# 2D P Colony for Vicinity Search Optimisation

Miroslav Langer[1][0000−0001−5990−7780], Daniel Valenta[2][0009−0005−0781−7755], and Petr Sosík[2][0000−0001−7624−3816]

[1] VSB-Technical University of Ostrava, Sokolská tř. 2416, Ostrava, Czech Republic
miroslav.langer@vsb.cz
[2] Silesian University in Opava, Bezručovo náměstí 1150/13, Opava, Czech Republic
{daniel.valenta,petr.sosik}@fpf.slu.cz

**Abstract.** P colony is a formal computational model suitable for modelling behaviour of simple agents acting in a shared environment. We build on the original concept of formal colonies where both the environment and agents were implemented by tools of formal grammars. P colonies transformed this concept into the framework of membrane systems, i.e., the environment and agents contain abstract discrete objects and formal rules acting upon them. Adding a 2D geometrical structure and evolution of the environment resulted in the model of 2D evolving P colonies. The model is suitable for simulation of phenomena like stigmergy, hence also for implementation of multi-agent optimisation strategies. The motivation for such an implementation lies in a possible future highly parallel and efficient bio-hardware implementation of P systems. In this paper we use a 2D P colony to implement an ant colony-inspired optimisation algorithm. The agents – ants – search the environment for food representing extrema of the objective function. The search is oriented with the help of pheromone trails left by previous agents. The trails are subject to a decay and they can eventually vanish. The original formulation of ant algorithms counts on ants immediately collecting found food in the nest. Here we allow the ants to decide randomly whether to collect the food or to continue the search for another food in the vicinity of an already found food source. We demonstrate experimentally that this behaviour improves the search results.

**Keywords:** 2D P colony · ant colony simulation · P system · evolving environment · optimisation · bio-inspired computation

## 1 Introduction

Nature has been a rich source of inspiration for solving complex problems. Researchers and practitioners in the field of optimisation have turned to natural phenomena and behaviours to develop powerful algorithms. In this article we focus on one of the prominent approaches – the swarm intelligence. The most commonly used methods in this area include:

(i) Ant Colony Optimisation (ACO) [4], inspired by the foraging behaviour of ants depositing pheromones to communicate and find optimal paths. ACO

algorithms construct solutions iteratively based on pheromone trails. They are effective for combinatorial optimisation problems.

(ii) Bee Colony Optimisation (BCO), modelled after the behaviour of honeybees exploring food sources (potential solutions) and communicating through dances. BCO balances exploration and exploitation and are well-suited for dynamic optimisation problems.

(iii) Firefly Algorithm (FA), inspired by the flashing patterns of fireflies. Fireflies attract each other based on brightness (fitness). FA works well for continuous optimisation tasks.

As these methods distribute the search across multiple agents, enabling parallel exploration, they are particularly suitable for implementation using distributed systems such as membrane systems. Membrane (P) systems [17] are bio-inspired computational models based on the structure of biological cells, using discrete objects abstracting from the way the chemicals interact and pass through cell membranes. One of the foundation ideas of membrane computing was its possible future implementation on a bio-hardware which would be massively parallel and energy efficient. Although these goals proved too demanding for current biotechnology, there is slow but visible progress, see, e.g., [1]. One can believe that with new technologies such as Bio-bricks [19] these possibilities could become real.

Membrane systems are frequently applied in the field of optimisation algorithms. To name a few examples, Deng et al. [5] used P systems to improve the efficiency and accuracy of reducer lubrication. Huang et al. [11] proposed a new variant of tissue P system for optimisation of processes with multiple productive objectives. The detection of malicious URLs was solved with the help of P systems in Bo et al. [2]. Another promising model used for solving combinatorial optimisation problems is the Optimisation Spiking Neural P system [6, 7, 23, 24].

Focusing on membrane implementations of swarm optimisation algorithms, particle swarm optimisation using monodirectional tissue-like P systems was presented in Wang et al. [22]. Luo et al (see [15]) proposed an ant colony P system $\Pi_{ACPS}$. $\Pi_{ACPS}$ providing a computational framework allowing for maximally parallel implementation of ACO algorithms. Ramachandranpillai and Arock used spiking neural P systems[18] to implement parallel optimisation technique based on foraging behaviour of ants. From a more general perspective, Gheorghe et al. [10] used P systems to model biological systems composed of many dynamic components.

