Journal of Membrane Computing (2023) 5:117–128
https://doi.org/10.1007/s41965-023-00123-y

RESEARCH PAPER

On evolving environment of 2D P colonies: ant colony simulation

Miroslav Langer[1] · **Daniel Valenta**[2]

Received: 31 January 2023 / Accepted: 17 June 2023 / Published online: 1 July 2023

Abstract
P colonies are very simple membrane systems originally derived from the P systems. The 2D P colonies, as a version of P colonies with a two-dimensional environment, were introduced as a theoretical model of the multi-agent system for observing the behavior of a community of very simple agents living in a shared environment. Each agent is equipped with a set of programs consisting of a small number of simple rules. These programs allow the agent to act and move in the environment. Although, the 2D P colonies proved to be suitable for the simulations of various (not only) multi-agent systems, and natural phenomena, like the flash floods, there are phenomena which they are not able to simulate without some additional features or characteristics. One of the ways the agents can share the information is to use the stigmergy, which means to leave some special symbols in the environment. In this paper, we follow our previous research on the 2D P colony. We present a model of the 2D P colony with evolving environment, which allows us to simulate phenomena like the stigmergy, hence to simulate an ant colony.

Keywords Ant colony simulation · 2D P colony · P systems · Evolving environment · Optimization · Bio-inspired computation

Mathematics Subject Classification 68Q01 · 68Q07 · 68Q42

1 Introduction

One of the approaches to solving optimization problems found its inspiration in nature. Those methods use a swarm intelligence and, hence, are inspired by social behavior of a pack or a swarm of animals or by a communication based on chemical substances, pheromones, left in an environment by the ants (see [1]).

Another approach is using bio-inspired computational models either for solving the optimization problems directly, or for simulating above mentioned multi-agent systems.

P systems and their variants are used for both approaches. A P system is a bio-inspired computational model. It is based upon the structure of biological cells, abstracting from the way the chemicals interact and pass through cell membranes (see [2]).

The field of application of the P systems is very wide. In [3], Xingqiao Deng et al. used P systems to improve the efficiency and accuracy of reducer lubrication. Liang Huang et al. proposed new variant of tissue P system (TPS) in order to systematically optimize the process with multiple productive objectives (see [4]). The detection of malicious URLs is solved with a help of P systems in a paper by Wang Bo et al. (see [5]). P systems, respectively, Optimization Spiking Neural P System (OSNPS), were also used for solving combinatorial optimization problems (see [6–9]).

P systems themselves were used to simulate (not only) ant colonies, as well.

Great piece of work in the field of simulation of P systems was done by the research group lead by Prof. Catalin Buiu. Open-source simulators of standard and enzymatic numerical P systems, and P colonies and P swarms are accessible

✉ Miroslav Langer
miroslav.langer@vsb.cz

Daniel Valenta
daniel.valenta@fpf.slu.cz

[1] Department of Applied Informatics, Faculty of Economics, VSB-Technical University of Ostrava, Sokolská tř. 2416, 702 00 Ostrava, Czech Republic

[2] Faculty of Philosophy and Science in Opava, Institute of Computer Science, Silesian University in Opava, Bezručovo náměstí 1150/13, 746 01 Opava, Czech Republic

on the webpage dedicated to the Membrane computing and robotics (see [10, 11]).

Luo et al. (see [12]) proposed an ant colony P system Π_{ACPS}. Π_{ACPS} provides a basic computational framework, and also can maximally perform the parallelism of ant colony algorithms. Ramachandranpillai and Arock introduced in [13] an optimization method that is based on parallelism to simulate the behavior of foraging ants using spiking neural P (SN P) systems (see [14]). The proposed method is designed by collaborating several SN P systems. In [15, 16], authors use P systems to model biological systems composed of many dynamic components.

In this paper, we follow up our previous research on 2D P colonies (see [17]), and their abilities to simulate optimization algorithms inspired by cooperating animals.

P colony (see [18]) is a very simple membrane system derived from P systems (see [2]). It is composed of a community of agents living in a shared environment represented by a multiset of objects. The agent, a membrane, contains a certain number of objects, from which it can evolve into other objects, or it can swap them for other objects with the environment. During research of these simple membrane systems, a two-dimensional variant was introduced.

The 2D P colony is a theoretical model of the multi-agent system designed for observing the behavior of a community of very simple agents living in a shared two-dimensional environment. Each agent of the 2D P colony is equipped with a set of programs consisting of a small number of simple rules. These programs allow the agent to act and move in the environment.

In our previous research, we equipped the 2D P colony with a blackboard as a communication device, to simulate the gray wolf algorithm (see [19, 20]). The blackboard allows the agents to store and read, hence share data. Another way the agents can share the information is to use the stigmergy, which means to leave some special symbols in the environment.

Although, the 2D P colonies were designed for the purpose of simulating multi-agent systems, some phenomena cannot be simulated in the scope of the original definition. The environment can be understood as a matrix of multisets of objects. The contents of these multisets can be changed only by the activity of the agents. From this point of view, the environment is static, thus it does not have its own developing rules. This characteristic of the 2D P colony limits its ability to simulate even such a simple zoocoenosis, like the ant colony, in a more detailed manner.

Let us focus on how the ants hunt for food. The worker ants leave their nest to search for food. During their search, they leave behind a trail of pheromones. The pheromones are chemical scents used for the purpose of communication. When an ant finds food, it turns around and goes back to the nest and it spreads more pheromone on the trail, reinforcing it. On the way back, the ant does not follow exactly the same path it came on, but it is able to find a direct path to its nest. When other worker ants come across the pheromone trail, they may abandon their own random search for food, and they follow the pheromone trail directly to the food source.

Indeed, some simulations of ant colony using 2D P colony had been already done (see [17]), but there are limits of this model, which do not allow us to simulate even the basic properties of pheromones.

To obtain a better result of the simulation of the ant colony, we need to solve two main issues. First, we need to ensure the ants orientation in the environment. This can be done by introducing a set of special objects, containing the position of each ant within the environment. The other issue concerns the pheromones. As we already mentioned, the pheromone is a chemical scent which naturally vanishes if not reinforced. To simulate this effect, we need to ensure that the environment can change its contents. Hence, we introduce the evolution rules of the environment.

2 The evolving environment in terms of the 2D P colony

In this section, we extend the definition of the 2D P colony. We will add the rules which allow the environment to evolve. We will not introduce the original definition of the 2D P colony, since the only difference is in a definition of an environment. The definition of the environment is only enriched by a set of evolution rules R. The evolving environment also affects the derivation step. In the definition of the 2D P colony, the changes in the environment can be done only by the agents using the communication rules.

