ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 9, No. 1, 2024

UNCREWED VEHICLE PATHFINDING APPROACH BASED ON ARTIFICIAL BEE COLONY METHOD

Oleh Sinkevych, Yaroslav Boyko, Bohdan Sokolovskyy, Mykhailo Pavlyk, Oleh Yarosh, Oleksandr Futey

Ivan Franko National University of Lviv, 50, Drahomanova Str, Lviv, 79005, Ukraine.
Author's e-mail*: oleh.sinkevych@lnu.edu.ua, yaroslav.boyko@lnu.edu.ua, bohdan.sokolovskyy@lnu.edu.ua, mykhailo.pavlyk@lnu.edu.ua, oleh.yarosh@lnu.edu.ua, oleksandr.futey@lnu.edu.ua*

https://doi.org/10.23939/acps2024.01.001

Submitted on 30.03.2024

***Abstract*: The presented study is dedicated to the dynamic pathfinding problem for UV. Since the automation of UV movement is an important area in many applied domains like robotics, the development of drones, autopilots, and self-learnable platforms, we propose and study a promising approach based on the algorithm of swarm AI. Given the 2D environment with multiple obstacles of rectangular shape, the task is to dynamically calculate a suboptimal path from the starting point to the target. The agent has been represented as UV in 2D space and should find the next optimal movement point from the current position only within a small neighborhood area. This area has been defined as a square region around the current agent's position. The size of the region has been determined by the attainability of the agent's scanning sensors. If the obstacle is detected by the agent, the latter should be taken into consideration while calculating the next trajectory point. To perform these calculations, the ABC metaheuristic, one of the best representatives of swarm AI, has been used. The validation of the proposed approach has been performed on several 2D maps with different complexity and number of obstacles. Also, to obtain the proper configuration, an inverse problem of identification of guided function weights has been formulated and solved. The outlined results show the perspective of the proposed approach and can complement the existing solutions to the pathfinding problem.**

***Index Terms*: Artificial bee colony, Metaheuristics, Numerical optimization, Swarm intelligence, UV pathfinding.**

I. INTRODUCTION

In the rapidly evolving landscape of modern technology, uncrewed vehicles (UVs) have emerged as pivotal instruments of innovation, pushing the boundaries of what is possible in fields ranging from environmental monitoring to disaster response.

The thing is getting even more challenging when it comes to dealing with a bunch of UVs. In order to fulfill the assigned task, such a group of vehicles must not only carry out mutual communication but also jointly coordinate their strategy [1]. This imposes additional difficulties on the design of even simple algorithms compared to a single agent.

The nature-inspired swarm AI algorithms cover many ideas, starting with artificial ant colonies and ending with newly found whale and gray wolf optimizations. Considering some of the shortcomings of the new approaches, classical and well-tested approaches are still of interest as a basis for designing swarm systems. We can highlight particle swarm optimization (PSO) [2], artificial bee colony (ABC) [3], artificial ant colony (AAC) [4], and the firefly algorithm [5] as the outstanding representatives of swarm techniques. Because based on published papers, it is still impossible to select the best one, here we would like to give a chance to the ABC method as a simple and simultaneously effective swarm AI algorithm.

II. LITERATURE REVIEW AND PROBLEM STATEMENT

First, let us consider just one of the relevant problems of developing UV's software, namely seeking an optimal trajectory. There are a number of scientific papers related to the mentioned problem. If one distinguishes those that concern ABC-based methods, there are many available at the given period.

In [6], authors proposed a local trajectory planning scheme which has been developed with ABC optimization algorithm to optimally obtain the next positions of all the robots in the world map from their current positions, so that the paths to be developed locally for n-robots are sufficiently small with minimum spacing with the obstacles, if any, in the world map. In [7], the hybrid approach of composing the ABC and evolutionary computing is considered. Local search based on ABC precedes global evolutionary search, which still incorporates single-objective optimization. Here, the authors insist on the inefficiency of the ABC in order to solve the full path optimization problem.

The paper [8] offers the ABC approach to the coordination of motions of multiple robots in parallel. Also, there is an interesting methodology for multiple agents' coordination. Dealing with the interaction of multiple robots requires collision-safe movement, which has been

studied in [9]. Here, researchers discuss the improved and efficient ABC algorithm for online robot path planning strategy.

In [10], authors modified a generic ABC algorithm to improve the exploitation phase using the Arrhenius strategy. The reported approach is declared to be better than standard implementation, but this modification does not include a sophisticated approach to optimization for the path searching ability. More recent publications mainly focus on basic ABC improvements in contrast to the complications of the objective function formulation. For instance, in [11], the authors propose the modification of ABC combined with conventional evolutionary programming (for refinement purposes) in order to improve the optimum path toward the goal position while taking into account the distance of the food point from the nearest obstacles. Paper [12] moves further toward the complex hybridization of ABC, EP (evolutionary programming), and the probabilistic roadmap method. Obtained outcomes conclude the superiority of the proposed path planning method against traditional ABC and ABC-EP algorithms applied to the sophisticated single objective function. In addition, work [13] profoundly explains the combination of modified ABC and EP.

