# Optimizing Wealth by a Game within Cellular Automata

Rolf Hoffmann<sup>1</sup>, Franciszek Seredyński<sup>2</sup>, Dominique Désérable<sup>3</sup>

 <sup>1</sup> Technical University Darmstadt, Germany hoffmann@informatik.tu-darmstadt.de
 <sup>2</sup> Institute of Computer Science, University of Siedlce, Poland franciszek.seredynski@uws.edu.pl
 <sup>3</sup> Institut National des Sciences Appliquées, Rennes, France domidese@gmail.com

#### Abstract

The objective is to find a Cellular Automata (CA) rule that can evolve 2D patterns that are optimal with respect to a global fitness function. The global fitness is defined as the sum of local computed utilities. A utility or value function computes a score depending on the states in the local neighborhood. First the method is explained that was followed to find such a CA rule. Then this method is applied to find a rule that maximizes social wealth. Here wealth is defined as the sum of the payoffs that all players (agents, cells) receive in a prisoner's dilemma game, and then shared equally among them. The problem is solved in four steps: (0) Defining the utility function, (1) Finding optimal master patterns with a Genetic Algorithm, (2) Extracting templates (local neighborhood configurations), (3) Inserting the templates in a general CA rule. The constructed CA rule finds optimal and near-optimal patterns for even and odd grid sizes. Optimal patterns of odd size contain exactly one singularity, a 2 x 2 block of cooperators.

**Keywords:** Pattern Optimization, Pattern Formation, Pattern Matching, Prisoner's Dilemma, Social Wealth, Probabilistic Cellular Automata, Matching Templates, Tilings.

## Contents

1	Introduction							
2	The problem and its solving method							
	2.1	The problem	5					
	2.2	The problem-solving method	6					
3	A special utility function: the payoff of a game							
	3.1	Prisoner's dilemma game with two players	7					
	3.2	Spatial PD game with K opponents	8					
	3.3	The pattern's value for the utility of the PD game	9					
	3.4	Estimation of the maximal reachable wealth	9					
4	Using a Genetic Algorithm to find optimal patterns							
5	Found optimal patterns							
	5.1	Patterns of size 6 x 6	12					
	5.2	Patterns of size 5 x 5	14					
	5.3	Patterns of size 7 x 7	16					
	5.4	Patterns' characteristics for odd $n > 7$	17					
6	Extracting templates from the optimal patterns 1							
7	The CA rule using the templates							
8	Patterns evolved by the CA rule							
	8.1	Even pattern sizes	23					
	8.2	Odd pattern sizes	26					
a	Cor	nelusion	21					

#### 1 Introduction

The general topic is to form certain patterns of colored cells arranged in a grid/field of discrete points. We aim at patterns that are optimal with respect to a global objective/fitness function over the field of cell states/colors. Specifically, we want to find the global optimum by the use of local operations only. Cellular Automata (CA) act locally and effect globally, and it is well known that the iterated application of a local rule can form complex patterns. Therefore we try to solve our problem with CA.

The goal is to find a CA rule that can form 2D patterns which are optimal with respect to a given fitness function. The fitness function shall be the sum of *local value functions* (also called *utilities*) computed by every cell taking into account their neighboring states. In this work we define the utility as the payoff/reward that a cell/agent/player receives out of a game played with his neighbors, the Prisoner's Dilemma Game. Note that only the CA rule determines the actions (cell state changes) of the "game" and there are no separate players. The payoff function of the game is only used as an example for the definition of a utility function.

In this article we use the following terms. A *cell* is a entity at a certain site in a 2D grid, compromising a *state*, and an *automaton* (a *rule*) with *connections* to its local neighbors. A array of cells is termed *Cellular Automata* (CA). A CA cell is also called *agent*, because it can change states, temporary variables, and optionally some states of the environment. The agents used here are simple cells that change their own state only, but they can compute in addition the *utility* function, a value for a certain local neighborhood configuration.

Interpretation of our special problem. The pattern we aim at will be a mixture of zeroes (cooperators in a game, displayed in white/green/yellow), and ones (defectors, displayed in black/blue). An optimal pattern will contain around 1/4 defectors and 3/4 cooperators locally arranged as "points"  $\binom{000}{010}$  and dominoes  $\binom{0000}{0110}$  of defectors. We can interpret the pattern as a community of agents, defectors and cooperators. We can name the defectors "dealers" or enterprisers, and the cooperators "consumers" or workers. We know from economics that the prosperity or wealth of the whole community increases if there is a certain amount of dealers among all agents. We require that the incomes/utilities of all dealers and all consumers of the community are shared, equally distributed among all of them. So we consider a systems where all agents behave fully social. The system tries to optimize its wealth as a whole and without giving any preference to individuals.

**Related Work.** The *Prisoner's Dilemma* is very well known and studied in the theory of games [1, 6, 24, 25, 26, 28, 29]. The prisoner dilemma's game is used as a mean to optimize systems and patterns in [20, 30, 32, 31, 33]. In contrast, the PD game is used here only to define a the utility function that in turn defines the fitness and structure of a pattern.

Cellular Automata. A CA is a grid of automata (cells). Every cell changes its state depending on its own state and the state of its local neighbors according to a local rule. The model of computation is parallel, simple, powerful, and has

a wide range of applications [35].

Genetic Algorithm (GA) for Pattern Generation. We used already a GA to generate patterns with a global fitness function based on local conditions and local pattern matching [17]. GA is a generally accepted method for optimization. It dates back to John H. Holland [18], some overviews are presented in [3, 19, 21]. We just used the classical techniques (crossover, mutation, selection) in a simple way, i.e., we randomly selected the second parent (the mate) without giving preference to parents with high fitness. This was sufficient for our purpose and it was not the intention to find a more efficient evolutionary algorithm or heuristic. This technique of finding optimal patterns with respect to a high rate of given local configurations was also used in previous works [7, 8, 9, 10] in which finite state machines for moving agents were evolved aiming at certain pattern structures.

Finding a CA rule. The difficulty is to find an effective and efficient CA rule that can solve a given global problem/objective, like density classification or natural phenomena. This issue is also called "global-to-local problem" [23, 36]. There are different ways to find a rule, such as

- Designing it by hand, for instance constructing self-replication rules [4], or by discretizing differential equations.
- Finding it with an evolutionary algorithm, like it was done in [22, 5, 7, 9].
- Designing it in a methodical way,
  - partially by hand and partially tool-supported. This work belongs to this category.
  - Or mainly automated using a tool-chain, like in [23, 36, 2].

In our previous work [11, 12, 13, 14, 15, 16, 17] CA rules were designed in a methodical way using templates (local matching patterns). The templates are derived from a given set of tiles that define the pattern class. The tiles are allowed to overlap, therefore the number of tiles can vary and the evolved patterns can easily adapt to different boundary structures. In [16] so-called "point patterns" are evolved under cyclic boundary conditions. These patterns contain points  $\begin{pmatrix} 000 \\ 000 \end{pmatrix}$  which cover the 2D space with a density that is not fixed, i.e. items of the zero-hull may overlap.

In this work, on the contrary, we shall evolve *dense* patterns of points (for even grid sizes), and points mixed with dominoes (for odd grid sizes). The concept of overlapping tiles is not used here, the patterns are evolved only on the basis of templates which are conditions that strictly have to be fulfilled in the final patterns.

## 2 The problem and its solving method

## 2.1 The problem

We assume a 2D grid of cells/agents (i,j) with  $i,j=0 \dots n-1$  and  $N=n^2$ . Each cell has a state (color)  $a_{ij} \in \{0,1\}$ . The array of cell states defines a field/pattern/configuration  $A=(a_{ij})$ . We use the following definitions and notations:

•  $a_{ij}^{+K} = (a_{ij}^0, a_{ij}^1, \dots, a_{ij}^k, \dots, a_{ij}^{K-1})$ 

The neighbors' states (or the neighborhood configuration) of (i, j).

The number of neighbors is K. The  $k^{th}$  neighbor's state is denoted by  $a_{ij}^k$ . In addition, the relative position of each neighbor k has to be declared. Usually the own cell's state  $a_{ij}$  is included, then we can declare  $a_{ij}^0 = a_{ij}$ . But it is possible to exclude it, then only the states of the *outer* neighborhood are considered.

- Remark. Another way to define the neighborhood configuration of  $a_{ij}$  is

$$a_{ij+P} = a_{ij+(P_0, P_1, \dots P_k \dots, P_{K-1})}$$

where P is the neighborhood index vector, with the elements (pointers, vectors)  $P_k$  defining the relative position  $(\Delta i, \Delta j)$  of the  $k^{th}$  neighbor. Then  $a_{ij+P_k} = a_{ij}^k$ .

•  $u: \{0,1\}^K \to [u_{min} \dots u_{max}]$  $u_{ij} = u(a_{ij}^{+K})$ 

The *utility function*, or *local value function*. The range is an interval in the set of real or integer numbers.

- $V(A) = \sum_{\forall ij} u_{ij}$ The (global) Value V(A) of a pattern A is the sum of all the cells' utilities.
- W(A) = V(A)/N

The Wealth W(A) of a pattern A, that is the global value normalized to the number of agents, i.e. the agents personal utility/income when the global value is totally socialized, i.e. the average utility of a cell.