Definition 1 A 2D P colony with evolving environment, $2D_{ev}$ P COL is a construct

$$\Pi = \left(V, e, Env, A_1, \ldots, A_d, f\right), d \geq 1,$$

where

- V is the alphabet of the colony. The elements of the alphabet are called objects.
- $e \in V$ is the basic environmental object of the 2D P colony,
- Env is a triplet $(m \times n, w_E, R)$, where
  - $m \times n, m, n \in \mathbb{N}$ is the size of the environment.
  - w_E is the initial contents of the environment, it is a matrix of size $m \times n$ of multisets of objects over $V \setminus \{e\}$.

  - R is a set of evolution rules. Each rule is of the form $S \to T$, where S is a multisets over the objects over $V \setminus \{e\}$, and where T is a multisets over the objects over V. We say, that the multiset S evolves into the multiset T.

- A_i, $1 \leq i \leq d$, is an agent. The number d is called a degree of the colony. Each agent is a construct $A_i = \big(O_i, P_i, [o,p]\big), 0 \leq o \leq m,\ 0 \leq p \leq n$, where

  - O_i is a multiset over V, it determines the initial state (contents) of the agent, $|O_i| = c, c \in \mathbb{N}$. The number c is called a capacity of the colony.
  - $P_i = \big\{p_{i,1}, \ldots, p_{i,l_i}\big\}, l \geq 1, 1 \leq i \leq k$ is a finite set of programs for each agent, where each program contains exactly $h \in \mathbb{N}$ rules, h is called a height. Each rule is in the following form:

    $* \ a \to b, a, b \in V$ is an evolution rule,
    $* \ a \leftrightarrow b, a, b \in V$ is a communication rule,
    $* \ \big[a_{q,r}\big] \to s, a_{q,r} \in V, 0 \leq q, r \leq 2, s \in \{\Leftarrow, \Rightarrow, \Uparrow, \Downarrow\}$ is a motion rule. $\big[a_{q,r}\big]$ is a matrix representing the vicinity of an agent.

  - $[o,p],\ 1 \leq o \leq m,\ 1 \leq p \leq n$, is an initial position of agent A_i in the 2D environment,

- $f \in V$ is the final object of the colony.

A configuration of the $2D_{ev}$ P COL is given by the state of the environment—the matrix of type $m \times n$ of multisets of objects over $V - \{e\}$, the states of all agents—the multisets of objects over V, and the coordinates of the agents. An initial configuration is given by the definition of the $2D_{ev}$ P colony.

A computational step of the $2D_{ev}$ P COL is a transition between two consecutive configurations. The computational step consists of four sub-steps. In the first sub-step, a set of the applicable programs of the agents is determined, according to the current configuration of the colony. In the second sub-step, for each agent, one program from this set is chosen. For the chosen programs, it must hold, that there is no collision between the communication rules of each two different programs. In the third sub-step, chosen programs are executed and the values of the environment are updated.

The fourth sub-step is the evolution of the environment. Let $E_{i,j}, 0 \leq i \leq m, 0 \leq j \leq n$ be a multiset representing the contents of the environment at the coordinates $[i, j]$. Let $A_{i,j}, 0 \leq i \leq m, 0 \leq j \leq n$ be a multiset of all the objects forming right sides of the communication rules of the programs chosen in the second sub-step for all the agents at the position $[i, j]$. Consider multisets $S^1_{i,j}, \ldots S^{o_{i,j}}_{i,j}, o_{i,j} \in \mathbb{N}$ such that $\cup_{k=0}^{o_{i,j}} S^k_{i,j} = E_{i,j} \backslash A_{i,j}$. Then, the evolution of the environment in this particular derivation step is provided by the application of the evolution rules $S^k_{i,j} \to T \in R$.

Generally speaking, the objects of the environment, which were not changed by the actions of the agents, are modified by the evolution rules of the environment. The application of the rules of the agents has higher priority over evolution rules of the environment.

A computation is non-deterministic and maximally parallel. The non-determinism means that there is a uniform distribution on all the applicable rules. Hence, if an agent can apply more rules, or there can be applied more rules of the environment in one particular derivation step, only one rule is non-deterministically chosen, but each and every rule can be applied with the same probability. The computation ends by halting, when there is no agent that has an applicable program.

The result of the computation is the number of copies of the final object placed in the environment at the end of the computation.

3 The simulation of the ant colony

Since we have defined the $2D_{ev}$ P COL, we can describe the idea of simulation of the ant colony, respectively the hunt for food. As we already mentioned, the food hunt can be divided into two phases. During the first phase, an ant randomly searches the environment for a food source. Once the ant finds some food source, it stops searching and heads directly to its nest. The path back to the nest does not have to correspond the one it used to find food. The ant uses the sun and visual information as well to find the shortest way back to the nest. On the way back, it also marks this path with pheromones, so the other ants can follow this path to the food source. The ants reinforce this path by adding more pheromone to it as long as the food source is not empty. Once, all food is moved to the nest, ants do not reinforce the path anymore, the pheromones vanish, and the path is destroyed.

To simulate this behavior, we introduce several sets of rules assigned to each agent. First, let us first define a division of the environment into the quadrants. We do not want to limit a position of the nest in the colony, so we are able, e.g., to put more colonies into one environment, but the ants have to be able to use their compass and find their way back to the nest. We do not want to introduce any additional component guiding the agents to the nest either.

