The Journal of Supercomputing (2021) 77:9069–9087
https://doi.org/10.1007/s11227-020-03549-8

A cellular automata rule placing a maximal number of dominoes in the square and diamond

Rolf Hoffmann[1] · **Dominique Désérable**[2] · **Franciszek Seredyński**[3]

Accepted: 28 November 2020 / Published online: 2 February 2021

Abstract
The objective is to demonstrate that a probabilistic cellular automata rule can place reliably a maximal number of dominoes in different active area shapes, exemplarily evaluated for the square and diamond. The basic rule forms domino patterns, but the number of dominoes is not necessarily maximal and the patterns are not always stable. It works with templates derived from domino tiles. The first proposed enhancement (Rule Option 1) can form always stable patterns. The second enhancement (Rule Option 2) can maximize the number of dominoes, but the reached patterns are not always stable. All rules drive the evolution by specific noise injection.

Keywords Pattern formation · Probabilistic cellular automata · Asynchronous updating · Matching templates · Overlapping tilings

1 Introduction

Pattern formation is an area of active research in various domains such as physics, chemistry, biology, computer science or natural and artificial life. Cellular automata (CA) are suitable and powerful tools for catching the influence of the microscopic scale onto the macroscopic behavior of such complex systems [1–3]. At the least, the 1-dimensional Wolfram's "Elementary" CA can be viewed as generating a large

✉ Rolf Hoffmann
hoffmann@informatik.tu-darmstadt.de

Dominique Désérable
domidese@gmail.com

Franciszek Seredyński
f.seredynski@uksw.edu.pl

[1] Technische Universität Darmstadt, Darmstadt, Germany

[2] Institut National des Sciences Appliquées, Rennes, France

[3] Department of Mathematics and Natural Sciences, Cardinal Stefan Wyszynski University, Warsaw, Poland

diversity of 2-dimensional patterns whenever the time evolution axis is considered as the vertical spatial axis, with patterns depending or not on the random initial configuration [4]. Regarding the agent-based Yamins–Nagpal "1D spatial computer" [5, 6] the authors emphasize therein how the local-to-global CA paradigm can turn into the inverse global-to-local question, namely "given a pattern, which CA rules will robustly produce it?" Such CA rules can be found by (i) proper design, (ii) by exhaustive search or (iii) by heuristics like Genetic Algorithm (GA) or Simulated Annealing, methods which were applied to solve the Density Classification Problem [7], for instance.

The arrangement of dominoes in a grid of cells is a special case of pattern formation. Possible applications are: parcel packing encountered in different logistics settings, such as loading boxes on pallets, arrangements of pallets in trucks or cargo stowage [8]; the design of a sieve for rectangular particles with a maximum flow rate; or an optimal arrangement of nanoparticles; and so forth. Unlike the *dimer* in statistical mechanics [9, 10]—a pure tiling problem—we do not allow dominoes to contact with one another. That means that space between dominoes is mandatory. As a result, the solution space is more complex than in the dimer case with tight compaction. Domino arrangement is closely related to short-range interaction couplings in spin systems [11].

1.1 Previous and related work

In further previous work [12–14], different patterns were generated by agents with embedded finite state control which was evolved by GA. Matching pattern templates were also applied during the training period, but are not part of the CA rule as in our current proposal. They were also defined in a different simple way in order to count the number of dominoes for the fitness function during the evolutionary process.

In [15, 16], domino patterns were formed by moving agents. Agents' behavior was controlled by a finite state machine, evolved by GA. The effort to find such agents was quite high, especially to find agents that work on any field size. In order to avoid such a computational effort, a novel approach to construct directly the required CA rule is proposed that will be presented thereafter. It has also the potential to be applied to further pattern formations. In addition, the new Rule Option 1 is defined that drives the evolution to a stable patterns.

In [17], this approach was already applied for the domino problem in a *square* field $n \times n$, n even. Herein the purpose is to prove the robustness of the model against field's shape changing. Now the square size is generalized to any odd–even n. Moreover, the rule is now extended to a rhombic *diamond* field's shape, a $\frac{\pi}{4}$–tilted square. Again in this case, no confusion must be made with the *Aztec* diamond, another pure dimer tiling problem [18].

Parallel Substitution Algorithm (PSA) [19] is a powerful generalization of CA, which was also inspiring this work. PSA allows to substitute small locally defined patterns P by other patterns Q in a non-conflicting way. Thereby, very complex computations and transformations can be performed in a decentralized and parallel way.

The problem of optimal domino placement is presented in Sect. 2. A basic probabilistic CA rule is designed in Sect. 3 that can form valid domino patterns. In Sect. 4, two rule enhancements (options) are presented. Rule Option 1 allows to stabilize the pattern, whereas Rule Option 2 drives the evolution to optimal patterns. Results of simulation, performance evaluation and robustness are discussed in Sect. 5 before Conclusion.

2 Optimal placement of dominoes

2.1 The problem

Given is an array of $N = (n + 2) \times (n + 2)$ cells with state values $s \in \{0, 1\}$. The objective is to find a CA rule that can place a maximal number of dominoes within a given *shape* of active cells $N_{\text{active}} \leq (n \times n)$ surrounded by inactive border cells with value 0. A domino is given by two orthogonal adjacent cells of value 1. The constraint is that between dominoes there shall be at least one empty cell with value 0, i.e., dominoes are not allowed to touch each other.

The dominoes that we will use here in our solution are "domino tiles," and we propose to use them to cover the given shape. A domino tile consists of two pixels with value 1 (the *kernel*, in blue) and 10 surrounding pixels with value 0 (the *hull*, in green). In order to avoid confusion with the regular cells, we call the elements of a tile (or the later introduced *templates*) "*pixels*." Two types of dominoes are distinguished, the horizontal oriented domino (D_{H}) and the vertical oriented (D_{V}) (Fig. 1a). It is allowed—and even necessary for a good solution—that green pixels from different domino tiles overlap. The possible levels of overlapping, from 2 to 4, are displayed in Fig. 1(b–d). We call the level of overlapping *cover level v*.

Our intention is to show that the CA rule is *robust* (insensitive) *against a change of the shape*, and in addition, to improve the behavior of the CA rule (Option 1, in Sect. 4.1). Two shapes are considered, the *square* and the *diamond*. The *diamond* can be defined in the following way. Cells inside the diamond at position (x, y) (measured from a corner cell of the whole field) are given by the conditions

$$|x - (n-1)/2| + |y - (n-1)/2| \leq (n-1)/2 \text{ if } n \text{ odd},$$
$$|x - n/2 - 1/2| + |y - n/2 - 1/2| \leq n/2 \text{ if } n \text{ even}.$$

The number of dominoes is denoted as $d = d_{\text{H}} + d_{\text{V}}$, where d_{H} is the number of horizontal dominoes and d_V is the number of vertical dominoes. A further requirement can be that the number of domino types shall be equal (or almost equal). We call

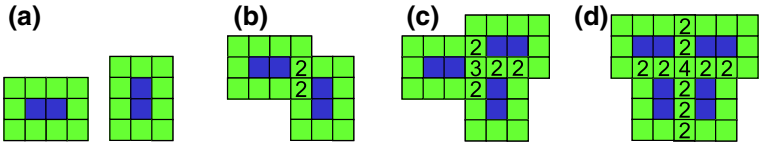

Fig. 1 **a** Horizontal and vertical domino tile, **b** two pixels of two domino hulls are overlapping, marked by *2*, **c** the pixel marked by *3* is the overlap of three domino hulls, **d** a case with *4* overlapping hull pixels (color figure online)

such patterns *balanced patterns* when $d_H = d_V$ if d_{max} is even, and $d_{\mathrm{H}} = d_{\mathrm{V}} \pm 1$ if $d_{\max}$ is odd, where $d_{max}(n)$ is the maximal possible number of dominoes that can be placed into the field with overlapping.