#### • Maximize W(A)

The *Objective Function* is W to be maximized.

The wealth W(A) of the whole community of agents is the objective function of our problem. Alternatively the value V(A) can be used if we do not need to compare patterns of different sizes N.

#### 2.2 The problem-solving method

This is an attempt to solve the problem with a probabilistic Cellular Automata (CA) rule as used before in [11, 12, 13, 15, 16, 17]. Such a rule uses a list of so-called templates  $B_i$  that are local matching patterns, often of size  $3 \times 3$  or  $5 \times 5$ . If a template matches at (i, j) in the aimed pattern A, then  $a_{ij}$  is kept. If a template matches except its center, then  $a_{ij}$  is set to the center value of the template. Otherwise  $a_{ij}$  is changed randomly.

In order to solve a given problem by this method, we need appropriate templates. Now the idea is to look first at optimal solutions and then extract templates from them by the use of a gliding window. How to find optimal solutions? A tool can be applied which was used already in a former works. This tool finds optimal patterns by a Genetic Algorithm for any global fitness that has to be defined. After having found optimal patterns for small field sizes, templates can be extracted. These templates are then inserted into a general CA rule with the expectation that optimal patterns can be evolved, also for larger sizes and different boundaries.

The problem depends heavily on the utility function being used, which has to be defined first. The used approach allows to use any utility function, but the success of it can not be guaranteed. Further research is needed to show the possibilities and limits. In our test case, this approach was successful.

In summary, our approach consists of the following steps.

- 1. (Utility Function) Define a utility function for the local neighborhood of cell states.
- 2. (Optimal Solutions) Find optimal patterns for a small pattern size, using a GA or other optimization techniques.
- 3. (Find Templates) Analyze the optimal patterns found in step 2 using a gliding window. Each local configuration within the window is a candidate for a template. In the list of observed templates duplicates are deleted and missing symmetric ones are added. Remark: We assume that all symmetric templates are allowed in our problem, i.e. the class of templates to which a template belongs shall be complete under all symmetries.
- 4. (Build CA Rule) Insert the templates into a general CA rule that can evolve patterns by increasing the number of matching templates.
- 5. (Evaluation) Compute the value of the evolved patterns. Evaluate if the values are optimal for patterns of sizes equal or different to the small size used in step 2. In order to test if the evolved patterns are optimal, the optimal fitness of larger patterns have to be known (Here we could find formula for that issue.) In general it is not easy to determine the value of a large pattern.

## 3 A special utility function: the payoff of a game

Our test case uses a utility function inspired by and derived from the Prisoner's Dilemma (PD) game. The PD game is a version in the 2D space with several players in the neighborhood, the so-called SPD (spatial PD) game. Note that we do not use the PD game for any gaming or as a method for optimization, like e.g. in [20, 34]. Here we just want to use the payoff matrix of the game as as an example for the definition of the global pattern's value V(A). We define V(A) as the the sum of the local payoffs (utilities) that all the agents receive together.

In some precedent experiments it turned out that the usual PD parameters lead to the trivial pattern all 0 (all are agents are cooperating). But when the parameters were changed, optimal patterns became a certain mixture of 0s and 1s (cooperators and defectors in terms of the game) that we aim at.

Now we will shortly explain the game and how the utility function is computed.

#### 3.1 Prisoner's dilemma game with two players

The Prisoner's Dilemma is a well-known and well-studied game. In its basic form there are two players or agents, X and Y. They play with each other, and for each player the other player is its opponent. Both players choose simultaneously an individual state (also called action), that is either C(ooperate) or D(efect). After a play we have the combined state/action  $(a_X, a_Y)$ . Now this tuple is evaluated using a payoff table which defines the payoffs  $(g_X, g_Y)$  for both players and for all combinations of actions.

$$(g_X, g_Y) = \begin{cases} (R, R) & \text{if } (a_X, a_Y) = (C, C) \\ (S, T) & \text{if } (a_X, a_Y) = (C, D) \\ (T, S) & \text{if } (a_X, a_Y) = (D, C) \\ (P, P) & \text{if } (a_X, a_Y) = (D, D) \end{cases}$$

If we are only interested in the payoff of the player X, depending on its own state  $a_X$  and on the opponents' state  $a_Y$  then we get, formulated in a different but semantically equivalent way:

$$g_X(a_X, a_Y) = \begin{cases} R & \text{if } (a_X, a_Y) = (C, C) \\ S & \text{if } (a_X, a_Y) = (C, D) \\ T & \text{if } (a_X, a_Y) = (D, C) \\ P & \text{if } (a_X, a_Y) = (D, D) \end{cases}$$

$$g_X(a_X, a_Y) = \begin{cases} R/S & \text{if } (a_X, a_Y) = (C, C/D) \\ T/P & \text{if } (a_X, a_Y) = (D, C/D) \end{cases}$$

$$g_X(a_X, a_Y) = \begin{cases} g_X(a_X = C) = \begin{cases} R & \text{if } a_Y = C \\ S & \text{if } a_Y = D \end{cases}$$

$$g_X(a_X = D) = \begin{cases} T & \text{if } a_Y = C \\ P & \text{if } a_Y = D \end{cases}$$

The payoff depends on the parameters T, R, P, S, staying for Treason/Temptation, Reward, Punishment, Sucker. For the classical PD the relation

$$T > R > P > S$$
 and  $2R > (S + T)$ 

is required. The condition 2R > T ensures that playing (C,C) gives the maximum of 2R if the agent's and the opponent's payoff  $(w = g_X + g_Y)$  are added. Divided by N = 2 (number of agents) gives the average (socialized/shared) "Wealth" W = w/N of each member of this small group of N = 2 agents.

*Remark*: If the players have no information about the opponents playing strategy then it would be most beneficial for both players to play (D, D) which is called a Nash Equilibrium (NE). Here we do not aim at a NE.

#### 3.2 Spatial PD game with K opponents

We assume a 2D grid where at each point (i, j) there is a player who plays the PD game with his K neighbors/opponents. Then a player receives a payoff  $g_{ij}^k$  for the game with the  $k^{th}$  neighbor, the total payoff is

$$u_{ij}^{total} = \sum_{k=0}^{K-1} g_{ij}^k(a_{ij}, a_{ij}^k)$$
.

where the payoff from the game with opponent k is

$$g_{ij}^k(a_{ij}, a_{ij}^k) = \begin{cases} R/S & \text{if } (a_{ij}, a_{ij}^k) = (C, C/D) \\ T/P & \text{if } (a_{ij}, a_{ij}^k) = (D, C/D) \end{cases}.$$

We divide the total payoff by the number K of opponents in order to normalize it, as we assume that the total wager is one unit only which has to be divided by the number of games/opponents. The result is the agent's utility  $u_{ij}$ .

$$u_{ij} = u_{ij}^{total}/K$$
 .

We can distinguish two cases: with and without self-play. In the case without self-play an agent plays only with others from the outer neighborhood. In the case of self-play an agent plays also with itself.

In the following we want to use a simplified game with the parameters (except T) set to (R = 1), (P = S = 0). According to [27] the features of the game are maintained.

Now we compute the utility for the SPD game. The payoff for a basic game becomes  $g(a_{Player}) = R$  if (C, C), and  $g(a_{Player}) = T$  if (D, C), because the payoff is zero for (C, D) and (D, D).

$$u_{ij}^{total} = (R[a_{ij}=C] + T[a_{ij}=D])N_C,$$
 where  $N_C = \sum_{k=0}^{K-1} [a_{ij}^k = C]$ 

We have used the notation [condition] which evaluates to 1 if the condition is true, else to 0. We declare C = 0 and D = 1, then we get  $a_{ij} = [a_{ij} = D]$  and  $\overline{a_{ij}} = (1 - a_{ij}) = [a_{ij} = C]$ , and with (R = 1) the formula becomes

$$u_{ij}^{total} = (\overline{a_{ij}} + Ta_{ij})N_C$$
 .

How to choose the parameters? The condition for the classical PD is now

$$(2R > (S+T)) = (2 > T > 1)$$
 for  $R = 1, S = 0$ .

This condition guarantees that the Wealth of the community is equal to 1 if all agents cooperate. Here we violate this condition by intention and change it to T>2, because we aim at patterns with a Wealth W>1. Such patterns will be a structured mixture of cooperators and defectors. We exclude T=2 because then the chosen actions would not matter and the Wealth would be 1 independently of the pattern.

In our application we use the spatial PD game with K=9 neighbors out of the MOORE neighborhood (with self-play), and the parameters are set to T=3, R=1, P=0, S=0. A defecting agent with 8 cooperating neighbors yields maximal a payoff of 24. A cooperating agent with 9 cooperating neighbors yields maximal a payoff of 9. The boundaries are

$$0 \le u_{ij}^{total} \le T \times (K-1) = 3 \times 8 = 24$$
$$0 \le u_{ij} \le T \times (K-1)/K = 24/9 = 2.67.$$

#### 3.3 The pattern's value for the utility of the PD game

The  $Total\ Payoff\ Sum$  that all N agents receive together is

$$TPS = \sum_{\forall ij} u_{ij}^{total}$$
, where  $u_{ij}^{total}$  is the player's total payoff.