The position of the nest divides the environment into already mentioned quadrants (see Fig. 1). Let the coordinates of the nest are being x, y, then into the quadrant $Q1$ belong all the cells with coordinates i, j, where $i < x, j < y$, similarly, into the quadrant $Q2$ belong all the cells with

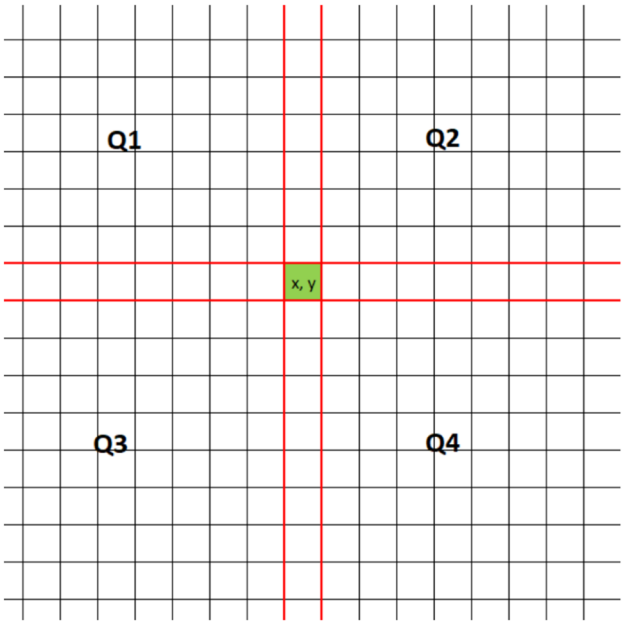

Fig. 1 Division of the environment into quadrants

coordinates i, j, where $i > x, j < y$, into the quadrant $Q3$ belong all the cells with coordinates i, j, where $i < x, j > y$, into the quadrant $Q4$ belong all the cells with coordinates i, j, where $i > x, j > y$. All the cells, where $i = x$, and/or $j = y$ have "special status", while they lay on the axes leading directly to the nest.

Let us present a general definition of our ant 2D P colony:

$$\Pi = \big(V, e, Env, A_1, \ldots, A_d, f\big),$$

where

- $V = \{a_{i,j}, P_0, \ldots P_9, F, E, N, e : 0 \le i \le m, 0 \le j \le n\}$ is the alphabet of the colony. The objects $a_{i,j}$ represent the position of an agent in the environment, the object E represents information, that the agent does not carry food, the object F represents food, and the object N represents the nest.
- Env is a triplet $(m \times n, w_E, R)$, where:
  - the size $m \times n$, and the initial contents w_E of the environment are defined for each particular case.
  - R contains following rules, which represent vanishing of the pheromones:

$$*\{P_i\} \to \{P_{i-1}\}, \text{for } 1 \le i \le 9,$$
$$*\{P_0\} \to \{e\}.$$

- The agent is $A_i = \big(\{a_{x,y}, E, e\}, P_i, [x, y]\big)$, where $[x, y]$ are the coordinates of the nest. We discuss the set of programs P_i in the following part.

3.1 Random motion

The first set of programs defines random motion of the agent in the environment. The object + represents an arbitrary object from the set V, which can occur in the environment, except food F and the pheromones $P_0, \ldots, P_9$. Random motion is based on a non-deterministic choice of one of the following programs.

$$P : \left\langle \begin{bmatrix} + & + & + \\ + & + & + \\ + & + & + \end{bmatrix} \to \Rightarrow ; E \to E; a_{i,j} \to a_{i+1,j} \right\rangle, \tag{1}$$

$$P : \left\langle \begin{bmatrix} + & + & + \\ + & + & + \\ + & + & + \end{bmatrix} \to \Leftarrow ; E \to E; a_{i,j} \to a_{i-1,j} \right\rangle, \tag{2}$$

$$P : \left\langle \begin{bmatrix} + & + & + \\ + & + & + \\ + & + & + \end{bmatrix} \to \Downarrow ; E \to E; a_{i,j} \to a_{i,j+1} \right\rangle, \tag{3}$$

$$P : \left\langle \begin{bmatrix} + & + & + \\ + & + & + \\ + & + & + \end{bmatrix} \to \Uparrow ; E \to E; a_{i,j} \to a_{i,j-1} \right\rangle. \tag{4}$$

3.2 Found the food

The second set of programs defines behavior next to food, and on food. The object $*$ represents an arbitrary object from the set V, which can occur in the environment.

$$P : \left\langle \begin{bmatrix} * & * & (F/*) \\ * & (*/F) & (F/*) \\ * & * & (F/*) \end{bmatrix} \to \Rightarrow ; E \to E; a_{i,j} \to a_{i+1,j} \right\rangle, \tag{5}$$

$$P : \left\langle \begin{bmatrix} (F/*) & * & * \\ (F/*) & (*/F) & * \\ (F/*) & * & * \end{bmatrix} \to \Leftarrow ; E \to E; a_{i,j} \to a_{i-1,j} \right\rangle, \tag{6}$$

$$P : \left\langle \begin{bmatrix} * & * & * \\ * & (*/F) & * \\ (F/*) & (F/*) & (F/*) \end{bmatrix} \rightarrow \Downarrow ; E \rightarrow E; a_{i,j} \rightarrow a_{i,j+1} \right\rangle, \tag{7}$$

$$P : \left\langle \begin{bmatrix} ((F/*)) & (F/*) & (F/*) \\ * & (*/F) & * \\ * & * & * \end{bmatrix} \rightarrow \Uparrow ; E \rightarrow E; a_{i,j} \rightarrow a_{i,j-1} \right\rangle. \tag{8}$$

By the notation $(F/*)$, we understand that at mentioned place is either food or some other symbol, and we consider also that at least one of the triplets of the symbols $(F/*)$ in each rule is F. In contrast, by $(*/F)$ we understand any symbol except F. Once the agent stands on food, it uses a rule:

$$P : \langle a_{i,j} \rightarrow a_{i,j}; E \leftrightarrow F; e \rightarrow e \rangle. \tag{9}$$

3.3 Homecoming

The third set of programs describes how the agent returns to the nest, and how it creates, or reinforces a pheromone path. Here, we must consider in which quadrant the agent is. We define the programs only for the $Q1$. For the other quadrants, the programs are similar. Let us recall that in this case are $i < x$ and $j < y$, hence the agent moves towards the nest by increasing its both coordinates.

$$P : \langle F \rightarrow F; e \rightarrow P_9; a_{i,j} \rightarrow a_{i,j} \rangle, \tag{10}$$

$$P : \left\langle P_9 \leftrightarrow e; \begin{bmatrix} * & * & * \\ * & P & P \\ * & (*/P) & * \end{bmatrix} \rightarrow \Rightarrow / \begin{bmatrix} * & * & * \\ * & P & * \\ * & * & * \end{bmatrix} \rightarrow \Rightarrow ; a_{i,j} \rightarrow a_{i+1,j} \right\rangle, \tag{11}$$

$$P : \left\langle P_9 \leftrightarrow e; \begin{bmatrix} * & * & * \\ * & P & (*/P) \\ * & P & * \end{bmatrix} \rightarrow \Downarrow / \begin{bmatrix} * & * & * \\ * & P & * \\ * & * & * \end{bmatrix} \rightarrow \Downarrow ; a_{i,j} \rightarrow a_{i,j+1} \right\rangle. \tag{12}$$

The second part of the rule after the slash can be applied by the agent only when the first part is not applicable. If the agent reaches one of the axes, hence $i = x$, or $j = y$, then programs are more simple.

