Complexity of Spatial Games

Krishnendu Chatterjee
Institute of Science and Technology Austria, Klosterneuburg, Austria

Rasmus Ibsen-Jensen
University of Liverpool, UK

Ismaël Jecker
University of Warsaw, Poland

Jakub Svoboda
Institute of Science and Technology Austria, Klosterneuburg, Austria

Abstract

Spatial games form a widely-studied class of games from biology and physics modeling the evolution of social behavior. Formally, such a game is defined by a square (d by d) payoff matrix M and an undirected graph G. Each vertex of G represents an individual, that initially follows some strategy $i \in \{1, 2, \ldots, d\}$. In each round of the game, every individual plays the matrix game with each of its neighbors: An individual following strategy i meeting a neighbor following strategy j receives a payoff equal to the entry (i, j) of M. Then, each individual updates its strategy to its neighbors' strategy with the highest sum of payoffs, and the next round starts. The basic computational problems consist of reachability between configurations and the average frequency of a strategy. For general spatial games and graphs, these problems are in PSPACE. In this paper, we examine restricted setting: the game is a prisoner's dilemma; and G is a subgraph of grid. We prove that basic computational problems for spatial games with prisoner's dilemma on a subgraph of a grid are PSPACE-hard.

2012 ACM Subject Classification Theory of computation

Keywords and phrases spatial games, computational complexity, prisoner's dilemma, dynamical systems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.11

Funding *Krishnendu Chatterjee*: The research was partially supported by the ERC CoG 863818 (ForM-SMArt).
Ismaël Jecker: The research was partially supported by the ERC grant 950398 (INFSYS).
Jakub Svoboda: The research was partially supported by the ERC CoG 863818 (ForM-SMArt).

1 Introduction

Spatial evolutionary games is a classic and well-studied model of evolutionary dynamics on graphs, which has been studied across fields, e.g., biology [9, 8], physics [10, 14], and computer science [3, 2].

While computer science studies games with few players and a large number of actions, evolutionary game theory studies games with few actions and strategies but with many players (see the survey [17]). Specifically, each spatial evolutionary game consists of a square, skew-symmetric, bimatrix game (i.e. the outcome in entry (i, j) for player 1 is the same as the outcome for player 2 in (j, i) for all i, j) and a finite graph. The game is played over a number of rounds. Each node of the graph corresponds to a player. Each node/player is associated with a current row and corresponding column. In each round, each player plays the matrix game against each of their neighbors, by playing their row against their neighbor's column (because of the skew-symmetry, who plays rows and who plays columns does not matter) and gets a payoff assigned, which is the sum of outcomes of the games they played

in that round. Each player then switches to the row played by their neighbor that had the highest payoff (or keeps their strategy). Since this is a deterministic dynamic, whenever we reach a round such that there is a previous round in which each player had the same row as now, the game "loops". All spatial evolutionary games will therefore loop after at most d^n rounds when the bi-matrix is d by d and there are n nodes.

Most studied setup: Grids and prisoner's dilemma. The standard study of spatial evolutionary games focuses on small (2 by 2 or 3 by 3) bi-matrices and on grids. A good understanding of the dynamics is known for a number of such setups (including all 2 by 2 bi-matrices on 2-dimensional grids). Grids are typically chosen since in biology they naturally model how cells interact with nearby cells. In particular, the focus has been on prisoner's dilemma (PD) matrices: A prisoner's dilemma bi-matrix is a 2 by 2 bi-matrix in which the two rows are called cooperation and defection, such that it is an advantage to defect if the other player cooperates, but it is better for both players if they both cooperate as compared to the mutual defection. We study these games to better understand why cooperation develops as we see in humans and many animal species. Indeed, this was the focus of the original paper [8] and later works extended this basic model in myriad ways:

1. What happens if you add in mobile agents [19, 5]?
2. What about age[11, 21]?
3. What if nodes/players do not have the same objectives [18, 13]?
4. What if there were different timescales [14, 22]?

See also the survey [12]. A common generalization considers more general graph types. We mention a few examples of the papers pursuing this direction:

1. Kabir et al. showed how increasing the network reciprocity changes the likelihood of cooperation in PD [6].
2. Hassell et al. considered how multiple different species on models with islands influence the outcome [4].
3. Santos and Pacheco showed how scale-free networks (when generated following specific paradigms) promote cooperation [15].
4. Yamauchi et al. showed how, if both the neighbors and strategies can change over time, cooperation can evolve [16].
5. Ohtsuki et al. gave a simple heuristic for when cooperation can evolve on a variety of different networks, including social networks [9].

Computational problems. The works mentioned previously usually examine how cooperation spreads when the process is applied many times. The question they are trying to decide is: Given a starting position, what is the average number of cooperators in the long run?

Open questions. The general spatial games problem is in PSPACE. This is because a configuration consists of a strategy for every vertex, which can be stored in polynomial space, and the update of the configuration according to the rules can also be achieved in PSPACE. The most well-studied problem for spatial games is the prisoner's dilemma, which has only two strategies, namely, cooperation and defection. Moreover, such games have been studied for special classes of graphs. The main open question is whether efficient algorithms can be obtained for prisoner's dilemma on graphs like grids.

Our results. In this paper, we consider spatial evolutionary games with a prisoner's dilemma matrix on *subsets*[1] of 2-dimensional grids. The subset models the situation when locations of e.g. cells or connections between them have been destroyed or are otherwise inaccessible.

Our main result is that for subgraphs of a two-dimensional grid and prisoners dilemma the reachability and average cooperation problems are PSPACE-hard. Additionally, we show that induced subgraphs can loop in loops of exponential length. Subsets of grids are simpler than scale-free or evolving networks, so our hardness result holds for more general graphs.

