Supervised Learning in Robotic Swarms: From Training Samples to Emergent Behavior

Gregory Vorobyev, Andrew Vardy, and Wolfgang Banzhaf

Abstract. Emergent behavior in swarm robotic systems is key to obtaining complex behavior by a group of relatively simple agents. The question is how to design the individual behaviors of agents in such a way that the desired global behavior emerges. Different approaches have been proposed to solve this problem: from biologically inspired probabilistic behavioral models to evolutionary techniques. In some situations, however, creating a complex probabilistic model of the behavior or developing a proper setup for an evolutionary process can be challenging. In this paper we propose a new method, based on supervised learning on a relatively small number of training samples. We apply our method to the well-known clustering problem and show that this approach yields the desired global clustering behavior.

1 Introduction

Emergent behavior in swarm robotic systems has been a subject of extensive research for the last two decades [1, 2]. A robotic swarm, provided that it has been designed in a particular way, can produce global behavior, that is often conceptually more complex than the behaviors of the individual agents. For example, a group of robots can collect scattered objects into a single cluster, although each individual robot follows a simple pick-up-and-deposit procedure without any explicit knowledge of where the cluster has to be formed [3, 4, 5]. This phenomenon is widely known in biology. For example, ants exhibit an astonishing degree of collaboration and coordination in wars against other ant colonies and even other animals, with attacks and retreats and, in the worst case, evacuation of the queen and larvae carried out by the whole ant colony as if it were controlled by someone who is always aware of the current situation in the world [6]. Honeybees have been observed

Gregory Vorobyev · Andrew Vardy · Wolfgang Banzhaf
Memorial University of Newfoundland, St. John's, NL A1B 3X5
e-mail: {gvorobyev,av,banzhaf}@mun.ca

M.A. Hsieh and G. Chirikjian (eds.), *Distributed Autonomous Robotic Systems,*
Springer Tracts in Advanced Robotics 104,
DOI: 10.1007/978-3-642-55146-8_31,

as they select a new place as home: this process is truly democratic and involves "voting", i.e., collective decision making, in which many individuals participate [7]. Due to the degree of self-organization, coordination and unity exhibited by these groups of animals, some scientists consider these groups as superorganisms [8].

The emergence of global behavior is a result of actions taken by individuals; thus, relatively simple behavioral patterns followed by the individuals eventually produce the more "intelligent" behavior of the swarm. The question is therefore how to design individual behaviors in such a way that they will construct the basis from which the desired global swarm behavior will emerge [9]. One way to do this is to take inspiration from biology. For example, clustering behavior is observed in ants as they collect their dead peers into piles [6], or in honeybees distributing pollen into cells in their hives [7]. The results of experiments conducted by biologists have led to probabilistic models of individual behavior [3]. The feasibility of these models is further confirmed by observing the global swarm behavior in simulated (or real) robotic systems and comparing it with what is observed in nature [10]. Another approach exploits evolutionary techniques. In this methodology, individual behaviors evolve in such a way that the global behavior is improved [11, 12, 13, 14].

While both methods of designing individual behaviors have proved to be successful, they have their own issues. In the first case, a swarm designer needs to have feasible behavioral models, which may not be available (for example, if a behavior which is desired for the swarm has not been observed in nature). In the other case, the problem of a proper set-up for the evolutionary process arises (for example, which parameters of the behavior are subject to evolution and which are not); moreover, computational costs are typically high for the evolutionary approach.

In this paper, we propose an alternative simple method of designing the individual behaviors of agents for the clustering problem. In our approach, the designer considers a small number of characteristic situations that an agent might encounter. While it is hard to predict each possible configuration of the environment in which the agent may find itself and to generate a corresponding rule for this situation, it is much easier to accomplish this task if the number of situations being considered is relatively small (in our work, only 4). Yet, as we demonstrate in this paper, such a small number of training samples is sufficient for the agents to learn the task of clustering in such a way that the swarm starts to produce the desired behavior. We conduct experiments to test our approach in a custom 3D simulator with a realistic physics engine, and we show that our agents are capable of accomplishing the clustering task without any explicit probabilistic models embedded into them.

The rest of the paper is organized as follows. In Section 2, a short review of the relevant work is given. Section 3 describes the methodology used to solve the behavioral design problem. In Section 4, we conduct experiments and discuss the results. Finally, conclusions and future work are given in Section 5.

2 Related Work

In one of the pioneering works in the area of swarm robotics, Deneubourg et al. consider a generic sorting problem, where a swarm of robots collect objects (pucks) of different types into homogeneous clusters [4]. Each agent moves randomly between cells in a grid-based environment. Whenever the agent encounters a puck (i.e., enters a cell with a puck), it decides whether or not to pick it up. This decision is based upon how many objects of the same type this agent has encountered in the recent past. This information is stored in a short-term memory which is represented as an array of 10 items, for instance 00AAA0B00A (4 pucks of type A, 1 puck of type B, and 5 empty cells encountered during the last 10 time steps). The more pucks of a given type the agent remembers from his recent experience, the less is the probability of picking up a puck of that type. Further on, if an agent carries a puck and enters an empty cell, it decides whether or not to put the puck down. Intuitively, the more pucks of the same type as the puck which is being carried the agent has encountered in the recent past, the larger is the probability of depositing this puck. Ultimately, these simple rules result in a global sorting behavior of the swarm. There is no communication between agents or centralized control in the swarm: agents effectively are not aware of each other and act completely independently.

