Computer Physics Communications 181 (2010) 750–755

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

Complete sets of initial vectors for pattern growth with elementary cellular automata

Joana G. Freire [a,b], Owen J. Brison [b,c], Jason A.C. Gallas [a,b,d,*]

[a] Instituto de Física, Universidade Federal do Rio Grande do Sul, 91501-970 Porto Alegre, Brazil
[b] Centro de Estruturas Lineares e Combinatórias, Universidade de Lisboa, 1649-003 Lisboa, Portugal
[c] Departamento de Matemática, Faculdade de Ciências, Universidade de Lisboa, 1749-016 Lisboa, Portugal
[d] Rechnergestützte Physik der Werkstoffe IfB, ETH Hönggerberg HIF E12, CH-8093 Zürich, Switzerland

ARTICLE INFO

Article history:
Received 22 August 2009
Received in revised form 7 December 2009
Accepted 8 December 2009

Keywords:
Pattern growth
Cellular automata
Complex systems
Spacewise algorithm

ABSTRACT

Computer simulations of complex spatio-temporal patterns using cellular automata may be performed in two alternative ways, the better choice depending on the relative size between the *spatial width* W of the expected patterns and their corresponding *temporal period* T. While the traditional *timewise* updating algorithm is very efficient when $W \ll T$, the complementary *spacewise* algorithm wins whenever $T \ll W$. Independently of the algorithm used, the key to obtaining exhaustive answers, not just statistical estimates, is to have explicit knowledge of the *complete sets of initial conditions* that need to be individually tested as sizes grow. This paper reports an efficient algorithm for generating complete sets (without redundancy) of k-vectors of initial conditions allowing one to perform definitive classifications of patterns in systems with a *minimal* characteristic length k, either spatial or temporal.

1. Introduction

The simulation of complex spatio-temporal patterns using cellular automata may be performed in two complementary ways: either by applying the traditional *timewise* updating algorithm [1–5], or by applying a complementary *spacewise* updating [6,7]. In both cases, the critical information required to start simulations is a set of *initial conditions* defining the initial state of the automaton. Even after discounting trivial repetitions, the number of initial conditions grows exponentially fast with the lattice size k. For instance, for the simplest possible class of automata, namely for *binary* automata,[1] an upper bound for the number $n_u(k)$ of initial conditions that need to be explicitly investigated to find exhaustively all possible dynamical behaviors of the automaton is $n_u(k) = 2^k$. This quantity provides simultaneously an estimate of the size of the *sampling space* that needs to be probed, as well as an indication of the computational complexity of the search that needs to be performed.

In many applications, particularly in statistical physics, one is interested in the dynamics at the so-called thermodynamic limit, i.e. the limit $k \to \infty$. The problem of investigating accurately the thermodynamic limit is that the exponential growth of the dimension of the sampling space frequently prevents an *exhaustive* search of all possible patterns supported even by the most elementary binary automata. So, instead of accurate exhaustive results, the thermodynamic limit forces one to be content with just statistical estimates of the asymptotic behaviors. This difficulty by no means prevented the useful application of cellular automata to many situations of interest in several disciplines, as amply described in several books and in the technical literature [1–5].

A totally different class of applications focuses not on the thermodynamic limit but on the simulation of complex spatio-temporal patterns and, with particular emphasis, on classifying exhaustively all patterns supported by specific rules. Of great interest are temporally periodic patterns like those typically present in, say, models of crystal growth or in the so-called "gliders", i.e. in spatially localized or traveling structures characteristic of complex "class 4" automaton rules [1,4]. Three representative examples of gliders are shown in Fig. 1, generated by rule 20, a popular rule among those known for their complex dynamics [8–12]. Under rule 20, the state of any given site i at time $t+1$ depends on the state of the site at time t as well as on the state of its nearest and next-nearest neighbors through the sum

$$\Sigma_i(t) \equiv \sigma_{i-2}(t) + \sigma_{i-1}(t) + \sigma_i(t) + \sigma_{i+1}(t) + \sigma_{i+2}(t) \tag{1}$$

and is synchronously updated as follows:

$$\sigma_i(t+1) = \begin{cases} 1 & \text{if } \Sigma_i(t) \in I, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

For rule 20, the set I contains just two numbers, namely $I = \{2, 4\}$.

Fig. 1 illustrates the asymmetry $T \ll W$ frequently observed between the period T and width W of complex gliders. To quantify

* Corresponding author.
E-mail address: jgallas@if.ufrgs.br (J.A.C. Gallas).

[1] A binary automaton is one in which individual cells may assume one of two possible values.