Dividing by the number of opponents gives the Value of the pattern A

$$V(A) = TPS/K = \sum_{\forall ij} u_{ij}$$
.

Further dividing by the number of agents gives the *Wealth* which is the income of each agent when all payoffs are normalized and shared (socialized):

$$W(A) = V(A)/N$$
.

#### 3.4 Estimation of the maximal reachable wealth

We want to estimate the wealth depending on (1) the rate of cooperators rsp. defectors in the whole population, and (2) the parameters T, R, P, S. Agents are in the cooperating state C with a rate of  $\pi_C$ , and in the defecting state D with a rate of  $\pi_D = 1 - \pi_C$ .

In order to compute the wealth, we can assume that  $N\pi_C$  cooperators and  $N\pi_D$  defectors play with all N agents as opponents. If the total payoff of an agent is divided by the number of N of opponents, then we get the normalized total payoff  $payoff(a_{ij}) = u_{ij} = u_{ij}^{total}/K$ . We declare

$$payoff(D) = u_{ij}(a_{ij} = D)$$
,  $payoff(C) = u_{ij}(a_{ij} = C)$ .

The expected wealth W is the sum of (i) the payoff(D) which a player in state D receives, multiplied by the rate of occurrence  $\pi_D$ , and (ii) the payoff(C) which a player in state C receives, multiplied by the rate of occurrence  $\pi_C$ .

$$W = \pi_D \times payoff(D) + \pi_C \times payoff(C)$$
, with  $payoff(D) = P\pi_D + T\pi_C$ , and  $payoff(C) = R\pi_C + S\pi_D$ .

Fig. 1 shows the expected wealth for different parameter settings. (a) This case is the one treated in this work. It shows a concave maximum with 1.125 at  $\pi_C=0.75$ . This approximation relates very good to the real optimal value (1.1944) obtained by the optimization procedures in the following sections. Note that the influence of the used  $3\times 3$  MOORE neighborhood was not taken into account. (b) This graph starts at P=0.5 and has its maximum with 1.166 at 0.65. (c) The wealth is constantly one, this parameter setting is not a case for optimization. (d – f) These settings are classical and are shown for information only. The maximal wealth increases steadily with the rate of cooperation and reaches 1 at  $\pi_C=1$ .

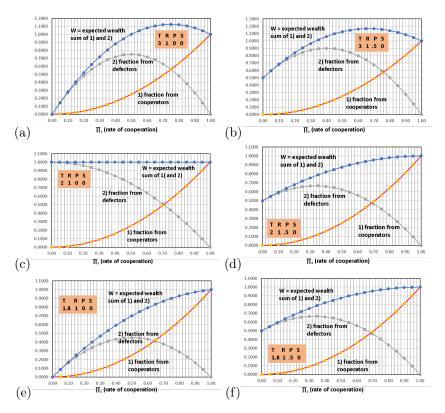


Figure 1: The expected wealth vs the rate of cooperation, for different parameter settings (T,R,P,S). The parameters shown for case (a) are used in our problem.

## 4 Using a Genetic Algorithm to find optimal patterns

#### Algorithm 1: Generating a Binary Pattern with a Genetic Algorithm

Types:  $Pattern = array[0 ... n_1 - 1, 0 ... n_2 - 1] \text{ of } \{0, 1\}$ 

Solution = (Pattern, Fitness)

Data & Output: S is an array [1 .. M] of Solution

Temporary: Offspring is a Solution

- (1)  $S.Pattern \leftarrow RandomPatterns$ 
  - (2) while not TerminationCondition
    - (3) foreach  $S_i$  in S
      - (4) Select randomly a Mate  $S_i$
      - (5)  $Offspring.Pattern = Mutate(Crossover(S_i, S_j))$
      - (6)  $Offspring.Fitness \leftarrow fitness(Offspring.Pattern)$
      - (7) **if** Offspring.Pattern  $\notin$  S.Pattern

and  $Offspring.Fitness > S_i.Fitness$ 

then  $S_i \leftarrow Offspring$ 

- (3) end foreach
- (2) end while
- (8)  $S \leftarrow SortByFitness(S)$

Figure 2: Algorithm that generates optimal patterns with respect to a global fitness function. The fitness function used here is the *Wealth* of the community, taking into account the local payoffs of all agents.

In a first step we use a GA to find optimal patterns with respect to the defined global fitness function. Such patterns will be used (a) as master solutions from which local matching templates are extracted to be inserted in a CA rule, and (b) for comparison with the fitness of the later CA evolved patterns.

The fitness function f(A) is computed as follows. At every site of the pattern the total payoff from the SPD game is computed and then summed up over all cells. It is

$$f(A) = \sum_{\forall ij} u_{ij}^{total} = TPS = W(A) \times N \times K$$
.

The algorithm used is given in Fig. 2. A possible solution (an individual) is a tuple (Pattern, Fitness). An array/list of M solutions is the data and output that is manipulated by the algorithm. (1) The pattern list is initialized randomly. (2) The while-loop is repeated until a termination condition becomes true, i.e. the number of iteration has reached a certain limit, and/or another global condition, like a certain fitness level or certain pattern features. (3) In the for each-block better solutions are searched for. For each current existing

individual (tentative solution) an offspring (a new individual) is generated by crossover and mutation. The current individual is replaced by the offspring if it has a higher fitness. (4) A mate  $S_j$  is selected at random from the list. (5) A new offspring pattern is computed by crossing over  $S_i$  with  $S_j$ , and then applying a mutation. (6) The fitness of the offspring pattern is computed and stored within the offspring. (7) The offspring replaces the current individual  $S_i$  if its fitness is higher. (8) The list of solutions is sorted for output.

The used GA is a simple form using the classical GA algorithm basics. The goal was to generate optimal patterns in a simple way and not to optimize the algorithm itself which is a topic for further research. Only one list of individuals is used and not two (old and new generation).

Each individual is treated separately and is expected to improve by crossover with any other individual, not depending on their fitness. Thereby a high diversity is supported though the speed of improvements may not be so high. Crossover is performed in the following way (uniform crossover with a certain probabilty): (1) Each bit of the offspring  $Crossover(S_i, S_j)$  (without mutation) is taken from the mate  $S_j$  with probability  $p_1$  otherwise unchanged from the parent  $S_i$ . (2) Then mutation is performed on each bit with probability  $p_2$  yielding  $Mutate(Crossover(S_i, S_j))$ . Then the fitness of the offspring is computed and used for replacement in the case of improvement.

The used size of the population (the number of possible solutions) was 40, and the probabilities were set to  $p_1 = 0.2, p_2 = 0.05$ . Optimal or near-optimal solutions were found within a number of iterations between 1 000 and 10 000 in a few minutes on a desktop PC with an Intel Core i5-3470 CPU @ 3.20GHz and 8 GB RAM under Windows 10. The programming language was Free Pascal using one thread only.

#### 5 Found optimal patterns

We want to characterize a pattern A by the characteristic

 $CC = (W(A), f(A) = TPS(A); n, n^2, b, b/n), \text{ where}$ 

b is the number of cells in state 1 (Defect).

#### 5.1 Patterns of size 6 x 6

The found optimal patterns for  $n=6\times 6$  are shown in Fig. 3. Their characteristic is CC=(1.19444,387;6,36,9,0.250). The payoffs of the cells are given below the patterns. Summing up all payoffs gives the fitness of 387, which corresponds to a wealth of 1.19444. The average wealth of each agent is 19.444% higher than the pattern all C (all 0) which is the baseline wealth of 1 for comparison. The sum of all payoffs is 387, and the number of Ds (ones) in the pattern is 9 out of 36 (25%). The patterns contains a maximal number of points, where a point is a '1' surrounded by 8 zeroes. We consider patterns equivalent if they are equal under the symmetries shift, rotation, and reflection. The patterns Fig. 3(b) and (c) are equivalents.

How equal is the community in terms of personal income/payoff. If all agents would cooperate then the pattern is all 0, and every agent has a base income (total payoff) of 9. (It becomes 1 if it is normalized through division by K). In an optimal pattern there are winners (income higher than 9) and losers (income lower than 9). For this example the winners are the defectors scoring 24, and the losers are the cooperators scoring 5, 6 or 7. But as we assume that all profits are shared the average income becomes  $TPS/n^2 = W \times K = 10.75$ , 1.75 units better than the base income.

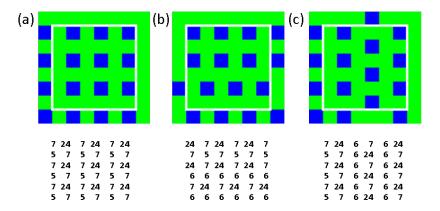


Figure 3: 6 x 6 Optimal pattern evolved by a Genetic Algorithm. The total payoffs are shown below the patterns. Fitness is 387, the maximum.

