
Modelling of Grey Wolf Optimization Algorithm Using 2D P Colonies

Daniel Valenta¹, Lucie Ciencialová^{1,2}, and Luděk Cienciala^{1,2}

¹ Institute of Computer Science, Silesian University in Opava, Czech Republic

² Research Institute of the IT4Innovations Centre of Excellence, Silesian University in Opava, Czech Republic
{daniel.valenta, lucie.ciencialova, ludek.cienciala}@fpf.slu.cz

Summary. In this paper, we investigate a possibility of Grey wolf optimization algorithm simulation by 2D P colonies. We introduce a new kind of 2D P colony equipped with a blackboard. It is used by agents to store information that is reachable by all the agents from every place in the environment.

Key words: 2D P colonies, blackboard, Grey wolf optimization algorithm.

1 Introduction

2D P colonies are kind of P colonies, very simple membrane systems inspired by colonies of formal grammars. The interested reader is referred to [8] for detailed information on membrane systems (P systems) and to [4] and [3] for more information to grammar systems theory. For more details on P colonies consult the survey [2].

2D P colony consists of a finite number of agents - finite collections of objects in a cell - and their joint shared environment. The environment of 2D P colony is represented by a 2D grid of square cells. In each cell, there is a multiset of objects. The agents have programs consisting of rules. These rules are of three types: they may change the objects of the agents and they can be used for interacting with the joint shared environment of the agents and movement rule. The direction of the movement of the agent is determined by the contents of cells surrounding the cell in which the agent is placed. The program can contain at most one motion rule. To achieve the greatest simplicity in agent behaviour, one other condition was set. If the agent moves, it cannot communicate with the environment. So if the program contains a motion rule, then the other rule is an evolution rule. The number of objects inside each agent is set by definition and it is usually a very small number: 1, 2 or 3.

When the agent is moving around the 2D environment it has no information about the states and placements of the other agents. For remote information ex-

change, we add the agents the possibility to store and read the information from a blackboard. It is a table with an unchangeable structure given by definition. Agents can change values inside cells but not captions and the number of rows or columns.

Grey wolf optimization algorithm (GWO) is a meta-heuristic optimization technology. Its principle is to imitate the behaviour of grey wolves in nature to hunt cooperatively. Four types of grey wolves such as alpha, beta, delta, and omega are used for simulating the leadership hierarchy. In addition, the three main steps of hunting, searching for prey, encircling prey, and attacking prey, are implemented. The algorithm was introduced by Mirjalili et al. in 2014 in [9].

2 Grey wolf optimization algorithm

This section is to explain the way the GWO works. Grey wolf optimization algorithm is inspired by social dynamics found in packs of grey wolves and by their ability to dynamically create hierarchies in which every member has a clearly defined role. We distinguish the following wolves:

- *Alpha* male and female make up the dominant pair. The pack follows their lead during hunts, while locating a place to sleep, and so on. The most important attributes are their organisational abilities and discipline.
- *Beta* wolves support and respect the Alpha pair during its decisions.
- *Delta* wolves are subservient to Alpha and Beta wolves, follow their orders, and control Omega wolves. There are three types of Delta wolves: *scouts* – they observe the surrounding area and warn the pack, *sentinels* – they protect the pack when endangered and *caretakers* – they provide aid to old and sick wolves.
- *Omega* wolves help to filter the pack’s aggression and frustrations by serving as scapegoats.

In GWO, the agents (wolves) primary goal in its environment is to find and hunt down prey, which in our case equals finding the optimal solution to the given problem. The environment is represented by a mathematical fitness function characterising the specific problem. A value found at the current position of the wolf refers to the highest-quality prey located. The wolf with the best value is ranked as Alpha, the second as Beta, third as Delta, and all the other as Omega.

The hunting technique of a wolf pack can be described in 5 steps:

- *Search for prey* (point A in Fig. 2.) – wolves are attempting to find the most valuable prey with respect to the effort required to successfully hunt it.
- *Exploitation of prey* (point B in Fig. 2.) – wolves are attempting to draw attention to themselves and to separate the prey from its herd.
- *Encircling prey* (point C in Fig. 2.) – the attempt to push the prey into a situation it cannot escape from.
- *The prey is surrounded* (point D in Fig. 2.) – it can no longer escape.