To resume, this work is motivated by the search of efficient implementation of computation-intensive optimization algorithms by simple cell-like P colonies, considering their possible future implementations using highly parallel and efficient bio-hardware. We present a novel implementation of the ACO algorithm which we have improved with the principle of vicinity search, using 2D P colonies with evolving environment. P colony [13] is an abstract discrete computing model based on principles of membrane computing [17] and multi-agent grammar systems called colonies [12]. It defines a community of agents acting in a shared environment represented by a multiset of abstract discrete objects. Each agent

contains its own multiset of objects, and is equipped with a set of programs containing a small number of simple rules acting upon these objects. These programs enable the agent to perform actions and to communicate via the environment (the blackboard architecture).

The rest of the paper is organised as follows. Section 2 recalls the definition and principles of 2D P colony with evolving environment. Section 3 shows how it can implement the Ant Colony Optimisation algorithm with vicinity search. Section 4 describes the simulation software used for experiments. Simulation results are presented in Sec. 5. The final section concludes the paper and discusses future research avenues.

## 2 2D P Colony with Evolving Environment

In the context of researching simple membrane systems, a 2-dimensional (2D) variant of P colony was introduced in [3]. It served as a theoretical model for observing the behaviour of a community of very simple agents residing in a shared 2-dimensional environment. In [14] we equipped the 2D P colony with a set of rules allowing the environment to evolve. This extension of the 2D P colony was designed to simulate a simple zoocoenosis like an ant colony, in a more detailed manner. The resulting 2D P colony with evolving environment was then used to simulate the behaviour of an ant colony with vanishing pheromone trail. We recall the definition here.

**Definition 1.** *A 2D P colony with evolving environment (2D$_{ev}$ P COL) is the construct*

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

*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 evolutionary rules. Each rule is of the form $S \to T$, where $S$ is a multiset of objects over $V \setminus \{e\}$, and where $T$ is a multiset of objects over $V$. We say that the multiset $S$ evolves to 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 $A_i = (O_i, P_i, [o, p])$, $0 \leq o \leq m$, $0 \leq p \leq n$, 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.*
  - *$P_i = \{p_{i,1}, \ldots, p_{i,l_i}\}, l \geq 1, 1 \leq i \leq k$ 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 evolutionary rule,*
  - $*$ $a \leftrightarrow b,\ a, b \in V$ *is a communication rule,*
  - $*$ $[a_{q,r}] \to s,\ a_{q,r} \in V,\ -1 \leq q, r \leq 1,\ s \in \{\texttt{Left}, \texttt{Right}, \texttt{Up}, \texttt{Down}\}$ *is a motion rule.* $[a_{q,r}]$ *is a* $3 \times 3$ *matrix representing the neighbourhood 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.*

The environment is a 2D $m \times n$ grid, where each cell (place) can hold any number of agents as well as a multiset of objects from $V$.

Let $O$ be a state (contents) of an agent at some moment. An evolutionary rule $a \to b$ of the agent is *applicable* if $a \in O$. When the rule is applied, $a$ is consumed and replaced with $b$. A communication rule $a \leftrightarrow b$ is *applicable* if $a \in O$ and $b \in E_{i,j}$, where $E_{i,j}$ is the contents of the environmental cell at coordinates $[i, j]$ where the agent is positioned. When the rule is applied, $a$ and $b$ are mutually exchanged. Finally, a motion rule $[a_{q,r}] \to s$ is *applicable* when each $a_{q,r}$, $-1 \leq q, r \leq 1$, either is or is not contained in $E_{i+q,j+r}$ – we use a specific notation described in Section 3 to distinguish between these cases. When the rule is applied, the agent moves one step in the direction $s$. A program is applicable if all of its rules are applicable.

A configuration of the $2D_{ev}$ P COL is given by the state of the environment – an $m \times n$ matrix 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 consisting of four sub-steps. In the first sub-step, a set of applicable programs of the agents is determined, according to the current configuration of the colony. In the second sub-step, for each agent, one applicable program is chosen, subject to the following restriction: each object in the environment or in an agent can be used in at most one evolutionary or communication rule in one computational step. In the third sub-step, chosen programs are executed, i.e., all their rules are applied in parallel.