For $i = x$,

$$P : \left\langle P_9 \leftrightarrow e; \begin{bmatrix} * & * & * \\ * & P & * \\ * & * & * \end{bmatrix} \rightarrow \Downarrow ; a_{i,j} \rightarrow a_{i,j+1} \right\rangle. \tag{13}$$

For $j = y$,

$$P : \left\langle P_9 \leftrightarrow e; \begin{bmatrix} * & * & * \\ * & P & * \\ * & * & * \end{bmatrix} \rightarrow \Rightarrow ; a_{i,j} \rightarrow a_{i+1,j} \right\rangle. \tag{14}$$

Once the agent reaches the nest, it drops the food and follows the pheromones to food, or, if there are not any pheromones, it searches for another source of food. Finding the nest and dropping the food represent following programs.

$$P : \left\langle P_9 \leftrightarrow e; \begin{bmatrix} * & * & * \\ * & P & * \\ * & N & * \end{bmatrix} \rightarrow \Downarrow ; a_{i,j} \rightarrow a_{i,j+1} \right\rangle, \tag{15}$$

$$P : \left\langle P_9 \leftrightarrow e; \begin{bmatrix} * & * & * \\ * & P & N \\ * & * & * \end{bmatrix} \rightarrow \Rightarrow ; a_{i,j} \rightarrow a_{i+1,j} \right\rangle, \tag{16}$$

$$P : \langle F \rightarrow f; e \rightarrow E; a_{x,y} \rightarrow a_{x,y} \rangle \tag{17}$$

$$P : \langle f \leftrightarrow e; E \rightarrow E; a_{x,y} \rightarrow a_{x,y} \rangle. \tag{18}$$

The agent evolves the F into the f, so the nest could not be considered as a food source, and we can also consider reaching the nest with food as a result of computational step.

3.4 Found the pheromone trail

Fourth set of programs is for the situation, when the agent runs into pheromone trail.

$$P : \left\langle \begin{bmatrix} * & * & (P/*) \\ * & (*/P) & (P/*) \\ * & * & (P/*) \end{bmatrix} \rightarrow \Rightarrow ; E \rightarrow E; a_{i,j} \rightarrow a_{i+1,j} \right\rangle, \tag{19}$$

$$P : \left\langle \begin{bmatrix} (P/*) & * & * \\ (P/*) & (*/P) & * \\ (P/*) & * & * \end{bmatrix} \rightarrow \Leftarrow ; E \rightarrow E; a_{i,j} \rightarrow a_{i-1,j} \right\rangle, \tag{20}$$

$$P : \left\langle \begin{bmatrix} * & * & * \\ * & (*\,/P) & * \\ (P/\,*) & (P/\,*) & (P/\,*) \end{bmatrix} \rightarrow \Downarrow ; E \rightarrow E; a_{i,j} \rightarrow a_{i,j+1} \right\rangle, \tag{21}$$

$$P : \left\langle \begin{bmatrix} (P/\,*) & (P/\,*) & (P/\,*) \\ * & (*\,/P) & * \\ * & * & * \end{bmatrix} \rightarrow \Uparrow ; E \rightarrow E; a_{i,j} \rightarrow a_{i,j-1} \right\rangle. \tag{22}$$

By the notation $(P/\,*)$, we understand that at mentioned place is either one of the pheromone symbols $P_0, \ldots, P_9$ or some other symbol, and we consider also that at least one of the triplets of the symbols $(P/\,*)$ in each rule is the pheromone symbol. In contrast, by $(*\,/P)$ we understand any symbol except P.

3.5 Following the pheromone trail

The fifth set of programs represents following the pheromone trail to food. Again, we present only the programs for the $Q1$.

$$P : \left\langle E \rightarrow E; \begin{bmatrix} * & P & * \\ * & P & * \\ * & * & * \end{bmatrix} \rightarrow \Uparrow ; a_{i,j} \rightarrow a_{i,j-1} \right\rangle, \tag{23}$$

$$P : \left\langle E \rightarrow E; \begin{bmatrix} * & * & * \\ P & P & * \\ * & * & * \end{bmatrix} \rightarrow \Leftarrow ; a_{i,j} \rightarrow a_{i-1,j} \right\rangle. \tag{24}$$

3.6 Final word on motion rules

All mentioned sets of programs form the set of programs P_i of each agent. The behavior of the agent is guided by its contents. Agent is aware of its position in the environment (object $a_{i,j}$, and it is also able to find the shortest way back to the nest. This simulates ant's orientation in the environment using the sun and visual information. The symbol F inside the agent represents the fact, that it carries food. It influences behavior of the agent in the term of its motion in the environment. Once the agent carries food, it follows or creates a pheromone path back to the nest. Otherwise, it searches the environment for food or pheromone path.

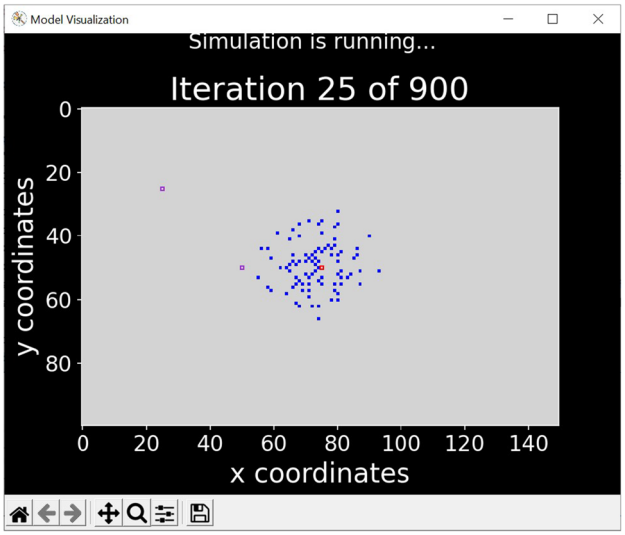

Fig. 2 Graphical interface

4 Computer simulation

For the purpose of testing and evaluating our proposed model, we developed Python application simulating the 2D P colony with evolving environment. The application allows to visualize the computational process of the colony. In the case of this paper, it allows us to verify our theoretical model.