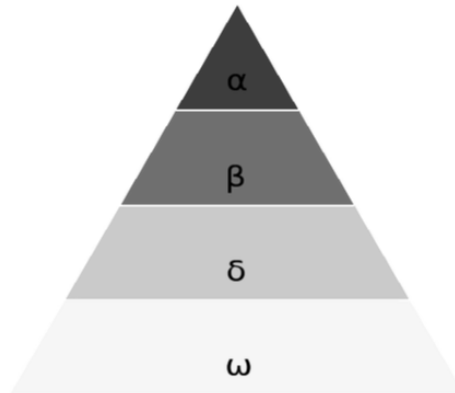


Fig. 1. wolf pack hierarchy

- *Attack* (point E in Fig. 2.) – wolves attack the prey’s weak spots (belly, legs, snout) until it succumbs to fatigue. Afterwards, they bring it down and crush its windpipe.



Fig. 2. Hunting technique of grey wolves in [6]

The algorithm is inspired by this process and smoothly transitions between scouting and hunting phases. In the scouting phase, the pack extensively scouts its environment through many random movements so that the algorithm does not

get stuck in a local extreme, while in the hunting phase, the influence of random movements is slowly reduced and pack members draw progressively closer to the discovered extreme. To maintain the divergence between those phases, each wolf is assigned vectors **A** and **C**.

$$\mathbf{A} \text{ is a vector with components } rand(-1, 1) * a,$$

where $rand(-1, 1)$ generates a random number between -1 and 1 and where

$$a = 2 - \left(\frac{2i}{i_{max}} \right)$$

while i is the algorithms current iteration, and i_{max} is the maximum number of iterations. It is random value between -2 and 2 . You can see its impact in the Fig. 3.

With growing iterations, it is more likely that its value will be between -1 and 1 . That makes it more likely for a wolf to be hunting.

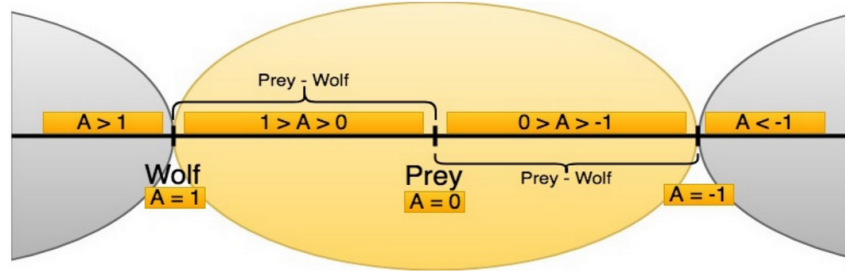


Fig. 3. Vector **A** and its impact in 1D

Another component supporting the scouting phase is vector

$$\mathbf{C} = rand(0, 2),$$

where $rand(0, 2)$, generates a random number between 0 and 2 . Vector **C** is similar to vector **A**, but iterations don't influence it. It helps the wolves behave more naturally. Similarly, in nature, wolves encounter various obstacles which prevent them from approaching prey comfortably.

The vectors **A** and **C** encourage wolves to prefer scouting, or hunting, and so to avoid local optima regardless of the algorithms current iteration.

Wolves' positions within the environment are updated based upon the estimated location of the prey using Alpha, Beta, and Delta wolves as guides.

Let $\mathbf{X}_j(i)$ be a positional vector of wolf j in i -th iteration. Positional vector of wolf j is updated as follows:

$$\mathbf{X}_j(i + 1) = \frac{\mathbf{X}_1 + \mathbf{X}_2 + \mathbf{X}_3}{3},$$

where i is the algorithms current iteration and $\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3$ are new potential position vectors of Alpha, Beta, and Delta wolves obtained from following formulas:

$$\begin{aligned}\mathbf{X}_1 &= \mathbf{X}_\alpha(i) - \mathbf{A}_1 * \mathbf{D}_\alpha \\ \mathbf{X}_2 &= \mathbf{X}_\beta(i) - \mathbf{A}_2 * \mathbf{D}_\beta \\ \mathbf{X}_3 &= \mathbf{X}_\delta(i) - \mathbf{A}_3 * \mathbf{D}_\delta\end{aligned}$$

where $\mathbf{X}_\alpha(i), \mathbf{X}_\beta(i), \mathbf{X}_\delta(i)$ are the position vectors of Alpha, Beta, and Delta wolves representing positions within the environment that are closest to the optimum in i -th iteration. Vectors $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3$ are calculated in the same way as vector $\mathbf{D}_\alpha, \mathbf{D}_\beta, \mathbf{D}_\delta$ are vectors defining the distance of the wolf j position from the prey as follows:

$$\begin{aligned}\mathbf{D}_\alpha &= |\mathbf{C}_1 * \mathbf{X}_\alpha(i) - \mathbf{X}_j(i)| \\ \mathbf{D}_\beta &= |\mathbf{C}_2 * \mathbf{X}_\beta(i) - \mathbf{X}_j(i)| \\ \mathbf{D}_\delta &= |\mathbf{C}_3 * \mathbf{X}_\delta(i) - \mathbf{X}_j(i)|\end{aligned}$$

where $|\mathbf{X}|$ is the vector whose components are the absolute values of the components of \mathbf{X} . Vectors $\mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3$ are computed in the same way as vector \mathbf{A} and influences the weight of the preys estimated position $\mathbf{X}_\alpha, \mathbf{X}_\beta, \mathbf{X}_\delta$ (increasing or decreasing it).

If wolves have the tendency to move closer towards prey, they will begin to encircle it (wolves approach from various directions), as you can see it in Fig. 4.

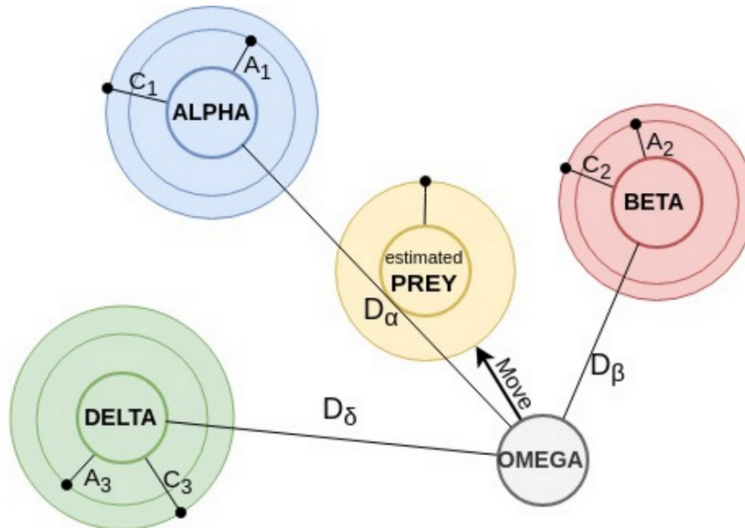


Fig. 4. Positional updates of Omega wolves as it is described by the mathematical formula

2.1 Algorithm pseudocode

In this subsection we describe the algorithm in pseudocode. Algorithms inputs are dimensions of the problems environment, boundaries of the problems environment, fitness function characterising the problem, size of the pack (number of wolves/agents), number of iterations of the algorithm, termination criteria and criteria of the fitness function.

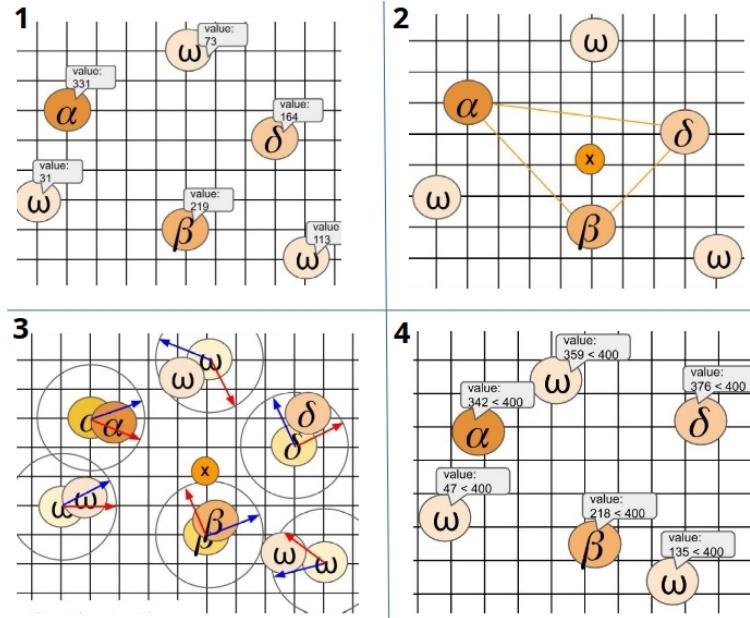


Fig. 5. Algorithm steps

The algorithms pseudocode follows:

- In the first step, agents (wolves) are randomly spread out across the environment.
- In each iteration i :
 - calculate the fitness value of each agent and determine the social hierarchy – Fig. 5. part 1. The agent with the best value (closest to the optimum) is Alpha, second best is Beta, third best is Delta, and all others are Omega.
 - calculate the best solution found thus far by Alpha, Beta and Delta ($\mathbf{X}_\alpha(i)$, $\mathbf{X}_\beta(i)$, $\mathbf{X}_\delta(i)$) and average it – Fig. 5. part 2,
 - update positions of all the wolves $X_j(i+1)$, while random vectors A and C are updated for each one – Fig. 5. part 3,
 - check the termination criterion – Fig. 5. part 4. Iterations terminate when fitness function value reaches a preset value.

3 2D P Colonies

In this section, we recall the definition of 2D P colonies and other terms related to them.

Definition 1. *A 2D P colony is a construct*

$$\Pi = (A, e, Env, B_1, \dots, B_k, f), k \geq 1, \text{ where}$$

- *A is an alphabet of the colony, its elements are called objects,*
- *$e \in A$ is the basic environmental object of the 2D P colony,*
- *Env is a pair $(m \times n, w_E)$, where $m \times n, m, n \in N$ is the size of the environment and w_E is the initial contents of environment, it is a matrix of size $m \times n$ of multisets of objects over $A - \{e\}$.*
- *$B_i, 1 \leq i \leq k$, are agents, each agent is a construct $B_i = (o_i, P_i, [o, p])$, $0 \leq o \leq m, 0 \leq p \leq n$, where*
 - *o_i is a multiset over A, it determines the initial state (contents) of the agent, $|o_i| = 2$,*
 - *$P_i = \{p_{i,1}, \dots, p_{i,l}\}, l \geq 1, 1 \leq i \leq k$ is a finite set of programs, where each program contains exactly 2 rules, which are in one of the following forms each:*
 - *$a \rightarrow b$, called the evolution rule, $a, b \in A$,*
 - *$c \leftrightarrow d$, called the communication rule, $c, d \in A$,*
 - *$[a_{q,r}] \rightarrow s, 0 \leq q, r \leq 2, s \in \{\leftarrow, \Rightarrow, \Uparrow, \Downarrow\}$, called the motion rule,*
- *$f \in A$ is the final object of the colony.*

A configuration of the 2D P colony is given by the state of the environment - matrix of type $m \times n$ with multisets of objects over $A - \{e\}$ as its elements, and by the state of all agents - pairs of objects from alphabet A and the coordinates of the agents. An initial configuration is given by the definition of the 2D P colony.

A computational step consists of three parts. The first part lies in determining the set of applicable programs according to the current configuration of the 2D P colony. In the second part, we have to select from this set one program for each agent, in such a way that there is no collision between the communication rules belonging to different programs. The third part is the execution of the chosen programs.

A change of the configuration is triggered by the execution of programs and it involves changing the state of the environment, contents and placement of the agents.

A computation is non-deterministic and maximally parallel. The computation ends by halting when there is at least one agent that has no 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.

The aim of introducing 2D P colonies is not studying their computational power but monitoring their behaviour during the computation.

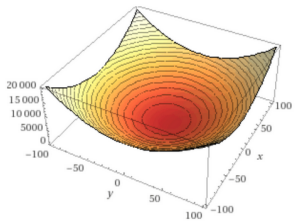
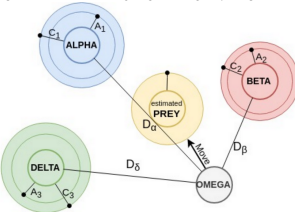
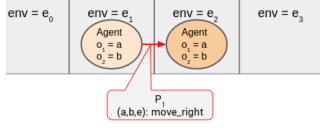
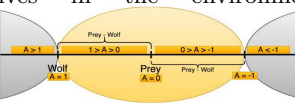
4 Modeling of Grey wolf algorithm using P colonies

As for the modeling of GWO, some similarities as well as a few differences have been found. Both are inspired by the nature, usable for solving optimisation problems, and both are multiagent system models. For comparison see the differences / problems:

- *Environmental problem,*
- *Communication problem,*
- *Randomness problem.*


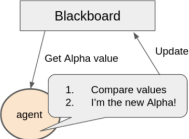
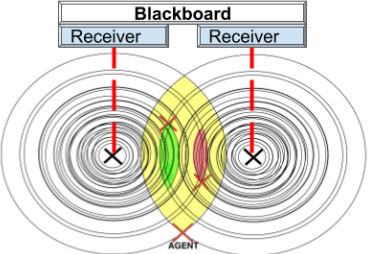
These problems are described in the Table 1.