2 Model and definitions

Graphs and grids. A graph $G = (V, E)$ consists of a set of vertices V and a set of undirected edges E. Every vertex is occupied by one individual. Edges determine pairs of individuals that interact. Two-dimensional grids are specific graphs where each vertex is assigned a unique pair of integers (i, j), and has (at most) 8 neighbors: vertices whose pairs differ by at most 1 in both coordinates. In our construction, we use graphs derived from grids: Given a grid (V, E), a *induced subgraph* of (V, E) is a graph $(V', (V' \times V') \cap E)$ with $V' \subseteq V$. An *subgraph* of (V, E) is a graph (V', E') with $V' \subseteq V$ and $E' \subseteq (V' \times V') \cap E$.

Games and individuals. An individual occupying a vertex has one of two types: cooperator if it plays C or defector if it plays D. In the figures, we use black for cooperators and white for defectors. We call *configuration* of a graph an assignment of each vertex to a strategy (cooperator or defector).

From all matrix games, we focus on prisoner's dilemma. The game is denoted by a matrix

	C	D
C	1	0
D	b	0

where $b > 1$. It means that C gets 1 for interacting with C, D gets b for interacting with C, and everyone gets 0 for interacting with D. We denote this game M^b.

Steps and updates. The evolution is simulated in rounds. In one round, every individual interacts with all neighbors and collects the total payoff. Then every individual compares the received payoff with the payoffs of neighbors. The individual keeps the strategy if it is the highest or changes the strategy to the strategy of a neighbor with the highest payoff (in a tie, defection is preferred).

We denote the proccess starting from position S on graph G with a game matrix M^b as $\mathcal{S}(G, M^b, S)$.

Complexity problems. We consider two complexity problems.

Reach : Given starting configuration, does the process reach a given configuration?

Avg : For a given starting configuration, what is the average number of cooperators (black vertices) in all succeeding configurations?

Note that for one configuration, there is only one possible succeeding configuration. This restricts the configuration graph. That means eventually the configuration graph creates a loop (as was noted before [20]).

[1] Either subgraphs or induced subgraphs

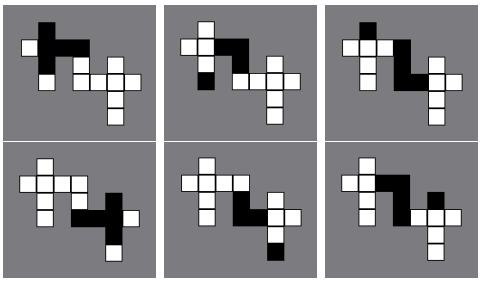

Figure 1 All configurations of the gadget c_3 initiated by the top left configuration. These configurations (in rows) show period 6.

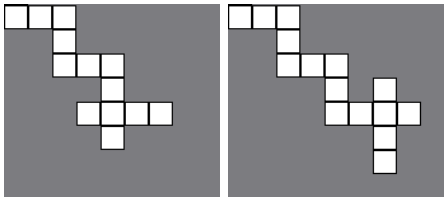

Figure 2 Extension of the gadget c_k by two cells. We extend the gadget to the lower right, the gadget itself can be as long as needed connected to the upper left which increases period by 2.

Ranges of b. There is a reasonable range for b in M^b. If $b < 1$, then the game is not a prisoner's dilemma. If b is larger than the maximal degree in a graph, the dynamic is trivial, a defector cannot become a cooperator. For our constructions in this paper, we suppose that $b \in (\frac{3}{2}, 2)$. In Appendix A, we show ideas explaining how our construction can be adapted to other values of b.

3 Exponential cycle

In this section, we show that even on an induced subgraph of a square grid, we observe a complex behavior. Namely, there exists a graph and a configuration that returns back to the starting configuration only after an exponential number of steps, we use $\tilde{(}\Omega)$ and $\tilde{(}\mathcal{O})$ which hide logarithmic factors.

▶ **Theorem 1.** *For $b \in (\frac{3}{2}, 2)$, there exists a graph G, induced subgraph of a square grid, with n vertices and a starting configuration S, such that it takes $2^{\tilde{\Omega}(\sqrt{n})}$ steps until $\mathcal{S}(G, M^b, S)$ reaches S again.*

Proof. We describe a family of gadgets and starting configurations with different periods, where *period* is the number of steps the gadget needs to return to the starting position again. Then we combine some of them to create a graph with a period equal to the lowest common multiple of the periods of its components.

For $k \geq 3$, we construct inductively c_k, a gadget that is an induced subgraph of a square grid with size $9 + 2k$ and period $2k$. The gadget c_3 in its starting configuration is depicted on Figure 1. By rearranging and adding two squares, we create the gadget c_{k+1} from c_k, as depicted on Figure 2.

For every integer $g > 1$, let us now define G_g as the disjoint union of the gadgets c_{p_2}, c_{p_3}, …, c_{p_g} where p_i is the i-th prime number. By the Chinese remainder theorem, the number of steps needed so that all gadgets are in the same state is the smallest common multiple of their periods, which is the product of the first g primes. We know that this number is $\Theta(e^{g \log g})$ from [7] and the Prime Number Theorem. Moreover, the number of squares (vertices) of G_g linear in the sum of the first g primes which is $\tilde{\mathcal{O}}(g^2)$.

Therefore, by using a induced subgraph of the square grid of size $n \in \mathbb{N}$, we can create a union of gadgets that has a period of size $2^{\tilde{\Omega}(\sqrt{n})}$. ◀

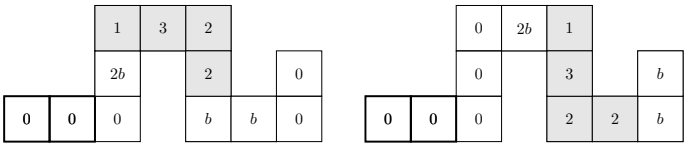

Figure 3 Sending signal through the wire with explicit payoffs with cooperators denoted by gray. Thicker vertices are input vertices.