The Fig. 2 shows a screenshot of the application. The nest is represented by the square with red borders, the agents by the squares filled with blue color. Food is represented by the square with purple borders. The color of the border of the pheromone trail varies from black, the strongest trail, through light green to light blue, which represents the weakest pheromone trail. In general, the simulation proceeded as we expected. The ants were exploring the environment, when they found food, they were taking it to the nest building or reinforcing the pheromone trail. The ants which found the trail were able to follow it to the source of food. Thanks to the development rules of the environment, no longer used pheromone trails were gradually disappearing.

Unfortunately, there were also some problems we had to face. The motion of the agents based on the proposed rules is too chaotic, what does not correspond to the real behavior of the ants. Even though the agents were able to reach the food, the path the agents followed corresponded more to the Brownian motion, and the time consumption was enormous. The other undesired, but expected behavior was that the agents were following the vanishing pheromone trail back to the nest. The idea of the solution for the first mentioned problem is very simple. We introduce extended set of motion rules. The agents will not be allowed to change the direction of the motion in every step freely, but with a

certain probability. The solution for the second issue requires changes in the set of rules for joining the pheromone trail. For each quadrant, the rules will be updated, so the agent will not be able to join or follow the trail, if there will not be a connection to the trail leading away from the nest.

4.1 Random move correction

As we mentioned, designed random motion did not fulfill our expectations, hence we decided to modify the rules of the random motion. To ensure, that the motion was not so chaotic, we considered two possibilities of the modification of the motion rules. First attitude was to change the direction with some particular possibility, e.g., with a probability of 60% an ant will continue in recent direction, and 40% it will change it. The other approach was to prevent an ant going in the opposite direction, and again, change the direction with some particular possibility. To introduce mentioned solutions, we need to keep the information of the direction an ant moves, and modify the rules of the random motion. Hence, for each direction we introduce a pair of new symbols, E_R, E_r for right, E_L, E_l for left, E_U, E_u for up, E_D, E_d for down. The E_R, E_L, E_U, E_D instruct the ant which direction to move, and the E_r, E_l, E_u, E_d provides the information, the motion in given direction was done. The motion rules mentioned in 3.1 are replaced by the following rules.

First set of rules is an initial choice of a direction. It is used at the very first step of an agent or when it drops a food in the nest.

$$P : \langle E \to z; a_{x,y} \to a_{x,y}; e \to e \rangle, \text{ where } z \in \{E_R, E_L, E_U, E_D\}. \tag{25}$$

Second set of rules executes the move itself. An agent moves in the desired direction and "remembers" it for next action. Let us recall that the object + represents an arbitrary object from the set V, which can occur in the environment, except food F and the pheromones $P_0, \ldots, P_9$.

$$P : \left\langle \begin{bmatrix} + & + & + \\ + & + & + \\ + & + & + \end{bmatrix} \to \Rightarrow ; E_R \to E_r; a_{i,j} \to a_{i+1,j} \right\rangle, \tag{26}$$

$$P : \left\langle \begin{bmatrix} + & + & + \\ + & + & + \\ + & + & + \end{bmatrix} \to \Leftarrow ; E_L \to E_l; a_{i,j} \to a_{i-1,j} \right\rangle, \tag{27}$$

$$P : \left\langle \begin{bmatrix} + & + & + \\ + & + & + \\ + & + & + \end{bmatrix} \to \Downarrow ; E_D \to E_d; a_{i,j} \to a_{i,j+1} \right\rangle, \tag{28}$$

$$P : \left\langle \begin{bmatrix} + & + & + \\ + & + & + \\ + & + & + \end{bmatrix} \to \Uparrow ; E_U \to E_u; a_{i,j} \to a_{i,j-1} \right\rangle. \tag{29}$$

Final set of rules represents modified behavior, the probability of keeping the direction, changing it, respectively. This set of rules represents mentioned variant, where the agent is forbidden to turn back. Hence, if it was moving right, the only allowed directions are right, for following the direction, up or down. The agent cannot move left.

Heading right:

$$P : \langle E_r \to E_R; a_{x,y} \to a_{x,y}; e \to e \rangle, \tag{30}$$

$$P : \langle E_r \to E_U; a_{x,y} \to a_{x,y}; e \to e \rangle, \tag{31}$$

$$P : \langle E_r \to E_D; a_{x,y} \to a_{x,y}; e \to e \rangle. \tag{32}$$

Heading left:

$$P : \langle E_l \to E_L; a_{x,y} \to a_{x,y}; e \to e \rangle, \tag{33}$$

$$P : \langle E_l \to E_U; a_{x,y} \to a_{x,y}; e \to e \rangle, \tag{34}$$

$$P : \langle E_l \to E_D; a_{x,y} \to a_{x,y}; e \to e \rangle. \tag{35}$$

Heading up:

$$P : \langle E_u \to E_U; a_{x,y} \to a_{x,y}; e \to e \rangle, \tag{36}$$

$$P : \langle E_u \to E_L; a_{x,y} \to a_{x,y}; e \to e \rangle, \tag{37}$$

$$P : \langle E_u \to E_R; a_{x,y} \to a_{x,y}; e \to e \rangle. \tag{38}$$

Heading down:

$$P : \langle E_d \to E_D; a_{x,y} \to a_{x,y}; e \to e \rangle, \tag{39}$$

$$P : \langle E_d \to E_L; a_{x,y} \to a_{x,y}; e \to e \rangle, \tag{40}$$

$$P : \langle E_d \to E_R; a_{x,y} \to a_{x,y}; e \to e \rangle. \tag{41}$$

Let us discuss how the probability of keeping and changing direction is ensured. Let us consider the first set of rules, the situation the agent moved right. The agent contains three symbols, E_r, e, and the information about its position $a_{x,y}$. There does not exist any other rule for such state of the agent, hence mentioned rules can only be applied. Desired probability is achieved by ratio between the numbers of rules 30, 31, and 32. If we consider, that there is no any

priority on the rules, hence there is the same probability of choosing any of those three rules that can be chosen in a particular configuration, then the probability can be set by introducing particular count of each rule. If each mentioned rule is in the set of rules introduced only once, the probability of continuing in the same direction, right, is one third, as well as the probability of turning up or down. Mentioned probability of 60% can be achieved by introducing the rule 30 three times, the rules 31, and 32 once in the set of rules.

The other considered solution that is the variant without forbidden turning back can be implemented in two ways. The first solution is to add one more rule allowing turning back into sets mentioned above, e.g., adding the rule

$$P : \langle E_d \to E_L; a_{x,y} \to a_{x,y}; e \to e \rangle \tag{42}$$

into the set for heading right.