Table 1. Differences between Grey wolf algorithm and P colony tables.

Difference System	Grey wolf algorithm	P colony
Environmental problem	<p>The environment is represented by a mathematical fitness function.</p> 	<p>The environment is represented by a multiset of symbols.</p> $Env = (6 \times 6, w_E),$ $w_E = \begin{bmatrix} D & D & D & D & D & D \\ D & S & S & D & D & D \\ D & S & S & D & D & D \\ D & D & D & S & S & D \\ D & D & D & S & S & D \\ D & D & D & D & D & D \end{bmatrix},$
Communication problem	<p>The agents have the knowledge of their global position in the environment.</p> 	<p>They are communities of simple reactive agents independently living and acting in a joint shared environment.</p> 
Randomness problem	<p>Random vectors A and C influence the movement of wolves in the environment.</p> 	<p>Each rule is deterministic, the only way to implement randomness is to randomise the choosing rule for identical configurations.</p> $P_1:$ <ol style="list-style-type: none"> 1. $(a, a, e): \text{action_1} \quad ?$ 2. $(a, a, e): \text{action_2} \quad ?$

2D P colony definition needs to be adjusted to meet the described requirements. Proposed solutions for those problems are described in Table 2.

Table 2. Proposed solution of differences / problems found

Difference Problem	Proposed solution																				
Environmental problem	<p>The environment will be a pair of matrix $m \times n$ and fitness function $f(x)$, where $m \times n, m, n \in \mathbb{N}$ is the size of the environment and $f(x)$ is a mathematical function with the initial contents of environment. Alphabet A will contains real numbers and environmental symbol, $A = \{\mathbb{R}\} \cup \{e\}$, The rules of the programs, which guide the agents, will compare the number values of objects using operators smaller "$<$" or greater "$>$" then, agents B_i will be defined as $B_i = o_i, P_i, [r_x, r_y], o_i = 2$.</p> <div style="text-align: center;">  </div> <p>Example: $Env = (6 \times 6, f(x)), f(x) =$</p>																				
Communication problem	<p>We extend the P system by adding the Blackboard that saves the agents' best fitness values and is always accessible to read and write by all agents. Agents must know their position in the environment, because the fitness value is not enough for calculating the preys estimated position.</p> <div style="display: flex; justify-content: space-around;"> <table border="1" data-bbox="597 1003 954 1129"> <thead> <tr> <th colspan="4">BlackBoard</th> </tr> <tr> <th>Index</th> <th>0</th> <th>1</th> <th>2</th> </tr> </thead> <tbody> <tr> <td>Agent</td> <td>B_{best}</td> <td>B_{best}</td> <td>B_{best}</td> </tr> <tr> <td>Position</td> <td>x_1, y_1</td> <td>x_2, y_2</td> <td>x_3, y_3</td> </tr> <tr> <td>Value</td> <td>Best value</td> <td>2nd best value</td> <td>3rd Best value</td> </tr> </tbody> </table> <div data-bbox="966 997 1153 1134">  </div> </div>	BlackBoard				Index	0	1	2	Agent	B_{best}	B_{best}	B_{best}	Position	x_1, y_1	x_2, y_2	x_3, y_3	Value	Best value	2nd best value	3rd Best value
BlackBoard																					
Index	0	1	2																		
Agent	B_{best}	B_{best}	B_{best}																		
Position	x_1, y_1	x_2, y_2	x_3, y_3																		
Value	Best value	2nd best value	3rd Best value																		
Randomness problem	<p>We will use the blackboard as means of giving feedback to agents. Agents do not need to know their position in the environment, all agents who can contribute to the search will send solution to the blackboard points called receivers, and estimation of prey position is calculated as average of distances collected by blackboard points from wolves Alpha, Beta and Delta. Omega wolves can ping the blackboard if changing position and get their distance from the prey. If the distance would decrease compared to the original distance, then the wolf will move. We plan to introduce randomness by only using two blackboard receivers and rounding the wolves' distances from the prey.</p> <div style="text-align: center;">  </div>																				

5 Proposed model of 2D P colony with the blackboard

As for the proposed definition of 2D P colony model with the blackboard, it is adapted to the suggestions from the previous chapter so that it allows simulation of the Grey wolf algorithm.