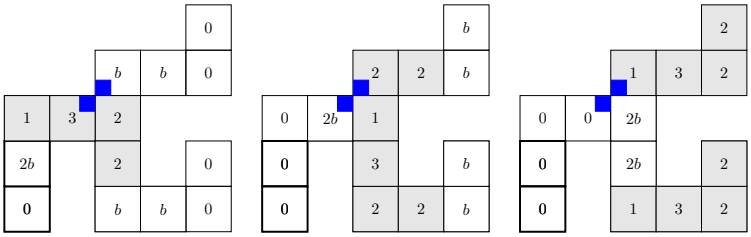

Figure 4 Splitting the signal in two. Blue boxes denote a deleted edge with cooperators denoted in black. Thicker vertices are input vertices.

4 Construction of a Turing machine

We show that both Reach and Avg are P-SPACE hard. For a polynomially bounded Turing machine and its input, we create a graph of a polynomial-size that is a subgraph of the square grid and an initial configuration of cooperators and defectors such that the process reaches a predefined configuration if and only if the Turing machine accepts the given input.

Since both problems, Reach and Avg , can be easily solved in P-SPACE by a simulation, that means these problems are P-SPACE complete.

▶ **Theorem 2.** *For $b \in (\frac{3}{2}, 2)$ holds:*

For any Turing machine T polynomially bounded by n and its input, there exists a subgraph of a square grid G with poly(n) vertices, a starting configuration S, and a target configuration Q, such that $\mathcal{S}(G, M^b, S)$ reaches Q if and only if T accepts the given input.

Moreover, for any Turing machine T and its input I, there exists a subgraph of an infinite square grid G, a starting configuration S with poly$(|I|)$ cooperators, and a target configuration Q, such that $\mathcal{S}(G, M^b, S)$ reaches Q if and only if T accepts the given input.

We construct G as a white (defector) graph with a few black (cooperator) vertices that carry signals and store data to simulate the behavior of T. On our figures, we use white for defectors, black for cooperators, gray for deleted vertices, and blue for *deleted edges*: To visualize deleted edges, we subdivide each square denoting some vertex v into 9 parts. The middle square corresponds to v itself, and the other squares denote the neighbors in relative position to v (upper-left, upper-middle, ...). To represent that an edge between two vertices v and w is deleted, we color in blue the subsquare corresponding to w in v's square and the subsquare corresponding to v in w's square.

First, we describe wires and basic logic gates. With that, we use a construction described in [1]. We use Lemma 7 and 9 from the paper, but explain technicalities emerging from more restrictive construction (the graph is a subgraph of a square grid). Both lemmas use simple gadgets to create functions and then a whole Turing machine.

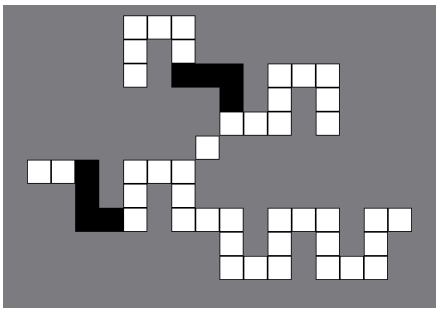

Figure 5 NOT gate: the upper signal is negated, the lower signal is a clock signal.

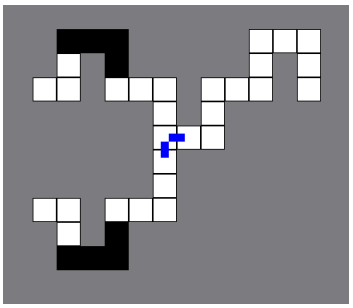

Figure 6 AND gate: both signals are essential to tunnel through. There are two deleted edges denoted by blue boxes.

4.1 Basic gadgets

We describe the computational gadgets used in our construction. Every gadget g has one or two inputs (I_1, I_2) and one or two outputs (O_1, O_2). We imagine the inputs being on the left and bottom and the outputs on the right and top. Every gadget fits into a constantly sized rectangle of the grid.

If at time t at least one input of g is true (input vertices are cooperators and other vertices are defectors), then at time $t + t_g$ it outputs true or false based on a function g computes.

One input (or output) consists of two connected vertices. We say that the input is true if both vertices are cooperators at time t and $t + 1$, and in the first step, they can convert all neighbors to cooperators. The previous gadget (connected by its output to the given input) is responsible for turning the input vertices back to defectors at time $t + 2$.

Here, some of the gadgets need to cross signals, we describe how to do it later:

Wire: it transmits a signal (See Figure 3). The wire has one input, one output, and if I_1 is true at time t, then O_1 is true at time $t + 2$.

Splitter: it splits one signal in two (See Figure 4). The splitter has one input and two outputs. If I_1 is true at time T, then O_1 and O_2 are true at time $t + 4$. An interesting property of the splitter is that it does not matter if the signal arrives from I_1 or O_1. From the graph perspective, these two are symmetric.

NOT gate: it computes the logical negation, but necessitates a clock signal (See Figure 5). The NOT gate has two inputs and one output. Input I_2 is a clock input: if I_1 and I_2 are true at time t, then O_1 is false at time $t + 8$, if only I_2 is true at time t, then O_1 is true at time $t + 8$. Note that the NOT gate actually computes the function $\neg I_1 \text{AND} I_2$, also the gates I_1 and O_1 are interchangeable.

AND gate: it computes the logical conjunction (See Figure 6). The AND gate has two inputs and one output. If both I_1 and I_2 are true at time t, then O_1 is true at time $t + 8$. Otherwise, O_1 is false.