2.2 Domino enumeration

We expose here a theoretical framework [20] to support the simulation results which will be presented in the sequel.

2.2.1 Dominoes in the square

Given a square array $\mathcal{S}_n$ of $(n+2) \times (n+2)$ cells including a border with perimeter $4n+4$ enclosing a $n \times n$ field of order n^2, the maximal domino number $\xi_n(\mathcal{S}_n)$ covering $\mathcal{S}_n$ is given by the inductive formula

- *n even*: $\xi_0 = 0,\ \xi_2 = 1,\ \xi_4 = 4$ and $\xi_n = \xi_{n-6} + 2(n-2)$
- *n odd*: $\xi_1 = 0,\ \xi_3 = 2,\ \xi_5 = 6$ and $\xi_n = \xi_{n-6} + 2(n-2)$

for $n \geq 6$, where $2(n-2)$ denotes the maximal number of dominoes in the crown surrounding the inner subgrid $\mathcal{S}_{n-6}$.

□

Setting $m = \lfloor n/2 \rfloor$ and $p = \lfloor m/3 \rfloor$, this hierarchy of configurations can then be divided into the three equivalence classes $\overline{\mathcal{S}}_0, \overline{\mathcal{S}}_1, \overline{\mathcal{S}}_2$ according to m mod 3, as illustrated in Fig. 2.

2.2.2 Dominoes in the diamond

A diamond $\mathcal{D}_n$ is given with perpendicular diagonals of length n and surrounded by a border with perpendicular diagonals of length $n+2$.

- n odd . The order $|\,\mathcal{D}_n\,|$ of $\mathcal{D}_n$ fulfills

$$|\,\mathcal{D}_n\,| = (n+2) + 2\,(1 + 3 + \cdots + n) = ((n+2)^2 + 1)/2$$

including a border of length $2n+2$ and $\mathcal{D}_n$ has an inner perimeter of length $2n-2$.

The maximal domino number $\psi_n(\mathcal{D}_n)$ is given by the inductive formula

$$\psi_1 = 0,\ \psi_3 = 1,\ \psi_5 = 2 \quad \text{and} \quad \psi_n = \psi_{n-6} + (n-3)$$

for $n \geq 6$, where $(n-3)$ denotes the maximal number of dominoes in the crown surrounding the inner subgrid $\mathcal{D}_{n-6}$.

This hierarchy of configurations can be divided into the three equivalence classes $\overline{\mathcal{D}}_0, \overline{\mathcal{D}}_1, \overline{\mathcal{D}}_2$ according to m mod 3, as illustrated in Fig. 3.

The domino capacities in the square and diamond satisfy this strong relationship

$$\psi_n = \frac{1}{2}\,(\xi_{n-1} + \mu) \quad \mu = m \bmod 3 \pmod 2$$

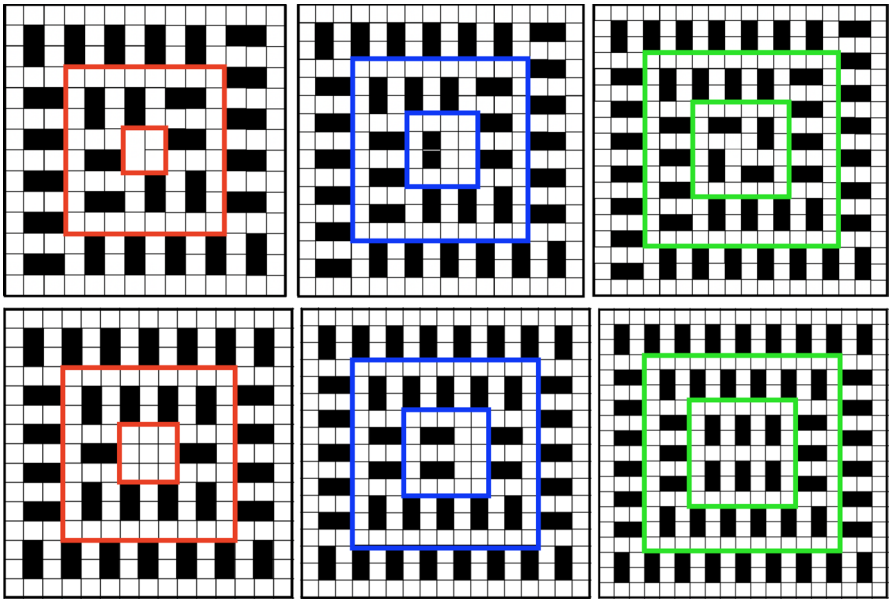

Fig. 2 Dominoes in the square. (↑) Equivalence classes $\overline{\mathcal{S}}_0, \overline{\mathcal{S}}_1, \overline{\mathcal{S}}_2$, according to m mod 3 with n even. From left to right: $(\mathcal{S}_0, \mathcal{S}_6, \mathcal{S}_{12})$, $(\mathcal{S}_2, \mathcal{S}_8, \mathcal{S}_{14})$, $(\mathcal{S}_4, \mathcal{S}_{10}, \mathcal{S}_{16})$. (↓) Equivalence classes $\overline{\mathcal{S}}_0, \overline{\mathcal{S}}_1, \overline{\mathcal{S}}_2$, according to m mod 3 with n odd. From left to right: $(\mathcal{S}_1, \mathcal{S}_7, \mathcal{S}_{13})$, $(\mathcal{S}_3, \mathcal{S}_9, \mathcal{S}_{15})$, $(\mathcal{S}_5, \mathcal{S}_{11}, \mathcal{S}_{17})$ (color figure online)

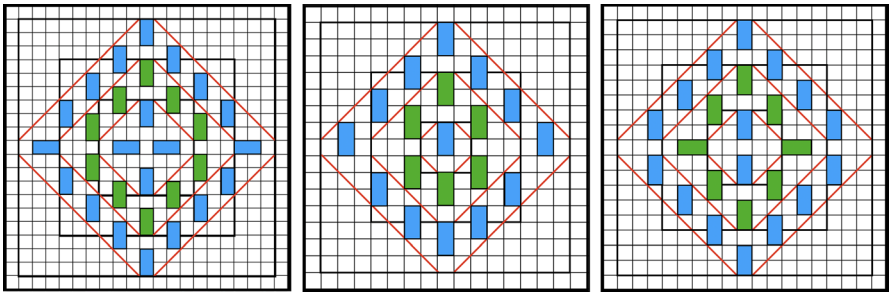

Fig. 3 Dominoes in the diamond Equivalence classes $\overline{\mathcal{D}}_0, \overline{\mathcal{D}}_1, \overline{\mathcal{D}}_2$ according to m mod 3 with n odd. From left to right: $(\mathcal{D}_7, \mathcal{D}_{13}, \mathcal{D}_{19})$, $(\mathcal{D}_3, \mathcal{D}_9, \mathcal{D}_{15})$, $(\mathcal{D}_5, \mathcal{D}_{11}, \mathcal{D}_{17})$, (color figure online)

- n even . The order $|\mathcal{D}_n|$ of $\mathcal{D}_n$ fulfills

$$|\mathcal{D}_n| = 2\,(2 + 4 + 6 + \cdots + (n+2)) = (n+2)(n+4)/2$$

including a border of length $2n + 4$ and $\mathcal{D}_n$ has an inner perimeter of length $2n$.