The goal of this study is to develop the pathfinding algorithm for UV in a dense environment with multiple obstacles. The UV is represented as the dimensionless agent on the 2D map. Using ABC method, the agent should build a path from starting to the target point in dynamic, step by step manner. In details, ABC algorithm is used to optimize special weighted guided function in each movement region within reach of the agent's sensors.

Firstly, the combined single-objective dynamic pathfinding problem in a 2D environment is considered. Secondly, the solutions are investigated in terms of the influence of objective weights. Thirdly, the inverse problem of finding the best objective weights is formulated and solved. All numerical experiments were implemented in Python 3.

III. SCOPE OF WORK AND OBJECTIVES

Let us consider two static 2D environments (Fig. 1, 2), where a single UV or agent should reach a predetermined goal starting from some initial position. Here are the movement maps, where rectangular sectors are forbidden regions, and two circles are starting and target points. The agent can travel on the map along the calculated optimal trajectory in such a way as to avoid obstacles.

Here, obstacles locations are static but not known in advance to the agent, which in some sense makes the problem dynamic. Being at some point on the map (x_i, y_i), the agent can move to a new point (x_i', y_i') along a given trajectory, and the length of the steps between the previous and next points may be different.

The agent can "see" or perform only in a bounded square region defined as a square with the center of the agent's current position. The size of that region can vary during the computations and is critical for the agent's pathfinding capabilities; the larger the value, the further the agent can operate. If there are no obstacles in the current region, agents can move anywhere within the area. If there are obstacles in the region or the agent's area intersects with obstacles, the agent must pick only the feasible position (Fig. 3).

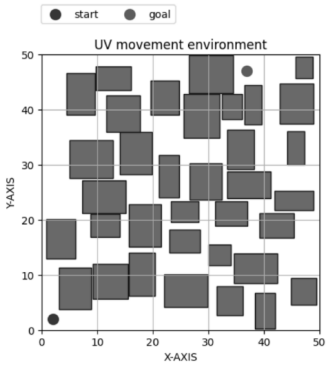


Fig. 1. Movement environment, map # 1

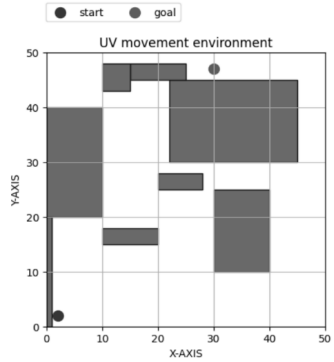


Fig. 2. Movement environment, map # 2

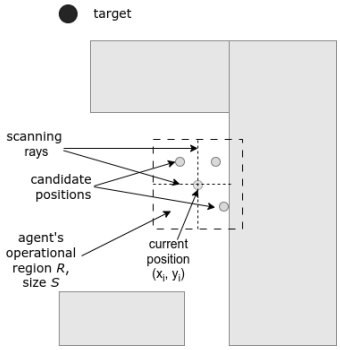


Fig. 3. Scheme of agent's region

The calculation of the optimal trajectory of the agent's movement is based on three criteria, which can be represented in the form of the following functions:

$$f_1(x_i, y_i) = \sqrt{(x_i - x^G) + (y_i - y^G)}\ , \qquad (1)$$

$$f_2(x_i, y_i) = \sum_{j=1}^{n} \frac{1}{d_j}, d_j = \min\_\text{dist}\left[(x_i, y_i), obst_j\right], \quad (2)$$

$$f_3(\mathbf{x}_i \mid \mathbf{x}_{i-1}, \mathbf{x}_{i-2}) = \arccos\left(\frac{(\mathbf{x}_i - \mathbf{x}_{i-1})(\mathbf{x}_{i-1} - \mathbf{x}_{i-2})}{|\mathbf{x}_i - \mathbf{x}_{i-1}||\mathbf{x}_{i-1} - \mathbf{x}_{i-2}|}\right), \quad (3)$$

where f_1 stands for the distance function from the current position to the goal point, (x^G, y^G) is the goal's coordinates; f_2 stands for the sum of minimum distances to obstacles in a current region and is responsible for "pushback" action, i.e., the agent is recommended to move some distance away from the obstacle for safe movement; f_3 defines the trajectory smoothness by preserving the smooth of the angle between subsequent segments of movement, $\mathbf{x}_i$ is the current position, $\mathbf{x}_{i-1}$ and $\mathbf{x}_{i-2}$ are two previous positions.