XOR gate: it computes logical exclusive disjunction. We can create it from other gadgets, but it makes our construction easier.

We compute XOR using connection of splitter and NOT gates (see Figure 7). It has two inputs I_1 and I_2 and one output O_1. First every input is split to get $I_{1,1}, I_{1,2}, I_{2,1}$ and $I_{2,2}$, then $I_{1,1}$ is sent to I_1 of NOT gate and $I_{2,1}$ is sent to O_1 of a splitter S. We already know that if only one signal equals true, then S splits the signal. If both are positive, signals annihilate each other. So O_2 of S has value $I_1 \text{XOR} I_2$. The only problem is when

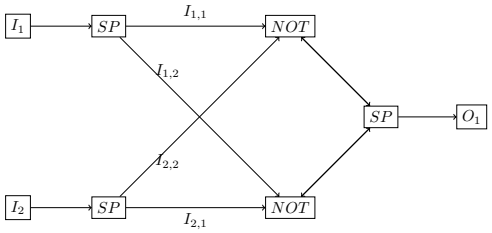

Figure 7 XOR gate: Combination of previous gates. Signals $I_{1,2}$ and $I_{2,2}$ are delayed such that they arrive at the NOT gate when signal $I_{1,1}$ or $I_{2,1}$ from the central splitter would.

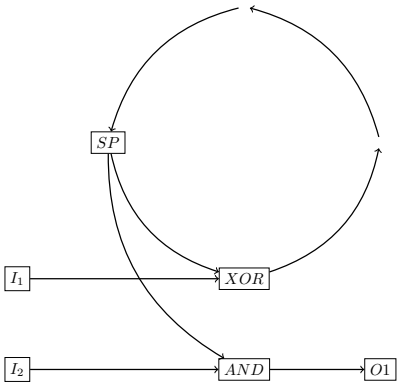

Figure 8 Storage unit.

I_1 is true, and I_2 is not (or symmetric). Then the signal travels back from the splitter through wire $I_{2,1}$, then we use the NOT gate with clock signal $I_{2,1}$ and signal $I_{1,2}$ that stops it.

OR gate: it computes logical disjunction. It has two inputs I_1 and I_2, and the output O_1 satisfies the function $(I_1 \text{AND} I_2)\text{XOR}(I_1 \text{XOR} I_2)$, it consists of gadgets described above. Note that we don't need the NOT gate directly, so clock signal is not necessary for that gadget.

Another gadget that the construction needs is a storage unit. It has an inner state $S \in \{0, 1\}$, two inputs and one output, see Figure 8. The storage unit consists of a big cyclic wire where a signal loops if the stored value S is 1. If a signal is sent via I_1, then the storage unit changes state: $S' = \neg S$, this is ensured by a XOR gate. On the cycle, there is a splitter that splits the signal towards an AND gate. If a signal is sent via I_2, S does not change, but the signal reaches AND and is sent to O_2 if and only if $S = 1$. The storage unit requires synchronicity, the signal from the input has to reach the splitter or XOR at the right time. But this is not hard to ensure by longer wires.

4.2 Connecting the graph

To make the graph a subgraph of a square grid, we need to prove two things: we can cross two signals, and we can ensure that the signals meet at the right place, at the right time.

Crossing. We use the structure described in Figure 13. Crossing accepts only one signal at a time and supposes that no other signal arrives for 10 steps afterward. Crossing ensures that in a square subset of a grid, the signal can travel only from upper left to lower right corner and from lower left to upper right without spreading or dying out. If a signal arrives, it is spreading to the other input and all the outputs, but the wires close to the active input get stopped, so only the wire that is not adjacent to the input wire continues to carry the signal.

Making the construction on square grid. On the Figure 9, we see that a signal starting at position $(0, 0)$ going to position (x, y) can take between $\max(x, y)$ and $\frac{1}{8} \cdot xy$ steps. The signal also does not leave the rectangle dentoted by points $(0, 0)$ and (x, y).

Every gadget that was described above can be padded by wires of different densities such that the gadget fits into a rectangle with predetermined width and height. Moreover, all inputs are in the same position (relative to the rectangle) and the time of evaluation is the same for all (constant).

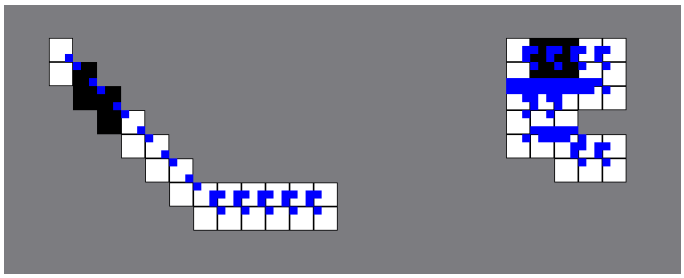

Figure 9 Two wires of different lengths connecting two points.

Then everything in the following construction can be viewed as placing a column of tiles (gadgets) one after another, where columns are connected by short wires.

4.3 Function construction

Here we construct a function using previously described gadgets. We bound the number of vertices and steps needed for the construction and the function evaluation.

We imagine a function as signals going from left to right. The input and output signals have constant horizontal distance.

▶ **Definition 3.** *We say that a function $f : \{0,1\}^k \to \{0,1\}^l$ is computed by a graph G (subgraph of a grid) in time g and space h if G has h vertices, k inputs I_1, I_2,..., I_k and l outputs O_1, O_2, ..., O_l spatially arranged, and such that for all $(x_1, x_2, \ldots, x_k) \in \{0,1\}^k$, if at the time t we have $I_j = x_j$ for all $1 \leq j \leq l$ (and all the other vertices are defectors), then at the time $t + g$ we have $O_j = y_j$ for all $1 \leq j \leq l$, where $f(x_1, x_2, \ldots, x_k) = (y_1, y_2, \ldots, y_l)$.*

Note that all our basic gadgets have a constant size. So, when analyzing the asymptotic space complexity, we can consider these gadgets and single vertices interchangeably.