The maximal domino number $\psi_n(\mathcal{D}_n)$ is given by the inductive formula

$$\psi_0 = 0,\ \psi_2 = 1,\ \psi_4 = 2 \quad \text{and} \quad \psi_n = \psi_{n-6} + O(n)$$

for $n \geq 6$, where $O(n)$ denotes the number of possible dominoes in the crown surrounding the inner subgrid $\mathcal{D}_{n-6}$. The case n even is more intricate because the "crown" parameter is fluctuating and only fulfills a weak property. It is possible to define a generic family of diamonds as a complement of embedded squares, as illustrated in Fig. 4. The following three cases are considered ($n \geq 6$).

- $\psi_n = \xi_{2p} + 4\,(W_1 + W_2 + \delta_p)$ for $m \equiv 0 \pmod 3$,

  where $W_1 = \dfrac{p\,(p+1)}{2}$ and $W_2 = \left\lfloor \dfrac{p\,(p-6)}{12} \right\rfloor$

and where $\delta_p = 1$ iff $p \equiv 0 \bmod 6$, $\delta_p = 0$ otherwise.

- $\psi_n = \xi_{m-\mu} + 4\,(W_1' + W_2' + q\,(1+\mu))$ for $m \equiv 1 \pmod 3$,
- $\psi_n = \xi_{m+\mu} + 4\,(W_1' + W_2' + q\,(2-\mu))$ for $m \equiv 2 \pmod 3$,

  where $W_1' = \dfrac{(p-q)(p-q+1)}{2}$ and $W_2' = \dfrac{3\,q\,(q-1)}{2}$

and where $q = \left\lfloor \frac{p+1}{4} \right\rfloor$ and $\mu = m \bmod 2$.

In each case, the first term ξ_η on the second side denotes the domino number in the square field $\eta \times \eta$ embedded into $\mathcal{D}_n$ and the second term stands for the capacities in dominoes of the four remaining wedges.

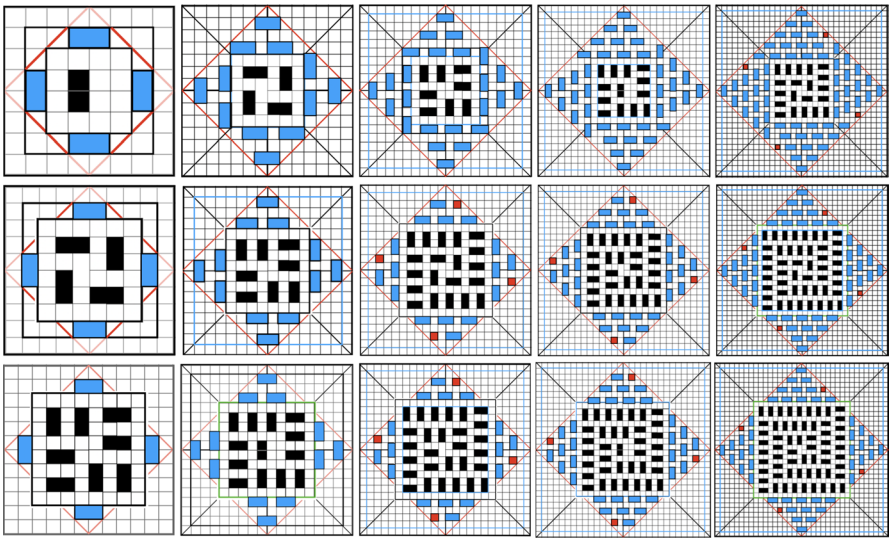

Fig. 4 Dominoes in the diamond A family of diamonds with square subfields S_η embedded in them. Equivalence classes $\overline{\mathcal{D}}_0, \overline{\mathcal{D}}_1, \overline{\mathcal{D}}_2$ according to $m \bmod 3$ with n even. From top to bottom: $(\mathcal{D}_6, \mathcal{D}_{12}, \mathcal{D}_{18}, \mathcal{D}_{24}, \mathcal{D}_{30})$, $(\mathcal{D}_8, \mathcal{D}_{14}, \mathcal{D}_{20}, \mathcal{D}_{26}, \mathcal{D}_{32})$, $(\mathcal{D}_{10}, \mathcal{D}_{16}, \mathcal{D}_{22}, \mathcal{D}_{28}, \mathcal{D}_{34})$. Isolated cells (in red) existing on `N-NE`, `E-SE`, `S-SW`, `W-NW`, borders (color figure online)

We should be aware that our construction can lead to a slight deficiency compared to the simulation. This can be explained by the fact that our theoretical construction is constrained by its own rule while the scenarios from the CA rule presented thereafter have more degrees of freedom. This deficit is due to the presence of isolated cells (Fig. 4) existing on some borders of our constrained system.

2.2.3 Space occupancy ratio

The space occupancy ratio ρ_k for a given domino k is defined as

$$\rho_k = \sum_{i=1}^{12} \tau_{k,i}$$

where $\tau_{k,i}$ is the occupancy ratio (the inverse $1/v$ of the level of overlapping) of its pixel i. Since two dominoes cannot overlap, $\tau_{k,1} = \tau_{k,2} = 1$ always holds.

For illustration, the occupancy ratio of the surrounded tiles "k" in Fig. 5 yields:

1. $\rho_k = 2 + 10 \times 1 = 12$
2. $\rho_k = 2 + 6 \times 1/2 + 4 \times 1/4 = 6$
3. $\rho_k = 2 + 4 \times 1/2 + 6 \times 1/3 = 6$
4. $\rho_k = 2 + 6 \times 1/2 + 4 \times 1/3 = 19/3 \approx 6.33$

Evidently, the second and third configurations are optimal, but an overall configuration is constrained by the boundary conditions.

