Bat Algorithm Swarm Robotics Approach for Dual Non-cooperative Search with Self-centered Mode

Patricia Suárez[1], Akemi Gálvez[1,2], Iztok Fister[3], Iztok Fister Jr.[3], Eneko Osaba[4], Javier Del Ser[4,5,6], and Andrés Iglesias[1,2(✉)]

[1] University of Cantabria, Avenida de los Castros s/n, 39005 Santander, Spain
iglesias@unican.es
[2] Toho University, 2-2-1 Miyama, Funabashi 274-8510, Japan
[3] University of Maribor, Smetanova ulica 17, 2000 Maribor, Slovenia
[4] TECNALIA, Derio, Spain
[5] University of the Basque Country (UPV/EHU), Bilbao, Spain
[6] Basque Center for Applied Mathematics (BCAM), Bilbao, Spain

Abstract. This paper presents a swarm robotics approach for dual non-cooperative search, where two robotic swarms are deployed within a map with the goal to find their own target point, placed at an unknown location of the map. We consider the self-centered mode, in which each swarm tries to solve its own goals with no consideration to any other factor external to the swarm. This problem, barely studied so far in the literature, is solved by applying a popular swarm intelligence method called bat algorithm, adapted to this problem. Five videos show some of the behavioral patterns found in our computational experiments.

1 Introduction

Swarm robotics is attracting a lot of interest because of its potential advantages for several tasks [1]. For instance, robotic swarms are well suited for navigation in indoor environments, as a swarm of simple interconnected mobile robots has greater exploratory capacity than a single sophisticated robot. Additional benefits of the swarm are greater flexibility, adaptability, and robustness. Also, swarm robotics methods are relatively simple to understand and implement, and are quite affordable in terms of the required budget and computing resources.

The most common case of swarm robotics is the cooperative mode, with several robotic units working together to accomplish a common task. Much less attention is given so far to the non-cooperative mode. In this paper we are particularly interested in a non-cooperative scheme that we call *egotist* or *self-centered* mode. Under this regime, each swarm tries to solve its own goals with little (or none at all) consideration to any other factor external to the swarm. In this work, we consider two swarms of robotic units deployed within the same spatial environment. Each swarm is assigned the task to reach its own target point, placed in an unknown location of a complex labyrinth with narrow corridors and several

H. Yin et al. (Eds.): IDEAL 2018, LNCS 11315, pp. 201–209, 2018.
https://doi.org/10.1007/978-3-030-03496-2_23

dead ends, forcing the robots to turn around to escape from these blind alleys. It is assumed that the geometry of the environment is completely unknown to the robots. Finding these unknown target points requires the robots of both swarms to perform exploration of the environment, hence interfering each other during motion owing to potential intra- and inter-swarm collisions. Under these conditions, the robots have to navigate in a highly dynamic environment where all moving robots are additional obstacles to avoid. This brings the possibility of potentially conflicting goals for the swarms, for instance when the robots are forced to move on the same corridors but in opposite directions. In our approach, there is no centralized control of the swarm, so the robot decision-making is completely autonomous and each robot takes decisions by itself without any external order from a central server of any other robot.

Similar to [2], in this paper we consider robotic units based on ultrasound sensors and whose internal functioning is based on a popular swarm intelligence method: the *bat algorithm*. To this aim, we developed a computational simulation framework that replicates very accurately all features and functionalities of the real robots and the real environment, including the physics of the process (gravity, friction, motion, collisions), visual appearance (cameras, texturing, materials), sensors (ultrasound, spatial orientation, cycle-based time clock, computing unit emulation, bluetooth), allowing both graphical and textual output and providing support for additional components and accurate interaction between the robots and with the environment.

The structure of this paper is as follows: firstly, we describe the bat algorithm, its basic rules and its pseudocode. Then, our approach for the dual non-cooperative search problem for the self-centered mode posed in this paper is described. Some experimental results are then briefly discussed. The paper closes with the main conclusions and some plans for future work in the field.