The fourth sub-step is the evolution of the environment. Let $A_{i,j}$, $0 \leq i \leq m$, $0 \leq j \leq n$ be the multiset of all the objects forming right sides of communication rules of the programs chosen in the second sub-step for all the agents at position $[i, j]$. Consider multisets $S^1_{i,j}, \ldots S^{o_{i,j}}_{i,j}$, $o_{i,j} \in \mathbb{N}$ such that $\bigcup_{k=0}^{o_{i,j}} S^k_{i,j} = E_{i,j} \setminus A_{i,j}$. Then all possible evolutionary rules $S^k_{i,j} \to T \in R$ are applied in parallel, i.e., only the objects of the environment not changed by actions of the agents in this step are modified by evolutionary rules of the environment.

The computation is non-deterministic and maximally parallel. The non-determinism means that if an agent can choose more programs in the second substep, or more rules of the environment can be applied in one particular 2D position in the fourth substep, only one program/rule is non-deterministically chosen. The maximal parallelism means that if an agent has a program that can be applied due to the principle described above, then the agent must not remain

inactive. 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.

**Example:** consider an agent with programs (21) and (22) described bellow. Let the agent contain objects $E, e$ and let the place where the agent is located contain objects $F$ and $V$. (a) Let this place contain no other agent. Then both programs (21) and (22) are applicable, one of them is non-deterministically chosen and executed. (b) If this place contains two agents of the same type, then one of them executes (21) and the other executes (22). (c) If there is a third agent of the same type, it remains inactive as the place contains only two symbols the agents can act upon.

The model of evolving 2D P colony might resemble cellular automata (CA) which are often coupled with (ant colony) optimization algorithm; let us cite [20] as one example of many, where the CA-based ACO algorithm is used to model DDoS attacks in ad hoc networks. The main difference between CA and evolving 2D P colonies is that the P colony contains agents which can move, change their states, carry information and communicate (via blackboard), therefore the ants can be directly implemented by these agents.

## 3 Vicinity Search with Evolving 2D P Colony

In the original formulation of the ACO, the ants were programmed to randomly search the environment for food. When an ant found the food, it returned back to the nest leaving a pheromone trail in the environment. When another ant found this trail, it could follow it to the location of the food. When heading back to the nest, it reinforced the existing trail. When the source of the food was exhausted, the trail subsequently evaporated.

However, it has been previously shown [8, 16] that a combination of ant optimisation with local search improves the performance of the ACO algorithm. Inspired by these findings, we designed a novel implementation of local vicinity search within the framework of 2D P colony with evolving environment. For the purpose of the vicinity search, we allowed the ant to opt between returning to the nest with a piece of the found food or searching the environment for another source of food. Recall that the food represents extremes of a function, so that the presence of an extreme suggests that another (and deeper) extreme might lay nearby.

The term "vicinity" has no precise geometrical meaning such as a search diameter or so. It just indicates that the ant starts a new search at the position of just found food, and since it moves randomly, the vicinity of this position is searched with a higher probability than more remote places.

### 3.1 $2D_{ev}$ P COL for Vicinity Search

We provide a formal definition of the $2D_{ev}$ P COL followed by a detailed description of groups of programs controlling different phases of the agents' behaviour, so that the reader could check the logic of the implemented ACO algorithm.

**Definition 2.** *$2D_{ev}$ P COL for Vicinity Search*

$$\Pi_{vic} = (V, e, Env, A, f)$$

*where:*

- $V = \{e, f, F_1, \ldots, F_5, P_0, \ldots, P_9,$
  $N, E, F, E_U, E_u, E_D, E_d, E_L, E_l, E_R, E_r, V, V_r, V_l, V_u, V_d\}$,
- *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,*
  - $R = \{P_i \to P_{i-1}$*, for* $1 \leq i \leq 9\} \cup \{P_0 \to e\}$.
- $A = (\{E, e\}, P, [m, n])$*, where* $P$ *contains programs described in following sections.*

**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 $F_1, \ldots, F_5$ and the pheromones $P_0, \ldots, P_9$. Random motion is based on a non-deterministic choice of one of the following programs.

$$P : \langle e \to e \ ; \ E \to E_R \rangle \tag{1}$$

$$P : \langle e \to e \ ; \ E \to E_L \rangle \tag{2}$$

$$P : \langle e \to e \ ; \ E \to E_U \rangle \tag{3}$$

$$P : \langle e \to e \ ; \ E \to E_D \rangle \tag{4}$$

$$P : \left\langle \begin{bmatrix} + & + & + \\ + & + & + \\ + & + & + \end{bmatrix} \to \texttt{Right} \ ; \ E_R \to E_r \right\rangle \tag{5}$$