For any field $\mathcal{F}$ of order $|\mathcal{F}|$ and containing d_{max} dominoes, then

$$\sum_{k=1}^{d_{max}} \rho_k = |\mathcal{F}| \quad \text{whence} \quad \sum_{k=1}^{\xi_n} \rho_k = |\mathcal{S}_n| \quad \text{and} \quad \sum_{k=1}^{\psi_n} \rho_k = |\mathcal{D}_n|$$

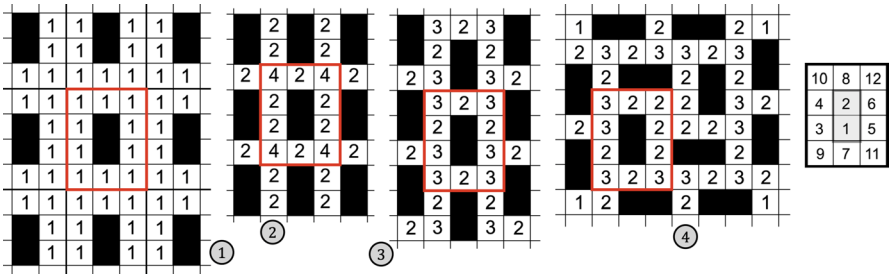

Fig. 5 Four typical arrangements of dominoes: (1) loosely coupled physical distancing configuration (2) tightly coupled orthotropic configuration (3) tightly coupled staggered configuration (4) tightly coupled isotropic configuration. *Inset*—The "domino tile" with numbered cells: the 2–cell kernel and the 10–cell hull

for square $\mathcal{S}_n$ and diamond $\mathcal{D}_n$, respectively. We observe that ratios $\mid \mathcal{S}_n \mid /\xi_n$ and $\mid \mathcal{D}_n \mid /\psi_n$ are decreasing Cauchy sequences which slowly converge towards the fixed limit of maximal occupancy as n approaches infinity

$$\lim_{n\to+\infty} \frac{\mid \mathcal{S}_n \mid}{\xi_n} = \lim_{n\to+\infty} \frac{12(n-1)}{2(n-2)} = 6 \quad ; \lim_{n\to+\infty} \frac{\mid \mathcal{D}_n \mid}{\psi_n} = \lim_{n\to+\infty} \frac{6(n-1)}{(n-3)} = 6$$

and more generally for *n even* in any case. Our theoretical results are displayed in Table 1.

Table 1 Domino enumeration for $n \times n$ fields with $m = n/2$ and $p = m/3$ Domino numbers—$\xi_n(\mathcal{S}_n)$ in the square—$\psi_n(\mathcal{D}_n)$ in the diamond. Space occupancy ratio—$\mid \mathcal{S}_n \mid /\xi_n$ in the square—$\mid \mathcal{D}_n \mid /\psi_n$ in the diamond

n	$m = n/2$	$p = m/3$	$\mid \mathcal{S}_n \mid$	ξ_n	$\mid \mathcal{S}_n \mid /\xi_n$	$\mid \mathcal{D}_n \mid$	ψ_n	$\mid \mathcal{D}_n \mid /\psi_n$
0	0	0	4	0	–	4	0	–
1	0	0	9	0	–	5	0	–
2	1	0	16	1	16.000	12	1	12.000
3	1	0	25	2	12.500	13	1	13.000
4	2	0	36	4	9.000	24	2	12.000
5	2	0	49	6	8.167	25	2	12.500
6	3	1	64	8	8.000	40	5	8.000
7	3	1	81	10	8.100	41	4	10.25
8	4	1	100	13	7.692	60	8	7.500
9	4	1	121	16	7.562	61	7	8.714
10	5	1	144	20	7.200	84	12	7.000
11	5	1	169	24	7.042	85	10	8.500
12	6	2	196	28	7.000	112	16	7.000
13	6	2	225	32	7.031	113	14	8.071
14	7	2	256	37	6.919	144	20	7.200
15	7	2	289	42	6.881	145	19	7.632
16	8	2	324	48	6.750	180	25	7.200
17	8	2	361	54	6.685	181	24	7.542
18	9	3	400	60	6.667	220	32	6.875
19	9	3	441	66	6.682	221	30	7.367
20	10	3	484	73	6.630	264	36	7.333
21	10	3	529	80	6.612	265	37	7.162
22	11	3	576	88	6.545	312	44	7.091
23	11	3	625	96	6.510	313	44	7.114
24	12	4	676	104	6.500	364	53	6.868
25	12	4	729	112	6.509	365	52	7.019
26	13	4	784	121	6.479	420	60	7.000
27	13	4	841	130	6.469	421	61	6.902
28	14	4	900	140	6.428	480	69	6.957
29	14	4	961	150	6.407	481	70	6.871

3 The design idea

The first approach was to design a *deterministic* rule with *synchronous* updating. After some experiments and experience from previous work, it showed to be very difficult if not even impossible to design such a rule that can converge always or with a high probability to the optimal or near-optimal aimed pattern.

The second approach was to construct a *probabilistic* rule with *synchronous* updating. Indeed, such a rule was found for a field of size 6×6 by GA, where each cell is modeled as an agent that can turn in any direction. But the effort to find such rules is high and the good behavior cannot be guaranteed in general. The third and successful approach used here is the design of a *probabilistic* rule with *asynchronous* updating in a methodical way.

3.1 The basic rule

The basic idea is to modify the current configuration in a systematic way such that increasingly more dominoes appear and at last the CA evolves to a stable pattern. To do this, the CA configurations are searched for domino tile parts (specific local patterns), and if an almost correct tile part is found, it is corrected; otherwise, some random noise is injected.

The domino tile parts are called "*templates*" A_i. They are systematically derived from the domino tiles (Fig. 6a). For each of the 12 pixels i of a domino (marked in red, carrying the domino pixel value $dval(i)$), a template A_i is defined. A template can be seen as a copy of the tile, but shifted in space in a way that the pixel i corresponds to the center of the template.

In the computation, we represent a *template* $\mathbf{A_i}$ as an array of size $(a' \times b')$ of pixels, where $a' = 2a - 1$, $b' = 2b - 1$ and $(a \times b)$ is the size of the tile (its bounding box). Our horizontal tile is of size (3×4); thus, their templates are of size (5×7) maximal (larger because of shifting). The pixels within a template are identified by relative coordinates $(\Delta x, \Delta y)$, where $\Delta y \in \{-(a-1), \ldots, a-1\}$ and $\Delta x \in \{-(b-1), \ldots, b-1\}$. The center pixel $(\Delta x, \Delta y) = (0, 0)$ is called "*reference pixel*." Each template pixel carries a value $val(A_i, \Delta x, \Delta y) \in \{0, 1, \#\}$. The value of

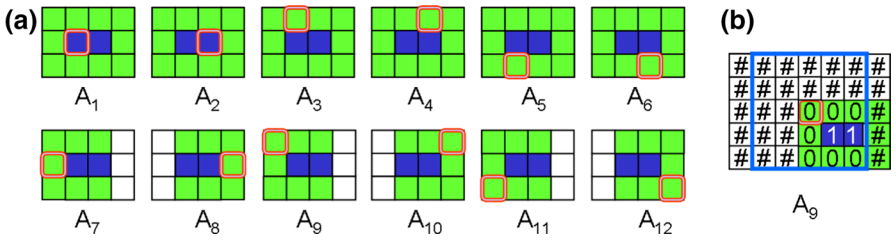

Fig. 6 **a** 12 templates of the horizontal domino tile. The value of the template center (marked, the so-called reference pixel) is used for cell updating if all other template pixel values match with the corresponding cell values of the current configuration. **b** Template A_9 represented as an array with Don't Cares (#). The templates size was reduced from (5×7) to a matching window of size (5×5) (color figure online)

the reference pixel is called "*reference value*," $refval(A_i) = val(A_i, 0, 0) \in \{0, 1\}$. Its value is equal to the red marked value of the corresponding tile pixel, $refval(A_i) = dval(i)$. The symbol # represents "Don't Care," meaning that a pixel with such a value is not used for matching (or does not exist, in another interpretation). Pixels with a value 0 or 1 are *valid* pixels; their values are equal to the values derived from the original tile. Some templates can be embedded into arrays smaller than $(a' \times b')$ when they have Don't Cares at their borders. Note that the valid pixels are asymmetrically distributed in a template because they are the result from shifting a tile.