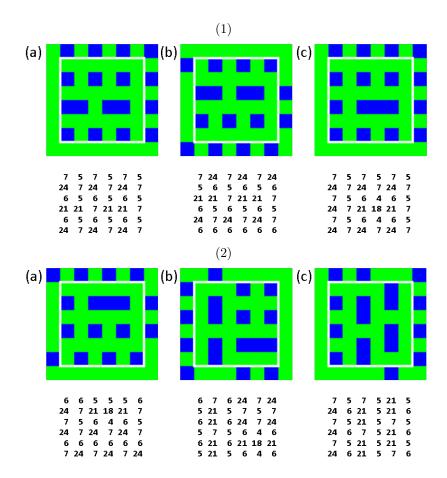


Figure 4:  $6 \times 6$  Near-optimal patterns evolved by a GA. Fitness is 386 for (1a)-(2a), and 385 for (2b)-(2c).

Near-optimal patterns can be of interest, too. The GA evolved second best patterns as depicted in Fig. 4. Their characteristic is

$$CC = (1.19136, 386; 6, 36, 10, 0.278)$$
.

It is remarkable that their fitness is only one resp. two less then the maximum. The number of points has incremented by one (from 9 to 10) compared to the optimum. Now the pattern contains not only points but also dominoes (11-lines/rectangles) or 111-lines.

#### 5.2 Patterns of size 5 x 5

There is only one optimal pattern (Fig. 5), suggested from many runs of the GA. The characteristic is CC = (1.17778, 265; 5, 25, 8, 8/25 = 0.32). The income of the defectors/winners with cell state 1 is 21, while that of the cooperators/losers is only 5 or 6. Recall that each agent would have an income of 9 (the baseline), if all of them would cooperate (corresponding to an all zero pattern). In this pattern there are 6 winners and 19 losers. As we assume that the winners are

social and altruistic they share their income with the whole community aiming at an equal or fair (close to equal) distribution. As the total income of the whole community is TPS=265, we divide it by the number of agents. The result is a shared common income of 265/25=10.6 which is an improvement of 1.6 compared to the baseline of 9. In order to motivate agents to become a dealer one could choose a distribution with partial sharing, for example income 10 for a consumer, and income 12.5 for a dealer.

We observe in the pattern 4 dominoes arranged in a "cycle". And we find a square block of 4 consumers (marked in red) within this cycle. We call such a block of  $2 \times 2$  zeroes, framed/confined by certain neighbors (typically dominoes or points) in a  $4 \times 4$  window, a *singularity*. In other non-optimal patterns, and in optimal patterns of a higher odd size n > 5, we can also find singularities.

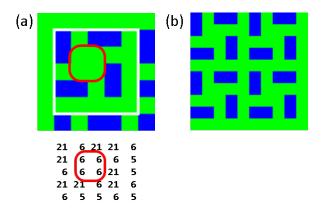


Figure 5: (a) 5 x 5 optimal pattern evolved by a Genetic Algorithm. The total payoffs are shown below it. Fitness is 265 and wealth is 1.17778, the maximum. Inside the red marked area there is a block of 4 zeroes. (b) The pattern (a) (inside the white marked boundaries) is 4 times replicated, twice horizontal and twice vertical. This "quad" representation emphasizes the inherent structures.

Fig. 6 shows some near-optimal patterns with a fitness of 262, three less then the optimum. The characteristic is CC = (1.16444, 262; 5, 25, 7, 7/25 = 0.28). We can observe patterns with (3 dominoes and 1 point), and (2 dominoes and 3 points where two of them are diagonally connected). In these patterns we find blocks (marked by red rectangles) of consumers of size  $2 \times 2$  or  $2 \times 3$ . If we analyze the optimal pattern (Fig. 5) with respect to such blocks, we find just one  $2 \times 2$  zero-valued block in the middle of a "circle" of 4 dominoes.

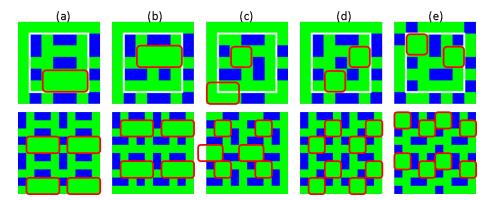


Figure 6:  $5 \times 5$  near-optimal patterns in normal and in quad representation. Fitness is 262 (3 less than the maximum) and wealth is 1.16444. The marked areas a small blocks of zeroes (cooperators/consumers).

#### 5.3 Patterns of size 7 x 7

The optimal  $7 \times 7$  pattern (Fig. 7) has the characteristic

$$CC = (1.18367, 522; 7, 49, 15, 15/49 = 0.306).$$

The pattern contains 6 dominoes and 3 points. We find exactly one singularity (marked).

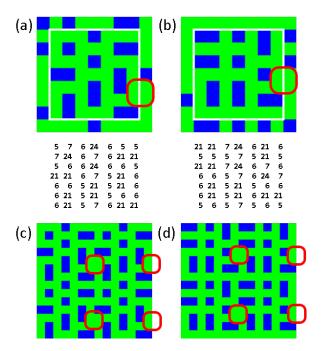


Figure 7:  $7 \times 7$  Optimal pattern evolved by a Genetic Algorithm. The total payoffs are shown below the pattern (a). Fitness is 522 and Wealth is 1.18367, the maximum. Patterns (c, d) are the quad representations of (a, b). The marked areas are singular points of 4 zeroes.

## 5.4 Patterns' characteristics for odd n > 7

More optimal patterns were evolved by the GA for n=9,11,13,15. Table 1 gives their characteristic data. All the given values increase with n, except the density  $b/n^2$  of the ones which decreases.

We can conclude from the table (without giving a proof) that the following relations hold, for odd(n) and  $n \ge 5$ :

$$\begin{split} N_{domino}(n) &= n-1 \\ N_{point}(n+2) &= N_{point}(n) + n-2, \text{ with } N_{point}(5) = 0 \\ N_{point}(n) &= m + m(m+1), \text{ where } m = (n-5)/2 \\ b &= 2N_{domino}(n) + N_{point}(n) \\ density &= b/n^2 \end{split}$$

As shown later in Sect. 8.2 we can also compute the total play sum TPS and the wealth W:

$$TPS = 265 + 128m + 43(m(m+1))$$
  
 $W = TPS/n^2/(K = 9).$ 

For very large n the wealth converges to 1.1944 and the density to 1/4, the same

values as for patterns of even size. This can also be explained by the fact that very large patterns are almost totally filled with points.

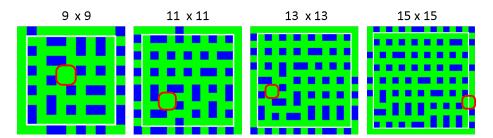


Figure 8: Optimal patterns for n = 9 to n = 15.

Table 1: The patterns' characteristics for odd n.

			we alth	TPS	number	number	number	density
m	n	$n^2$	W		dominoes	points	b of ones	$b/n^2$
-	3	9	1.12346	91	2	0	4	0.444
0	5	25	1.17778	265	4	0	8	0.320
1	7	49	1.18367	522	6	3	15	0.306
2	9	81	1.18656	865	8	8	24	0.296
3	11	121	1.18825	1294	10	15	35	0.289
4	13	169	1.18935	1809	12	24	48	0.284
5	15	225	1.19012	2410	14	35	63	0.280

## 6 Extracting templates from the optimal patterns

We want to use the found optimal patterns as masters and use their inherent local structures to generate similar (hopefully optimal or near-optimal) patterns. In this second step we extract from the optimal patterns local matching patterns (templates) that we will use later (in the third step) in a CA rule to generate patterns that are similar to the given optimal ones.

A special software tool was developed that can analyze a given pattern with respect to their inherent local neighborhoods. A small window (typically a Moore or von-Neumann neighborhood) is gliding over the whole pattern. All found neighborhood configurations (we call them found templates)  $T_i^{found}$  are stored in a list  $(T_i^{found})$ . Multiples are deleted. We check for symmetries (rotations, mirror against horizontal/vertical center line), and then we form classes of symmetric templates. If a class is not complete, we add the missing symmetric templates. We have assumed for our problem that each templates belongs to a certain class (under symmetries) of templates, i.e. each template is a representative of a certain class of equivalent templates.

Remark: In general, it is not necessary to complete the classes symmetrically,

especially if we know that some symmetric templates will never occur in the given pattern or a similar pattern that we aim at and if we want to generate patterns on the basis of extracted templates only.

For our problem we assume that the inherent local structures can be expressed by templates of size  $3 \times 3$ . Indeed, for our problem, this size seems to be sufficiently large, but this cannot be stated in general. All extracted templates are shown in Tab. 2, where T0–T7 define point-only patterns for an even field size n, and T0–T51 the more complex patterns for n odd.

Table 2: T0–T7: These templates define the possible neighborhoods if the field size n is even, i.e. when then patterns contain points only. T0 represents a point (a one surrounded by zeroes). T2 is a 90° rotation of T1, T4–T7 is a class of equivalents under rotation. T0–T51: The templates for an odd field size.