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2009.12.007

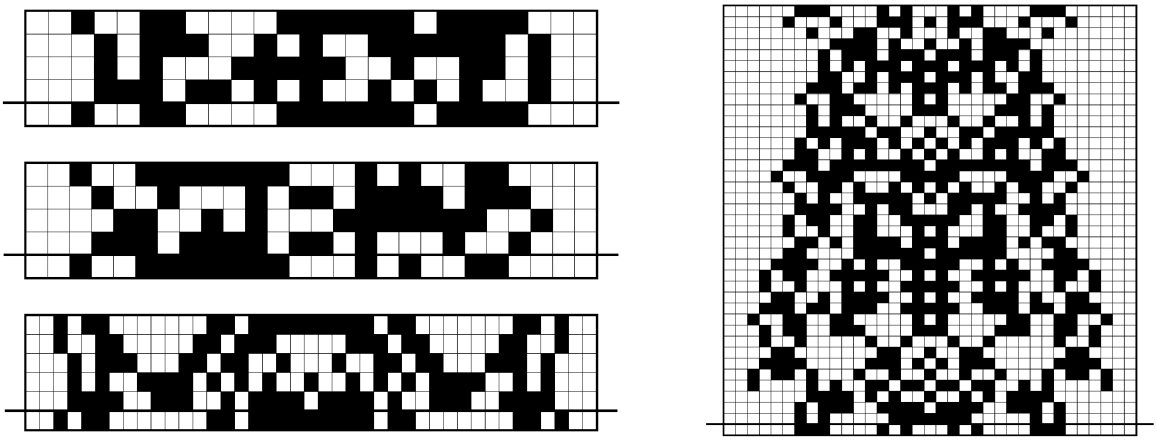

Fig. 1. Typical size asymmetries between period T and width W observed in gliders supported by rule 20. Time runs vertically downwards. Left column, from top to bottom: $(T, W) = (4, 21), (4, 22), (5, 37)$. Right column: example of a situation with $(T, W) = (38, 31)$. The left column illustrates situations where $T \ll W$ while in the right we show the only known glider with $T > W$ (see Table 1).

Table 1
Comparison between width and period of all known static patterns supported by the rule 20. Values taken from Wolfram [4], except for the situation marked with an asterisk, reported in Ref. [7]. Note that only one glider has $T > W$.

Period T	1	2	3	4	4	5	6	6*	8	10	10	10	22	38
Width W	8	9	42	21	22	37	66	73*	45	57	61	73	28	31
$W - T$	7	7	39	17	18	32	60	67*	37	47	51	63	6	−7

how "typical" this asymmetry is, Table 1 collects T and W for all known gliders of rule 20. With the exception of the glider marked by an asterisk, all data is taken from a classification presented by Wolfram [4], obtained with timewise updating. The missing glider indicated by the asterisk was found recently during a systematic search using the spacewise algorithm [7]. From Table 1 one sees that for the majority of known cases one indeed has $T \ll W$, suggesting spacewise updating to be the efficient algorithm to used in these situations.

2. The two complementary methods of automata updating

In this section we outline and contrast the two complementary methods for investigating cellular automata, namely "timewise" and "spacewise" updating.

Consider a general binary automaton, represented by a rectangular matrix of indefinite size (an "indefinite sheet of squared paper") in the lower half-plane; each cell may contain either a 0 or a 1. Temporal evolution is recorded vertically downwards while spatial dimension is recorded horizontally, increasing to the right. Denote by $\sigma_i(t)$ the state (or value) of the automaton at (spatial) site i at time t. For the evolution rules that interest us here, the state of any site i at time $t+1$ is given by a function that depends on the sum of the states at time t of the site i and its nearest and next-nearest neighbors. This sum is defined by Eq. (1) and the state of each site i is synchronously updated by Eq. (2), where I is a suitably-defined set; for example, $I = \{2, 4\}$ for rule 20, while $I = \{2, 4, 5\}$ for rule 52, etc.

The classical timewise updating algorithm for such an automaton starts with the selection of a convenient spatial dimension, say L cells, and by the imposition of the periodic boundary condition $\sigma_i(t) = \sigma_{i+L}(t)$ for all i and all t; this amounts to defining the automaton on a cylinder whose circumference has L cells. A choice is made for the state of the automaton at time $t = 0$ and the values for $t = 1$, $t = 2$, and so on, are calculated. There are of the order of $2^L/L$ choices for the state at $t = 0$: see Ref. [7] for details.

One of the questions that might be asked about such an automaton is whether there exist gliders (non-zero time-periodic structures on an all-zero background) or (time-periodic) interfaces (between the all-zero state to the left of a fixed site i and a non-zero state to the right). An exhaustive implementation of the above procedure for a given value of L and all possible choices for time $t = 0$ will probably discover a number of such gliders and interfaces. This approach has two main problems: for L moderately large, the number of choices for $t = 0$ becomes unwieldy, while the procedure will miss structures of spatial dimension greater than L. One might successively increase L in an attempt to compensate for the second problem, but this just makes the first problem worse.