Many of these templates are similar under mirroring, which can facilitate an implementation. For the vertical domino, a similar set of 12 templates is defined by 90° rotation; altogether, we need 24 templates.

The templates $A_7 - A_{12}$ show white pixels that are not used because the template size (for the later described matching process) was restricted to (5×5). As an example, the reduced template A_9 is marked in Fig. 6b by the blue square. The implementation with these incomplete templates worked very well, but further investigations are necessary to prove to which extent templates can be incomplete.

We need also to define the term "*neighborhood template*" that is later used in the matching procedure. The neighborhood template A_i^* is the template A_i in which the reference value is set to #, in order to exclude the reference pixel from the matching process. The cell processing scheme is:

1. A cell is randomly selected.
2. The rule is applied asynchronously. The new cell state $s' = f(s, B^*)$ is computed and immediately updated without buffering, where $s(x, y) \in \{0, 1\}$ is the cell's state at position (x, y). $B^*(x, y)$ denotes the states of the neighbors of (x, y) within a (5×5)-window, excluding the center. A new generation at time-step $t + 1$ is declared after N_{active} cell updates (sub-steps) during the compute interval between t and $t + 1$. We will update each cell once in a time interval in a random order which is re-computed for each new time-step.

The following rule is applied:

$$s'(x, y) = \begin{cases} refval(A_i) & \textbf{if } \exists A_i^* \textit{ matches with CA–Neighbors } B^*(x, y) \quad (a) \\ & \textbf{otherwise} \\ random \in \{0, 1\} & \textit{with probability } \pi_0 \quad (b1) \\ s(x, y) & \textit{with probability } 1 - \pi_0 \quad (b2) \end{cases}.$$

The neighborhood templates A_i^* are tested against the corresponding cell neighbors $B^*(x, y)$ in the (5×5)–window at current position (x, y). Thereby, the marked reference position $(\Delta x, \Delta y) = (0, 0)$ of a neighborhood template is aligned with the center of the window. If all values match, then the state of cell (x, y) is set to the value $refval(A_i)$. Otherwise, with probability π_0, the cell is set randomly to either 0 or 1, or remains unchanged with probability $1 - \pi_0$. The rule corrects the state of a

cell to the reference value if a matching neighborhood is detected; otherwise, cell's state is changed randomly. It is possible that several neighborhood templates match (then tiles are overlapping), but there can be no conflicts because then all templates have the same reference value as derived from the tile. As no conflicts can arise, the sequence of testing the neighborhood templates does not matter, and one could skip further tests after a first match.

It is important to note that the rule obeys the criterion of stability, which means that a field filled with dominoes without gaps (uncovered cells) is stable because we have matching at every site. Otherwise some random noise is injected in order to drive the evolution to an aimed pattern.

3.1.1 Test on the square

The rule was tested for $N_{\text{run}} = 1000$ runs on (5×5)–fields with random initial configuration, $T_{\text{Limit}} = 50$ (simulation time-step limit), with $\pi_0 = 0.5$. We know that there exist valid solutions with 6, 5, or 4 dominoes. The system converges after a few iterations into one of two solution classes. In the class I (stable), the 4 dominoes are covering the square totally without gaps and the reached pattern is stable. In the class II (partially stable), the 4 dominoes are covering the diamond not totally with at least one gap. The gap cells are toggling their state values ($0 \leftrightarrow 1$) due to the injected noise that never ends. Nevertheless, the class II patterns contains 4 dominoes which do not change (neither position, nor orientation or number).

The number of stable patterns (class I) with $d = 6, 5, 4$ dominoes was 41, 514, and 225, respectively. The average number of time-steps to reach stability for the class I patterns was $t_{\text{avrg}} = 4.0$. The number of partially stable class II patterns with an isolated toggling state was 194. The evolution of a class I and a class II pattern is depicted in Fig. 7b, c.

3.1.2 Test on the diamond

The rule was tested on (7×7) initially random fields for $N_{\text{run}} = 1000$, with $T_{\text{Limit}} = 50$ and $\pi_0 = 0.5$. This diamond has the same number of active cells (25) as the (5×5)–square. We know that there exist only solutions with 4 dominoes. The system converges after a few iterations into one of the two solution classes; 166/1000 runs evolved into a stable class I pattern, and the rest evolved into class II patterns. The average number of time-steps to reach stability for the class I patterns was $t_{\text{avrg}} = 5.1$. A simulation of a class I and a class II pattern is depicted in Fig. 7d, e.

These tests showed that always either class I (stable) or class II (partially stable with gaps (isolated uncovered cells)) patterns evolved rapidly. The number of dominoes was ranging from the minimum to the maximum, also validated by further experiments not outlined here.

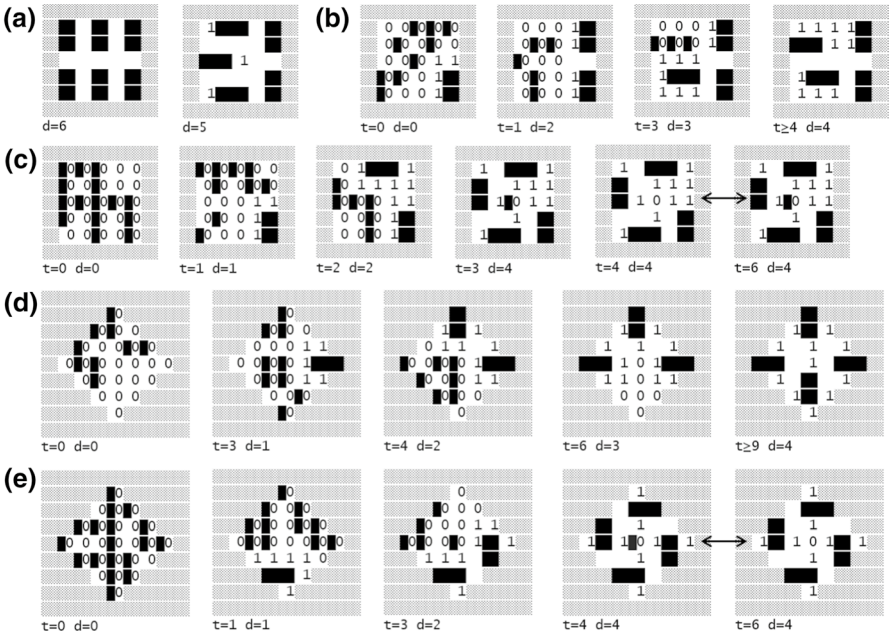

Fig. 7 Basic rule. **a**–**c** (5 × 5)—square. **d**–**e** (7 × 7)—diamond. **a** Stable patterns with 6 resp. 5 dominoes. **b**, **d** Evolution of a stable pattern. **c**, **e** Evolution of a partially stable pattern, containing a gap that toggles between 0 and 1. Representation: (gray) border cell, (black) domino cell, (0) uncovered zero-state cell/gap, (black half square with 0) uncovered one-state cell, (1) zero-state cell with cover level 1, and (white) zero-state cell with cover level ≥ 2

4 Rule enhancements

The basic rule was enhanced by adding optional rules on top:

- (Option 1) The aim was to reach only stable class I patterns.
- (Option 2) The aim was to reach preferably patterns with a maximal number of dominoes (max patterns).