To determine the minimum distance to the obstacle, we use the ray concept, i.e., the agent emits four perpendicular scanning rays within the region, after which the points of intersection of the rays with the rectangular obstacle and the corresponding length of the resulting segment are calculated.

The complete pathfinding problem is formulated as minimizing guided function $\mathbf{\Phi}(\mathbf{x}_i)$ which consists of weighted sum of f_1, f_2 and f_3, and these functions provide flexibility of interdependence between functions and tuning capabilities:

$$\mathbf{\Phi}(\mathbf{x}_i) = \omega_1 f_1(\mathbf{x}_i) + \omega_2 f_2(\mathbf{x}_i) + \omega_3 f_3(\mathbf{x}_i) \to \min_{\mathbf{X},\boldsymbol{\omega}}, \quad (4)$$

where $\omega_1, \omega_2, \omega_3 \in \boldsymbol{\omega}$ are weights and $\mathbf{x}_i \in \mathbf{X}$.

Using eq. (4), the definition of pathfinding optimization is outlined as follows: given initial agent's position (x_0, y_0), region size R_0, target coordinates (x^G, y^G) and coordinates of obstacles (x_i^{ob}, y_i^{ob}) inside the current region, find a) the best (sub-optimal) movement position (x_1, y_1) based on the single-objective optimization problem and b) continue this step until the trajectory to the target (x^G, y^G):

$$\mathbf{T} = \left\langle (x_0, y_0), (x_1, y_1), \ldots, (x_n = x^G, y_n = y^G) \right\rangle$$

will be built.

IV. PATHFINDING METHODOLOGY

Using the discussed assumptions and the provided weighted function as our foundation, it is possible to develop a systematic approach to address the pathfinding problem.

After the map being generated, the agent A, initialized in the position (x_0, y_0), operates inside the region R_0 by producing N candidate positions (x_i', y_i'), $i = 1, N$. This step is performed as a standard solution initialization approach of cABC [16] algorithm. Then bounded by region R_0, constraint cABC (which will be cover in the next section) optimization cycle runs predefined number of iterations I.

Let us describe the proposed routine. By starting with some initial position (x_0, y_0), the agent A generates N *possible* positions (x_i', y_i') using the standard initialization cABC phase. The generation region is bounded by square R_0 defined by the attainability of agent sensors, where agent's position is the center point of the square. After the executing some number of iterations I, i.e., performing employed, onlooker and scout bees phases to evolve the positions (x_i', y_i') according to guided function (4) we obtain one best agent's position (x_1, y_1) as the best one among all (x_i', y_i'). If there are obstacles inside the region R_1 the agent should be aware of them and can not move to the position which collides with the obstacle. After moving in (x_1, y_1), the agent again generates N candidate positions and the routine continues until the predefined number of outer iterations. In this step cABC is applied from scratch, assuming the new candidate solutions exist inside the new square region R_1. If the agent finds the target position, then the routine stops. Now, let us briefly outline cABC steps.

In this paper, we use classical constrained ABC algorithm proposed in [14]. According to [14], to optimize (4) we should take in account possible constraints (obstacles) which are represented as rectangular objects. That is, the constrained problem is formulated as follows:

$$\mathbf{\Phi}(\mathbf{x}_i) \to \min_{\mathbf{X},\boldsymbol{\omega}}, \quad (5)$$

$$\mathbf{x}_i \notin Obstacle_j, \forall Obstacle_j \; in \; R_k, \quad (6)$$

where $k \in [0, ..., M]$, M is the maximum number of total iterations, $\mathbf{X}$ is the solution space and $\boldsymbol{\omega}$ is the known vector of weights, which will be discussed further.

To solve this problem, the initial phase and three cABC steps should be applied.

Firstly, we generate N candidate solution inside the R_k by formulae

$$x_{i,j} = x_j^{lb} + \text{rand}(0,1)\left(x_j^{ub} - x_j^{lb}\right), \quad (7)$$

where $i = 1, N$, x_j^{lb} and x_j^{ub} are the lower and upper bounds of each component determined via R_k, $\text{rand}(0,1)$ gives the random value in range [0, 1] and $j = 1, 2$ (because we consider 2D space). Each of solution has a special variable $trial_i$, which indicates how many times this solution has not been improved.