▶ **Lemma 4.** *Computing the function $f(x_1, x_2, \ldots, x_c) = (x_1, x_2, \ldots, x_c, x_1, x_2, \ldots, x_c)$ takes time $\mathcal{O}(c)$ and space $\mathcal{O}(c^2)$.*

Proof. We describe the gadget computing f. First, we split every signal and then cross every copy of it with others to the right place. One signal needs to cross $\mathcal{O}(c)$ others, so that is the number of steps and we have c signals going through a wire of length c, that makes $\mathcal{O}(c^2)$ gadgets. ◀

▶ **Lemma 5.** *Let $c \in \mathbb{N}$. Every function $f : \{0,1\}^c \to \{0,1\}$ mapping to 0 every tuple whose first component is 0 can be computed in space $\mathcal{O}(3^c)$ and time $\mathcal{O}(c^2)$.*

Proof. We use an induction on the dimension c of the domain. If $c = 1$, since f satisfies $f(0) = 0$ by supposition, either f is the identity, which is realised by a wire, or f is the zero function, which is realised by simply disconnecting the input and output.

Now, suppose that $c > 1$, and we can realise any function $f : \{0,1\}^{c-1} \to \{0,1\}$ that satisfies the condition. In particular, there exist two gadgets g_0 and g_1 computing

$$\begin{array}{rccc} f_0 : & \{0,1\}^{c-1} & \to & \{0,1\}, \\ & (x_1, x_2, \ldots, x_{c-1}) & \mapsto & f(x_1, x_2, \ldots, x_{c-1}, 0), \\ f_1 : & \{0,1\}^{c-1} & \to & \{0,1\}, \\ & (x_1, x_2, \ldots, x_{c-1}) & \mapsto & f(x_1, x_2, \ldots, x_{c-1}, 1). \end{array}$$

Having these values, the computation is straightforward as the value $f(x_1, x_2, \ldots, x_c)$ is given by the formula

$$(f_0(x_1, x_2, \ldots, x_{c-1}) \wedge \neg x_c) \vee (f_1(x_1, x_2, \ldots, x_{c-1}) \wedge x_c).$$

To construct a gadget g that is a subgraph of a grid g and computes f, we proceed as follows.

Computing the function. We apply the gadget described in Lemma 4 to the input, and we use a splitter to get one more signal x_1. At this point, we have the arranged signals $x_1,\ x_1,\ x_2,\ x_3,\ \ldots,\ x_c,\ x_1,\ x_2,\ x_3,\ \ldots,\ x_c$. We apply the gadgets computing f_0 and f_1 to get the signals $f_0(x_1, x_2, \ldots, x_c)$ and $f_1(x_1, x_2, \ldots, x_c)$, now the signals are arranged as $x_1, f_0(x_1, x_2, \ldots, x_{c-1}), x_c, f_1(x_1, x_2, \ldots, x_{c-1}), x_c$. We cross the signals x_1 and $f_0(x_1, x_2, \ldots, x_{c-1})$, and then we use x_1 as clock signal towards a NOT gate with x_c. By this, we either get the signal $\neg x_c$, or x_1 is zero, then the result should be zero anyway.

Now we send $f_0(x_1, x_2, \ldots, x_{c-1})$ and $\neg x_c$ towards an AND gate, similarly we send $f_1(x_1, x_2, \ldots, x_{c-1})$ and x_c towards another AND gate, and finally we send both results towards an OR gate.

Size of the gadget. Now, we show that the computation is fast and requires a reasonable number of vertices. Let there be a recurrent formula R_s that maps any integer c to the maximal number of gadgets needed to compute the function f. Using induction, we see that it satisfies:

$$R_s(c) = \mathcal{O}(c^2) + 2R_s(c-1) + \mathcal{O}(1).$$

Therefore, $R(c) \in \mathcal{O}(3^c)$. Similarly, we use recurrent formula for the time needed

$$R_t(c) = \mathcal{O}(c) + R_t(c-1) + \mathcal{O}(1)$$

and the solution for this recurrence is $\mathcal{O}(c^2)$. ◀

▶ **Lemma 6.** *Let $c, d \in \mathbb{N}$. Every function $f : \{0,1\}^c \to \{0,1\}^d$ mapping to $(0, 0, \ldots)$ every tuple whose first component is 0 can be computed in space $\mathcal{O}(d3^c)$ and time $\mathcal{O}(c^2 + c\log(d))$.*

Proof. The idea is easy, we split the inputs d times using Lemma 4 and then we use Lemma 5 for every copy.

Multiplying the inputs needs $\mathcal{O}(d)$ described in Lemma 4. We can arrange them in layers where every layer doubles the input, so the whole multiplying takes time $\mathcal{O}(\log(d) \cdot c)$.

Then we use d gadgets described in Lemma 5 in parallel which gives time $\mathcal{O}(c^2 + c\log(d))$ and space $\mathcal{O}(d3^c)$. ◀

4.4 Blob and connections

▶ **Lemma 7.** *Let T be a Turing machine. For every input u evaluated by T using $C \in \mathbb{N}$ cells of the tape, there exists a subgraph of a grid G on $\mathcal{O}(C)$ vertices and an initial configuration c_0 of G such that T stops over the input u if and only if $\mathcal{S}(G, M^b, c_0)$ for $b \in (\frac{3}{2}, 2)$ eventually reaches a configuration without cooperators.*

Proof. We suppose that the Turing machine T has a single final state, which can only be accessed after clearing the tape. We present the construction of the graph G simulating T through the following steps. First, we encode the states of T, the tape alphabet, and the transition function in binary. Then, we introduce the notion of a blob, the building block of G, and we show that blobs accurately simulate the transition function of T. Afterward, we approximate the size of a blob, and finally, we define G as a composition of blobs.