The other option is a bit simpler. First and second sets remain the same, we change only the "keeping and changing direction" parts. First rule of each of this part remains the same, but the second ones evolve the symbols E_r, E_l, E_u, E_d, respectively, into the symbol E, hence the first set, "initial choice of a direction" can be used. Let us introduce only the rules for heading right. The other directions are obvious.

Heading right:

$$P : \langle E_r \to E_R; a_{x,y} \to a_{x,y}; e \to e \rangle, \tag{43}$$

$$P : \langle E_r \to E; a_{x,y} \to a_{x,y}; e \to e \rangle. \tag{44}$$

As well as in the previous variant, the probability is controlled by the ratio between the counts of the rules 43 and 44.

5 On patterns of behavior

The main objective of this paper is to introduce a new model able to simulate common phenomena of a natural environment and to provide a computer simulation of this model. Our intention is not to provide great changes to the 2D P colony concluding in the design of a powerful model capable to easily simulate any behavior or phenomena, but rather keep the model as simple and as close to the original one as possible, and show that even such a simple model is able to provide the results.

After the corrections of some undesirable behavior, we obtained the results we were expecting. $2D_{ev}$ P COL provides an enhanced communication channel tin terms of 2D P colonies. Of course, the agents are able to leave a message on the environment of the 2D P colonies as well. However, it has to be collected by another (or the same) agent, or it would remain in the environment until the end of the computation. The evolving environment, vanishing messages, allows us to control the behavior of the agents in a restricted area and for a limited period of time.

Further extension of the evolution rules is possible as well. Allowing the context evolution rules would enable to spread the pheromones into the near vicinity.

The behavior of an agent is strictly controlled be the contents of the environment in his vicinity. The priority of the action, the pattern of its behavior, is coded in its rules. An agent is allowed to freely explore the environment only if there is no food or pheromone in its range. In the case, that there is food or the pheromone, it has no option, but go in the direction of the pheromone or food. In case of reaching the food, again, it must pick it up and head back to the ants nest.

Introducing other sets of rules and particular pheromones would allow us to define roles of the ants. An ant producing particular pheromone and releasing it into the environment would be able to force the other ants to follow it. An ant warrior can be immune to the pheromones, but it would not be allowed to leave the nest, hence guard the borders of the nest. A forager can be defined as the agent who has no rules for following the pheromone path; hence, it would search for food and set new pheromone paths to the sources of a food. It can ignore of course the sources already found, if they are marked by a certain pheromone.

6 Implementation

In this section, we discuss the implementation and visualization of our model. We focus on the specifics of the implementation using the programming language Python 3.9. One of the benefits of the Python language is the wide range of modules that extend its functionality and facilitate a programmer's work. For graphical visualization of the simulation, we have chosen the following modules:

- Matplotlib ([21])—a comprehensive library for creating static, animated, and interactive visualizations in Python.
- Random ([22])—a pseudo-random number generator for various distributions.

The Matplotlib module provides the basis for our work. It provides a real-time view of the simulation data on the user's screen. Thanks to this module, a user can see the current configuration of the model (positions of the agents, food, and pheromone trails in the environment), analyze the behavior of the theoretical model, and evaluate the conclusions.

The next crucial module is Random, which we use to implement a random movement of agents. The following subsections describe the specifics of the implementation of each object and function.

6.1 Environment

The environment is represented by a matrix of size $m \times n$, where $m, n > 0$ are user-defined variables. Another editable parameter is a position of a nest. The nest represents also the starting positions of all the agents at the point of an initialization.

For each position on the environment can be assigned a multiset of symbols. By default, each position in the environment contains the symbol e, except the positions of the food. However, the environment can contain an arbitrary multiset of symbols upon the alphabet. For example, the symbol p indicates a pheromone trail. Such a symbol can be injected into the environment using a communication rule of an agent.

6.2 Agent

The agent is implemented as a class that is defined by the following attributes:

- x, y—the position of the agent in the environment.
- $obj1$, $obj2$, $obj3$—the internal objects of an agent.

The attributes x and y define the position of an agent in the environment, so their values must be within a range corresponding to the size of the environment. When an agent is initialized (each of them), the position of the nest is set as its default x and y values. Attributes $obj1$, $obj2$ and $obj3$ can have any value defined in the model alphabet. The initial value of the object is given by the definition of the model. The internal objects of an agent can be changed by evolution and communication rules.

Agents are generated when the program is launched and placed right in the environment. The number of agents is user-defined and the default setting is 100. The rules are the same for every agent and are defined for different quadrants of the environment.

6.3 Running the simulation

In the initialization phase, the environment is loaded and the required number of agents is created and placed in the environment. The initial position of all agents is the same as the nest. Then, food positions are identified (or randomly generated) and labeled with the f symbol in the environment. As this is an iterative algorithm and the number of iterations defines the termination criterion, it is necessary to set this parameter first. The default number of iterations is 500. Each agent is allowed to apply only one program in each iteration of the algorithm. The agent that has already applied the program was set the "done" state and waited for the next iteration of the algorithm, hence it was not allowed to apply other programs. In each iteration of the algorithm, the programs are applied by each agent as follows:

1. For a given agent configuration, find a list of all applicable programs (rules) for the quadrant in which the agent is currently located,
2. If there exists an applicable program, randomly select one and apply it,