$$P : \left\langle \begin{bmatrix} + & + & + \\ + & + & + \\ + & + & + \end{bmatrix} \to \texttt{Left} \ ; \ E_L \to E_l \right\rangle \tag{6}$$

$$P : \left\langle \begin{bmatrix} + & + & + \\ + & + & + \\ + & + & + \end{bmatrix} \to \texttt{Down} \ ; \ E_D \to E_d \right\rangle \tag{7}$$

$$P : \left\langle \begin{bmatrix} + & + & + \\ + & + & + \\ + & + & + \end{bmatrix} \rightarrow \texttt{Up} \; ; \; E_U \rightarrow E_u \right\rangle \tag{8}$$

$$P : \langle e \rightarrow e \; ; \; E_r \rightarrow E_R \rangle \tag{9}$$

$$P : \langle e \rightarrow e \; ; \; E_r \rightarrow E \rangle \tag{10}$$

$$P : \langle e \rightarrow e \; ; \; E_l \rightarrow E_L \rangle \tag{11}$$

$$P : \langle e \rightarrow e \; ; \; E_l \rightarrow E \rangle \tag{12}$$

$$P : \langle e \rightarrow e \; ; \; E_d \rightarrow E_D \rangle \tag{13}$$

$$P : \langle e \rightarrow e \; ; \; E_d \rightarrow E \rangle \tag{14}$$

$$P : \langle e \rightarrow e \; ; \; E_u \rightarrow E_U \rangle \tag{15}$$

$$P : \langle e \rightarrow e \; ; \; E_u \rightarrow E \rangle \tag{16}$$

The first group of programs 1, 2, 3, 4 serves as an initial choice of direction in which an ant is going to move. These rules can be applied only if the agent contains symbols $E$ and $e$.

According to the choice of the direction, one of the motion programs 5, 6, 7, 8 is applied. Note that after an application of the program, the capital subscript is made lowercase. This allows to control the probability of change of the direction in which the ant moves.

Programs 9, 11, 13 and 15 ensure that an ant will continue in chosen direction. The programs 10, 12, 14, 16 serve as a return to the initial state in which an ant chooses the direction of its further move. The probability of the change of the direction is given by the ratio of the number of programs 9 and 10, (11 and 12, 15 and 16, or 13 and 14 respectively). As a program of the agent is chosen non-deterministically from all applicable programs, each of the applicable programs has the same probability to be chosen. Hence, to avoid very chaotic motion, we need to control the probability of the direction change. For instance, if there is one program 9 to continue to the right, and one program 10 to change the direction, then the probability of change is 0.5. If the ant contains three copies of the program 9 and one copy of 10, the probability of change is only 0.25. The topic of random motion was discussed in more details in [14]. Let us recall, that if we introduce a set of rules without the probability of following the same direction, the motion of an agent is chaotic and reminds the Brownian motion. For this reason, we introduce this mechanism where an ant can change its direction with given probability.

**Found the Food** The second set of programs defines behaviour next to food, and when food is found. The object $*$ represents an arbitrary object from the set $V$, which can occur in the environment. The object $F$ represents the occurrence of food.

$$P : \left\langle \begin{bmatrix} * & * & (F/*) \\ * & (*/F) & (F/*) \\ * & * & (F/*) \end{bmatrix} \rightarrow \texttt{Right} \ ; \ E_A \rightarrow E \right\rangle \tag{17}$$

$$P : \left\langle \begin{bmatrix} (F/*) & * & * \\ (F/*) & (*/F) & * \\ (F/*) & * & * \end{bmatrix} \rightarrow \texttt{Left} \ ; \ E_A \rightarrow E \right\rangle \tag{18}$$

$$P : \left\langle \begin{bmatrix} * & * & * \\ * & (*/F) & * \\ (F/*) & (F/*) & (F/*) \end{bmatrix} \rightarrow \texttt{Down} \ ; \ E_A \rightarrow E \right\rangle \tag{19}$$

$$P : \left\langle \begin{bmatrix} ((F/*)) & (F/*) & (F/*) \\ * & (*/F) & * \\ * & * & * \end{bmatrix} \rightarrow \texttt{Up} \ ; \ E_A \rightarrow E \right\rangle \tag{20}$$

where $A = \{R, L, U, D, r, l, u, d, \varepsilon\}$, hence arbitrary subscript, including no subscript. By the symbol $F$ we understand an arbitrary symbol for the food $F_1, \ldots, F_5$.