Instead of the above "timewise" updating, it has proved practicable to implement "spacewise" updating in the search for gliders and interfaces, provided we first fix a temporal period k. Typically, we start on the left with $\sigma_i(t) = 0$ for $1 \leqslant i \leqslant 4$ and for all t; this corresponds to 4 vertical columns containing only the value 0. We then choose an "initial vector" v_k, a column vector with k entries which are, reading from the top, the values of $\sigma_5(t)$ for $t = 0, \ldots, k-1$. We impose the periodicity condition that $\sigma_i(t) = \sigma_i(t+k)$ for all i and all t. We then note that if $\sigma_i(t)$ is known, and if the values of $\sigma_j(t-1)$ are also known for $j = i-2, \ldots, i+1$, then the value of $\sigma_{i+2}(t-1)$ is restricted because of Eq. (2). The value of $\sigma_{i+2}(t-1)$ might be either "necessarily 0" or "necessarily 1", it might be "either 0 or 1", or it might be "impossible". In the latter case, the updating stops. In the ambiguous case, we will have to develop each possibility separately. Thus if we know all columns up to $i+1$, then as we know $\sigma_i(k)$ by periodicity, the values for column $i+2$ are restricted as just described. It might seem that this sidewise updating would itself rapidly become unmanageable because of the ambiguities, but in practice we have found [6] it to be workable. It can be (and was) even used by hand with pencil and squared paper to prove that rule 52 admits no static gliders or interfaces of minimal temporal period 4 or 5, when all sidewise updatings ended in the "impossible" case. It is the purpose of the present work to describe an algorithm, recursive in k, to systematically generate all possible initial vectors v_k mentioned above.

Table 2
The number $n(k)$ of independent initial conditions that need to be spatially grown to ascertain unambiguously the periodicity, as a function of the characteristic length k, with or without considering divisors of k. The number $n_{tot}(k)$ includes all possible divisors, while $n_u(k) \equiv 2^k$ gives the upper bound for $n_{tot}(k)$, including all possible redundant cyclic permutations. The last line gives the net "gain" $g(k) \equiv n_u(k) - n_{tot}(k)$.

k	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$n(k)$	1	2	3	6	9	18	30	56	99	186	335	630	1161	2182
$n_{tot}(k)$	3	4	6	8	14	20	36	60	108	188	352	632	1182	2192
$n_u(k) \equiv 2^k$	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16 384	32 768
$g(k)$	1	4	10	24	50	108	220	452	916	1860	3744	7560	15 202	30 576

3. The complexity of testing all initial conditions

How could a glider have been missed during the timewise search mentioned above? We argue the answer to be likely related with the total number $n(k)$ of initial conditions (without repetitions) that need to be investigated in order to find rigorously all possible periodic patterns with a "characteristic length" k, either temporal or spatial.

For elementary 1D totalistic (deterministic) cellular automata and independently of the automaton rule, we have recently shown that $n(k)$ is given by

$$n(k) = \frac{1}{k}\sum_{d|k} \mu\left(\frac{k}{d}\right)\left(2^d - 1\right), \tag{3}$$

where $\mu(k/d)$ is the Möbius function [7]. This expression is valid for both timewise and spacewise algorithms. It does not contain the divisors d of k. For each divisor d of k one needs to investigate an additional $n(d)$ possibilities, making the total number of initial conditions to be tested to grow to

$$n_{tot}(k) = 1 + \sum_{d|k} n(d), \tag{4}$$

where the extra 1 counts the trivial vector $(0, \ldots, 0)$.

Now, from Table 1 one sees that the missing glider has $T = 6$ and $W = 73$. This means that even considering just lower bounds, the number of initial conditions that one needs to investigate using spacewise and timewise updating are, respectively,

$$n_{tot}(6) - 1 = 13, \tag{5}$$

$$n_{tot}(73) - 1 = 129\,379\,903\,640\,264\,252\,431 \simeq 10^{20}, \tag{6}$$

without counting the extra 4 sites needed by the timewise updating to enforce periodic boundary conditions and subtracting 1 from $n_{tot}(k)$, to account for the trivial initial condition $(0, 0, \ldots, 0)$. Thus, while spacewise updating requires studying the dynamical evolution for 13 initial conditions, timewise updating requires studying of the order of 10^{20} conditions.

The huge number of initial conditions normally required when dealing with timewise updating made totally superfluous questions concerning size and completeness of the set of initial conditions. However, independently of the updating algorithm used, there are situations where one can provide exhaustive answers. In Table 2 we compare $n(k)$ and $n_{tot}(k)$ with $n_u(k) \equiv 2^k$, the upper bound of $n_{tot}(k)$. The table also contains the net "gain" $g(k)$, i.e. the number of *non-primitive* initial conditions that do not need to be considered because they would just repeat situations already represented. Note that while for $k = 10$ one only needs to test about 10% of the total initial conditions, for $k = 15$ this number already drops to 6.6%. As mentioned above, the strong reduction as k increases allows one to perform definitive classifications for a number of problems without having to rely on statistical estimates. The purpose of this paper is to describe an efficient algorithm that allows one to sort out efficiently the *minimal* set of initial conditions that need to have their dynamics investigated explicitly.

4. The generation of initial conditions

The aim of this section is to describe the algorithm for producing without redundancy the *minimal set* $I(k)$ of initial "k-vectors" or, equivalently, "k-seeds" that must to be investigated individually in order to classify exhaustively all periodic patterns having a characteristic length k (either temporal or spatial) for a given binary cellular automaton of interest. The total number of k-vectors in the minimal set $I(k)$ is given by Eq. (4). The algorithm produces the $(k+1)$-seeds from a knowledge of the k-seeds. This recursive procedure was tested explicitly up to $k = 25$ and found to be quite efficient. It should be noted that $I(k+1)$ is less than double the size of $I(k)$, and that it is trivial to start the recursive process.