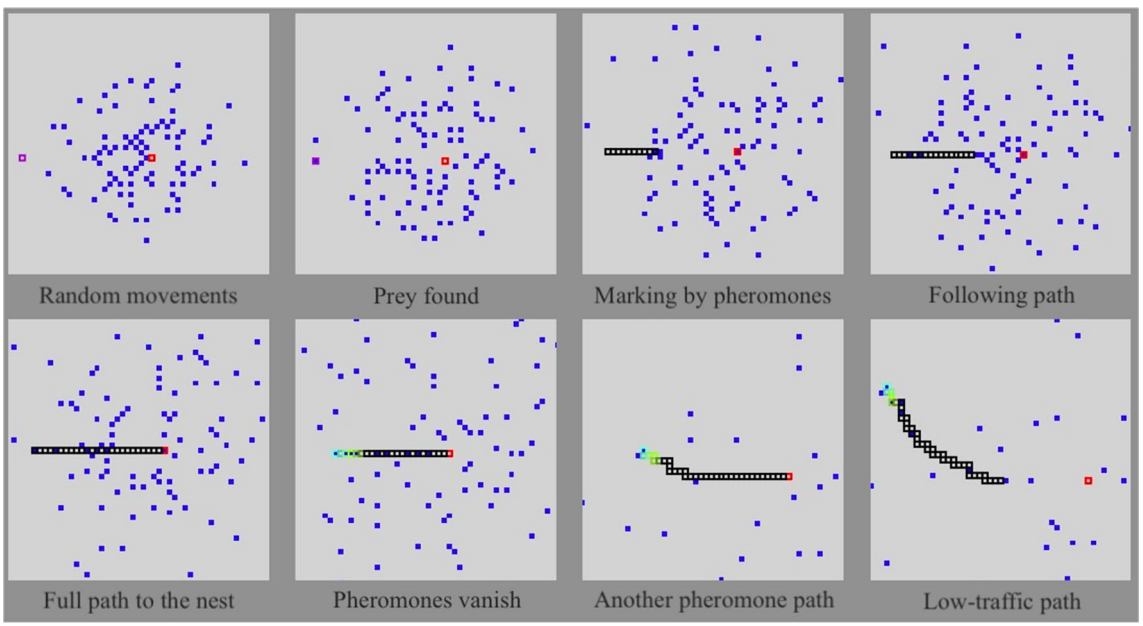

Fig. 3 Significant configurations captured during the simulation

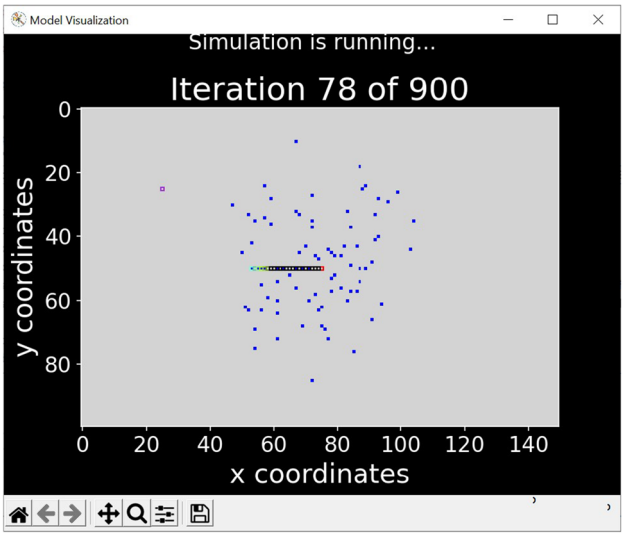

Fig. 4 Gradually disappearing (vanishing) pheromone trail

3. Switch the agent to the "done" state, in which it waits for the next iteration of the algorithm.

When all the agents are in the state "done", update the environment according the evolution rules.

6.4 Graphical interface

The graphical interface can be seen in the Fig. 2, and a graphical visualization of some of the significant situations captured during the simulation can be seen in Fig. 3.

It contains several elements. In the top middle, the user is informed about the current status and the iteration of the algorithm. The largest element located in the middle of the window is a canvas for rendering the simulation using the Matplotlib module. At the bottom, the Matplotlib toolbar is accessible for scaling, zooming in or out, or saving a screenshot. The canvas is redrawn at the end of each iteration of the algorithm. At this time, all the changes of the environment are updated. The user can monitor any changes in the environment in real time. In other words, they have an overview of the current configuration of the model. The rendering speed can be set by editing the variable "animation_delay". By default, it is rendered every 0.015 s.

The first rendering is right after the Initialization. First, the environment and the nest area are rendered, then the positions of the agents and the food. As the algorithm iterates, agents may find food and also pheromones may appear in the environment. Each object in the environment is marked with a specific color:

- the nest is red,
- the Agents are blue,
- food is in shades of purple, the more food occurs at the position, the darker the color is.
- The pheromones are in black if their intensity is greater than 5, if it is less, it gradually changes into shades of green, yellow and then light blue.

Let us note that both the pheromones and food are marked with different shades, which allows us to monitor their gradual vanishing from the environment (see Fig. 4).

The simulation can be terminated at any time by pressing the "c" button. Otherwise, the simulation runs until the termination criterion is met, which in our case defines the maximum number of iterations of the algorithm. When the last iteration of the algorithm is finished, the "End of simulation" information is displayed at the top of the window and the last visited agent positions remain on the canvas until the program is ended by pressing the mouse in the canvas area.

7 Conclusion

P systems and their variants are used for solving optimization problems. In general, there are two basic approaches. The first one is the use of a P system or its variant to solve an optimization problem directly. The other one is to use a P system to simulate some of the optimization algorithm, the approach we followed in this paper.

The P colonies, as well as their 2D variant are already well-established computational models. Especially, the 2D P colonies were used for modeling various multi-agent systems or natural phenomena. Yet, some systems could not be simulated properly, while some properties of those systems cannot be simulated in the terms of recent definition. One of such systems is also an ant colony. The vanishing of the pheromone trail could not be satisfiable simulated using current model. The newly introduced version $2D_{ev}$ P COL allows us to simulate these kinds of phenomena. In this paper, we designed a $2D_{ev}$ P COL successfully simulating the behavior of the ant colony. We identified some problems, which occurred during a computer simulation, and we proposed the solutions.

One of the approaches to identify the solution given by ACO is the strength of a pheromone trail. The stronger the trail, the better the solution. Since the primary goal of recent research was to simulate an ant colony, we did not implement any tool for recording or storing found value of the source of food, or the strength of the pheromone trails. But, thanks to the color shades of the pheromone trail and the food, we are able to follow and evaluate the behavior of the colony.

As we stated, the result of the ACO can be defined by the strongest pheromone trail. We plan to focus on this

aspect and guide our further research this way. Inspired by the Gray wolf optimization algorithm, where the wolves are able to search the vicinity of considered solution, we plan to modify the behavior of the agents, and enable them to search the surroundings to find a source of food. The source of a food may represent searched extremes of a function, and in a near vicinity other extreme, hence better solution may occur.

Other direction of further research can be focused on simulation itself. We plan to insert the obstacles into the environment, and push the simulation further. Monitor the behavior of agents in the environment, where direct path to the nest does not have to be available.

Acknowledgements Research was supported by the SGS/8/2022 Project of the Silesian University in Opava. Research was also supported by the Project of VSB - Technical University in Ostrava, SP2022/74, Computational Intelligence in the Prediction of Economic Quantities, Data Mining and Economic Process Modeling.