TO	T1	T2	Т3	T4	T5	Т6	T7
A	B0	B1	C	D0	D1	D2	D3
0 0 0	0 0 0	0 1 0	1 0 1	1 0 1	0 1 0	0 0 1	1 0 0
0 1 0	1 0 1	0 0 0	0 0 0	0 0 0	0 0 0	1 0 0	0 0 1
0 0 0	0 0 0	0 1 0	1 0 1	0 1 0	1 0 1	0 0 1	1 0 0
T8	Т9	T10	T11	T12	T13	T14	T15
E0	E1	E2	E3	FO	F1	F2	F3
0 0 0	0 0 0	0 1 0	0 0 0	1 1 0	0 1 1	1 0 1	0 0 0
1 1 0	0 1 1	0 1 0	0 1 0	0 0 0	0 0 0	1 0 1	1 0 1
0 0 0	0 0 0	0 0 0	0 1 0	1 1 0	0 1 1	0 0 0	1 0 1
T16	T17	T18	T19	T20	T21	T22	T23
GO	G1	G2	G3	G4	G5	G6	G7
1 1 0	0 1 0	0 1 1	0 1 0	0 0 1	0 0 0	1 0 0	0 0 0
0 0 0	0 0 0	0 0 0	0 0 0	1 0 1	1 0 1	1 0 1	1 0 1
0 1 0	1 1 0	0 1 0	0 1 1	0 0 0	0 0 1	0 0 0	1 0 0
T24	T25	T26	T27	T28	T29	T30	T31
HO	H1	H2	НЗ	H4	Н5	Н6	H7
1 1 0	1 0 1	0 1 1	1 0 1	1 0 1	1 0 0	1 0 1	0 0 1
0 0 0	0 0 0	0 0 0	0 0 0	0 0 1	0 0 1	1 0 0	1 0 0
1 0 1	1 1 0	1 0 1	0 1 1	1 0 0	1 0 1	0 0 1	1 0 1
T32	T33	T34	T35	T36	T37	T38	T39
IO	I1	12	13	J0	J1	J2	J3
1 1 0	0 1 1	0 0 1	1 0 0	1 1 0	1 0 0	0 1 1	0 0 1
0 0 0	0 0 0	1 0 1	1 0 1	0 0 0	0 0 0	0 0 0	0 0 0
0 1 1	1 1 0	1 0 0	0 0 1	1 0 0	1 1 0	0 0 1	0 1 1
T40	T41	T42	T43	T44	T45	T46	T47
J4	J5	J6	J7	KO	K1	K2	КЗ
1 0 1	0 0 0	1 0 1	0 0 0	0 1 0	0 0 1	0 1 0	1 0 0
0 0 1	0 0 1	1 0 0	1 0 0	0 0 0	0 0 0	0 0 0	0 0 0
0 0 0	1 0 1	0 0 0	1 0 1	0 0 1	0 1 0	1 0 0	0 1 0
T48	T49	T50	T51				
K4	K5	K6	K7				
0 0 0	1 0 0	0 0 0	0 0 1				
0 0 1	0 0 1	1 0 0	1 0 0				
1 0 0	0 0 0	0 0 1	0 0 0				

The templates for n even. Let us extract the templates from the optimal patterns shown in Fig. 3. As the optimal patterns in Fig. 3 are simple structured we can find the templates also by visual inspection without using a special tool. We find easily the template T0 as a "point" surrounded by zeroes. In addition we extract the templates T1, T2, T3 from Fig. 3(a). They define the neighborhood of a zero valued cell when two lines of points are not shifted against each other. For point lines that are shifted against each other, we find T4, T5 in Fig. 3(b), and then T6, T7 in Fig. 3(c). So we have found all relevant templates for the optimal patterns shown in Fig. 3. Note that other optimal patterns differ only slightly (lines of points can be shifted against each other), but it is obvious that the set of found templates is complete in the sense that we can construct any optimal pattern with them by overlaying them without conflict and thereby covering the whole area. Not all templates need to be part of an optimal pattern, for instance the pattern in Fig. 3(a) uses templates (T0–T3) only, and Fig. 3(b) uses only the templates (T0-T3), T4, T5, and Fig. 3(c) uses only the templates (T0-T3), T6, T7. We can also imagine an optimal pattern of size  $8 \times 8$  which contains 4 rows that are shifted row by row. Such a pattern can be defined by the templates T0, T6, T7 only.

The templates for n odd. Optimal patterns of size  $5 \times 5$  and  $7 \times 7$  were generated by the GA pattern generator. Then several optimal patterns were analyzed by a tool using the gliding window technique. All found templates were gathered and enhanced by missing symmetric ones. Then the list of templates was sorted into categories by hand as shown in Table 2. Templates T0-T7 are the basic templates also used for patterns with even size. E0-E3 appear as result of dominoes fitting into  $3 \times 3$  target window (not all surrounding zeroes fit into that window). So we have it to do with partial neighborhoods which are induced by certain local configurations of dominoes and points. F0-F3 arise as the result of two parallel dominoes. G0-G7 and H0-H7 arise as the result of one domino and one point (or alternatively a part of a domino). I0–I3 arise as the result of two skewed dominoes. J0-J7 and K0-K7 are derived from singularities observed in optimal patterns. A singularity is a  $4 \times 4$  local pattern that includes a  $2 \times 2$  square block of zeroes. We carefully inspected the optimal patterns generated by the GA and found exactly one singularity in each optimal pattern of odd size. A proof would be welcome giving evidence to this observation.

Fig. 9 shows the three detected singularities, 4 times replicated in each row. From each singularity we derived four  $3 \times 3$  templates using a gliding window of this size, because our aim is to solve our problem with this limited target size. It would be more natural to use the three singularities directly as  $4 \times 4$  templates. But as we can assemble a singularity by overlaying four  $3 \times 3$  templates (the marked windows) we can get along with the smaller templates of size  $3 \times 3$ . In this way we found the templates J0, J3, J4, J7, K1, and K5. The missing symmetric ones were added resulting in J0–J7 and K0–K7 as shown in Table 2.

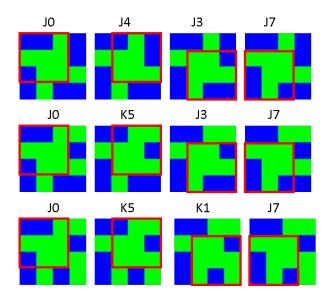


Figure 9: Three singularities of size  $4\times 4$  were detected in optimal  $7\times 7$  GA patterns, shown in the three rows. A singularity is a  $2\times 2$  block of zeroes enclosed by its direct neighbors. Templates of size  $3\times 3$  are derived from the singularities using a gliding window (marked in red). We find the templates J0, J3, J4, J7, K1, and K5. The missing symmetric ones have to be added resulting in J0–J7 and K0–K7.

## 7 The CA rule using the templates

The here designed CA rules are tailored to the given problem, they enhances the construction principles presented already in [11, 12, 13, 15, 16, 17]. The rule is *probabilistic* and uses *asynchronous* updating. A new generation at the next time step t+1 is a sequence of  $n^2$  micro time-steps. The cell state is the tuple

$$(a_{ij}, h_{ij})$$

where  $a_{ij} \in \{0,1\}$  is the pattern state and  $h_{ij} \in \{0,1\}$  is the hit state.

The rule uses a given set of templates, meaning that the templates are parameters of a general rule. It works in the following way.

#### • (Select a cell)

A cell ij = (i, j) is selected at random or sequentially.

#### • (Compute template hits)

Each template TQ (except its center value  $TQ_{center}$ ) is tested for a match at the site (i,j) in the pattern field A. More specifically, the outer neighbors of  $a_{ij}$  are compared to the spatial corresponding values (also called pixels) of each template taken from the given list of templates.

For each template an associated, temporary hit  $h_{ij}^Q$  is stored. It is set to 1 for a hit, otherwise it is set to 0.

Then all  $h_{ij}^Q$  are summed up, and if the sum is larger than zero, the hit state  $h_{ij}$  is set to one, otherwise to zero.

if 
$$(\sum_{\forall Q} h_{ij}^Q) > 0$$
 then  $h_{ij} := 1$  else  $h_{ij} := 0$ 

#### • (Adjust or Inject Noise)

#### - (If there is a hit: Adjust)

If  $h_{ij} = 1$  then there exists a template hit  $h_{ij}^Q = 1$ .

\* If there is only one template hit, the pattern state is set to the center value of the template.

if 
$$h_{ij}^Q = 1$$
 then  $a_{ij} := TQ_{center}$ .

if  $h_{ij}^Q = 1$  then  $a_{ij} := TQ_{center}$ . This operation is not necessary if the pattern state  $a_{ij}$  is already equal to the correct value  $TQ_{center}$ , which means that the complete template was found in the pattern.

\* If there are several template hits, one of them is selected at random for adjustment.

#### - (If there is no hit: Inject Noise)

In order to drive the evolution to optimal patterns, noise needs to be injected into the CA. This is performed by changing the pattern state  $a_{ij}$  with some randomness. By default the current state of a cell shall be unchanged. In general, for each possible state transition  $A \rightarrow$  $B, A \neq B$  a specific probability needs to be defined, that may further depend on an additional condition. In our case we have only the two states zero and one, and so we need to define the state transitions  $0 \to 1$ , and  $1 \to 0$ :

```
if a_{ij} = 0 then a_{ij} := 1 with probability \pi_{01}
if a_{ij} = 1 then a_{ij} := 0 with probability \pi_{10}.
```

In our simulations, the best results were obtained with the settings  $\pi_{01} = 0.04$  and  $\pi_{10} = 1.0$ . So there is an unconditional change  $1 \to 0$ , and a state change  $0 \to 1$  with a low probability of 0.04. In terms of the PD game, the state 1 means "Defect", and 0 means "Cooperation".