Binary encoding. Let $T_s \in \mathbb{N}$ be the number of states of T, and $T_a \in \mathbb{N}$ be the size of its tape alphabet. We pick two small integers s and n satisfying $T_s \leq 2^{s-1}$ and $T_a \leq 2^{n-1}$. We encode the states of T as elements of $\{0,1\}^s$, and the alphabet symbols as elements of $\{0,1\}^n$, while respecting the following three conditions: the blank symbol maps to 0^n, the final state of T maps to 0^s, and all the others map to strings starting with 1. Then, for these mappings, we modify the transition function of T to:

$$F : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^s \times \{0,1\}^s \times \{0,1\}^n.$$

Instead of using one bit to denote if the head is going left or right, we use $2s$ bits to store the state and signify the movement: if the first s bits are zero, the head is moving right; if the second s bits are zero, it is moving left; if the first $2s$ bits are zero, the computation ended. Moreover, the last n bits of the image of F do not encode the new symbol, but the symmetric difference between the previous and the next symbol: if the i-th bit of the tape symbol goes from y_i to z_i, then F outputs $d_i = y_i \oplus z_i$ (XOR of these two).

Constructing blobs. We construct the graph G by simulating each cell of the tape with a blob. Blob stores a tape symbol, and after receiving a signal corresponding to a state of T it computes the transition function. The main components of a blob are as follows.

- Memory: n storage units (m_1, m_2, …, m_n) are used to keep in memory a tape symbol $a \in \{0,1\}^n$ of T.
- Receptor: $2s$ inputs (I_1, I_2, …, I_{2s}) are used to receive states $q \in \{0,1\}^s$ of T either from the left or from the right.
- Transmitter: $2s$ outputs (O_1, O_2, …, O_{2s}) are used to send states $q \in \{0,1\}^s$ of T either to the right or to the left.
- Transition gadget: We use gadget from Lemma 6, it needs $\mathcal{O}((n+s)3^{n+s})$ space and $\mathcal{O}((n+s)^2)$ time.

Blobs are connected in a row to act as a tape: for every $1 \leq i \leq s$, the output O_i of each blob connects to the input I_i of the blob to its right, and the output O_{s+i} of each blob connects to the input I_{s+i} of the blob to its left. When receiving a signal, the blob transmits the received state and the tape symbol stored in memory to the transition gadget g_F, which computes the corresponding transition, and then apply its results. We now detail this inner behavior. Note that when a gadget is supposed to receive simultaneously a set of signals coming from different sources, it is always possible to add wires of adapted length to ensure that all of them end up synchronized.

Simulating the transition function. To simulate the transition function of T, a blob acts according to the three following steps:

1. Transmission of the state. A blob can receive a state either from the left (through inputs $I_1, I_2, \ldots, I_s$) or from the right (through inputs $I_{s+1}, I_{s+2}, \ldots, I_{2s}$), but not from both sides at the same time, since at every point in time there is at most one active state. Therefore, if for every $1 \leq i \leq s$ we denote by x_i the disjunction of the signals received by I_i and I_{s+i}, then the resulting tuple $(x_1, x_2, \ldots, x_s)$ is equal to the state received as signal (either from the left or the right), which can be fed to the gadget g_F. Formally, the blob connects, for all $1 \leq i \leq s$, the pair I_i, I_{s+i} to an OR gate whose output is linked to the input I_i of g_F.
2. Transmission of the tape symbol. Since the first component of any state apart from the final state is always 1, whenever a blob receives a state, the component x_1 defined in the previous paragraph has value 1. The tape symbol $(y_1, y_2, \ldots, y_n)$ currently stored in the

blob can be obtained by sending, for every $1 \leq i \leq n$, a copy of x_1 to the input I_2 of the storage unit s_i, causing it to broadcast its stored state y_i. The tuple continues to the gadget g_F. Formally, the blob uses n splitters to transmit the result of the OR gate between I_1 and I_{s+1} to the input I_1 of each storage unit. Then, for every $1 \leq i \leq n$, the output O_1 of the storage unit s_i is connected to the input I_{s+i} of g_F.

3. Application of the transition. Upon receiving a state and a tape symbol, g_F computes the result of the transition function, yielding a tuple $(r_1, r_2, \ldots, r_{s+n})$. The blob now needs to do two things: send a state to the successor blob and update the element of the tape.

Connecting the output O_i of g_F to the output O_i of the blob for every $1 \leq i \leq 2s$ ensures that the state is sent to the correct neighbor: the values $(r_1, r_2, \ldots, r_s)$ are nonzero if the head is supposed to move to the next block on the right (outputs $O_1, O_2, \ldots, O_s$ are connected that blob). Conversely, $(r_{s+1}, r_{s+2}, \ldots, r_{2s})$ are nonzero if the head is supposed to move to the left, (outputs $O_{s+1}, O_{s+2}, \ldots, O_{2s}$ are connected to the left blob).

Finally, connecting the output O_{2s+i} of g_F to the input I_1 of s_i for all $1 \leq i \leq n$ ensures that the state is correctly updated: this sends the signal d_i to the input I_1 of the storage unit s_i. Since d_i is the difference between the current bit and the next, the state of s_i will change if it has to.

The size of blob. The size of the blob is determined by the size of the transition gadget g_F: one blob is composed of $\mathcal{O}(3^{n+s})$ vertices, and evaluating a transition requires $\mathcal{O}((n+s)^2)$ steps by Lemma 6. Since n and s are constants (they depend on T, and not on the input u), the blob has constant size. Crossing wires to get them to the right place also takes space $\mathcal{O}((n+s)^2)$.