4.1. Preliminaries

Because cyclic permutations of a k-seed give the same result, it is clear that the first three minimal sets $I(k)$ of initial conditions can be taken as

$$I(1) = \big\{(1), (0)\big\}, \tag{7}$$

$$I(2) = \big\{(1, 1), (1, 0), (0, 0)\big\}, \tag{8}$$

$$I(3) = \big\{(1, 1, 1), (1, 1, 0), (1, 0, 0), (0, 0, 0)\big\}. \tag{9}$$

This shows that the number of 1-, 2-, and 3-vectors are, respectively, 2, 3, and 4. In general, a k-vector may be represented by $u = (a_1, \ldots, a_k)$ with entries a_j from $\mathbb{F}_2 = \{0, 1\}$. The main goal of the algorithm is to produce $I(k+1)$ from knowledge of $I(k)$. When no confusion arises, we will say "vector" instead of k-vector.

A very useful property of a pair of k-vectors $u = (a_1, \ldots, a_k)$ and $v = (b_1, \ldots, b_k)$ is that they may be always ordered *lexicographically*. In other words, we may write $u > v$ if there exists $j \geqslant 1$ such that $a_j = 1$ and $b_j = 0$ while $a_i = b_i$ if $i < j$. For example,

$$(1, 0, 1, 1, 0, 1, 1) > (1, 0, 1, 0, 1, 1, 1)$$

because $a_1 = b_1$, $a_2 = b_2$, $a_3 = b_3$, $a_4 = 1$, $b_4 = 0$. We will write $u \geqslant v$ if either $u > v$ or $u = v$. Then $\geqslant$ is the so-called "lexicographic" order [13] on the set of k-vectors with entries in $\mathbb{F}_2$. Given k-vectors u and v, either $u > v$ or $v > u$ or $u = v$. The set of initial conditions $I(k)$ will be "minimal" in the sense that all its elements are always distinct: $u \neq v$. It is well known that any non-empty set of k-vectors with entries from $\mathbb{F}_2$ has a maximum (or greatest) element with respect to this order [13].

We will say that two k-vectors are *equivalent* if one may be obtained from the other by a suitable number (between 0 and $k-1$ inclusive) of applications of the cyclic permutation $(1, 2, \ldots, k)$. For example, the vectors

$$u_1 = (1, 1, 0, 1), \tag{10}$$

$$u_2 = (1, 1, 1, 0), \tag{11}$$

$$u_3 = (0, 1, 1, 1), \tag{12}$$

$$u_4 = (1, 0, 1, 1) \tag{13}$$

form a complete class of equivalent vectors which is ordered lexicographically as follows:

$$u_2 > u_1 > u_4 > u_3.$$

We take u_2, the maximum element with respect to the lexicographic order, as the *representative* of the class. It is clear that, apart from the "exceptional" vectors $(1,\ldots,1)$ and $(0,\ldots,0)$, a chosen representative $(a_1,\ldots,a_k)$ of an equivalence class will always have $a_1 = 1$ and $a_k = 0$.

From Eqs. (7)–(9) one might have the impression that $I(k+1)$ may be obtained from $I(k)$ by adjoining a 1 to the left side of each vector of $I(k)$ and finally adding the vector $(0,\ldots,0)$. However, $I(4)$ is given by

$$I(4) = \{(1,1,1,1), \tag{14}$$
$$(1,1,1,0), \tag{15}$$
$$(1,1,0,0), \tag{16}$$
$$(1,0,1,0), \tag{17}$$
$$(1,0,0,0), \tag{18}$$
$$(0,0,0,0)\}. \tag{19}$$

From this, one sees that the vector $(1,0,1,0)$ is not obtained from $I(3)$ by the above procedure. Note that $(1,0,1,0)$ is a repetition of the 2-vector $(1,0)$. In other words, we may consider that $(1,0)$ is a *divisor* of $(1,0,1,0)$ as 2 divides 4.

We say that a k-vector v has *minimal period* d if d is the smallest number such that v is invariant under a cyclic-shift of d places; such a d must be a divisor of k.

The set $I(k)$ is formed by the union of two disjoint sets

$$I(k) = N(k) \cup D(k). \tag{20}$$

The first set, $N(k)$, is the set of all initial vectors which correspond to minimal period k. That is, $N(k)$ is the complete set of chosen representatives of the various classes of those vectors that correspond to minimal period k. The second set $D(k)$ is related with the divisors of k. If d is a divisor of k with $1 \leqslant d < k$ then certain k-vectors correspond to minimal period d. From each equivalence class of vectors with minimal period d where $1 \leqslant d < k$ we choose the maximal vector with respect to the lexicographic order as the representative, and write $D(k)$ for the set of all such representatives. We include the case $d = 1$, which corresponds to the k-vector $(1,\ldots,1)$. The vector $(0,\ldots,0)$ belongs to $D(k)$ because it is invariant under shifts by 1 place.