By $(F/*)$ we denote that the mentioned place contains either food or some other symbol, and we assume 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 square with food, it can opt between taking the food back to the nest, or continue in searching for another source. This behaviour is implemented by the programs:

$$P : \langle E \leftrightarrow F \ ; \ e \rightarrow e \rangle \tag{21}$$

$$P : \langle E \leftrightarrow V \ ; \ e \rightarrow e \rangle \tag{22}$$

The probability of choosing the behaviour is controlled in the same way as in the case of the motion change.

**Vicinity Search** If the ant chose to search the vicinity, it must abandon the found food. To loose track of the found food, it must move at least two steps away, otherwise its rules would immediately attract it back to the recently found food. First, the ant chooses in which direction it moves and it makes the first step:

$$P : \left\langle \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \rightarrow \texttt{Right} \ ; \ V \rightarrow V_r \right\rangle \tag{23}$$

$$P : \left\langle \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \rightarrow \texttt{Left} \; ; \; V \rightarrow V_l \right\rangle \tag{24}$$

$$P : \left\langle \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \rightarrow \texttt{Down} \; ; \; V \rightarrow V_d \right\rangle \tag{25}$$

$$P : \left\langle \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \rightarrow \texttt{Up} \; ; \; V \rightarrow V_u \right\rangle \tag{26}$$

Then the ant makes another step in the chosen direction and decides whether it continues the same direction or changes it:

$$P : \left\langle \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \rightarrow \texttt{Right} \; ; \; V_r \rightarrow E_r \right\rangle \tag{27}$$

$$P : \left\langle \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \rightarrow \texttt{Left} \; ; \; V_l \rightarrow E_l \right\rangle \tag{28}$$

$$P : \left\langle \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \rightarrow \texttt{Down} \; ; \; V_d \rightarrow E_d \right\rangle \tag{29}$$

$$P : \left\langle \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \rightarrow \texttt{Up} \; ; \; V_u \rightarrow E_u \right\rangle \tag{30}$$

**Homecoming** If the ant chooses to pick the food and bring it to the nest, while creating or reinforcing a pheromone trail, the following set of programs is activated:

$$P : \langle F \rightarrow F \; ; \; e \rightarrow P_9 \rangle \tag{31}$$

$$P : \left\langle P_9 \leftrightarrow e \; ; \; \begin{bmatrix} * & * & * \\ * & P & P \\ * & (*/P) & * \end{bmatrix} \rightarrow \texttt{Right}/ \; \begin{bmatrix} * & * & * \\ * & P & * \\ * & * & * \end{bmatrix} \rightarrow \texttt{Right} \right\rangle \tag{32}$$

$$P : \left\langle P_9 \leftrightarrow e \; ; \; \begin{bmatrix} * & * & * \\ * & P & (*/P) \\ * & P & * \end{bmatrix} \rightarrow \texttt{Down}/ \; \begin{bmatrix} * & * & * \\ * & P & * \\ * & * & * \end{bmatrix} \rightarrow \texttt{Down} \right\rangle \tag{33}$$

The object $P$ represents the pheromone trail. 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 a border of the environment, then the following programs are applied:

$$P : \left\langle P_9 \leftrightarrow e \; ; \; \begin{bmatrix} * & * & * \\ * & P & * \\ * & * & * \end{bmatrix} \rightarrow \texttt{Down} \right\rangle \tag{34}$$

$$P : \left\langle P_9 \leftrightarrow e \; ; \; \begin{bmatrix} * & * & * \\ * & P & * \\ * & * & * \end{bmatrix} \rightarrow \texttt{Right} \right\rangle \tag{35}$$

The border of the environment can be lined using a special symbol that is not used in any rule, therefore an ant cannot interact with it.

Once the ant reaches the nest, it drops the food and follows a pheromone trail to food, or, if there are no pheromones, it randomly searches for another source of food. This is controlled by the following programs:

$$P : \left\langle P_9 \leftrightarrow e \; ; \; \begin{bmatrix} * & * & * \\ * & P & * \\ * & N & * \end{bmatrix} \rightarrow \texttt{Down} \right\rangle \tag{36}$$

$$P : \left\langle P_9 \leftrightarrow e \; ; \; \begin{bmatrix} * & * & * \\ * & P & N \\ * & * & * \end{bmatrix} \rightarrow \texttt{Right} \right\rangle \tag{37}$$

$$P : \langle F \rightarrow f \; ; \; e \rightarrow E \rangle \tag{38}$$

$$P : \langle f \leftrightarrow e \; ; \; E \rightarrow E \rangle \tag{39}$$

The agent changes the symbol $F$ to $f$, so as the nest could not be considered as a food source.