During the employed phase each of candidate solutions is evolved by the following step

$$v_{i,j} = x_{i,j} + \phi\left(x_{i,j} - x_{k,j}\right). \quad (8)$$

Here ϕ is the random value which is chosen from range [-1, 1], $x_{k,j}$ is another neighboring position. To select the better position, a special fitness function should be calculated. This function is given as follows:

$$fit\left(\boldsymbol{v}_i\right) = \begin{bmatrix} 1 + \left|\boldsymbol{\Phi}\left(\boldsymbol{v}_i\right)\right| : \boldsymbol{\Phi}\left(\boldsymbol{v}_i\right) \geq 0, \\ 1/\left(1 + \boldsymbol{\Phi}\left(\boldsymbol{v}_i\right)\right) : \boldsymbol{\Phi}\left(\boldsymbol{v}_i\right) < 0 \end{bmatrix}. \quad (9)$$

If value of this function for $\boldsymbol{v}_i$ is larger than that for $\mathbf{x}_i$, and both solutions are feasible according to obstacles, then $\mathbf{x}_i \leftarrow \mathbf{v}_i$ and $trial_i \leftarrow 0$. If position $\boldsymbol{v}_i$ is infeasible, but $\mathbf{x}_i$ is feasible, then $\mathbf{x}_i$ remains the same and $trial_i$ should be incremented. If both solutions are infeasible, than we choose that solution which violates less constraints/obstacles (if it is not $\boldsymbol{v}_i$, then $trial_i += 1$).

During the onlooker phase, firstly the probability score for each solution is calculated as

$$p\left(\mathbf{x}_i\right) = \begin{cases} 0.5 + \dfrac{0.5\, fit\left(\mathbf{x}_i\right)}{\sum_j fit\left(\mathbf{x}_j\right)}, if\ \mathbf{x}_i\ is\ feasible \\ 0.5\left(1 - \dfrac{violation\left(\mathbf{x}_i\right)}{\sum_j violation\left(\mathbf{x}_j\right)}\right), else, \end{cases} \quad (10)$$

where $violation\left(\mathbf{x}_i\right)$ indicates how many obstacles/constraints $\mathbf{x}_i$ does not satisfy. Once the probabilities have been calculated, then each of N solutions can be improved based on the condition $rand\left(0,1\right) < p\left(\boldsymbol{x}_i\right)$. The improvement loop which incorporates employed phase process the solution by solution until the improvement counter equals N. This phase helps to strengthen neighbors search.

The final step is a scout phase. Each variable $trial_i$ is checked if it equals some predefined limit. If so, the corresponding solution must be reinitialized using (7). Generally, the limit is set to $2*N$ or $2*N//2$.

V. DIRECT PATHFINDING PROBLEM

To study the proposed approach, we conducted several numerical experiments based on two movement environment (Fig. 1, 2) and guided function $\boldsymbol{\Phi}\left(\mathbf{x}_i\right)$ (4). Firstly, we have investigated the impact of two weighted functions f_1 and f_2 on pathfinding capabilities. For these experiments the number of cABC runs during each iteration equals to 8. Since cABC is stochastic in nature and may generate different solutions for each run, we performed 20 experiments, $M = 200$. The results are outlined in Tables 1 for the first map (Fig. 1).

Table 1

Pathfinding results for f_1 and f_2 (map # 1)

ω_1	ω_2	*target*	*avgPath*	*stdPath*	*avgT*	*stdT*
0.7	0.3	5	113	31	1.9	0.98
0.9	0.1	17	68	0.8	0.4	0.06
1.0	0.0	16	66	1.9	0.35	0.05

Here *target* stands for the number of times the agent reaches the goal point, *avgP* is the average path lengths, *stdP* is the standard deviation of path lengths, *avgT* is the average execution time and *stdT* is the standard deviation of path execution time.

What can be learned from these results is that for the map # 1 the increase of the first weight ω_1 improves pathfinding capabilities while the smaller values do not help to reach the target. Also, decreasing weight ω_2 down to zero, i.e., elimination of the impact of "pushback" provides better search due the large number of obstacles. For the second map, such a weight configuration does not produce any good results, i.e., the agent did not reach the target during all experimental runs.

Table 2

Pathfinding results for f_1 and f_3 (map # 1)

ω_1	ω_3	*target*	*avgP*	*stdP*	*avgT*	*stdT*
0.5	0.05	6	141	32	0.92	0.3
0.6	0.05	5	201	30	1.5	0.3
0.7	0.03	17	118	44	0.96	0.6
0.7	0.1	5	153	53	1.0	0.53
0.8	0.05	12	155	57	1.1	0.6
0.9	0.1	3	180	58	1.2	0.6
0.9	0.05	12	155	51	1.23	0.7

Table 3

Pathfinding results for f_1 and f_3 (map # 2)

ω_1	ω_3	*target*	*avgP*	*stdP*	*avgT*	*stdT*
0.6	0.01	1	94.6	0.0	0.27	0.0
0.7	0.03	14	166	53	0.5	0.2
0.7	0.1	10	178	61	0.36	0.2
0.8	0.05	13	179	46	0.4	0.1
0.9	0.1	8	188	83	0.4	0.2
0.9	0.05	10	154	35	0.35	0.1
1.0	0.05	14	173	70	0.4	0.2

Results for incorporating weighted f_1 and f_3 are shown in Tables 2, 3 for each map, respectively.