A variable *hit*(*x*, *y*) was added to the cell's state then becoming (*s*, *hit*). All hits can be seen as an additional layer, also called *hit matrix.* The hit matrix stores the number of template hits for every site (*x*, *y*) that was selected for computation and updating. The number of hits on a site *hit*(*x*, *y*) is:

- 0, if no neighborhood template matches or there is a gap.
- 1, if exactly one neighborhood template with the reference value 0 matches,
- 2–4, if there are 2–4 neighborhood templates with reference values 0 that match at the same site (*x*, *y*). This means that 2–4 tiles are overlapping there.

- 100, if one neighborhood template with the reference value 1 matches. Note that 1-valued pixels are not allowed to overlap. The number 100 was arbitrarily chosen in order to differentiate such a hit from the other.

Recall that we are using asynchronous updating, where a time-step interval consists of N_{active} sequential substeps. The value of a hit can be up-to-date, computed already during the current time interval, or it can be the old value from the previous time interval. In the case of a stable pattern the hit is equal to the cover level. In the case of a transient pattern a hit can be equal to the cover level in regions where the pattern shows already stable dominoes. So the hit matrix approximates the cover level values.

The enhanced execution mode is now for each cell: (1) compute the new state s' according to the basic rule, (2) compute the hit value, (3) modify the new state to s'' by Option 1 and then (4) modify the new state to s''' by Option 2. Because of the asynchronous updating scheme, the new state replaces the state s then immediately.

4.1 Rule option 1: stabilizing the pattern

We have seen in Sect. 3.1 that the basic rule can evolve class I and class II patterns. We define now an additional rule that will turn class II patterns into class I patterns. Analyzing the class II patterns, we can see isolated toggling cells with hit = 0. The idea is to disseminate this information to the cells in the von Neumann neighborhood. If a cell in the neighborhood detects a hit-zero cell, it will produce additional noise in order to drive the evolution to a stable pattern without gaps. The optional rule is

$$s''(x,y) = \begin{cases} random \in \{0,1\} & with\ probability\ \pi_1\ \textbf{if}\ \ \exists hit(x \pm 1, y \pm 1) = 0 \\ s'(x,y) & \textbf{otherwise} \end{cases}$$

Two tests were performed with 100 runs and $T_{\text{limit}} = 300$, one on a (5×5)–square and another on a (7×7)–diamond.

4.1.1 Test on square

All 100 reached patterns were now stable (class I). The frequency of patterns with $d = 6, 5, 4$ was 2, 70, 28 and $t_{\text{avrg}} = 10.2$ (min 2 – max 41).

4.1.2 Test on diamond

All 100 reached patterns were stable. The number of dominoes was $d = 4$, the only possibility as expected, with $t_{\text{avrg}} = 37.0$ (min 2 – max 197).

4.2 Rule option 2: maximizing the number of dominoes

The idea is to maximize the overlap between tiles by destroying non–overlapping situations (hit = 0) through additional noise, allowing a reordering with higher hit

rates. First, the new state s'' (or s' if Option 1 is not applied) is computed and then the hit matrix. Then, the new state is modified to s''':

$$s'''(x,y) = \begin{cases} random \in \{0,1\} \ with\ probability\ \pi_2 & \textbf{if}\ \ hit(x,y) = 1 \\ s''(x,y) & \textbf{otherwise} \end{cases}$$

When this option is applied it is not clear whether a reached domino pattern remains stable. In fact, it turned out that stability can only be reached if there exists a tiling, where every tile overlaps with at least another one or the border (called *totally overlapping tiling*), i.e., the cover level is $v \geq 2$ everywhere inside the active area, e.g., a totally overlapping tiling exists for (10×10) but not for (8×8) square fields. Therefore, the number of dominoes will reach the maximum and remain stable in a (10×10) field, whereas the number of dominoes in a (8×8) field is reaching a maximum, and then it is decreasing and fluctuating and is driving again towards another maximum, and so forth.

Four tests were performed on the (5×5)–square and the (7×7)–diamond, each with 1000 runs and $T_{\text{limit}} = 500$.

4.2.1 Test with option 2 only

- *Square.* The frequency of patterns with $d = 6, 5, 4$ was 96.0%, 4.0%, and 0%. The max patterns with $d = 6$ were stable. The patterns with $d = 5$ were changing between different solutions but did not reach a max pattern because of the limited time. $t_{\text{avrg}}(d = 6, 5) = 120\ (3 - 497)$. For averaging, the time of the first appearance of that number of dominoes was used.
- *Diamond.* All patterns contained 4 dominoes, the only number possible, and all of them were not stable, changing from one solution to another because no totally overlapping solution exists. The average time of the first appearance of a valid solution was $t_{avrg}(d = 4) = 59.9$.

4.2.2 Test with option 2 and option 1

- *Square.* The frequency of patterns with $d = 6, 5, 4$ was 94.7%, 5.3%, and 0%. All max patterns with $d = 6$ were stable. The patterns with $d = 5$ were changing between different solutions. $t_{\text{avrg}}(d = 6, 5) = 135.1\ (3 - 500)$. For averaging, the time of the first appearance of that number of dominoes was used.
- *Diamond.* All patterns contained 4 dominoes, the only number possible, and all of them were not stable, changing from one solution to another because no totally overlapping solution exists. The average time of the first appearance of a valid solution was $t_{\text{avrg}}(d = 4) = 40.7$.

These tests suggest that Option 1 is not really helpful when Option 2 is used in order to find a max pattern. One can see Option 1 only as another unnecessary source of noise. Therefore, in the following we will use only Option 2 because our objective is to find max patterns.