For any $v \in I(k)$ we define $b \equiv b(v)$ as the size of a maximal block of *cyclically-consecutive* entries which are all equal to 1 in the vector v. For example, if $v = (1,0,0,1,1)$ then $b = 3$ because, due to the boundary conditions, we consider the three entries "1" to be cyclically-consecutive, while if $v = (1,0,1,1,0)$ then $b = 2$.

Next, for a given $v \in I(k)$ one needs to be able (1) to calculate b; (2) to locate all blocks of $b(v)$ cyclically-consecutive 1s in v; (3) provided that $b \geqslant 1$, to find the blocks of exactly $b-1$ cyclically-consecutive entries equal to 1, if any. In this context, a "block of exactly 0 entries equal to 1" is defined to be an entry equal to 0.

It may happen that v possesses more than one block of b cyclically-consecutive entries equal to 1. For example, if $v = (1,1,1,0,1,1,1,0,1,0,0)$ then $b = 3$ and v possesses 2 blocks of 3 cyclically-consecutive entries equal to 1.

If v possesses just one block of b cyclically-consecutive entries equal to 1, we will say that v possesses a "unique maximal block" and we will say that v is a UMB vector. For example, $v_1 = (1,0,1,1,1,0,1,0,0,1)$ is a UMB vector with $b = 3$.

If a vector v is a UMB vector and if its unique maximal block is situated at the beginning (i.e. at the left-hand end) of v, then v must be an initial vector because v is greater (in the lexicographic order) that any shift of v. For example, $v_2 = (1,1,1,0,1,0,0,1,1,0)$ is an initial vector of $I(10)$ with $b = 3$. Obviously, v_2 may be obtained by performing two shifts on v_1 above.

If a vector v is initial it must commence with a maximal block of b consecutive entries equal to 1, because of the lexicographic order, but it need not be a UMB vector. For example, $v = (1,1,1,0,1,1,1,0,0,0)$ is an initial 10-vector. Here $b = 3$ and v has two blocks of 3 cyclically-consecutive entries equal to 1.

4.2. The algorithm

It is assumed that a lexicographically ordered $I(k)$ is known for some $k \geqslant 1$. As mentioned before, the aim is to generate an ordered $I(k+1)$ from the knowledge of $I(k)$.

For convenience, during the computation we will use $I(k+1)$ as a place-holder for the vectors generated, possibly containing redundancies. Such redundancies will be eliminated by performing a lexicographic ordering at the end.

Start by adjoining a 1 to $(0,\ldots,0) \in I(k)$ to get $(1,0,\ldots,0) \in I(k+1)$. Add the $(k+1)$-vector $(0,\ldots,0)$ to $I(k+1)$.

Proceeding in lexicographic order, for each $v = (a_1,\ldots,a_k) \in I(k)$, $v \neq (0,\ldots,0)$, determine the value of $b = b(v)$. Further, calculate the minimum period, d, to which v corresponds; d is necessarily a divisor of k (possibly equal to k). Store this minimum period d. By definition of $b(v)$, v has one or more maximal blocks of exactly $b = b(v)$ consecutive 1s. In particular, provided $v \neq (0,\ldots,0)$, v must begin with a maximal block of $b \geqslant 1$ consecutive entries equal to 1. After this, the algorithm consists essentially of performing two tasks:

(a) Shift v successively through $0,1,\ldots,i,\ldots,d-1$ places to obtain vectors $v_0 = v, v_1, \ldots, v_i, \ldots, v_{d-1}$. For each $i = 0,\ldots,d-1$, check to see if v_i starts with a block of b consecutive 1s.[2] If v_i starts with a block of b consecutive 1s, adjoin a 1 to obtain $(1, v_i)$. This vector must start with a unique maximal block of $(b+1)$ 1s and so must be initial. Place this vector $(1, v_i)$ into $I(k+1)$. If v_i does not start with a block of b consecutive 1s, proceed to the next i.

(b) Check whether v has one or more blocks of exactly $b-1$ consecutive 1s. If so, then shift v an appropriate number of times up to a maximum of $d-1$ places so that each such block, in its turn, appears at the front. After each such shift, test to see whether the resulting vector, v', ends in a 0 or a 1. If the vector v' ends in 1, then discard it[3]; else create a $(k+1)$-vector v'_1 by adjoining a 1 at the front. After that: Test to see whether v'_1 is initial or not: If it is not initial, discard it.[4] If v'_1 is initial, add it to the set $I(k+1)$.

Proceed to the next $v \in I(k)$.

Finally, put the vectors of $I(k+1)$ into lexicographic order. This ends the algorithm. Note that in step (a), all $(k+1)$-vectors produced are distinct and are UMB vectors. Step (b) never produces

[2] Of course, v_0 does.

[3] It might seem that this should not occur; it will not if $b \geqslant 2$, but might occur if $b = 1$ because a block of exactly 0 1s is, by our definition, a 0. For example, $(0,1,0) \to (1,0,1,0)$ but $(0,0,1)$ is discarded. The potential new vector if $(0,0,1)$ was not discarded, $(1,0,0,1)$, is equivalent to the vector we obtain by adding 1 to the front of $(1,0,0)$.