A combination of f_1 and f_3 functions for the first map produces, in general, better results than f_1 and f_2. Such combination not only leverages impact of the first function, but also adds influence regarding the sharpness of the angle of rotation of the agent when generating the next position. The best result is obtained for $\omega_1 = 0.7$, $\omega_3 = 0.03$ applied to the map # 1 and is visualized in Fig. 4.

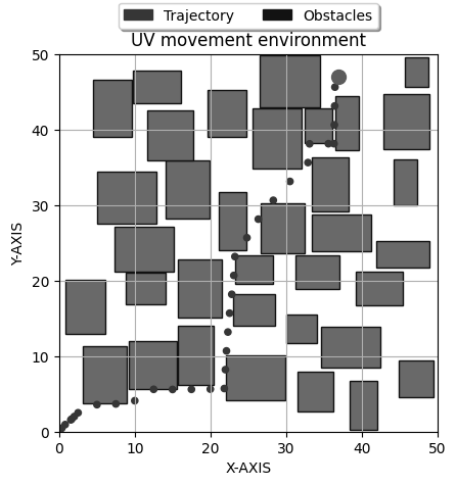


Fig. 4. Path trajectory for $\omega_1 = 0.7$, $\omega_3 = 0.03$, *map # 1*

The main point here is that weights $\boldsymbol{\omega}$ play a crucial role in guided function (4) and are of considerable interest related to tuning the algorithm.

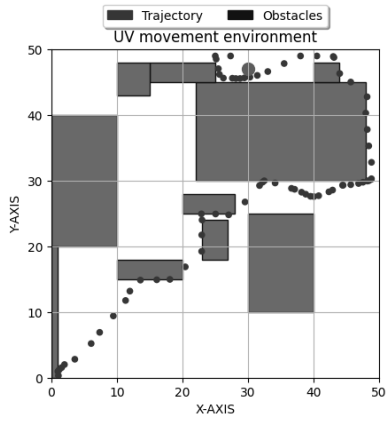


Fig. 5. Path trajectory for $\omega_1 = 0.7$, $\omega_3 = 0.03$, *map # 2*

The last conducted experiment relates to the involvement of all three weighted functions f_1, f_2 and f_3 (Tables 4, 5). Most of the weights chosen for map # 1 turn into adequate results, but the best one obtained for weights $\omega_1 = 1.0$, $\omega_2 = 0.2$ and $\omega_3 = 0.03$ due to lowest average path value.

The second map # 2 is more complicated, because to reach the target agent should worsen guided function value for f_1 which stands for a target attraction. Therefore, the value *target* in average lower than *target* calculated for the map # 1. The best weight obtained for map # 2 is $\omega_1 = 0.8$, $\omega_2 = 0.1$, $\omega_3 = 0.03$ and results in *target=16.* Visualized tracks are presented in Fig. 6, 7 for each map, respectively.

The interesting behavior of the agent is shown in Fig. 7. The proposed approach allows reaching the target even in case when the agent traps in corner/local minima. To get out of the trap, the agent should generally worsen target function f_1. This behavior becomes possible because the other two functions '*balance*' the first one. Thus, the pressure of the first function on the function (4) becomes smaller.

Table 4

Pathfinding results for f_1, f_2 and f_3 (map # 1)

ω_1	ω_2	ω_3	*target*	*avgP*	*stdP*	*avgT*	*stdT*
0.8	0.1	0.05	16	146	39	1.1	0.5
0.8	0.2	0.1	9	186	50	1.4	0.5
0.8	0.3	0.2	7	164	30	1.2	0.3
0.9	0.2	0.05	14	171	41	1.6	0.5
0.9	0.1	0.1	6	133	25	0.95	0.3
0.9	0.3	0.03	17	105	30	0.9	0.4
1.0	0.1	0.1	7	168	48	1.2	0.5
1.0	0.2	0.03	17	90	30	0.7	0.4
1.2	0.1	0.05	17	132	36	1.1	0.5

Table 5

Pathfinding results for f_1, f_2 and f_3 (map # 2)

ω_1	ω_2	ω_3	*target*	*avgP*	*stdP*	*avgT*	stdT
0.8	0.1	0.05	11	141	41	0.8	0.4
0.8	0.2	0.05	5	195	50	1.4	0.4
0.8	0.3	0.05	10	203	53	1.43	0.4
0.9	0.2	0.05	14	165	58	1.2	0.5
0.9	0.1	0.03	12	153	55	1.3	0.6
0.8	0.1	0.03	16	183	54	1.5	0.6
0.7	0.1	0.03	14	166	60	1.3	0.6
1.0	0.1	0.05	14	187	55	1.4	0.5
1.2	0.1	0.05	13	146	34	1.1	0.3