2 The Bat Algorithm

The *bat algorithm* is a bio-inspired swarm intelligence algorithm originally proposed by Xin-She Yang in 2010 to solve optimization problems [4–6]. The algorithm is based on the echolocation behavior of microbats, which use a type of sonar called *echolocation*. The idealization of this method is as follows:

1. Bats use echolocation to sense distance and distinguish between food, prey and background barriers.
2. Each virtual bat flies randomly with a velocity $\mathbf{v}_i$ at position (solution) $\mathbf{x}_i$ with a fixed frequency f_{min}, varying wavelength λ and loudness A_0 to search for prey. As it searches and finds its prey, it changes wavelength (or frequency) of their emitted pulses and adjust the rate of pulse emission r, depending on the proximity of the target.
3. It is assumed that the loudness will vary from an (initially large and positive) value A_0 to a minimum constant value A_{min}.

Require: (Initial Parameters)
Population size: $\mathcal{P}$; Maximum number of generations: $\mathcal{G}_{max}$; Loudness: $\mathcal{A}$
Pulse rate: r ; Maximum frequency: f_{max} ; Dimension of the problem: d
Objective function: $\phi(\mathbf{x})$, with $\mathbf{x} = (x_1, \ldots, x_d)^T$; Random number: $\theta \in U(0, 1)$
$g \leftarrow 0$
Initialize the bat population $\mathbf{x}_i$ and $\mathbf{v}_i$, $(i = 1, \ldots, n)$
Define pulse frequency f_i at $\mathbf{x}_i$
Initialize pulse rates r_i and loudness $\mathcal{A}_i$
while $g < \mathcal{G}_{max}$ **do**
  for $i = 1$ **to** $\mathcal{P}$ **do**
    Generate new solutions by using eqns. (1)-(3)
    if $\theta > r_i$ **then**
      $\mathbf{s}^{best} \leftarrow \mathbf{s}^g$ //select the best current solution
      $\mathbf{ls}^{best} \leftarrow \mathbf{ls}^g$ //generate a local solution around $\mathbf{s}^{best}$
    end if
    Generate a new solution by local random walk
    if $\theta < \mathcal{A}_i$ *and* $\phi(\mathbf{x_i}) < \phi(\mathbf{x}^*)$ **then**
      Accept new solutions, increase r_i and decrease $\mathcal{A}_i$
    end if
  end for
  $g \leftarrow g + 1$
end while
Rank the bats and find current best $\mathbf{x}^*$
return $\mathbf{x}^*$

Algorithm 1. Bat algorithm pseudocode

Some additional assumptions are advisable for further efficiency. For instance, we assume that the frequency f evolves on a bounded interval $[f_{min}, f_{max}]$. This means that the wavelength λ is also bounded, because f and λ are related to each other by the fact that the product $\lambda.f$ is constant. For practical reasons, it is also convenient that the largest wavelength is chosen such that it is comparable to the size of the domain of interest (the search space for optimization problems). For simplicity, we can assume that $f_{min} = 0$, so $f \in [0, f_{max}]$. The rate of pulse can simply be in the range $r \in [0, 1]$, where 0 means no pulses at all, and 1 means the maximum rate of pulse emission. With these idealized rules indicated above, the basic pseudo-code of the bat algorithm is shown in Algorithm 1. Basically, the algorithm considers an initial population of $\mathcal{P}$ individuals (bats). Each bat, representing a potential solution of the optimization problem, has a location $\mathbf{x}_i$ and velocity $\mathbf{v}_i$. The algorithm initializes these variables with random values within the search space. Then, the pulse frequency, pulse rate, and loudness are computed for each individual bat. Then, the swarm evolves in a discrete way over generations, like time instances until the maximum number of generations, $\mathcal{G}_{max}$, is reached. For each generation g and each bat, new frequency, location and velocity are computed according to the following evolution equations:

$$f_i^g = f_{min}^g + \beta(f_{max}^g - f_{min}^g) \quad (1)$$

$$\mathbf{v}_i^g = \mathbf{v}_i^{g-1} + [\mathbf{x}_i^{g-1} - \mathbf{x}^*]\, f_i^g \quad (2)$$