[4] There is no need here to shift v'_1 to get an initial vector in $I(k+1)$: such an initial vector will arise elsewhere. For example, suppose $v = (1,1,0,1,0,0) \in I(6)$. Here, $b(v) = 2$. Shift to $v' = (1,0,0,1,1,0)$. Adjoin 1 to get $v'_1 = (1,1,0,0,1,1,0)$. This vector is not initial: its initial representative is $w = (1,1,0,1,1,0,0)$. However, w arises by adjoining 1 to $(1,0,1,1,0,0)$ and this is a shift of $(1,1,0,0,1,0) \in I(6)$ and so we will obtain w anyway.

UMB vectors and so there is no possibility that vectors from (a) and (b) might coincide.

Examples.

(i) Take $v = (1, \ldots, 1) \in I(k)$. Then step (a) gives $(1, 1, \ldots, 1) \in I(k+1)$.
(ii) Take $v = (1, 1, 0, 1, 1, 0, 0) \in I(7)$. Then step (a) gives distinct vectors $(1, 1, 1, 0, 1, 1, 0, 0)$ and $(1, 1, 1, 0, 0, 1, 1, 0) \in I(8)$.
(iii) Consider $k = 14$ and $v = (1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0) \in I(14)$, period $d = 7$. We append a 1 to the front of v to get $(1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0) \in I(15)$. If we were to shift v by 7 places the result would be v again, which is of no further interest; we avoided this by only shifting v through $0, 1, \ldots, d-1$ places. However, a shift of 3 places gives $v_3 = (1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0) \neq v$; when we adjoin a 1 to the front of v_3 we get $(1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0) \in I(15)$, and this is different to the vector found above. All other possible shifts will give either v or v_3 or else a vector that does not start with a block of 2 1s. For example, a shift of 10 places gives $(1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0)$.

Example. Take $v = (1, 0, 0, 1, 0, 1, 0)$, with $b = 1$. Each of the 4 entries equal to 0 counts as a block of $b - 1 = 0$ consecutive 1s. We will shift each in turn to the front:

$(0, 0, 1, 0, 1, 0, 1)$

$(0, 1, 0, 1, 0, 1, 0)$

$(0, 1, 0, 1, 0, 0, 1)$

$(0, 1, 0, 0, 1, 0, 1)$.

Of these, we discard the first, third and fourth. We take the second, $v' = (0, 1, 0, 1, 0, 1, 0)$, and adjoin a 1 at the front: $v'_1 = (1, 0, 1, 0, 1, 0, 1, 0)$. This vector is initial and has $d = 2$; we place it in $I(8)$.

Step (b) in the above algorithm may be replaced by the following alternative step:

(b) (*alternative*) Check whether v has one or more blocks of exactly $b-1$ consecutive 1s. If so, then if $b = 1$ then shift v (up to a maximum of $d-1$ places) so that each 0 in succession appears at the front, to give vectors generically called v'. If v' ends in 1 then discard it. Otherwise create a $(k+1)$-vector v'_1 by adjoining 1 to the front of v'. Test to see if v'_1 is initial; if so, place it in $I(k+1)$ and if not, discard it. On the other hand, if $b \geqslant 2$ shift v (up to a maximum of $d-1$ places) so that each block of $b-1$ cyclically-consecutive 1s, in its turn appears at the front.[5] On each occasion, create a $(k+1)$-vector v'_1 by adjoining a 1 at the front of v'. Check to see whether v'_1 is initial. If it is initial, add it to the set $I(k+1)$, otherwise discard it.

Proceed to the next $v \in I(k)$.

To conclude this section, we remark that although convenient for humans, lexicographic order is not strictly necessary for the machine. In fact, the ordering may be omitted without altering the final results. What matters is that each vector be a representative of its class.

5. Validation

The purpose of this section is to show that the above algorithm: (1) produces $I(k+1)$ containing a complete set of initial vectors, and (2) does not give repetitions.

(1) Suppose that $v_1 \in I(k+1)$, $v_1 \neq (1, 0, \ldots, 0)$ and $v_1 \neq (0, 0, \ldots, 0)$. We wish to show that v_1 arises from some vector $v \in I(k)$ in either (a) or (b) above. We may also assume $v_1 \neq (1, 1, \ldots, 1)$ because this vector is produced by (a) from the k-vector $(1, \ldots, 1) \in D(k)$.

Thus v_1 starts with $b = b(v_1) \geqslant 1$ consecutive 1s, followed by at least one 0.

If there are no other blocks of b consecutive 1s in v_1 then v_1 arises by adjoining a 1 to the left of a k-vector v where $b(v) = b-1$ and such that v starts with a block of $b-1$ consecutive 1s; such a vector v can, if necessary, be shifted to a vector $v^* \in I(k)$ and so v_1 will have been constructed in (a) or (b).

Suppose that v_1 possesses further blocks of b consecutive 1s. Again, v_1 arises by appending a 1 to the left of a k-vector v. In this case, $b(v) = b$ and v must have started with a block of $b-1$ consecutive 1s; such a vector v can be shifted to give a vector v^* which starts with a block of b consecutive 1s and which belongs to $I(k)$, so that v_1 arises in (b).