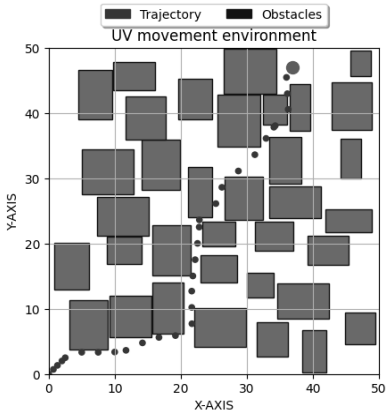


Fig. 6. Path trajectory for $\omega_1 = 1.0$, $\omega_2 = 0.2$ *and* $\omega_3 = 0.03$

Another interesting direction is to study the impact of local cABC iterations inside each independent region R_k. Since this is not clear in advance how many iterations must be executed to get the best result, we have conducted several numerical experiments presented in Tables 6, 7.

Judging from Table 6 (map # 1) it is seen that the best *target* value was obtained when 2 iteration of cABC

were executed inside each independent R_k due to the shorter path. What is also interesting is that even if a simple greedy search on randomly generated solutions is used without any run of cABC, it is still possible to reach the target.

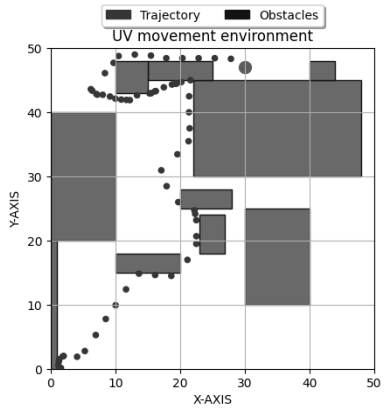


Fig. 7. Path trajectory for $\omega_1 = 0.8$, $\omega_2 = 0.1$ *and* $\omega_3 = 0.03$

Table 6

Local number of cABC iterations (map # 1)

Inner iter.	*avgP*	*stdP*	*target*
0	34	137	30
1	32	120	28
2	32	126	30
5	32	125	26
8	24	132	20
10	24	116	12

For map # 2 things are getting more complicated, because this map is more advanced. The best results are obtained for five cABC iterations which produced the smallest average path length.

Table 7

Local number of cABC iterations (map # 2)

Inner iter.	*avgP*	*stdP*	*target*
0	205	72	13
1	200	60	15
2	202	50	14
3	146	60	17
5	137	36	19
8	142	37	14

The reason why greedy search can sometimes produce decent results is the fact that all regions R_k *are independent* and optimal trajectory point for R_k cannot be any good for the next R_{k+1}. This consequence leads us to the construction of the path with memory, when the agent will remember the regions where he has already been. And the generation of a new position will take place in an area consisting of several regions. Nevertheless, for now the proposed approach is still quite effective in a partially unknown environment with obstacles.

VI. INVERSE PATHFINDING PROBLEM

The last question that arises in this research is how to determine the proper set of weights for the guided function (4).

In this paper, we consider this issue as the formulation of an inverse pathfinding problem. Let us study the possibility of calculating the weights based on the solution of the inverse problem (11):

$$\boldsymbol{\omega}^* = \arg\min_{\boldsymbol{\omega}} \frac{avgP}{1+target} \qquad (11),$$

where *avgP* is the average path lengths obtained during K experiments, *target* indicates how many times of K the agent has reached the target, $\boldsymbol{\omega}^*$ is the optimal weight vector. The inverse problem (11) can be solved by running the direct pathfinding routine (5), (6) K times and gathering the respective *avgP* and *target* values. The essence of (11) function is the minimization of average path length in accordance with the best *target* value. The number of cABC iterations was set equal to 3 in order to reduce computational efforts and $K = 30$.

To solve this problem for each map, we used a greedy brute-force algorithm applied to the predefined grid of possible weights.

For map # 1 we got some sub-optimal results $\boldsymbol{\omega}^* = [1.05, 0.2, 0.03]$, *target=30, avgP=108* and *stdP=28.* The path trajectory for $\boldsymbol{\omega}^* = [1.05, 0.2, 0.03]$ is shown in Fig. 8.

