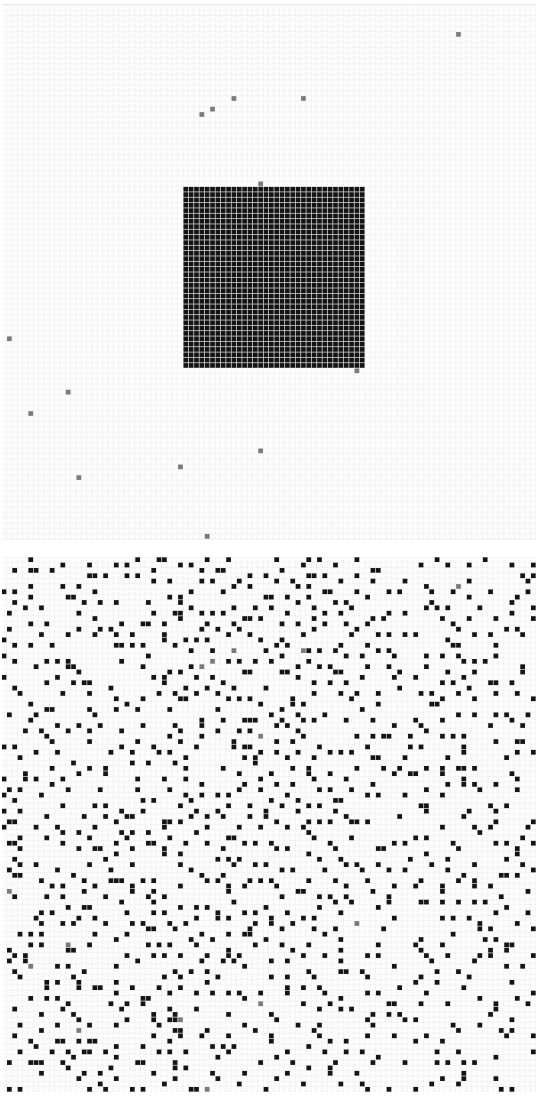

Fig. 3. Initial and desired states of an automaton with 10,000 cells and 729 sensors. Note that despite the simplicity of the rules, the emergent behaviour is clear: it strives for sparsity while maintaining coverage and connectivity. Regardless of the initial state, the transition rules reward good coverage and connectivity, and punish the contrary, ensuring that the desired state (which looks similar to the example above) is always reached.

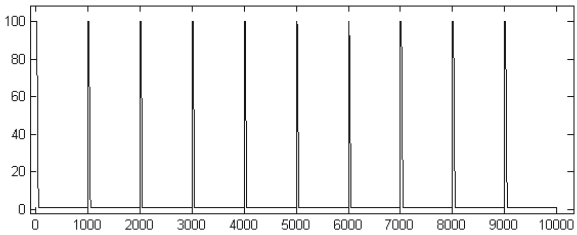

Fig. 4. Periodic simulated annealing process with 10 periods in 10,000 cycles. Each period shown here contains 15 cycles at the maximum probability (100% of the original values), 35 cycles of decreasing probability and 950 cycles at the minimum probability (1% of the original values).

4.2 Node Lifetime

Regardless of the placement algorithm, nodes will eventually have to deplete their energy and die. This causes a progressive decrease in coverage and connectivity until our fault tolerant algorithm can no longer adapt and they fall below acceptable levels. Therefore, a node replacement strategy is essential, unless the system is no longer needed beyond its lifetime (which is a very rare case).

We propose a node replacement strategy that matches the simulated annealing strategy mentioned in the previous section. It assumes that the maximum node lifetime is known, and that nodes die randomly sometime before their maximum lifetime. Based on this assumption, it replaces subsets of the nodes progressively throughout that lifetime in order to maintain a minimum acceptable number of nodes at all times. The nodes are replaced right before the probability of movement is increased, so that the increase in probability helps them find the best locations in the system. They can added anywhere in the system, although ideally they would be randomly spread.

Figure 5 shows a random decay of 850 nodes with a 10,000 cycle lifetime and replenishment every 1,000 cycles. If the nodes are inexpensive and unintrusive, they can simply die in-situ when their batteries are depleted; otherwise, they can be instructed to move to a charging station when their battery levels reach critical values.