**Found the Pheromone Trail** The next set of programs controls the situation when an ant runs into pheromone trail.

$$P : \left\langle \begin{bmatrix} * & * & (P/*) \\ * & (*/P) & (P/*) \\ * & * & (P/*) \end{bmatrix} \rightarrow \texttt{Right} \; ; \; E_A \rightarrow E \right\rangle \tag{40}$$

$$P : \left\langle \begin{bmatrix} (P/*) & * & * \\ (P/*) & (*/P) & * \\ (P/*) & * & * \end{bmatrix} \rightarrow \texttt{Left} \; ; \; E_A \rightarrow E \right\rangle \tag{41}$$

$$P : \left\langle \begin{bmatrix} * & * & * \\ * & (*/P) & * \\ (P/*) & (P/*) & (P/*) \end{bmatrix} \rightarrow \texttt{Down} \; ; \; E_A \rightarrow E \right\rangle \tag{42}$$

$$P : \left\langle \begin{bmatrix} (P/*) & (P/*) & (P/*) \\ * & (*/P) & * \\ * & * & * \end{bmatrix} \rightarrow \texttt{Up} \; ; \; E_A \rightarrow E \right\rangle \tag{43}$$

By the notation $(P/*)$ we understand that the corresponding place contains either one of the pheromone symbols $P_0, \ldots, P_9$ or some other symbol, and we assume 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.

**Following the Pheromone Trail** The last set of programs controls the situation, when an ant follows the pheromone trail.

As discussed in section 5, to simplify the colony definition, the nest is located in the right lower corner of the environmental grid, hence we need to define only the rules of the situation, when the food is located in the left and/or above direction from the nest. The full scope of the rules was discussed in [14].

$$P : \left\langle E \rightarrow E \; ; \; \begin{bmatrix} * & P & * \\ * & P & * \\ * & * & * \end{bmatrix} \rightarrow \texttt{Up} \right\rangle \tag{44}$$

$$P : \left\langle E \rightarrow E \; ; \; \begin{bmatrix} * & * & * \\ P & P & * \\ * & * & * \end{bmatrix} \rightarrow \texttt{Left} \right\rangle \tag{45}$$

## 4 Software Implementation

Although there exist simulation software bundles for various types of membrane systems, and even for the case of P colonies there is a solution provided by Florea and Buiu [9], it would be difficult to implement the model of P colony with 2D environment and evolutionary rules. Therefore, we have developed a simple simulator in Python supporting 2D P colonies with evolving environment, and with extended capabilities for ACO simulations. The simulator operates in discrete time and space. To simulate the parallel behaviour of the 2D P colony model, agent rules are applied sequentially but in random order. This approach mimics the parallel actions of the agents and provides a realistic simulation of the colony behaviour.

At the moment, the application is sufficient to run experiments with a small number of agents and a not very large environment. Testing in a 150×100 environment with 100 agents took about one minute for 400 iterations on a regular PC with an Intel i7 processor. However, the application is not yet optimised for time and memory requirements, so larger environments and more agents and their programs could be simulated in the future.

The application also modular, allowing future integration of new features. For example, it currently supports graphical visualisation using the Mathplotlib

library. We plan to replace it with the PyGame library which is more suitable for real-time visualisation. Visualisation can be completely disabled to speed up computation if needed.

In summary, the developed software package serves as a versatile platform for simulating 2D P colonies and provides capabilities for dynamic / evolving environments and ACO simulations. Its modular design and planned optimisations ensure adaptability and scalability for future enhancements and specialised applications. The simulator design and usage is described in detail in [21].

## 5 Results

We first describe the testing methodology. We use the same 2D P colony configuration for both basic ACO search and the vicinity search, only the agent programs are different. The number of agents is 100. The agents/ants act in an environment of size $150 \times 100$. An ant is aware of its position in the environment. The extremes (food) are generated using the Python normal distribution function $numpy.random.normal(10, 5)$, where parameter 10 is the mean value and 5 is the standard deviation. Function values greater than 21 are marked as food (outlier) in the environment. The approximate number of extremes (food) generated by this function can be seen in Figure 1 on the left. The environment is re-generated each time the simulation is run. The number of iterations of the algorithm for each run is 400.