Author contributions ML wrote the main manuscript text and DV prepared all the figures and he is also author of the software. Both authors reviewed the manuscript.

Funding Open access publishing supported by the National Technical Library in Prague.

Data availability Not applicable.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Colorni, A., Dorigo, M., & Maniezzo, V. (1991). Distributed optimization by ant colonies. In: European Conference on Artificial Life; str 134–142.
2. Păun, G. (2000). Computing with membranes. *Journal of Computer and System Sciences, 61*, 108–143.
3. Deng, X., Dong, J., Wang, S., Luo, B., Feng, H., & Zhang, G. (2022). Reducer lubrication optimization with an optimization spiking neural P system. *Information Sciences, 604*, 28–44. https://doi.org/10.1016/j.ins.2022.05.016. ISSN 0020-0255.
4. Huang, L., Sun, L., Wang, N., & Jin, X. (2007). Multiobjective optimization of simulated moving bed by tissue P system. *Chinese Journal of Chemical Engineering, 15*(5), 683–690. https://doi.org/10.1016/S1004-9541(07)60146-3. ISSN 1004-9541.
5. Bo, W., Fang, Z. B., Wei, L. I., Cheng, Z. F., & Hua, Z. X. (2021). Malicious URLs detection based on a novel optimization algorithm. *IEICE Transactions on Information and Systems, 104*(4), 513–516. https://doi.org/10.1587/transinf.2020EDL8147. Released on J-STAGE April 01,. (2021). Online ISSN 1745–1361. Print ISSN, 0916–8532.
6. Dong, J., Zhang, G., Luo, B., Yang, Q., Guo, D., Rong, H., Zhu, M., & Zhou, K. (2022). A distributed adaptive optimization spiking neural P system for approximately solving combinatorial optimization problems. *Information Sciences, 596*, 1–14. https://doi.org/10.1016/j.ins.2022.03.007. ISSN 0020-0255.
7. Dong, J., Zhang, G., Luo, B., et al. (2022). Multi-learning rate optimization spiking neural P systems for solving the discrete optimization problems. *Journal of Membrane Computing, 4*, 209–221. https://doi.org/10.1007/s41965-022-00105-6
8. Zhang, G., Rong, H., Neri, F., & Pérez-jiménez, M. J. (2014). An optimization spiking neural P system for approximately solving combinatorial optimization problems. *International Journal of Neural Systems*. https://doi.org/10.1142/S0129065714400061
9. Zhu, M., Yang, Q., Dong, J., Zhang, G., Gou, X., Rong, H., Paul, P., & Neri, F. (2021). An adaptive optimization spiking neural P system for binary problems. *International Journal of Neural Systems*. https://doi.org/10.1142/S0129065720500549
10. Buiu, C., et al. http://membranecomputing.net/. Accessed 1 June 2023.
11. Florea, A. G., & Buiu, C. (2016). Development of a software simulator for P colonies. Applications in robotics. *International Journal of Unconventional Computing, 12*(2–3), 189–205.
12. Luo, Y., Guo, P., & Zhang, M. (2019). A framework of ant colony P system. *IEEE Access, 7*, 157655–157666. https://doi.org/10.1109/ACCESS.2019.2949952
13. Ramachandranpillai, R., & Arock, M. (2020). Spiking neural P ant optimisation: A novel approach for ant colony optimisation. *Electronics Letters, 56*, 1320–1322. https://doi.org/10.1049/el.2020.2144
14. Ionescu, M., Păun, G., & Yokomori, T. (2006). Spiking neural P systems. *Fundamenta Informaticae, 71*(2–3), 279–308.
15. Gheorghe, M., Stamatopoulou, I., Holcombe, M., & Kefalas, P. (2004). Modelling dynamically organised colonies of bio-entities. In J. P. Banâtre, P. Fradet, J. L. Giavitto, & O. Michel (Eds.), *Unconventional programming paradigms. UPP. Lecture Notes in Computer Science.* (Vol. 3566). Springer. https://doi.org/10.1007/11527800_17
16. Kefalas, P., Stamatopoulou, I., Eleftherakis, G., & Gheorghe, M. (2008). Transforming state-based models to P systems models in practice. In D. W. Corne, P. Frisco, G. Păun, G. Rozenberg, & A. Salomaa (Eds.), *Membrane computing. WMC. Lecture Notes in Computer Science.* (Vol. 5391). Springer. https://doi.org/10.1007/978-3-540-95885-7_19
17. Cienciala, L., Ciencialová, L., & Perdek, M. (2012). 2D P colonies. In E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, & Gy. Vaszil (Eds.), *Membrane computing. CMC 2012. Lecture Notes in Computer Science* (Vol. 7762, pp. 161–172). Berlin: Springer. https://doi.org/10.1007/978-3-642-36751-9_12
18. Kelemen, J., Kelemenová, A., & Păun, G. (2004). Preview of P colonies: A biochemically inspired computing model. In: Workshop and Tutorial Proceedings. In: Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX). pp. 82–86. Boston, Massachusetts, USA (September 12–15 2004)
19. Valenta, D., Langer, M., Ciencialová, L., & Cienciala, L. (2021). On Numerical 2D P colonies with the blackboard and the gray wolf algorithm. In R. Freund, T. O. Ishdorj, G. Rozenberg, A. Salomaa, & C. Zandron (Eds.), *Membrane computing. CMC*

2020. Lecture Notes in Computer Science. (Vol. 12687). Springer. https://doi.org/10.1007/978-3-030-77102-7_10
20. Valenta, D., & Langer, M. (2021). On Numerical 2D P colonies modelling the grey wolf optimization algorithm. *Processes., 9*(2), 330. https://doi.org/10.3390/pr9020330
21. Hunter, J. D. (2007). Matplotlib: "A 2D Graphics Environment". *Computing in Science and Engineering, 9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55
22. Rossum, G., & Drake, F. L. Python 3 Reference Manual. Scotts Valley, CA: CreateSpace. https://docs.python.org/3/library/random.html. Accessed 1 June 2023.

Miroslav Langer is Assistant Professor at the Department of Applied Informatics, Faculty of Economics, VSB – Technical University in Ostrava. He teaches subjects related to programming, design and development of information systems, and artificial intelligence. His scientific interests include, but are not limited to formal grammars and languages, their application, biologically inspired computation, artificial intelligence and natural language processing.

Daniel Valenta is Assistant Professor at the Institute of computer science at Silesian University specializing in membrane systems and optimization problem-solving systems. With a keen interest in programming, he also teaches courses on databases and information systems.