5 Performance and robustness

5.1 Performance for different field sizes

The basic rule with Option 2 (maximizing dominoes) was simulated for the square and diamond with $\pi_0 = 0.5$ and $\pi_2 = 0.05$. The simulation time limit T_{limit} was set large enough to yield max patterns for every simulation run in each test set of runs, e.g., $T_{\text{limit}} = 40\,000$ for $n = 12, 16$, two slow converging cases. The number of runs in a test set for averaging was $N_{\text{run}} = 100$ for $n \leq 10$ and 20 for $n > 10$. The average time t_{avrg} is shown in Table 2. It corresponds to the *Work per Cell* if each cell is considered as a processor, because the total (parallel) *Work* is $t_{\text{avrg}} \times N_{\text{active}}$. We divide further the work per cell by the problem size N_{active}. This quotient $E = t_{\text{avrg}}/N_{\text{active}}$ is a constant if the work per cell would increase linear with N_{active}. So this measure can show us a superlinear increase in work per cell if it increases with problem size. The minimal and maximal times were also recorded for each run and related to the average time. On average, the minimal/maximal time recorded was 0.085/4.02 times t_{avrg}. These extensive simulations confirmed that the basic CA rule with Option 2 converges to patterns with a maximal number of dominoes if the time limit is large enough, no matter whether the square shape or the diamond shape was used. It was surprising that E shows some

Table 2 For the square and the diamond with $N_{\text{active}}(n)$ cells, the average time t_{avrg} to yield a maximum number d_{max} of dominoes for each simulation in a set of simulation runs was evaluated. The quotient $t_{\text{avrg}}/N_{\text{active}}(n)$ gives the number of necessary time-steps per active cell

n	Square				Diamond			
	d_{max}	N_{active}	t_{avrg}	$t_{\text{avrg}}/N_{\text{active}}$	d_{max}	N_{active}	t_{avrg}	$t_{\text{avrg}}/N_{\text{active}}$
2	1	4	0.93	0.23	1	4	0.89	0.22
3	2	9	3.22	0.36	1	5	2.4	0.48
4	4	16	75	4.7	2	12	49	4.08
5	6	25	126	5.0	2	13	75	5.77
6	8	36	986	27.4	5	24	60	2.5
7	10	49	90	1.8	4	25	55	2.2
8	13	64	258	4.0	8	40	178	4.5
9	16	81	375	4.6	7	41	84	2.1
10	20	100	1 185	11.9	12	60	2 497	41.6
11	24	121	2 272	18.8	10	61	116	1.9
12	28	144	8 477	58.9	16	84	12 598	145.0
13	32	169	1 305	7.7	14	85	168	2.0
14	37	196	2 563	13.1	20	112	491	4.4
15	42	225	2 920	13.0	19	113	432	3.8
16	48	256	11 220	43.8	26	144	3 348	23.3
17	54	289	8 790	20.0	25	145	14310	98.69
18	60	324	26 971	83.2	32	180	3 802	21.12

remarkable peaks and drops, in addition of a weak increase with problem size. In the investigated range E was highest for $n = 12$ in the diamond and for $n = 18$ in the square. An explanation is that max patterns have a certain frequency among all valid patterns, and that there a harder tasks (field sizes) where the frequency of max patterns is very low and therefore more difficult to reach by the walk through the pattern space. For example, there are only 4 highly symmetric optimal solutions in the diamond $\mathcal{D}_{12}$ but a lot of non-symmetric optimal solutions in $\mathcal{D}_{14}$. In Fig. 4, only symmetric solutions were shown. Further research is necessary to make a sound statement about the time complexity for large n. Figure 8 shows some patterns evolved by the CA rule with Option 2 for the square and the diamond for $n = 12, 13$. All are max patterns except (d). (a, c, e, g) are balanced. (a, b, c, f) are totally overlapping (no $v = 1$ is visible). Some larger balanced max patterns evolved with Option 2 are shown in Fig. 9.

Our theoretical results displayed in Table 1 should be compared with simulation results in Table 2. $|\mathcal{S}_{n-2}|$ and $|\mathcal{D}_{n-2}|$ in Table 1 correspond with N_{active} in Table 2 while ξ_n and ψ_n tie in $d_{\max}$. Let us observe the slight deficit for ψ_{16}, ψ_{17}. Looking at Fig. 9, let us observe a slight deficit for ψ_{24}, ψ_{25} and a big deficit for ψ_{26}. Looking back to Fig. 4, let us observe again the presence of isolated cells on the diamond's border, that is likely to explain this phenomenon of domino's deficit.

5.2 Robustness

We have seen that the rule works reliable for two different shapes, and it works also for the circle and rectangle when being tested. Further experiments have shown that the CA rule is also robust (insensitive) against the initial configuration and the order in which the cells are updated.

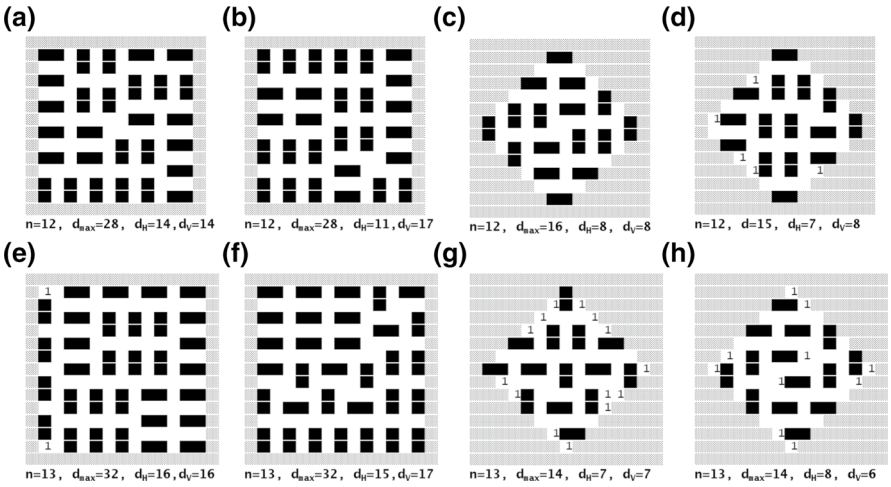

Fig. 8 Some patterns evolved with Option 2. All are max patterns except (**d**). **a**, **c**, **e**, **g** are balanced. **a**, **b**, **c**, **f** are totally overlapping (cover level $v > 1$)