To simplify the colony definition, the nest is located in the right lower corner of the environmental grid. The position of the nest in the middle of the environment would be more efficient in the terms of faster exploration of the environment, but the primary goal here is to verify the assumption whether the vicinity search can improve the search results.

We ran the simulator 100 times with original agent programs and then 100 times with programs augmented by the vicinity search. The results in Table 1 show how the vicinity search increases the probability of finding extremes of the studied function. In the case of original agent programs, the agents found an average of 31 extremes over 100 runs, corresponding $< 26$ percent of the total number of extremes. In contrast, for the 2D P colony with the vicinity search, agents found an average of 66 extremes, corresponding to 55 percent of the total number of extremes. This means an improvement of more than 29 percent.

Another consequence of the vicinity search is the ability of agents to search a larger part of the environment in the same number of iterations. The situation is illustrated in Figure 1 in the middle and to the right. This verifies the hypothesis that the vicinity search has a positive search effect for environments with extremes located with normal spatial distribution.

The area marked as 'Deeply searched part of the environment' in Figure 1 is the part of the environment where agents spend more than 90% of their exploitation time, hence a global optimum can be found with a high probability when placed here. It turns out that this area increased by about 29% after introduction of the vicinity search. Table 1 confirms that in the case of extremes

distributed according to the normal distribution, the improvement in the number of found extremes corresponds to the proportion of the deeply searched area. Finally, it should be noted that for a different distribution of extremes the benefit of the vicinity search may be different.

**Table 1.** Comparison of the original ACO implementation and the vicinity search version (results averaged over 100 runs).

| **Summary** | **Original ACO** | **Vicinity search** | **Difference** |
|---|---|---|---|
| Average number of found extremes | 31.38 | 66 | 34.62 |
| Percentage of total No. of extremes: | 25.75% | 55.14% | 29.40% |

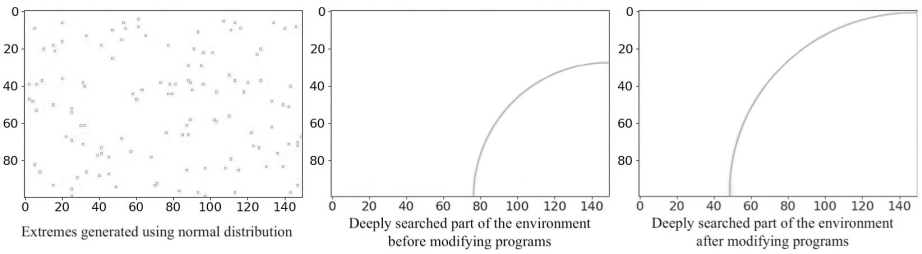


**Fig. 1.** Environment in detail. Left: extremes located according to a normal distribution. Middle and on the right: deeply searched part of the environment without and with the vicinity search.

## 6 Conclusion

We have introduced a model of 2D P colony with evolving environment ($2D_{ev}$ P COL) implementing an ant colony optimisation(ACO) algorithm enhanced with the vicinity search principle. The model is inspired by improved versions of ACO with local search. We have provided a detailed formal description of the $2D_{ev}$ P COL using the vicinity search, and run a batch of simulations of search of function extremes (represented by food) placed in a 2D environment according to the normal distribution. Then we compared the results with the original ACO simulations implemented also by a $2D_{ev}$ P COL and with identical experimental settings.

The obtained results confirmed that the introduction of vicinity search improved substantially the search efficiency. The agents were able to deeply explore much larger part of the environment, which we did not quantify precisely but illustrated in Fig. 1 (as this is rather a qualitative than a quantitative characteristics). Furthermore, an average number of found extrema increased from

approx. 26% to 55%. This significant improvement indicates an applicability of the $2D_{ev}$ P COL-based algorithms to real life optimisation problems.

There is a room for improvements both in theoretical and implementation part of this method. Other variants of local search should be implemented and tested to fine-tune the search efficiency. Also on the software simulator side, there is a room for speed/memory optimization, and primarily also for implementation of true parallel processing of individual agents using a GPU.

**Acknowledgments.**

This work was supported by the Silesian University in Opava under the Student Funding Plan, project SGS/9/2024.
Research was also supported by the Project of VSB - Technical University in Ostrava, SP2025/052.

# Bibliography


[1] Arteta Albert, A., Díaz-Flores, E., López, L. F. d. M., and Gómez Blas, N. (2021). An in vivo proposal of cell computing inspired by membrane computing. *Processes*, 9(3).