Constructing G. Now that we have blobs that accurately simulate the transition function of T, constructing the graph G mimicking the behavior of T over the input u is straightforward: we take a row of C blobs (remember that $C \in \mathbb{N}$ is the number of tape cells used by T to process u). Since the size of a blob is constant, the size of G is polynomial in C. We define the initial configuration of G by setting the states of the $|u|$ blobs on the left of the row to the letters of u, and setting the inputs I_1 to I_s of the leftmost blob to the signal corresponding to the initial state of T as if it was already in the process. As explained earlier, the blobs then evolve by following the run of T. If the Turing machine stops, then its tape is empty, the final state is encoded by 0^s, and the blank symbol is encoded by 0^n. So G reaches the configuration where all vertices are defectors. Conversely, if T runs forever starting from the input u, there will always be some cooperators vertices in G to transmit the signal corresponding to the state of T. ◀

Proof of Theorem 2,part 1. By Lemma 7, we can reduce any problem solvable by a polynomially bounded Turing machine into Reach , asking whether we reach the configuration with only defectors. Or into Avg , asking whether the long-run average is strictly above 0. ◀

Proof of Theorem 2,part 2. Again, by Lemma 7, we can create an infinite chain of blobs and then we can reduce any problem solvable by any Turing machine into Reach , asking whether we reach the configuration with only defectors, or into Avg , asking whether the long-run average is strictly above 0. ◀

5 Conclusion

Our result shows that going beyond the basic grid model even slightly makes spatial games PSPACE-hard. It ensures that there is no efficient (polynomial-time) algorithm for Reach or Avg , even for the simplest game of prisoner's dilemma.

We studied games with synchronous and deterministic updating. It poses a question: does lifting any restriction permit an efficient algorithm? Moreover, we can ask whether it is possible to construct graphs on which the game has certain properties, such as: are there graphs where the cooperative behavior is favored?

References

1 Krishnendu Chatterjee, Rasmus Ibsen-Jensen, Ismaël Jecker, and Jakub Svoboda. Simplified game of life: Algorithms and complexity. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 22:1–22:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.22.

2 Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player nash equilibria. *Journal of the ACM (JACM)*, 56(3):1–57, 2009.

3 Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

4 Michael P. Hassell, Hugh N. Comins, and Robert M. May. Species coexistence and self-organizing spatial dynamics. *Nature*, 370:290–292, 1994.

5 Dirk Helbing and Wenjian Yu. The outbreak of cooperation among success-driven individuals under noisy conditions. *Proceedings of the National Academy of Sciences*, 106(10):3680–3685, 2009.

6 KM Ariful Kabir, Jun Tanimoto, and Zhen Wang. Influence of bolstering network reciprocity in the evolutionary spatial prisoner's dilemma game: a perspective. *The European Physical Journal B*, 91(12), December 2018.

7 Sebastián M. Ruiz. 81.27 a result on prime numbers. *The Mathematical Gazette*, 81:269, July 1997. doi:10.2307/3619207.

8 Martin A. Nowak and Robert M. May. Evolutionary games and spatial chaos. *Nature*, 359(6398):826–829, October 1992. doi:10.1038/359826a0.

9 Hisashi Ohtsuki, Christoph Hauert, Erez Lieberman, and Martin A. Nowak. A simple rule for the evolution of cooperation on graphs and social networks. *Nature*, 441(7092):502–505, May 2006. doi:10.1038/nature04605.

10 Hisashi Ohtsuki, Martin A. Nowak, and Jorge M. Pacheco. Breaking the symmetry between interaction and replacement in evolutionary dynamics on graphs. *Phys. Rev. Lett.*, 98:108106, March 2007. doi:10.1103/PhysRevLett.98.108106.

11 Matjaž Perc and Attila Szolnoki. Social diversity and promotion of cooperation in the spatial prisoner's dilemma game. *Phys. Rev. E*, 77:011904, January 2008.

12 Matjaž Perc and Attila Szolnoki. Coevolutionary games—a mini review. *Biosystems*, 99(2):109–125, 2010.

13 Matjaž Perc and Zhen Wang. Heterogeneous aspirations promote cooperation in the prisoner's dilemma game. *PLOS ONE*, 5(12):1–8, December 2010.

14 Carlos P. Roca, José A. Cuesta, and Angel Sánchez. Time scales in evolutionary dynamics. *Phys. Rev. Lett.*, 97:158701, October 2006.

15 Francisco C. Santos and Jorge M. Pacheco. Scale-free networks provide a unifying framework for the emergence of cooperation. *Phys. Rev. Lett.*, 95, August 2005.

16 Francisco C. Santos, Jorge M. Pacheco, and Tom Lenaerts. Cooperation prevails when individuals adjust their social ties. *PLOS Computational Biology*, 2(10):1–8, October 2006. doi:10.1371/journal.pcbi.0020140.

17 Gyorgy Szabo and Gabor Fath. Evolutionary games on graphs. *Physics Reports*, 446(4):97–216, 2007.

18 Attila Szolnoki and Gyorgy Szabó. Cooperation enhanced by inhomogeneous activity of teaching for evolutionary prisoner's dilemma games. *Europhysics Letters (EPL)*, 77(3):30004, January 2007. doi:10.1209/0295-5075/77/30004.

19 Mendeli H. Vainstein, Ana T.C. Silva, and Jeferson J. Arenzon. Does mobility decrease cooperation? *Journal of Theoretical Biology*, 244(4):722–728, 2007.

20 Virgil. *Eclogue IV*. Cambridge University Press, 2008.

21 Zhen Wang, Zhen Wang, Xiaodan Zhu, and Jeferson J. Arenzon. Cooperation and age structure in spatial games. *Phys. Rev. E*, 85:011149, January 2012.