Depending on the used subset of all the 52 templates (Table 2), we defined the following specialized rules

- Rule 8: This rule uses only the templates T0-T7, it is dedicated to work for patterns of even size n only.
- Rule 36: This rule uses the templates T0-T35. It forms transient optimal or near-optimal patterns of odd size. The templates describing singularities are not used.
- Rule 52: This rule uses the full set of templates T0-T51. It forms stable optimal or near-optimal patterns of odd size.

The CA rule is interpreted as follows.

• Initialize

```
for all (i, j) do
             a_{ij} \leftarrow 1 with probability 1/4, else a_{ij} \leftarrow 0
             h_{ij} \leftarrow 0 —The hits are reset
           end for
           Evaluate global objective (*)
• for t \leftarrow 1 to t_{limit} do
          repeat n^2 times — compute a new generation, n^2 micro time-steps
             Select a cell (i, j) at random
             \forall Q: h_{ij}^Q \leftarrow 0/1 — The results of testing all templates TQ
             h_{ij} \leftarrow \text{Number of template hits}
             a_{ij} \leftarrow \text{Rule}(h_{ij}, \text{ Center of template TQ with hit } h_{ij}^Q = 1) - Adjust \text{ or Noise}
          end repeat
          ^{(*)} Evaluate\ global\ objective
             \forall (i,j) : compute \ u_{ij} \leftarrow (a_{ij}^{+9})/9
             STP(A) and W(A) depending on \sum u_{ij} are computed.
   end for
```

Recall that  $n^2$  micro time-steps are performed in asynchronous updating mode per generation. The utilities  $u_{ij}$ , STP(A) and W(A) are only computed for evaluation and statistics, they do not influence the pattern evolution. The pattern states  $a_{ij}$  and the hit states  $h_{ij}$  are stored between the generations. During the computation of a new generation not all cells are updated, roughly 1/3 never, 1/3 once, and 1/3 multiple times, according to the binomial distribution. Therefore the states  $(h_{ij}^k$  and  $u_{ij}^k)$  of a neighbor k can be depreciated because changes of their neighborhoods between their last update and the current updating may not be reflected. This effect can be seen as an additional source of noise. Fortunately, in our simulations, this effect did not disturb the overall convergence to a stable pattern. When the pattern becomes stable, this perturbing effect vanishes.

## 8 Patterns evolved by the CA rule

### 8.1 Even pattern sizes

Application of Rule 8. At first we want to use Rule 8 that was designed to evolve patterns of even size n. The initial pattern is generated randomly. The patterns cell states are set to 1 with the probability of 1/4. This probability was used because optimal pattern have a density close to 1/4 and this setting may help to speed-up the evolution. Note that the real initial rate of ones can deviate from  $n^2/4$  because the distribution is binomial. The Rule 8 drives the evolution to a stable optimal pattern. Fig. 10 shows two sample evolutions with 4 generations only, from t=0 to t=4. The payoffs for every cell are also given. In an optimal pattern the 1-cells receive a payoff of  $N_{zero} \times T = 24$ , and the 0-cells  $(1+N_{zero}) \times R = 5,6$ , or 7, where  $N_{zero}$  denotes the number of zeroes in the outer Moore-neighborhood with 8 neighbors.

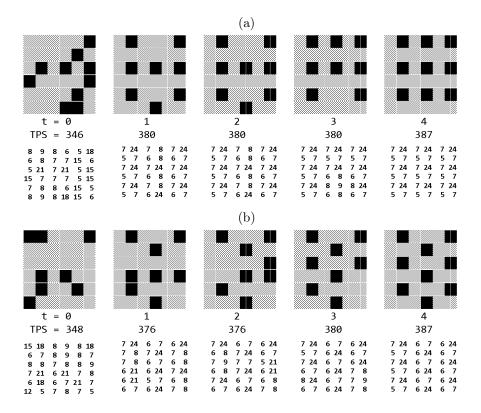


Figure 10: The evolution of two optimal  $6 \times 6$  patterns (a) and (b) using rule 8. Wealth at t=4 is W=1.1944, and TPS=387. The total payoffs are shown below the patterns.

100 runs (experiments) were performed in order to compute the average time (the number of time-steps) to reach a stable optimum. The simulation time limit was set large enough to reach the optimum in every run.

The measured values are:

$$\begin{split} t_{avrg}^{opt}(n=6) &= 34 \ (3_{min} - 200_{max}). \\ t_{avrg}^{opt}(n=8) &= 198 \ (5_{min} - 733_{max}). \\ t_{avrg}^{opt}(n=10) &= 1421 \ (31_{min} - 4998_{max}). \end{split}$$

Note that this values are not absolutely certain because of the statistical variations that have to be taken into account. Nevertheless we can see that the time is increasing very rapidly. What can we do against it? If we are searching only for one or a few optimal solution we can make a number of runs  $N_{run}$  with a limited number of generations  $t_{limit}$  and then select the optimal or best ones.

For performance evaluation we can compute the rate  $N_{found}^{opt} \leq N_{run}$  of found optimal solutions within  $N_{run}$  runs for a limited number  $t_{limit}$  of generations.

Another way to evaluate the performance is to compute for a given Rule 8/36/52, a given size n, and a given time limit  $t_{limit}$ 

- (i)  $W_{run}^{max}$ , the maximal wealth found in each run, found first at  $t=t_{max}$ , and
- (ii)  $W_{avrg}^{max} = 1/N_{run} \sum_{\forall run} W_{run}^{max}$ , the maximal obtained wealth averaged over all runs.
- (iii)  $W_{max}^{max}$ , the best pattern found during all runs.
  - Example: For n=10 we run  $N_{run}=100$  evolutions (runs) with different  $t_{limit}$ . For  $t_{limit}=(50,100,200,500,1000,5000)$  we found

```
N_{found}^{opt} = (4, 6, 13, 30, 55, 100) optimal solutions.
```

The ratio  $N_{found}^{opt}/N_{run}=expect(N^{opt})$  approximates the expectation of finding an optimal solution, which is in our example

```
expect(N^{opt}, t_{limit}) = (4/100, 6/100, 13/100, 30/100, 55/100, 100/100)
= (4\%, 6\%, 13\%, 30\%, 55\%, 100\%).
```

The achieved maximal wealth, averaged over all runs, and for the given time limits are

```
W_{avrg}^{max} = (1.1709, 1.1745, 1.1814, 1.1870, 1.1905, 1.1944). \\
```

For this example, during  $t_{limit} = 50$  iterations, we can expect with a probability of 4% an optimal pattern with wealth 1.1944, and a pattern with wealth of 1.1709 on average. This means that we can find very good performing patterns within a short time. Recall that the baseline pattern has the wealth of 1, it is completely filled with zeroes (cooperators in terms of the PD game).

**Application of Rule 52.** We want to test whether the rules designed for patterns of odd size can also work. First we test Rule 52. We use the same example with n = 10,  $N_{run} = 200$ . The result is:

- All patterns reach quickly a *stable near-optimal* configuration. The time needed is  $t_{avrg} = 10 \ (2 74)$ .
- The best pattern found has  $W_{max}^{max}=1.1922$ . It is close to the optimum. The average of the run's maxima is  $W_{avrg}^{max}=1.1841$ .
- The measured wealth distribution ((W, rate)) was

```
(1.1755, 1),(1.1766, 1),(1.1777, 2), (1.1788, 3), (1.1800, 10),(1.1811, 10), (1.1822, 8),(1.1833, 6),(1.1844, 17),(1.1855, 15),(1.1866, 9), (1.1877, 9), (1.1888, 2),(1.1900, 3),(1.1911, 2), (1.1922, 2) .
```

We found a variety of good patterns close to the optimum which we did not reach, although it can be reached in principle. Some of these patterns are shown in Fig. 11(a).

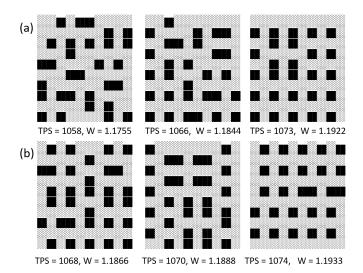


Figure 11: (a) Some near-optimal  $10\times 10$  patterns found with (a) Rule 52, and (b) with Rule 36.

**Application of Rule 36.** We want also test Rule 36, using the same example with n = 10,  $N_{run} = 250$ . The result is:

- All evolved patterns were *stable* and *near-optimal*. The time needed is  $t_{avrg} = 54 \ (4 236)$ .
- The best pattern found has  $W_{max}^{max}=1.1933$ . It is very near to the optimum, probably it is the nearest. The average of the run's maxima is  $W_{avrg}^{max}=1.1897$ .
- The measured wealth distribution ((W, rate)) was

```
(1.1866, 3), (1.1877, 13), (1.1888, 30), (1.1900, 23), (1.1911, 19), (1.1922, 8), (1.1933, 4).
```

We found a variety of stable near-optimal patterns, significantly better than with Rule 52. The reason is that we do not allow singularities, therefore these patterns are more dense. Compared to Rule 52, the evolution lasts longer (54:10). Some of these patterns are shown in Fig. 11(b).