(2) In (a) and (b) we produce new vectors by appending a 1 to the left of vectors which are shifts, by at most $d-1$ places (where d is the minimum period) of vectors in $I(k)$. Suppose the vector v_1 has been produced by appending a 1 to the left of a k-vector v. Then v is equivalent to a vector $v^* \in I(k)$, and arises only once in the process of successively shifting v^* by $0, 1, \ldots, d-1$ places. It follows that the vectors produced in the algorithm are distinct among themselves.

6. Algorithm overview

We now give an abridged description of the algorithm of Section 4.2. This abridged version generates the same set generated by the algorithm of Section 4.2. It is however shorter because it relies on concepts and notation introduced in the previous sections. As above, the goal is to generate $I(k+1)$ assuming that the lexicographically ordered set $I(k)$ is known.

Apart from the "exceptional" vectors $(1, \ldots, 1)$ and $(0, \ldots, 0)$, a representative vector $(v_1, \ldots, v_k)$ will always have $v_1 = 1$ and $v_k = 0$.

The algorithm consists of the following steps:

1. Adjoin 1 to the exceptional vector $(0, \ldots, 0)$ to get $(1, 0, \ldots, 0)$ and place this vector in $I(k+1)$. Place the $(k+1)$-vector $(0, \ldots, 0)$ into $I(k+1)$.
2. For each vector $v \in I(k)$ we need to:
   (a) Calculate the minimal period d of v.
   (b) Calculate $b \equiv b(v)$, defined as the size of a maximal block of cyclically-consecutive entries which are all equal to 1 in the vector v.
   (c) Locate all blocks of b cyclically-consecutive 1s in the first d places of v.
   (d) Provided that $b \geqslant 1$, locate all the blocks of exactly $b-1$ cyclically-consecutive entries equal to 1 in the first d places of v.
   (e) Shift each block of size b found in (c) to the leftmost position and produce new vectors $v_1 \in I(k+1)$ by appending a 1 to the left of the shifted vectors $v \in I(k)$. Drop all non-initial vectors, if any.
   (f) If $b \geqslant 1$ and if $(b-1)$-blocks exist, repeat (e) but for all the blocks with $b-1$ cyclically-consecutive entries equal to 1 found in (d). Drop all non-initial vectors, if any.
3. Order $I(k+1)$ lexicographically.

7. Conclusions and outlook

In conclusion, we have presented an explicit algorithm to generate recursively *complete sets of initial conditions* that need to be

[5] After each rotation, the resulting vector, v', must end in a 0 because $b > 1$.

tested individually in order to obtain exhaustive classification of patterns, not mere statistical estimates. The algorithm may be applied equally well to classify spatial or temporal patterns in systems with a *minimal* characteristic length k. From the discussion in Section 3 and a comparison between the numbers given in Eqs. (5) and (6) one realizes that the choice of an adequate updating algorithm plays a decisive role: while the thermodynamic limit is certainly out of reach for exhaustive classifications, much remains to be discovered regarding the classification of periodic patterns with finite sizes. In particular, spacewise updating might be now efficiently used to grow patterns under rules of arbitrary complexity. A recent systematic search combining the spacewise updating algorithm with a brute force scan of the initial conditions has produced unexpected results, revealing patterns that were so far overlooked in previous classifications [7]. Now, using the minimal set of initial conditions as described here, it should be possible to conduct much wider searches of complex patterns with longer periods. It should be also possible to adapt the spacewise updating algorithm to classify automatically *traveling gliders* and to investigate what sort of gliders are capable of surviving repeated collisions in the lattice and, thus, may be used as carriers of useful information across extended spatial domains. The present algorithm combined with spacewise search is also expected to help finding exhaustive answers concerning the spatio-temporal organization of cellular automaton models of interesting applications such as, for example, computer networks [14–17], secure schemes to share encrypted color images [18], modeling of biological patterns and processes like, e.g., tumor growth [19,20], efficient means of simulating mixing and segregation of granular media [21] as well as in a number of fundamental open questions in physics [22–31].

Acknowledgements

J.G.F. thanks Fundação para a Ciência e Tecnologia, Portugal, for a Postdoctoral Fellowship and Instituto de Física da UFRGS for hospitality. O.J.B. and J.A.C.G. thank the Centro de Estruturas Lineares e Combinatórias, Universidade de Lisboa, for partial support. J.A.C.G. thanks Hans J. Herrmann for a fruitful month spent in his Institute in Zürich. He is supported by the Air Force Office of Scientific Research, Grant FA9550-07-1-0102, and by CNPq, Brazil. The authors thank CESUP-UFRGS for access to the Sun Fire X2200 and X4600 clusters.