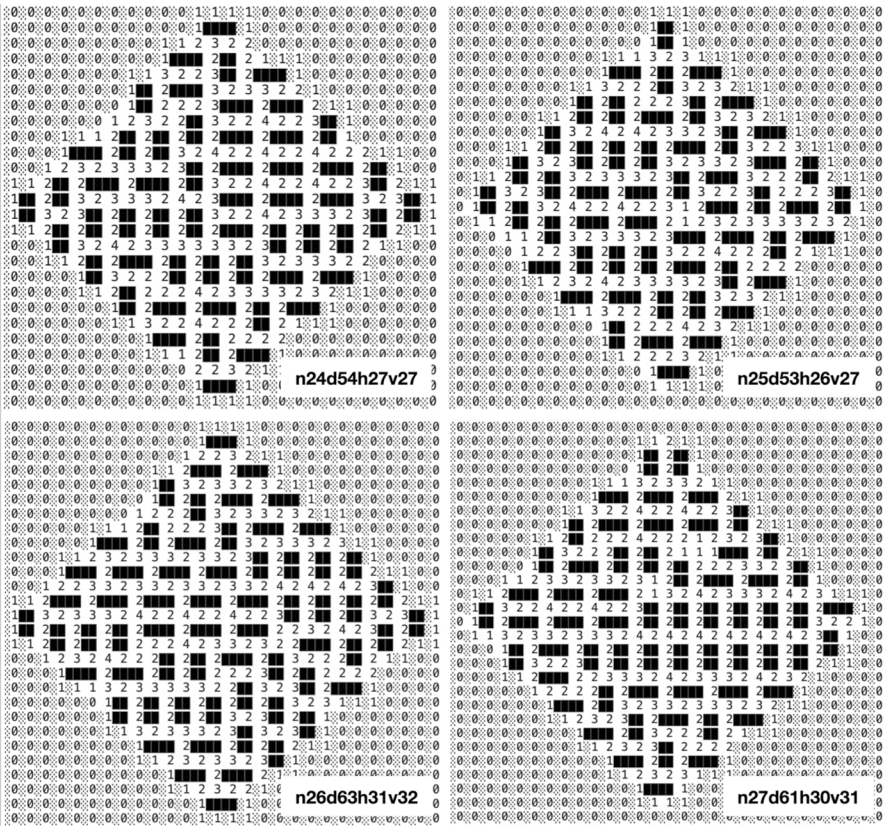

Fig. 9 Some balanced max patterns evolved with Option 2 in the diamond. Numbers represent the cover levels. Diamonds of size $n = 24, 25, 26, 27$, with $d_{\max} = 54, 53, 63, 61$

5.2.1 Initial configuration

In all the simulations conducted here, the initial cell states were random. As already shown in [17], the rule works similarly well if the initial configuration is black or white everywhere.

5.2.2 Self-stabilization

This term was introduced in 1973 by Dijkstra [21, 22]. It means that a system will always converge to a desired system state even if it is disturbed. This is a very important feature for systems to be reliable. When in our system failures appear from time to time, they will not influence the overall dynamics, because if the system is in an early stage it will result in just another random initial state, and if the system is in a late stage, then errors are corrected or the system will drive to another valid solution.

5.2.3 Updating sequence

We used here for each time interval a different random order in such a way that every cell is updated once. In the former work [17], there was no ordering, cells were picked up at random and updated. Thereby, a cell can be updated never or multiple times in a time interval (updating N_{active} times). By this method, the domino patterns evolved similarly.

In addition we tested also the row-by-row sweep sequence and the "checkerboard sequence." For the checkerboard sequence, first the cells are updated for $(x + y)$ even, then for $(x + y)$ odd. Also these tests suggest that the cell-by-cell updating order does not significantly influence the overall evolution. An explanation is that the rule is probabilistic, and any deterministic order is destroyed by the rule's inherent randomness. This is useful when the computation is executed on a parallel supercomputer with several computing nodes / cores. Then all cores can compute in parallel and the order of processing the border cells between the subareas (including data exchange between neighboring cores) is not critical and can be scheduled in a way that the slowdown is minimal.

6 Conclusion

A probabilistic CA rule with two options was designed that can form high quality domino patterns. The basic rule uses 24 matching templates derived from the two (3×4) domino tiles. Each selected cell is tested against the templates and is adjusted in the case that a template matches in the neighborhood. The basic rule is sub-optimal with respect to the number of placed dominoes. The evolved patterns are not always stable if there exist isolated gaps. Rule Option 1 distributes the gap information to the neighborhood, then the additional noise drives the evolution to stable patterns. Rule Option 2 injects noise where there is no overlap (overlap level 1) which drives the evolution to a maximal number of dominoes. A reached optimal pattern remains stable, if it is totally overlapping (no overlap level 1 exists). The basic rule and its options are robust against the active shape (square, diamond) and the order in which the cells are updated.

Funding Open Access funding enabled and organized by Projekt DEAL..

References

1. Chopard B, Droz M (1998) Cellular automata modeling of physical systems. Cambridge University Press, Cambridge
2. Deutsch A, Dormann S (2005) Cellular automaton modeling of biological pattern formation. Birk, East Lymeäuser
3. Désérable D, Dupont P, Hellou M, Kamali-Bernard S (2011) Cellular automata in complex matter. Complex Syst 20(1):67–91
4. Wolfram S (1983) Statistical mechanics of cellular automata. Rev Mod Phys 55(3):601–644
5. Nagpal R (2008) Programmable pattern-formation and scale-independence. In: Minai AA, Bar-Yam Y (eds) Unifying themes in complex sytems IV. Springer, Berlin, pp 275–282
6. Yamins D, Nagpal R (2008) Automated Global-to-Local programming in 1-D spatial multi-agent systems, In: Proceedings of the 7th International Joint Conference. AAMAS, (pp 615–622)
7. Tomassini M, Venzi M (2002) Evolution of asynchronous cellular automata for the density task PPSN, 2002. In: Guervós JJM, Adamidis P, Beyer HG, Schwefel HP, Fernández-Villacañas JL (eds) Parallel problem solving from nature-PPSN VII. Springer, Berlin, pp 934–943
8. Birgin EG, Lobato RD, Morabito R (2010) An effective recursive partitioning approach for the packing of identical rectangles in a rectangle. J Oper Res Soc 61:303–320
9. Temperley HNV, Fisher ME (1961) Dimer problem in statistical mechanics - an exact result. Philos Mag 6(68):1061–1063
10. Kasteleyn PW (1961) The statistics of dimers on a lattice. Physica 27:1209–1225
11. Niss M (2005) History of the Lenz-Ising model 1920–1950: from ferromagnetic to cooperative phenomena. Arch Hist Exact Sci 59:267–318
12. Hoffmann R (2014) How agents can form a specific pattern. In: Sirakoulis G, Bandini S, Wąs J (eds) Cellular automata. Springer, Cham, pp 660–669
13. Hoffmann R (2016) Cellular automata agents form path patterns effectively. Acta Phys Pol B Proc Suppl 9(1):63–75
14. Hoffmann R, Désérable D (2016) Line patterns formed by cellular automata agents. In: Bandini S, Wąs J, El Yacoubi S (eds) Cellular automata. Springer, Cham, pp 424–434
15. Hoffmann R, Désérable D (2017) Generating maximal domino patterns by cellular automata agents, PaCT 2017. In: Malyshkin V (ed) Parallel computing technologies. Springer, Cham, pp 18–31
16. Hoffmann R, Désérable D (2019) Domino pattern formation by cellular automata agents. J Supercomput 75:7799–7813
17. Hoffmann R, Désérable D, Seredyński F (2019) A probabilistic cellular automata rule forming domino patterns. In: International Conference on Parallel Computing Technologies. Springer, Cham (pp 334-344)
18. Fendler M, Grieser D (2016) A new simple proof of the Aztec diamond theorem. Gr Combinatorics 32:1389–1395
19. Achasova S, Bandman O, Markova V, Piskunov S (1994) Parallel substitution algorithm. World Scientific, Singapore
20. Désérable D (2020) On arrangement of dominoes in square and diamond and on occupancy ratio, (*personal communication*)
21. Dijkstra Edsger W (1974) Self-stabilization in spite of distributed control. Commun ACM 17(11):643–644
22. Schneider M (1993) Self-stabilization. ACM Comput Surv 25(1):45–67