[2] Bo, W., Fang, Z., Wei, L., Cheng, Z., and Hua, Z. (2021). Malicious urls detection based on a novel optimization algorithm. *IEICE Transactions on Information and Systems*, E104.D(4):513–516.

[3] Cienciala, L., Ciencialová, L., and Perdek, M. (2012). 2D P colonies. In *Membrane Computing*, volume 7762, pages 161–172. Springer.

[4] Colorni, A., Dorigo, M., and Maniezzo, V. (1991). Distributed optimization by ant colonies. In *European Conference on Artificial Life*, pages 134–142.

[5] Deng, X., Dong, J., Wang, S., Luo, B., Feng, H., and Zhang, G. (2022). Reducer lubrication optimization with an optimization spiking neural P system. *Information Sciences*, 604:28–44.

[6] Dong, J., Zhang, G., Luo, B., et al. (2022a). A distributed adaptive optimization spiking neural P system for approximately solving combinatorial optimization problems. *Information Sciences*, 596:1–14.

[7] Dong, J., Zhang, G., Luo, B., et al. (2022b). Multi-learning rate optimization spiking neural P systems for solving the discrete optimization problems. *Journal of Membrane Computing*, 4:209–221.

[8] Dorigo, M. and Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66.

[9] Florea, A. G. and Buiu, C. (2016). Development of a software simulator for P colonies. applications in robotics. *International Journal of Unconventional Computing*, 12(2-3):189–205.

[10] Gheorghe, M., Stamatopoulou, I., Holcombe, M., and Kefalas, P. (2004). Modelling dynamically organised colonies of bio-entities. In *Unconventional Programming Paradigms*, volume 3566. Springer.

[11] Huang, L., Sun, L., Wang, N., and Jin, X. (2007). Multiobjective optimization of simulated moving bed by tissue P system. *Chinese Journal of Chemical Engineering*, 15(5):683–690.

[12] Kelemen, J. and Kelemenová, A. (1992). A grammar-theoretic treatment of multiagent systems. *Cybernetics and System*, 23(6):621–633.

[13] Kelemen, J., Kelemenová, A., and Păun, G. (2004). Preview of P colonies: A biochemically inspired computing model. In *Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX)*, pages 82–86.

[14] Langer, M. and Valenta, D. (2023). On evolving environment of 2D P colonies: ant colony simulation. *Journal of Membrane Computing*, 5(3):117–128.

[15] Luo, Y., Guo, P., and Zhang, M. (2019). A framework of ant colony P system. *IEEE Access*, 7:157655–157666.

[16] Mavrovouniotis, M., Müller, F. M., and Yang, S. (2016). Ant colony optimization with local search for dynamic traveling salesman problems. *IEEE transactions on cybernetics*, 47(7):1743–1756.

[17] Păun, G. (2000). Computing with membranes. *Journal of Computer and System Sciences*, 61:108–143.

[18] Ramachandranpillai, R. and Arock, M. (2020). Spiking neural P ant optimisation: a novel approach for ant colony optimisation. *Electron. Lett.*, 56:1320–1322.

[19] Smolke, C. D. (2009). Building outside of the box: igem and the biobricks foundation. *Nature biotechnology*, 27(12):1099–1102.

[20] Thilak, K. D. and Amuthan, A. (2018). Cellular automata-based improved ant colony-based optimization algorithm for mitigating ddos attacks in vanets. *Future Generation Computer Systems*, 82:304–314.

[21] Valenta, D. and Langer, M. (2023). On 2D P colony simulator. In Ciencialová, L., editor, *Proceedings of the Twenty-fourth International Conference on Membrane Computing (CMC2023)*, pages 177–190. Silesian University in Opava. https://cmc2023.slu.cz/user/pages/files/Proceedings_CMC2023.pdf.

[22] Wang, L., Liu, X., Qu, J., Zhao, Y., Gao, L., and Ren, Q. (2023). An extended membrane system with monodirectional tissue-like P systems and enhanced particle swarm optimization for data clustering. *Applied Sciences*, 13(13):7755.

[23] Zhang, G., Rong, H., Neri, F., and Pérez-jiménez, M. (2014). An optimization spiking neural P ssystem for approximately solving combinatorial optimization problems. *International Journal of Neural Systems*, 24(5).

[24] Zhu, M., Yang, Q., Dong, J., Zhang, G., Gou, X., Rong, H., Paul, P., and Neri, F. (2021). An adaptive optimization spiking neural P system for binary problems. *International Journal of Neural Systems*, 31(01).