Appendix A. MAPLE code for computing $n(k)$ and $n_{tot}(k)$

The following MAPLE code evaluates Eqs. (3) and (4) and may be used to extend Table 2. The program is set here to generate all values of $n(k)$ and $n_{tot}(k)$ up to $k = 73$, the value given in Eq. (6). It runs in a small fraction of a second. It is instructive to run it and see how fast these numbers grow.

```
with(numtheory):
n(1) := 1:
n(2) := 1:
for k from 3 to 73  do
  pdiv := divisors(k)[1..(tau(k)-1)]:
      sr(k) :=0:
      ntot(k) :=1:
  for d in pdiv do sr(k):=sr(k)+d*n(d) od:
      n(k) := (2^k - (sr(k)+1))/k:
  pdiv := divisors(k);
  for d in pdiv do ntot(k):=ntot(k)+n(d) od:
  print(  k, n(k), ntot(k)  );
od:
```

Alternatively, one may also use a MAPLE function to obtain $n(k)$ as follows:

```
with(numtheory):
for k from 1 to 73  do
  pdiv := divisors(k);   soma(k) := 0:
  for d in pdiv do
    soma(k):=soma(k)+mobius(k/d)*(2^d - 1) od:
  n(k) := soma(k)/k:
  print(  k, n(k)  );
od:
```

References

[1] B. Chopard, M. Droz, Cellular Automata Modeling of Physical Systems, Cambridge University Press, Cambridge, 1998.
[2] R. Badii, A. Politi, Complexity: Hierarchical Structures and Scaling in Physics, Cambridge University Press, Cambridge, 1997.
[3] A. Ilachinski, Cellular Automata, World Scientific, Singapore, 2001.
[4] S. Wolfram, A New Kind of Science, Wolfram Media, Champaign, 2002.
[5] For recent surveys see: A.E. Motter, Z. Toroczkai (Eds.), Focus Issue: Optimization in Networks, Chaos 17 (2007);
S. Havlin, M. Nekovee, Y. Moreno (Eds.), Focus on Complex Networked Systems: Theory and Application, New J. Phys. 9 (2007).
[6] J.G. Freire, O.J. Brison, J.A.C. Gallas, Chaos 17 (2007) 026113.
[7] J.G. Freire, O.J. Brison, J.A.C. Gallas, J. Phys. A: Math. Theor. 42 (2009) 395003.
[8] S. Wolfram, Physica D 10 (1984) 1.
[9] J.A.C. Gallas, H.J. Herrmann, Internat. J. Modern Phys. C 1 (1990) 181.
[10] J.A.C. Gallas, P. Grassberger, H.J. Herrmann, P. Ueberholz, Physica A 180 (1992) 19.
[11] J.A.C. Gallas, H.J. Herrmann, Physica A 356 (2005) 78.
[12] J.G. Freire, J.A.C. Gallas, Phys. Lett. A 366 (2007) 25.
[13] G. James, A. Kerber, The Representation Theory of the Symmetric Group, Addison-Wesley, New York, 1981, p. 23.
[14] Z. Ren, Z. Deng, Comput. Phys. Comm. 144 (2002) 243, 310.
[15] D.A. Meyer, Comput. Phys. Comm. 146 (2002) 295.
[16] C.R. Calidonna, S. Di Gregorio, M.M. Furnari, Comput. Phys. Comm. 147 (2002) 724.
[17] D.R. Paula, A.D. Araujo, J.S. Andrade, H.J. Herrmann, J.A.C. Gallas, Phys. Rev. E 74 (2006) 017102.
[18] G. Alvarez, A. Hernández Encinas, L. Hernández Encinas, A. Martín del Rey, Comput. Phys. Comm. 173 (2005) 9.
[19] A. Deutsch, S. Dormann, Cellular Automaton Modeling of Biological Pattern Formation, Birkhäuser, Basel, 2005.
[20] P. Gerlee, A.R.A. Anderson, J. Theoret. Biol. 259 (2009) 67.
[21] D.V. Ktitarev, D.E. Wolf, Comput. Phys. Comm. 121–122 (1999) 303.
[22] B.L. Costello, R. Toth, C. Stone, A. Adamatzky, L. Bull, Phys. Rev. E 79 (2009) 026114.
[23] T. Kahle, E. Olbrich, J. Jost, N. Ay, Phys. Rev. E 79 (2009) 026201.
[24] F. Bagnoli, R. Rechtman, Phys. Rev. E 79 (2009) 041115.
[25] C.R. Shalizi, R. Haslinger, J.-B. Rouquier, K.L. Klinkner, C. Moore, Phys. Rev. E 73 (2006) 036104.
[26] H. Janicke, A. Wiebel, G. Scheuermann, W. Kollmann, IEEE Trans. Visualization Computer Graphics 13 (2007) 1384.
[27] J.T. Lizier, M. Prokopenko, A.Y. Zomaya, Phys. Rev. E 77 (2008) 026110.
[28] M. Prokopenko, F. Boschetti, A.J. Ryan, Complexity 15 (2009) 11.
[29] A.B. Bracic, I. Grabec, E. Govekar, Eur. Phys. J. B 69 (2009) 529.
[30] B. Fernandez, B. Luna, E. Ugalde, Phys. Rev. E 80 (2009) 025203.
[31] M. Gonzalez, D. Dominguez, F.B. Rodriguez, Neurocomputing 72 (2009) 3795.