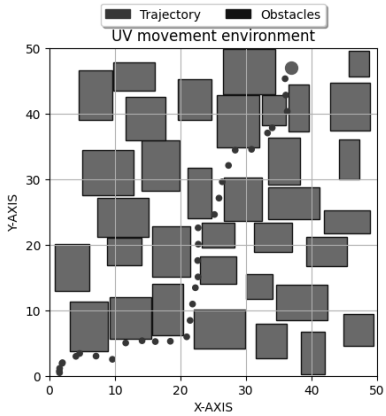


Fig. 8. Path trajectory for $\omega_1 = 1.05$, $\omega_2 = 0.2$ *and* $\omega_3 = 0.03$, *map # 1*

For map # 2 we obtained another sub-optimal parameters $\boldsymbol{\omega}^* = [0.73, 0.09, 0.03]$, *target=22, avgP=147* and *stdP=53.* The path trajectory of for $\boldsymbol{\omega}^* = [0.73, 0.09, 0.03]$ is presented in Fig. 9.

It is evident that more sophisticated map # 2 leaves its mark on the optimization results. Hence, in general, to get the stable values for $\boldsymbol{\omega}^*$ during solving the inverse problem requires more experiments than $K = 30$, which

in turn leads to more exhaustive computations. Nevertheless, this is a good starting point for future numerical experiments.

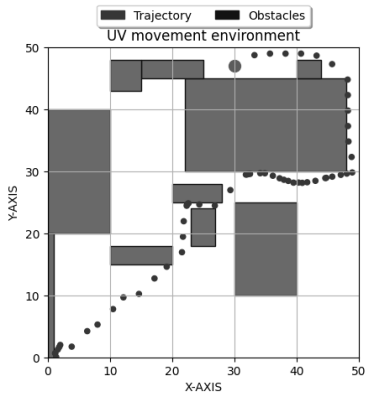


Fig. 9. Path trajectory for $\omega_1 = 0.73$, $\omega_2 = 0.09$ *and* $\omega_3 = 0.03$, *map # 2*

VII. CONCLUSION

In this paper, we propose and study the approach to pathfinding problems in the dense 2D environment which is a partially unknown in advance.

To validate the proposed scheme, we have generated two maps of different complexity. During the numerical experiments, we have considered different weight and function combinations as well as the influence of cABC iterations inside each bounded region. The results led us to several essential conclusions: 1) weights that are used to balance the guided function are crucial and radically affect search capabilities, which necessitates a more thorough study of this issue; 2) since the search is bounded by independent regions at each iteration, the agent does not incorporate memory of previously visited areas; hence, in future work, this improvement will be widely studied; 3) the problem of proper weights that can be applied to the arbitrary maps is relevant in the course of the conducted research; to set up the starting point for this issue, we proposed a simple approach based on solving the inverse problem, which requires the decent number of runs of the direct problem; 4) the latter problem can be solved using certain rules and adaptive weights to avoid solving a long-running inverse problem; 5) machine learning techniques may be used to transform this problem into a reinforcement learning stack.

References

[1] Yanmaz, E., Yahyanejad, S., Rinner, B., Hellwagner, H., & Bettstetter, C. (2018). Drone networks: Communications, coordination, and sensing. *Ad Hoc Networks*, *68*, 1-15. DOI: https://doi.org/10.1016/j.adhoc.2017.09.001.

[2] Gad, A. G. (2022). Particle swarm optimization algorithm and its applications: a systematic review. *Archives of computational methods in engineering, 29(5),* 2531-2561. DOI: https://doi.org/10.1007/s11831-021-09694-4.

[3] Abu-Mouti, F. S., & El-Hawary, M. E. (2012, March). Overview of Artificial Bee Colony (ABC) algorithm and its applications. In *2012 IEEE International Systems Conference SysCon 2012* (pp. 1-6). IEEE. DOI: https://doi.org/10.1109/syscon.2012.6189539.

[4] Dorigo, M., & Blum, C. (2005). Ant colony optimization theory: A survey. *Theoretical computer science*, *344*(2-3), 243-278. DOI: https://doi.org/10.1016/j.tcs.2005.05.020.

[5] Fister, I., Fister Jr, I., Yang, X. S., & Brest, J. (2013). A comprehensive review of firefly algorithms. *Swarm and evolutionary computation*, *13*, 34-46. DOI: https://doi.org/10.1016/j.swevo.2013.06.001.

[6] Bhattacharjee, P., Rakshit, P., Goswami, I., Konar, A., & Nagar, A. K. (2011, October). Multi-robot path-planning using artificial bee colony optimization algorithm. In *2011 Third World Congress on Nature and Biologically Inspired Computing* (pp. 219-224). IEEE. DOI: https://doi.org/10.1109/nabic.2011.6089601.

[7] Contreras-Cruz, M. A., Ayala-Ramirez, V., & Hernandez-Belmonte, U. H. (2015). Mobile robot path planning using artificial bee colony and evolutionary programming. *Applied Soft Computing*, *30*, 319-328. DOI: https://doi.org/10.1016/j.asoc.2015.01.067.