5.1 Definition

$P_{gw} = (A, e, env, B_1, B_2, \dots, B_n, X, f)$, where:

- $A = \{\mathbb{R}\} \cup \{e, m, f\}$,
- $e \in A$ is the basic environmental object,
- env is a pair $(m \times n, f(x))$, where $m \times n, m, n \in \mathbb{N}$,
- B_1, B_2, \dots, B_n are the agents, $B_i = (O_i, P_i, [r_x, r_y])$, where:
 - $O_i = 2$,
 - $P_1 = P_2 = \dots = P_n$, P_i rules are defined below,
 - r_x, r_y are the initial coordinates,
- X is the blackboard defined as structure in Fig. 6.,
- f is the final object, $f \in A$.

BlackBoard							
Index	0	1	2	3	4	...	5
Agent	B_{Alpha}	B_{Beta}	B_{Delta}	B_1	B_2	...	B_n
Value	Best value	2nd best value	3rd Best value				
Distance	Estimation of prey position (calculated by the BlackBoard)			distance of prey	distance of prey	...	distance of prey

Fig. 6. Blackboard structure

The initial agents' configuration is: $(O_1[e], O_2[e], env[i])$, where $i \in \mathbb{N}$.

Programs P_i associated with i-th agent are:

1. $(e_1, e_2, x) : e_1, e_2 \leftrightarrow x; x \in \mathbb{R}$
2. $(x, y, e); x, y \in \mathbb{R}$:
 - a) $y \leftarrow Get(BB[Alpha])$ and $Compare(x, y)$:
 - i. $x > y$: I'm new Alpha, $Update(BB[Alpha])$;
 - ii. $x < y$: $y \leftarrow Get(BB[Beta])$ and $Compare(x, y)$:
 - A. $x > y$: I'm new Beta, $Update(BB[Beta])$;
 - B. $x < y$: $y \leftarrow Get(BB[Delta])$ and $Compare(x, y)$:
 - $x > y$: I'm new Delta, $Update(BB[Delta])$;
 - $x < y$: I'm Omega: $y \leftrightarrow e; e \rightarrow m; m \leftrightarrow y$;
3. $(x, y, m); x, y \in \mathbb{R}$:
 - a) $PingBB[i]$ and $y \leftarrow Get(BB[i])$;

- b) $PingBB[i] + mv1; mv1 = (\leftarrow, \Rightarrow, \uparrow, \downarrow)$ and $y \leftarrow Get(BB[i]);$
 - c) $Compare(x, y);$
 - i. $x > y: Do(mv1);$
 - ii. $x < y:$
 - A. $PingBB[i] + mv2; mv2 = (\leftarrow, \Rightarrow, \uparrow, \downarrow) - mv1; mv1 \neq mv2$
 - B. $y \leftarrow Get(BB[i])$ and $Compare(x, y);$
 - $x > y: Do(mv2);$
 - $x < y: \dots$ (try it with $mv3$ and $mv4$)
 - Can't move: $y \leftrightarrow m; m \rightarrow f; f \leftrightarrow m;$
4. $(x, y, f); x, y \in \mathbb{R}: \text{Stop the agent. } \}$

P_i rules definition use the following symbols:

- \leftarrow means Get from the blackboard,
- \rightarrow means Rewrite agent's object,
- \leftrightarrow means Change agent's object with environment object,
- " \leftarrow " = *LEFT*, " \Rightarrow " = *RIGHT*, " \uparrow " = *UP*, " \downarrow " = *DOWN*

At this point it is important to focus on the use of the blackboard by the agents. Agents can use blackboard functions:

- $Get(BB[i]); i = \text{agent's index}$ - agent i gets its distance from the prey (it is calculated by the blackboard),
- $Update(BB[x]); x = \text{Alpha, Beta, Delta}$ - agent update $B_{alpha}, B_{beta},$ or B_{delta} field value,
- $Ping[BB[i]]; i = \text{the agent's index}$ - agent can ping the blackboard from some position, its distance from the prey is recalculated.