$$\mathbf{x}_i^g = \mathbf{x}_i^{g-1} + \mathbf{v}_i^g \quad (3)$$

where $\beta \in [0, 1]$ follows the random uniform distribution, and $\mathbf{x}^*$ represents the current global best location (solution), which is obtained through evaluation of the objective function at all bats and ranking of their fitness values. The superscript $(.)^g$ is used to denote the current generation g. The best current solution and a local solution around it are probabilistically selected according to some given criteria. Then, search is intensified by a local random walk. For this local search, once a solution is selected among the current best solutions, it is perturbed locally through a random walk of the form: $\mathbf{x}_{new} = \mathbf{x}_{old} + \epsilon\mathcal{A}^g$, where ϵ is a uniform random number on $[-1, 1]$ and $\mathcal{A}^g = <\mathcal{A}_i^g>$, is the average loudness of all the bats at generation g. If the new solution achieved is better than the previous best one, it is probabilistically accepted depending on the value of the loudness. In that case, the algorithm increases the pulse rate and decreases the loudness (lines 13–16). This process is repeated for the given number of generations. In general, the loudness decreases once a bat finds its prey (in our analogy, once a new best solution is found), while the rate of pulse emission decreases. For simplicity, the following values are commonly used: $\mathcal{A}_0 = 1$ and $\mathcal{A}_{min} = 0$, assuming that this latter value means that a bat has found the prey and temporarily stop emitting any sound. The evolution rules for loudness and pulse rate are as: $\mathcal{A}_i^{g+1} = \alpha\mathcal{A}_i^g$ and $r_i^{g+1} = r_i^0[1 - exp(-\gamma g)]$ where α and γ are constants. Note that for any $0 < \alpha < 1$ and any $\gamma > 0$ we have: $\mathcal{A}_i^g \to 0$, $r_i^g \to r_i^0$ as $g \to \infty$. Generally, each bat should have different values for loudness and pulse emission rate, which can be achieved by randomization. To this aim, we can take an initial loudness $\mathcal{A}_i^0 \in (0, 2)$ while r_i^0 can be any value in the interval $[0, 1]$. Loudness and emission rates will be updated only if the new solutions are improved, an indication that the bats are moving towards the optimal solution.

3 Bat Algorithm Method for Robotic Swarms

In this work we consider two robotic swarms $\mathcal{S}_1$ and $\mathcal{S}_2$ comprised by a set of μ and ν robotic units $\mathcal{S}_1 = \{\mathcal{R}_1^i\}_{i=1,\ldots,\mu}$, $\mathcal{S}_2 = \{\mathcal{R}_2^j\}_{j=1,\ldots,\nu}$, respectively. For simplicity, we assume that $\mu = \nu$ and that all robotic units are functionally identical, i.e., $\mathcal{R}_1^i = \mathcal{R}_2^j$, $\forall i, j$. For visualization purposes, the robots in $\mathcal{S}_1$ and $\mathcal{S}_2$ are displayed graphically in red and yellow, respectively. They are deployed within a 3D synthetic labyrinth, $\mathbf{\Omega} \subset \mathbb{R}^3$ with a complex geometrical configuration (unknown to the robots and shown in Fig. 1) to perform dynamic exploration. The figure is split into two parts for better visualization, corresponding to the side view (left) and top view (right). As the reader can see, the scene consists of a large collection of cardboard boxes arranged in a grid-like structure and forming challenging structures for the robots such as corridors, dead ends, bifurcations and T-junctions to simulate the walls and corridors of a labyrinth. Figure 1 also

shows a set of 10 robotic units for each swarm, scattered throughout the environment. The goal of each swarm $\mathcal{S}_k$ is to find a static target point $\mathbf{\Phi}_k$ $(k = 1, 2)$, placed in a certain (unknown) location of the environment. We assume that $||\mathbf{\Phi}_1 - \mathbf{\Phi}_2|| > \delta$ for a certain threshold value $\delta > 0$, meaning that both target points are not very close to each other so as to broaden the spectrum of possible interactions between the swarms. They are represented in Fig. 1 by two spherical-shaped points of light in red and yellow for $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. Target $\mathbf{\Phi}_1$ is located inside a room at a relatively accessible location, while target $\mathbf{\Phi}_2$ is placed at the deepest part of the labyrinth (the upper left corner in top view). The scene also includes many dead ends to create a challenging environment for the robotic swarms.