#### 8.2 Odd pattern sizes

Rule 8, designed for even pattern sizes, does not evolve stable patterns for odd sizes, it tries to place points surrounded by zeroes of size  $3\times3$ , but these points are transients only. Therefore Rule 36 and 52 were designed which can evolve optimal patterns. Rule 52 forms stable optimal and near-optimal patterns, and Rule 36 transient optimal and near-optimal patterns. In the following some evolved patterns are presented which can be found with both rules. The CA rules can evolve the same patterns that we found by the Genetic Algorithm.

**Size**  $5 \times 5$ . There is only one optimal pattern, shown in Fig. 12(a, b). (a) and (b) are symmetric under reflection, so they are counted only once. We can see the red marked  $2 \times 2$  square of zeroes which marks a singularity. Two non-optimal patterns are shown in Fig. 12(c, d).

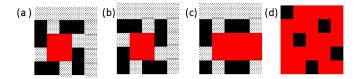


Figure 12:  $5\times 5$  patterns evolved by the Rules 52 and 36. (a, b) are optimal patterns with W=1.1777, (c, d) are near-optimal ones. Wealth is W=1.1644 for (c), and 1.1555 for (d).  $2\times 2$  zero-blocks are marked in red.

Size  $7 \times 7$ . Four optimal patterns are depicted in Fig. 13. The  $4 \times 4$  singularities in (a, b) are similar but different to (c, d).

Near-optimal patterns (Fig. 14) can also be of interest, for instance from the artistic point of view. Pattern (a) contains dominoes only, has only one singularity and is highly symmetric, a rare case. This case shows that patterns with one singularity need not necessarily to be optimal, though we observed *exactly one* singularity in *optimal* patterns of odd size.

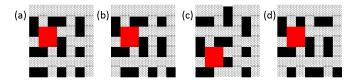


Figure 13: Some optimal  $7\times 7$  patterns. Wealth is W=1.1836 and  $TPS=522.\ 2\times 2$  zero-squares are marked in red.

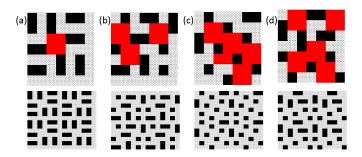


Figure 14: Some near-optimal  $7 \times 7$  patterns. Wealth of (a,b,c,d) is (1.1814,1.1768,1.1723,1.1700) and TPS=(521,519,517,516). The corresponding quad representation is shown in the bottom row. Zero-blocks are marked in red.

Size  $9 \times 9$ . Fig. 15(a-d) shows optimal patterns, and (e-f) near-optimal ones.

The optimal patterns contain only one singularity.

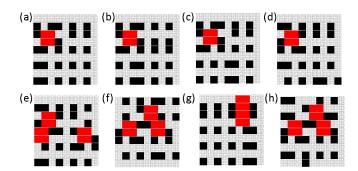


Figure 15:  $9 \times 9$  patterns evolved by CA. (a)–(d) are optimal patterns with W=1.16615, (e)–(f) are near-optimal ones. (e)–(g) have a wealth W=1.1824, and (h) a wealth of 1.180. Zero-blocks are marked in red.

Constructing optimal odd patterns. Can optimal patterns be constructed? Yes, see Fig. 16. We start with the  $5 \times 5$  pattern, and then build a  $7 \times 7$  pattern. (i) A row  $0^5$  and a column  $0^1(01)^1$  is added. (ii) A row  $0^1(01)^1$  and a column  $0^1(01)^1$  is added. (iii) The missing square  $0^1(01)^1$  is added. In a similar way the  $0^1(01)^1$  pattern can be constructed, and so on. This construction principle is simple and can easily be generalized.

Let m = (n-5)/2,  $n \ge 5$ , defining the relation (n,m) = (5,0), (7,1), (9,2)...Let  $n \times n$  be the size of the next larger pattern to be constructed. Then the construction principle is (i) add row  $110(10)^m$  and column  $0^{n-2}$ , (ii) add row  $110(10)^m$  and column  $0^{n-2}$ , (iii) add the missing square  $\frac{00}{10}$ .

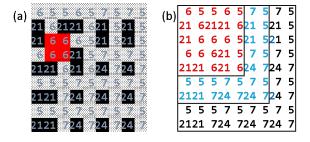


Figure 16: (a)  $9\times9$  optimal pattern with payoffs. (b) This specific pattern can recursively be constructed from the  $5\times5$  pattern (with payoffs in red) by adding two rows and two columns (blue marked payoffs) to get the  $7\times7$  pattern, and then adding again two rows and columns (black marked payoffs) to get the  $9\times9$  pattern.

A general formula for the total play sum TPS. We want to know the maximal TPS for any odd size n. Recall that TPS (the fitness in the Genetic Algorithm) is

$$TPS = W \times n^2 \times (K = 9).$$

To provide the TPS of our PD game we regard the payoffs shown in Fig. 16. We count 265 for the basic  $5 \times 5$  area, 64 for a  $2 \times 3$  rectangles at the borders with the values  $_{21}^{5}$   $_{21}^{5}$   $_{7}^{5}$ , and 43 for squares with  $_{24}^{7}$   $_{7}^{5}$ . In total we get

```
TPS = 265 + 128m + 43(m(m+2)).
```

This formula gives us the exact value of an optimal solution. For very large m=(n-5)/2, TPS approximates  $43m^2$ . Dividing TPS by  $Kn^2$  and computing the limit for large n gives the limit wealth  $W^{limit}=43/36=1.19444$ . This means that for odd pattern the TPS is always smaller than this limit which is the optimal value for the even case.

By visual inspection we observe that such an optimal odd pattern contains n-1 dominoes at two neighboring border sides, and is elsewhere (in the remaining square of the whole pattern field) filled with points  $000 \\ 000 \\ 000$ .

**Performance of Rule 52.** We simulated the case n = 9,  $N_{run} = 100$ ,  $t_{limit} = 100$ . The result is:

- All patterns reach quickly a *stable* optimal or near-optimal configuration. The time needed was  $t_{avrg} = 11 \ (2 49)$ .
- The best patterns found have  $W_{max}^{max}=1.1865$ , the optimum. The average of the run's maxima was  $W_{avrg}^{max}=1.1834$  (99.74%).
- The measured wealth distribution ((W, rate)) was

```
(1.1769, 5), (1.1783, 5), (1.1796, 2), (1.1810, 17), (1.1824, 22), (1.1838, 1), (1.1851, 14), (1.1865, 34)!.
```

34 of 100 of the found pattern are optimal, the remaining a near-optimal.

Now we want to know how the Rule 52 performs for a larger field size. Another simulation with n=27,  $N_{run}=100$ ,  $t_{limit}=100$  was conducted. The number of cell states to be computed is now 9 times higher compared to n=9. The results are:

- All patterns reach quickly a stable configuration. The time needed was  $t_{avrg}=31\ (9-68).$
- No optimal pattern with W = 1.1922, TPS = 7822 was found.
- The average of the run's maxima is  $W_{avrg}^{max} = 1.1833$  (99.25%),  $W^{max}$  ranges between 1.1806 and 1.1867 (*TPS* between 7748 and 7786).

**Performance of Rule 36.** We simulated the case n = 9,  $N_{run} = 100$ ,  $t_{limit} = 100$ . The result is:

- All evolved patterns are *transient* but of very high quality. The average time needed to find the best in every run was  $t_{avrq} = 26 \ (2 88)$ .
- $\bullet$  The optimal pattern with  $W_{max}^{max}=1.1865$  was found 96 times. The average of the run's maxima was  $W_{avrg}^{max}=1.1864$ .

• The wealth distribution ((W, rate)) was (1.1851, 4), (1.1865, 96).

Comparing Rule 36 to Rule 52, we found (96:34) optimal patterns, though they are transients. The average time to find the solutions was (26:11).

A second simulation case was conducted with  $t_{limit} = 50$  instead of 100. Optimal patterns were found 80 times, and  $t_{avrg} = 20 \ (3-50)$ ,  $W_{avrg}^{max} = 1.1858$ . Then a third simulation experiment was conducted with  $t_{limit} = 10$ . Optimal patterns were found 32 times, and  $t_{avrg} = 5 \ (3-10)$ ,  $W_{avrg}^{max} = 1.1825$ . The results demonstrate that Rule 36 is able to outperform Rule 52 although the evolved optimal patterns are not stable.

Now we want to know how the Rule 36 performs for a larger field size. Another simulation with n=27,  $N_{run}=100$ ,  $t_{limit}=100$  was conducted. The results are:

- The evolved patterns are not stable. The time needed for the best patterns within the time limit was  $t_{avrg} = 76 (37 100)$ .
- No optimal pattern with W = 1.1922, TPS = 7822 was found.
- The average of the run's maxima was  $W_{avrg}^{max} = 1.1880$  (99.65%),  $W^{max}$  ranges between 1.1870 and 1.1899 (*TPS* between 7788 and 7807).