22 Zhi-Xi Wu, Zhihai Rong, and Petter Holme. Diversity of reproduction time scale promotes cooperation in spatial prisoner's dilemma games. *Phys. Rev. E*, 80:036106, September 2009.

A Construction for a more general b

We can make a similar construction for different b's than those from the range $(\frac{3}{2}, 2)$. By selecting different b, the construction is not a subgraph of a square grid.

We can interpret the wire from Figure 3 as a path (lower row of vertices) with auxiliary vertices (upper row) that empower the tip of the cooperator signal and also help the first defector behind it. For different b's, we can add more auxiliary vertices. Details are on Figure 10.

For setting $b \in (\frac{3}{2}, 2)$, the path and auxiliary vertices interchangeable, it's not the case for different bs. Our construction uses this, but only to preserve the topology properties. When we add k auxiliary vertices, we need to ensure that $kb > k - 1$ and $k - 1 > b$. For small b, we change the signal that it does not span two consecutive slices, but one (as on Figure 10.

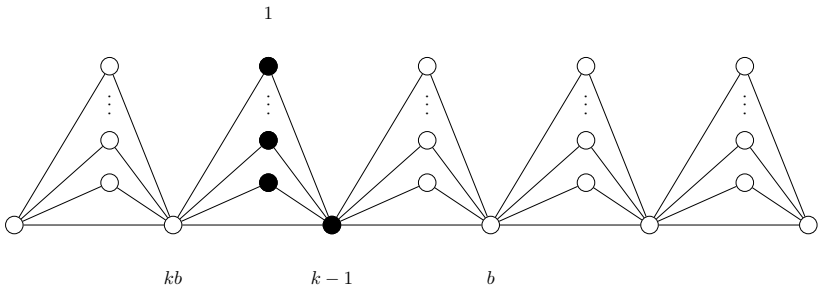

Figure 10 Schematic wire construction for general b with the slices of size k. The numbers are payoffs of important vertices.

B More gadgets

In this section, we provide more detailed figures for some gadgets. Moreover, you can see videos of some gadgets in action at https://pub.ist.ac.at/~jsvoboda/research/spatial_games/.

On Figure 11 and Figure 12 we see explicit graph shown on Figure 3 and Figure 4.

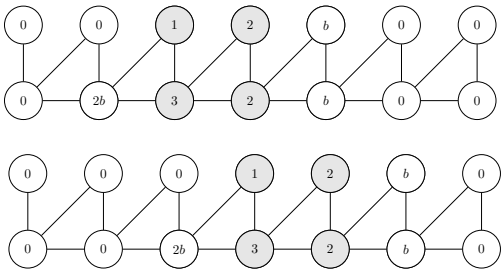

Figure 11 Sending signal throught the wire with explicit payoffs.

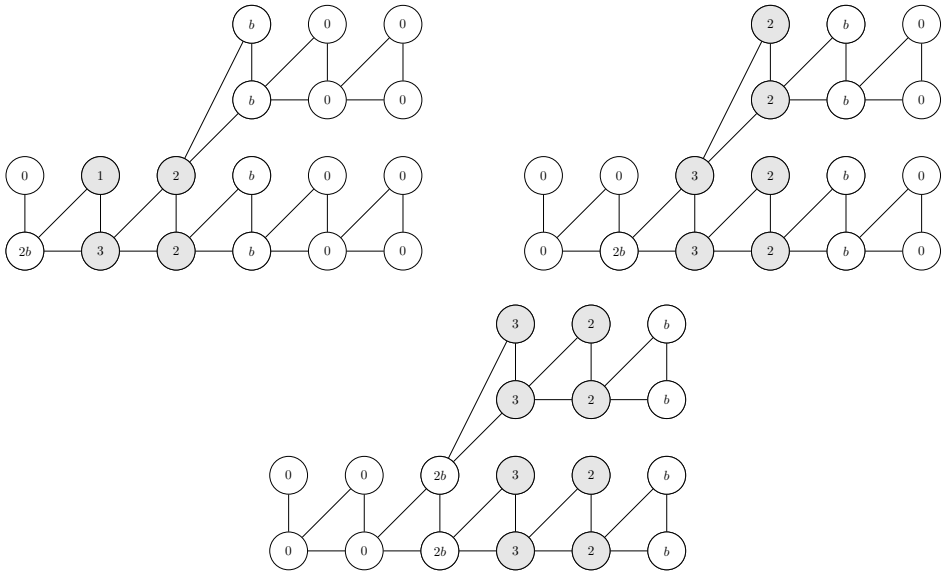

Figure 12 Splitting the signal in two. Note that the splitter does not have a direction (a signal from any input/output is split and sent by the other two inputs/outputs).

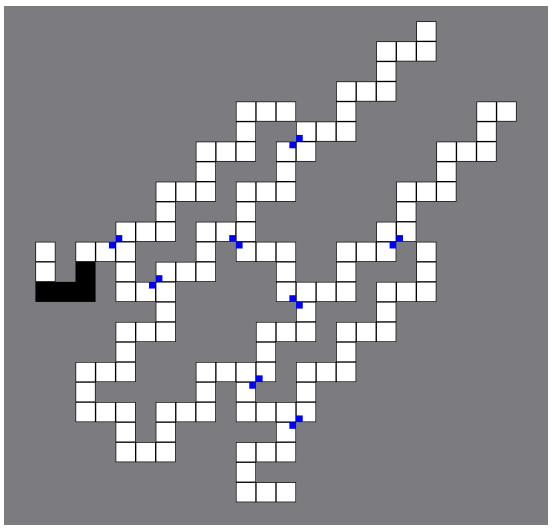

Figure 13 Crossing of two wires. Signal coming from left leaves the gadget by the right wire and signal coming from bottom leaves the gadget by the top wire.