4.3 Testing the Enhancements

We modified our system to consider the enhancements presented above. We started with 850 sensors to cover the cellular automaton containing 10,000 cells and gave them full freedom to move for 2,000 cycles in order to establish an initial position. We then considered a node lifetime of 10,000 cycles, during which all 850 nodes die progressively at random times. 85 nodes appeared at random locations (replenishment with random positioning) every 1,000 cycles. Therefore on average, the cellular automaton contained between 765 (1.38 times the minimum) and 850 (1.53 times the minimum) sensors. In addition, we started

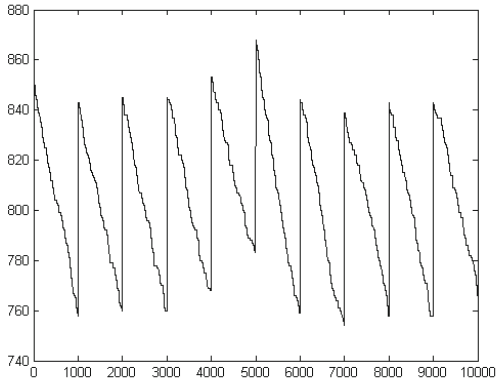

Fig. 5. Random decay of 850 nodes with a lifetime of 10,000 cycles. One tenth of the total nodes (85 in this case) are added at every one tenth of the lifetime (1,000 cycles).

with probabilities of movement of 50%, 20% and 5% for 15 cycles, and smoothly dropped them down respectively to 0.5%, 0.2% and 0.05% during 35 cycles, then kept them at these values for 950 cycles before raising them back up to 50%, 20% and 5% when 85 nodes are added. We then measured the average coverage and connectivity rates over 10,000 cycles, which were respectively about 99.25% and more than 99.99%. These are promising values, and given that under this scheme the average node is moving less than 1% of the time, we believe that this algorithm is also practical from an energy consumption perspective.

4.4 Open Problems

Chen and Koutsoukos present some open problems related to node placement in wireless sensor networks [3]. Our unconventional design and choice of cellular automata as a platform mitigates several of them:

Sensors with Irregular Sensing or Communication Ranges. Most node placement algorithms assume that all sensors have the same sensing and communication ranges. However, this is often not true in practice where different kinds of sensors are combined in one system. Since our transition rules are simply based on the number of other sensors every individual node can locally see within its communication range, this problem is inherently taken care of. Sensors with irregular sensing and communication ranges can also be simulated in cellular automata using non-uniform transition rules.

Coverage Solutions for Mobile Sensor Networks. Mobility is at the core of the presented system. It enables desired initial positioning as well as fault tolerance when changes in the environment (or problems with individual sensors) cause reduced coverage or connectivity. Therefore, this problem is solved by definition.

Other Energy Conservation Methods (besides scheduling). Our simulated annealing approach is based on scheduling; therefore, regardless of its effectiveness, it does not solve this open problem. Chen and Koutsoukos propose communication range reduction as an example measure aimed at energy conservation [3]; while this measure is not part of our system, it could be accommodated by locally adjusting the communication range probabilistically depending on the number of other nodes in that range.

Fault Tolerance. Thanks to its local probabilistic design, our system is inherently fault tolerant. We prove this in the section above after implementing node decay and still getting promising coverage and connectivity results thanks to the system's periodic adjustments through simulated annealing.

4.5 Conclusions

We have shown how our system can achieve its objectives of maximizing coverage and connectivity while aiming for sparsity, provided the right number of sensors is initially deployed. We have also shown how we can quickly estimate that number. While energy conservation seemed to be the only potential major weakness of our system, it is no longer an issue thanks to the simulated annealing and node replenishment enhancements. It is possible that energy consumption and convergence speed could be further improved with a more careful choice of movement direction. Future work could also consider cases where mobility is somewhat restricted as well as communication protocols that best complement the presented node placement scheme.

References

1. Younis, M., Akkaya, K.: Strategies and techniques for node placement in wireless sensor networks: A survey. Elsevier Ad Hoc Network Journal 6(4), 621–655 (2008)
2. Mitchell, M.: Computation in cellular automata: A selected review. In: Non-Standard Computation. John Wiley & Sons, Inc., New York (1997)
3. Chen, J., Koutsoukos, X.: Survey on coverage problems in wireless ad hoc sensor networks. In: IEEE SouthEastCon, Richmond, VA (March 2007)
4. Torbey, S.: Towards a framework for intuitive programming of cellular automata. Parallel Processing Letters 19(01), 73 (2009)
5. Wolfram, S.: A New Kind of Science, 1st edn. Wolfram Media (May 2002)