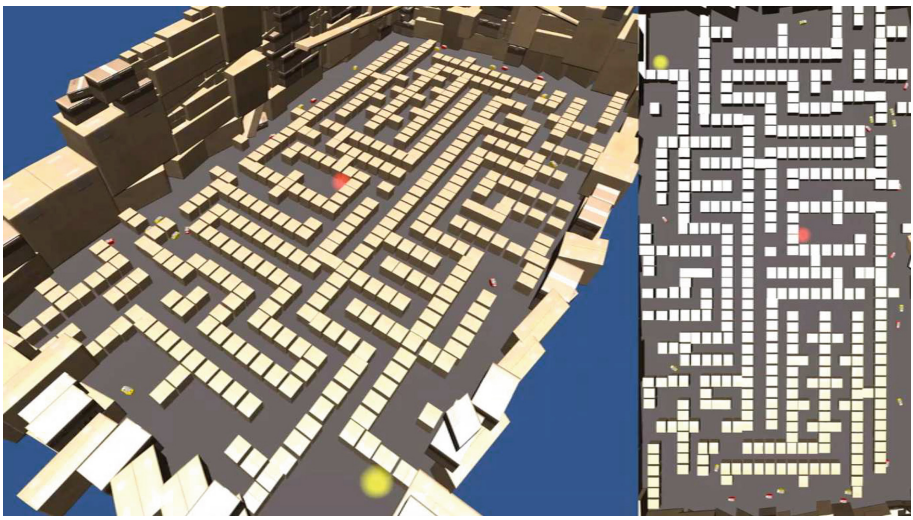

Fig. 1. Graphical representation of the labyrinth: side view (left); top view (right). The image corresponds to the initialization step of our method, when two robotics swarms are deployed at random positions in the outermost parts of the map. (Color figure online)

In our approach, each robot moves autonomously, according to the current values of its fitness function and its own parameters. To this aim, each virtual robot $\mathcal{R}_k^i$ is mathematically described by a vector $\mathbf{\Xi}_k^{i,j} = \left\{\varphi_k^{i,j}, \mathbf{x}_k^{i,j}, \mathbf{v}_k^{i,j}\right\}$, where $\varphi_k^{i,j}$, $\mathbf{x}_k^{i,j} = (x_k^{i,j}, y_k^{i,j})$ and $\mathbf{v}_k^{i,j} = (v_{k,x}^{i,j}, v_{k,y}^{i,j})$ represent the fitness value, position, and velocity at time instance j, respectively. Note here that, although the environment is a 3D world, we consider the case of mobile walking robots, therefore moving on a two-dimensional map $\mathcal{M} = \mathbf{\Omega}|_{z=0}$. The robots are deployed at initial random positions $\mathbf{x}_k^{i,0}$ and with random but bounded velocities $\mathbf{v}_k^{i,0}$ so that the robots move strictly within the map $\mathcal{M}$. Moreover, the robots are initialized in the outermost parts of the map to avoid falling down very near to the target.