[8] Contreras-Cruz, M. A., Lopez-Perez, J. J., & Ayala-Ramirez, V. (2017, June). Distributed path planning for multi-robot teams based on artificial bee colony. In *2017 IEEE congress on evolutionary computation (CEC)* (pp. 541-548). IEEE. DOI: https://doi.org/10.1109/cec.2017.7969358.

[9] Liang, J. H., & Lee, C. H. (2015). Efficient collision-free path-planning of multiple mobile robots system using efficient artificial bee colony algorithm. *Advances in Engineering Software*, *79*, 47-56. DOI: https://doi.org/10.1016/j.advengsoft.2014.09.006.

[10] Nayyar, A., Nguyen, N. G., Kumari, R., & Kumar, S. (2020). Robot path planning using modified artificial bee colony algorithm. In *Frontiers in Intelligent Computing: Theory and Applications: Proceedings of the 7th International Conference on FICTA (2018), Volume 2* (pp. 25-36). Springer Singapore. DOI: https://doi.org/10.1007/978-981-13-9920-6_3.

[11] Kumar, S., & Sikander, A. (2022). Optimum mobile robot path planning using improved artificial bee colony algorithm and evolutionary programming. *Arabian Journal for Science and Engineering*, *47*(3), 3519-3539. DOI: https://doi.org/10.1007/s13369-021-06326-8.

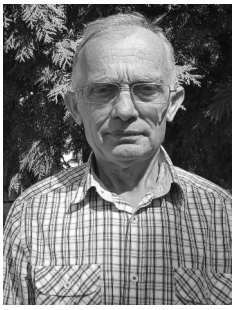
[12] Kumar, S., & Sikander, A. (2024). A novel hybrid framework for single and multi-robot path planning in a complex industrial environment. *Journal of Intelligent Manufacturing*, *35*(2), 587-612. DOI: https://doi.org/10.1007/s10845-022-02056-2.

[13] Faridi, A. Q., Sharma, S., Shukla, A., Tiwari, R., & Dhar, J. (2018). Multi-robot multi-target dynamic path planning using artificial bee colony and evolutionary programming in unknown environment. *Intelligent Service Robotics*, *11*, 171-186. DOI: https://doi.org/10.1007/s11370-017-0244-7.

[14] Karaboga, D., & Akay, B. (2011). A modified artificial bee colony (ABC) algorithm for constrained optimization problems. *Applied soft computing*, *11*(3), 3021-3031. DOI: https://doi.org/10.1016/j.asoc.2010.12.001.

Oleh Sinkevych, PhD, was born in Lviv, Ukraine, in 1988. Starting from 2023, he has been working as Associate Professor at the Faculty of Electronics and Computer Technologies of Ivan Franko National University of Lviv. His research interests encompass machine learning, natural language processing, swarm AI algorithms, numerical optimization, and metaheuristics.

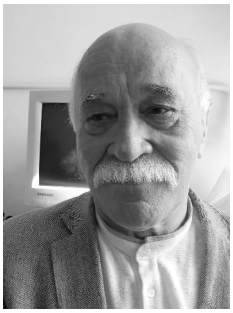
Yaroslav Boyko, PhD, was born in Lviv region, Ukraine, in 1967. Since 2017, he has been working as Asssociate Professor at the Faculty of Electronics and Computer Technologies of Ivan Franko National University of Lviv. His research interests encompass Internet of Things and Fog/Edge computing.
.



Bohdan Sokolovskyy, PhD, was born in Kulykiv, Lviv region, Ukraine, in 1950. Since 2014, he has been working as Associate Professor at the Faculty of Electronics and Computer Technologies of Ivan Franko National University of Lviv. His research interests encompass computer modeling of nonuniform semiconductor structures and methods of stochastic optimization.

Mykhailo Pavlyk, PhD, was born in Davydiv, Lviv district, Lviv region, in 1986. From 2023 he has been working as Associate Professor at the Electronics and Computer Technologies faculty of Ivan Franko National University of Lviv. His research interests include machine learning, embedded machine learning and embedded systems.

Oleh Yarosh was born on August 26, 2003, in the city of Dobromil, Ukraine. In 2020, he entered the faculty of electronics and computer technologies at Ivan Franko University of Lviv. His research interests include machine learning, embedded machine learning and embedded systems. Now he works in swarm AI subfield of machine learning.
.



Oleksandr Futey was born on May 10, 1955, in Buchach, Ternopil region. Since 2013 he has been working as Assistant Professor at the Department of Radioelectronic and Computer Systems of the Faculty of Electronics and Computer Technologies. His research interests include electronics and embedded machine learning.