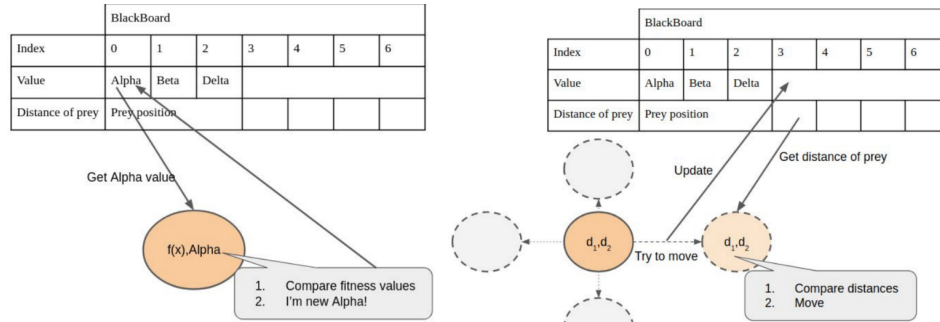


Fig. 7. Blackboard in use

The agent concludes it is Alpha and it rewrites the corresponding blackboard field in Fig. 7. on the left side. On the right side of Fig. 7., the agent concludes it is Omega and it will try to move with blackboard's assistance.

Finally, the following derivation can be created. Fig. 8., a sequence of derivation steps is depicted.

The first step (iteration) (point 1 in Fig. 8.) starting with two agents randomly initialized into the environment. They are in the initial configuration - two objects e inside the agent.

In the second iteration (point 2 in Fig. 8.), agents exchange their objects for objects placed in the environment (computed by fitness function).

In next iterations (point 3 in Fig. 8.), agents get Alpha value from the blackboard and try to compare it to their own value. If Alpha value is empty, the first agent to try to compare its value will become the Alpha and update the blackboard. If more than one agent tries to write value into the blackboard in the same position the winner is non-deterministically chosen.

The same process is used for declaring Beta and Delta agents (point 4 in Fig. 8.). If the agent's fitness value is lower than Delta, this agent becomes Omega. At this point (point 5 in Fig. 8.) this agent rewrites the environmental object to m . Afterwards, it will try to move (point 6 in Fig. 8.). Before moving, however, it will compare the distances. The algorithm iterates until no more movement which would improve the fitness value is possible.

6 Conclusion

In this paper we introduce extended model of 2D P colonies that can simulate solving optimization problem by Grey wolf optimization algorithm. The model will be improved in the near future. It is assumed that the function *Compare()* can be replaced by special equivalent rules, like $(x > y, e) : action$. It is also assumed that the other functions, such as *Do(mv1)*, can be replaced by the rules in a form closer to the common rules of P systems.

When it comes to testing the model and comparing it with the original version of GWO, the observation of the behaviour of the modified model is the aim of our further work. Potentially, the modified model could be faster or more efficient (in its approach towards the divergence between scouting for prey and hunting) compared to the original algorithm.

Acknowledgments.

The work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science - LQ1602 and by the Silesian University in Opava under the Student Funding Scheme, project SGS/11/2019.

References

1. Cenciala, L., Cencialová, L., Perdek, M.: 2D P colonies. In: Csuhaj-Varjú E., Gheorghie M., Rozenberg G., Salomaa A., Vaszil Gy. (eds) Membrane Computing. CMC

2012. Lecture Notes in Computer Science, vol 7762. Springer, Berlin, Heidelberg, pp. 161–172 (2012)
2. Ciencialová, L., Csuhaj-Varjú, E., Cienciala, L., and Sosík, P.: P colonies. *J Membr Comput* 1, 178–197 (2019)
 3. Csuhaj-Varjú, E., Kelemen, J., Păun, Gh., Dassow, J.(eds.): *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*. Gordon and Breach Science Publishers, Inc., Newark, NJ, USA (1994)
 4. Kelemen, J., Kelemenová, A.: A Grammar-Theoretic Treatment of Multiagent Systems. *Cybern. Syst.* 23(6), 621–633 (1992)
 5. Kelemen, J., Kelemenová, A., Păun, Gh.: Preview of P colonies: A biochemically inspired computing model. In: *Workshop and Tutorial Proceedings. Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX)*. pp. 82–86. Boston, Massachusetts, USA (September 12-15 2004)
 6. Muro, C., Escobedo, R., Spector, L., Coppinger, R.P.: Wolf-pack (*Canis lupus*) hunting strategies emerge from simple rules in computational simulations, *Behavioural Processes* 88(3), 192–197 (2011)
 7. Păun, Gh.: Computing with membranes. *J. Comput. Syst. Sci.* 61(1), 108–143 (2000)
 8. Păun, Gh., Rozenberg, G., Salomaa, A.(eds.): *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA (2010)
 9. Mirjalilia, S., Mirjalilib, S.M., Lewisa, A.: Grey Wolf Optimizer. *Advances in Engineering Software* 69, 46–61(2014)