For the robot motion, we assume that the two-dimensional map $\mathcal{M}$ is described by a tessellation of convex polygons $\mathcal{T}_{\mathcal{M}}$. Then, we consider the set $\mathcal{N}_{\mathcal{M}} \subset \mathcal{T}_{\mathcal{M}}$ (called the *navigation mesh*) comprised by all polygons that are fully traversable by the robots. At time j the fitness function $\varphi_k^{i,j}$ can be defined as the distance between the current position $\mathbf{x}_k^{i,j}$ and the target point $\mathbf{\Phi}_k$, measured *on* $\mathcal{N}_{\mathcal{M}}$ as $\varphi_k^{i,j} = ||\mathbf{x}_k^{i,j} - \mathbf{\Phi}_k||_{\mathcal{N}_{\mathcal{M}}}$. In this case, our 2D robot navigation can be seen as an optimization problem, that of minimizing the value of $\varphi_k^{i,j}$, $\forall i, j, k$, which represents the distance from the current location to the target point. This problem is solved through the bat algorithm described above. About the parameter tuning, our choice has been fully empirical, based on computer simulations for different parameter values. We consider a population size of 10 robots for each swarm, as larger values make the labyrinth too populated and increase the number of collisions among robots. Initial and minimum loudness and parameter α are set to 0.5, 0, and 0.6, respectively. We also set the initial pulse rate and parameter γ to 0.5 and 0.4, respectively. However, our results do not change significantly when varying these values. All executions are performed until all robots reach their target point. Our method is implemented in *Unity 5* on a 3.8 GHz quad-core Intel Core i5, with 16 GB of DDR3 memory, and a graphical card AMD RX580 with 8 GB VRAM. All programming code in this paper has been created in *JavaScript* using the *Visual Studio* programming framework.

4 Experimental Results

The proposed method has been tested through several computational experiments. Five of them are recorded in five MPEG-4 videos (labelled as *Video1* to *Video5*) (generated as accompanying material of this paper and stored as a single 18.8 MB ZIP file publicly available at [3]), selected to show different behavioral patterns obtained with our method and corresponding to as many random initial locations of the robots.

Video 1 shows that the robots are initially wandering to explore the environment, searching for their target points. As explained above, at every iteration each robot receives the value of its fitness as well as the position of the best member of the swarm. After a few iterations some robots are able to find a path that improves their fitness (this fact is clearly visible for the red robots in lower right part of top view) and then they move to follow it. One of the members of the red swarm is successfully approaching its target so, at a certain iteration, it becomes the best of the swarm. At this point, the other members of the swarm try to follow it according to Eqs. (1)–(3). This behavioral pattern is not general, however, as other robots exhibit different behaviors. For instance, we can also see an example of a grouping pattern, illustrated by several yellow robots gathering in the upper part of the map in top view. Also, we find a case of collision between robots in central area of the scene, where two red robots trying to move to the south to approach its best, and two yellow robots trying to gather with other members of their swarm collide each other, getting trapped in a narrow corridor for a while. Note also that, although the yellow robot on the right of this group

does not visually collapse with any other when moving ahead and hence might advance, it keeps idle because its ultrasound sensor detects the neighbor robots even if they are not exactly in front. When a possible collision is detected, the robots involved stop moving and start yawing slightly to left and right trying to escape from the potentially colliding area. But since the yawing motion angle is small, this situation can last for a while, as it actually happens in this video for the four robots in this group, until one of the yellow robots is able to turn around and moves in the opposite direction to its original trajectory. In its movement, this yellow robot attracts its fellow teammate. Once freed, the two red robots follow their original plan and move to the south as well. In the meanwhile, all other red robots reached their target point and all other yellow robots gathered in north part of the map and try to find a passage to advance towards their target point. Eventually, all robots find a path to their target points, and the simulation ends successfully.