One of the found near-optimal patterns is shown in Fig. 17. There is no singularity in this pattern.

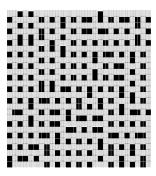


Figure 17: A near-optimal pattern of size  $27 \times 27$  with W=1.1899, TPS=7807, with 98 points, 56 dominoes, 519 zero-states, and 210 one-states

Another experiment was conducted in order to find easily optimal patterns. The initial configuration was totally filled with points with the mandatory extra spaces at the borders (because n is odd). From this configuration optimal configurations (Fig. 18) evolve rapidly within approx. 3–50 generations using Rule 36 or Rule 52. The singularity in Fig. 18 for the final generation (t=3) can better be detected after having shifted it.

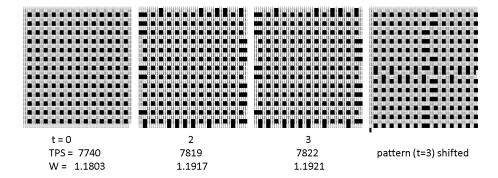


Figure 18: A fast evolution of an optimal  $27 \times 27$  pattern. with  $W=1.11921,\ TPS=7821.$  The initial pattern at (t=0) was filled with points.

## 9 Conclusion

A CA rule was developed in a methodical way that can evolves optimal patterns with respect to a global fitness function. The fitness function was declared as the sum of local utilities computed by the cells/agents. A utility is a local value function over all states in the neighborhood. Here the utility was defined as the payoff that a player would receive in a Spatial Prisoner's Dilemma Game with a special parameter setting. The used method performs in three steps: (1) optimal "master" patterns are generated by a genetic algorithm, (2) templates (local matching patterns) are extracted from the master patterns, and (3) the templates are inserted into a general probabilistic CA rule.

The CA rule for even grid sizes needs only 8 templates of size  $3 \times 3$ , whereas it needs 36 or 52 templates to cope with grids of odd sizes. The rules can evolve optimal or near-optimal patterns. Depending on the selected rule, the patterns are stable or transient. Optimal patterns of even size contain only points (a one surrounded by zeroes). Optimal patterns of odd size are a mixture of points and dominoes, and they contain exactly one singularity, a framed square of four zeroes.

In future work the following topics could be addressed.

- $\bullet$  How do other parameters T and P impact the resulting patterns?
- Another utility function not necessarily related to a game could be used.
- Where are the limits of this approach?
- What is the minimal necessary size of the templates, and can larger sizes speed-up the evolution?
- Templates could be defined by local utility configurations instead of cell state configurations. Is there an advantage?
- The design process could be further automated.

#### References

- [1] Axelrod, R., The Evolution of Cooperation, Basic Books, Inc., New York (1984)
- [2] Bandini, Stefania, et al. Cellular Automata Pattern Recognition and Rule Evolution Through a Neuro-Genetic Approach. J. Cell. Autom. 4.3 (2009): 171–181.
- [3] Beasley, D.; Bull, D.R.; Martin, R.R. An overview of genetic algorithms: Part 1, fundamentals. Univ. Comput. 1993, 15, 56-69.
- [4] Beuchat, J-L., and Jacques-Olivier Haenni. Von Neumann's 29-state cellular automaton: a hardware implementation. IEEE Transactions on Education 43.3 (2000): 300–308.
- [5] Capcarrere, Mathieu. Cellular automata and other cellular systems: design and evolution. No. 2541. EPFL, 2002.
- [6] Grim, P., The greater generosity of the Spatialized Prisoner's Dilemma, J. Theor. Biol. 173(4) (1995) 353–359
- [7] Hoffmann, R. (2014). How Agents Can Form a Specific Pattern. ACRI Conf. 2014, LNCS 8751, pp. 660–669
- [8] Hoffmann, R. (2016). Cellular Automata Agents form Path Patterns Effectively. Acta Physica Polonica B Proceedings Supplement, Vol. 9 (2016) No.1
- [9] Hoffmann, R. and Désérable, D. (2016). Line Patterns Formed by Cellular Automata Agents. ACRI Conf. 2016, LNCS 9863, pp. 424–434
- [10] Hoffmann, R., Désérable, D. (2017). Generating Maximal Domino Patterns by Cellular Automata Agents. PaCT 2017, Malyshkin, V., ed., LNCS 10421 (2017) 18–31
- [11] Hoffmann R., Désérable D., Seredyński F. (2019). A Probabilistic Cellular Automata Rule Forming Domino Patterns. In: Malyshkin V. (eds) Parallel Computing Technologies. PaCT 2019. Lecture Notes in Computer Science, vol 11657, pp. 334–344. Springer
- [12] Hoffmann, R. and Seredyński, F. (2020). Covering the space with sensor tiles. In International Conference on Cellular Automata for Research and Industry (pp. 156–168). Springer, Cham.
- [13] Hoffmann, R., Désérable, D. and Seredyński, F. (2021). A cellular automata rule placing a maximal number of dominoes in the square and diamond. J Supercomput 77, 9069 — 9087
- [14] Désérable, D., Hoffmann, R., Seredyński, F. (2023). Maximal Arrangement of Dominos in the Diamond. arXiv preprint arXiv:2305.04544.
- [15] Hoffmann, R., Désérable, D., and Seredyński, F. (2021). Minimal Covering of the Space by Domino Tiles. In International Conference on Parallel Computing Technologies (pp. 453–465). Springer, Cham.
- [16] Hoffmann, R., Forming Point Patterns by a Probabilistic Cellular Automata Rule. Article arXiv:2202.06656 (2022).
- [17] Hoffmann, R., Generating Loop Patterns with a Genetic Algorithm and a Probabilistic Cellular Automata Rule. Algorithms 16.7 (2023): 352.
- [18] Holland, J.H. Genetic algorithms. Sci. Am. 1992, 267, 66-73.
- [19] Holland, J.H. Genetic algorithms. Scholarpedia 2012, 7, 1482.
- [20] Katsumata, Y., Ishida, Y., On a membrane formation in a spatio-temporally generalized Prisoner's Dilemma, Cellular Automata, Umeo, H., Morishita, S., Nishinari, K., Komatsuzaki, T., Bandini, S. (eds) LNCS 5191 (2008) 60-66
- [21] Mitchell, M. Genetic algorithms: An overview. In Complex; Wiley: Hoboken, NJ, USA, 1995; Volume 1, pp. 31–39

- [22] Mitchell, Melanie, James P. Crutchfield, and Rajarshi Das. Evolving cellular automata with genetic algorithms: A review of recent work. Proceedings of the First international conference on evolutionary computation and its applications (EvCA'96). Vol. 8. 1996.
- [23] Nagpal, R., Programmable pattern-formation and scale-independence, Unifying Themes in Complex Systems IV, Minai, A.A, Bar-Yam, Y., eds., (2008) 275–282
- [24] Nash, J.F., Equilibrium points in n--person games, PNAS USA 36(1) (1950) 48-49
- [25] Nowak, M., May, R., Evolutionary games and spatial chaos, Nature 359 (1992) 826-829
- [26] Nowak, M., Sigmund, K., Tit—for—Tat in heterogeneous populations, Nature 355 (1992) 250–253
- [27] Nowak, M.A., May, R.M. The spatial dilemmas of evolution. Int. J. of Bifurcation and Chaos 3(1), 35–78 (1993)
- [28] Nowak, M., Five rules for the evolution of cooperation, Science 314 (2006) 1560–1563
- [29] Rapoport, A., Chammah, PL., Prisoner's Dilemma, Univ. Michigan Press (1965)
- [30] Tretyakova, A., Seredyński, F., Bouvry, P., Cellular automata approach to maximum lifetime coverage problem in wireless sensor networks, Cellular Automata, W, as, J., Sirakoulis, G., Bandini, S., (eds.) LNCS 8751 (2014) 437–446
- [31] Seredyński, F., Kulpa, T., Hoffmann, R., and Désérable, D., Coverage and Lifetime Optimization by Self-optimizing Sensor Networks, Sensors 23(8): 3930 (2023)
- [32] Seredyński, F., Kulpa, T., Hoffmann, R., Evolutionary self-optimization of large CA-based multi-agent systems, J. Computational Science 68 (2023) 101994
- [33] Szaban, M., Seredyński, M., Hoffmann, R., Désérable, D. and Seredyński, F. Pattern Formation by Collective Behavior of Competing Cellular Automata-Based Agents. In International Conference on Cellular Automata for Research and Industry ACRI (pp. 34-44). (2024, September). Cham: Springer Nature Switzerland.
- [34] Wang, Jianrui, et al. Cooperative and competitive multi-agent systems: From optimization to games. IEEE/CAA Journal of Automatica Sinica 9.5 (2022): 763–783.
- [35] Wolfram, S. (2002). A New Kind of Science. Wolfram Media, Champaign, IL
- [36] Yamins, D., Nagpal, R., Automated Global—to—Local programming in 1D spatial multi-agent systems, Proc. 7th Int. Conf. AAMAS (2008) 615–622