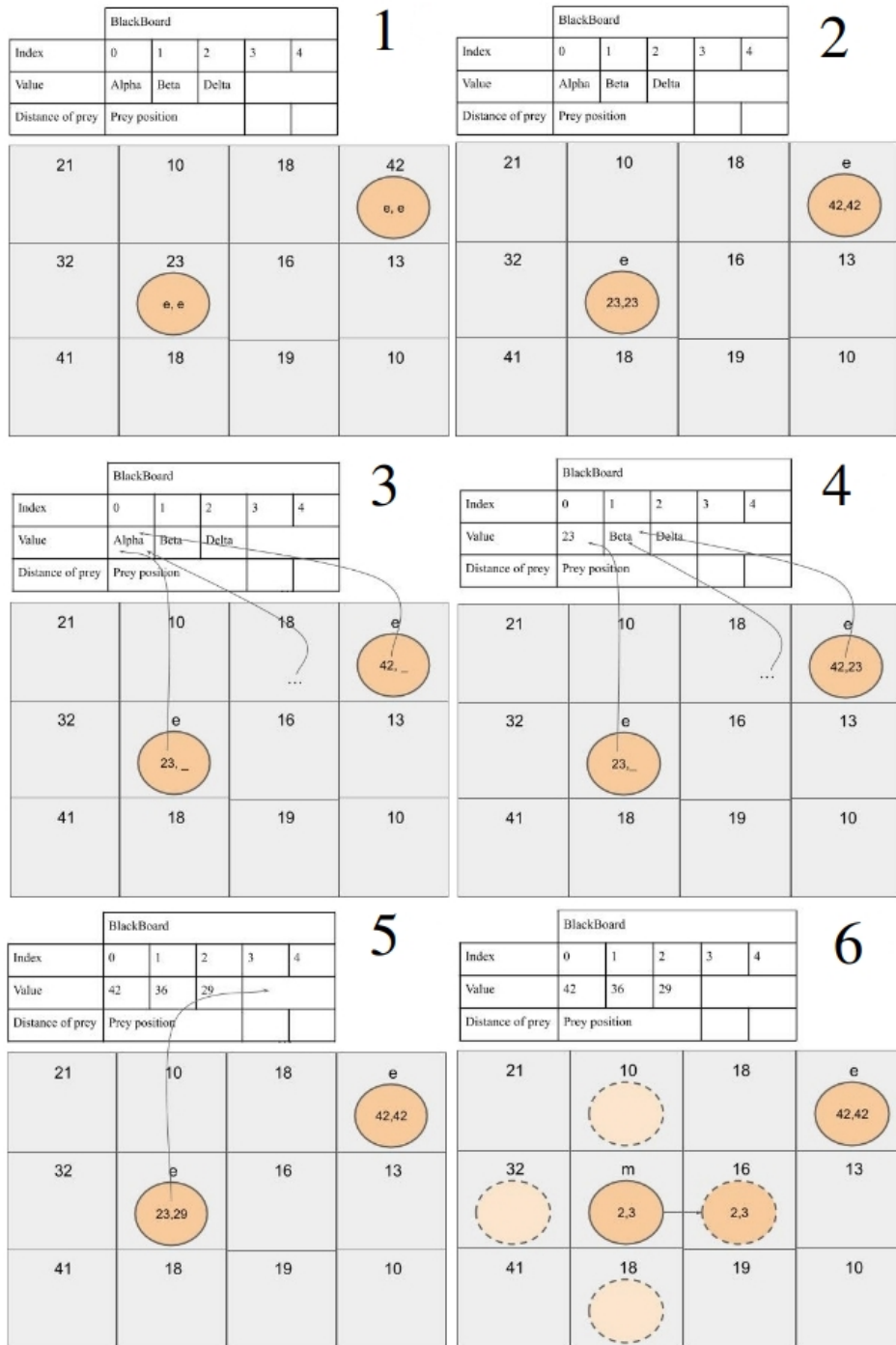


Fig. 8. Example of derivation