Some other behavioral patterns can be seen in the other videos. For instance, *Video 2* shows an interesting example of the ability of the robots to escape from dead ends. In the video, several yellow robots gather in the upper part of the map in top view but in their exploration of the environment they move to a different corridor, getting trapped in a dead end. Unable to advance, the robots in the rear turn around and after some iterations all robots get rid of the alley, and start moving to the previous corridor in the north for further exploration. However, other robots of the group follow the opposite direction getting closer to the target point. As their fitness value improves, they attract the other members of the swarm in their way to the target point. This kind of wandering behavior where the robots move back and forth apparently in erratic fashion, also shown in *Video 3* to *Video 5*, can be explained by the fact that the robots do not have any information about the environment, so they explore it according to their fitness value at each iteration and the potential collisions with the environment and other robots of the other and their own swarm. *Video 2* also shows some interesting strategies for collision avoidance. For example, how some robots entering a corridor stop their motion to allow other robots to enter first. This behavioral pattern is even more evident in the initial part of *Video 4*, where a red robot in the upper part trying to enter into a corridor to move to the south finds four yellow robots in front trying to move to the north and forcing the red robot to back off and wait until they all pass first. This video also shows some of the technical problems we found in this work. At the middle part of the simulation, one red robot gets trapped at a corner, unable to move because the geometric shape of the corner makes the ultrasound signal very different to left and right. Actually, the robot was unable to move until other robots arrived to the area, modifying its ultrasound signal and allow it to move. A different illustrative example appears in *Video 5*, where a red robot in the central part of the map stays in front of two yellow robots moving in a small square with an obstacle in the middle. Instead of avoiding the yellow robots by moving around the obstacle in the opposite direction to them, the red robot stays idle for a while waiting for the other robots to pass first. We also found these unexpected

situations with the real robots, a clear indication that our simulations reflect the actual behavior of the robots in real life very accurately. Of course, these videos are only a few examples of some particular behaviors, and additional behavioral patterns can be obtained from other executions. We hope however that they allow the reader to get a good insight about the kind of behavioral patterns that can be obtained from the application of our method to this problem.

5 Conclusions and Future Work

This paper develops a swarm robotics approach to solve the problem of dual non-cooperative search with self-centered mode, a problem barely studied so far in the literature. In our setting, two robotic swarms are deployed within the same environment and assigned the goal to find their own target point (one for each swarm) at an unknown location of the map. This search must be performed under the self-centered regime, a particular case of non-cooperative search in which each swarm tries to solve its own goals with little (or none at all) consideration to any other factor external to the swarm. To tackle this issue, we apply a popular swarm intelligence method called bat algorithm, which has been adapted to this particular problem. Five illustrative videos generated as supplementary material show some of the behavioral patterns found in our computational experiments.

As discussed above, the bat algorithm allows the robotic swarms to find their targets in reasonable time. In fact, after hundreds of simulations we did not find any execution where the robots cannot get a way to the targets. We also remark the ability of the robots to escape from dead ends and other challenging configurations, as well as to avoid static and dynamic obstacles. From these observations, we conclude that our method performs very well for this problem.

There also some limitations in our approach. As shown in the videos, some configurations are still tricky (even impossible) to overcome for individual robots. Although the problem is solved for the case of swarms, it still requires further research for individual robots. We also plan to analyze all behavioral patterns that emerge from our experiments, the addition of dynamic obstacles and moving targets and larger and more complex scenarios.

Acknowledgements. Research supported by project PDE-GIR of the European Union's Horizon 2020 (Marie Sklodowska-Curie grant agreement No. 778035), grant #TIN2017-89275-R (Spanish Ministry of Economy and Competitiveness, Computer Science National Program, AEI/FEDER, UE), grant #JU12 (SODERCAN and European Funds FEDER UE) and project EMAITEK (Basque Government).

References

1. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, New York (1999)
2. Suárez, P., Iglesias, A., Gálvez, A.: Make robots be bats: specializing robotic swarms to the bat algorithm. Swarm and Evolutionary Computation (in press)
3. https://goo.gl/JUbBYw, (password: RobotsIDEAL2018)
4. Yang, X.S.: A new metaheuristic bat-inspired algorithm. In: González, J.R., Pelta, D.A., Cruz, C., Terrazas, G., Krasnogor, N. (eds.) NICSO 2010. SCI, vol. 284, pp. 65–74. Springer, Berlin (2010). https://doi.org/10.1007/978-3-642-12538-6_6
5. Yang, X.S., Gandomi, A.H.: Bat algorithm: a novel approach for global engineering optimization. Eng. Comput. **29**(5), 464–483 (2012)
6. Yang, X.S., He, X.: Bat algorithm: literature review and applications. Int. J. Bio-Inspired Comput. **5**(3), 141–149 (2013)