The idea of creating probabilistic behavioral controllers similar to what was proposed by Deneubourg et. al. has been applied to many different problems [1, 2]. For example, the task of collective aggregation has been solved by a group of cockroach-like robots with probabilistic controllers [15, 16]. The robots move randomly and stop with a certain probability which is a function of the number of other robots in immediate proximity (note that this mechanism is very similar to what has been used by Deneubourg et. al., although the task is slightly different). Thus, the behavior `Stop` is activated with a certain probability P_{stop}. In [17], the aggregation task is accomplished by robots with 4 atomic behaviors: `ObstacleAvoidance`, `Repel`, `Wait`, and `Approach`, with the last three organized into a probabilistic finite-state automaton. A robot approaches the largest group of robots with a probability P_{return}, waits for a random period of time, and then runs away from it with the probability P_{leave}. The results of this work demonstrate that the best performance is achieved with the $P_{return} = P_{leave} = 1$; in this case, the probabilistic behavior is reduced to deterministic, or procedural, behavior.

Deterministic behaviors of swarm agents have also been systematically studied in [9], where 6 basic behaviors are presented and tested: `Aggregation`, `Homing`, `CollisionAvoidance`, `Following`, `Dispersion`, and `Flocking`. Each behavior is a simple procedure; for example, `CollisionAvoidance` can be summarized as *"If there is another robot on the right, turn left; otherwise, turn right"*. In more recent work, [18], relatively simple deterministic behaviors of the agents have been applied to the chain formation problem.

In [5], similar deterministic rules have been embedded into a subsumption architecture to solve the clustering problem. For example, if an obstacle is detected in front of the robot, the `ObstacleAvoidance` behavior is activated. Different behaviors are activated depending on certain conditions. Similar experiments have

been conducted in [19]. However, the condition checks used to trigger the behaviors in this work are "encoded" as the weights of neural connections going from the sensors rather than hard-coded procedural boolean expressions. The neural networks approach for activating behaviors based on a certain perception snapshot has been more explicitly used in [12] for the aggregation problem. In [20], the similar approach has been applied to the problem of self-assembly in a swarm-bot.

The impact of different parameters of atomic behaviors on the overall performance is commonly estimated through systematic experiments in these works. For example, in case of probabilistic controllers, the parameters that are subject to testing may include probabilistic thresholds, such as P_{leave} or P_{return} [17]. Such tests proved to be important, because it is tricky to predict which values for these parameters will be optimal for each particular experimental configuration. If the number of parameters is large, the designer's task becomes even more challenging.

Evolutionary techniques have been introduced in swarm robotics as another approach to designing individual behaviors. In [12] the aggregation problem was solved by evolving the weights of a perceptron using a fitness function which computes the average distance from a robot to the largest group. This work has been further improved in [11], where the authors deduced general rules for selecting evolutionary parameters in the swarm design problem. In the most recent work, [13], the evolutionary approach has been applied in swarm robotics to obtain emergent self-organizing behavior inspired by the collective flashing behavior of fireflies.

While evolutionary algorithms allow to avoid difficulties with fine-tuning parameters of the individual behaviors, they raise new issues. For example, as it is stressed in [13], the designer of a robotic swarm should determine which behavioral parameters are fixed and which are subject to evolution. The most difficult part, however, is probably the fitness function. Fitness functions, like those used in [12], tend to require some global knowledge (for example, distance between robots), which sometimes could hardly be obtained. Finally, the computational costs are usually large for evolutionary algorithms: for example, in [13], 500 experimental trials have been executed to evolve the individual behaviors.

In this paper, we present an alternative approach to designing the individual behaviors with application to the clustering problem. Our agent's controller is based on a neural network, which is similar to the networks described in [19] and [12]. However, we do not use hard-coded neural weights (as in, e.g., [19]), and we do not use evolutionary algorithms to evolve the weights (as in, e.g., [12]). Rather, we consider a set of 4 training samples. Each sample represents a perceptual snapshot. From a set of 3 behaviors - `BackUpAndTurn`, `Turn`, and `MoveStraightAhead` - we select the most suitable. For example, if a robot "sees" a large number of pucks in front of it, it should activate `BackUpAndTurn`. We show that this approach, being extremely simple and easy to follow, yields the desired clustering behavior.

3 Methodology

In this work we revisit the clustering problem, in which agents collect initially scattered pucks into a single pile. Similar to [21], our agents have no specialized grippers to manipulate the pucks. Rather, the agents push the pucks with a plow.

As mentioned above, a neural network is a central part of the agent's architecture in our work. We use a simple single-layer perceptron, with 2 inputs and 3 outputs. Each time step, sensory data is used as an input to this neural network. The output of the neural network is then interpreted as a code of the basic behavior to activate.

In the rest of this section, we discuss what kind of sensory data we use, describe the basic behaviors, and explain how sensory data is normalized to be fed into the neural network and how the network's output is interpreted. Finally, we describe the samples used to train the neural network.

3.1 Perception Areas

We assume that a robot has a sensor that can detect pucks and their position relative to it. It can be implemented in physical robotic systems with a calibrated camera, some image processing, and exploiting some knowledge about the geometry of the local environment (i.e. that the floor is planar) [23].

Sensory data in our work are the number of pucks in perception areas. Two such areas are provided for an agent (see Fig. 1). The `Central` area is important for detecting clusters of pucks in the immediate vicinity in front of the agent. The size of the `Central` area is approximately 6x6 puck diameters. The `Exploration` area stretches forward and is used for detecting pucks that are relatively far from the robot. The size of the `Exploration` area is approximately 4x25 puck diameters.

The input fed to the neural network reflects the number of pucks in the perception areas. This input, however, must be normalized within the range $[0,1]$ (which is conventional for neural networks). The normalization is done by dividing the actual number of pucks in the area by the maximum number of pucks for that area. Thus, the **relative density** of pucks in an area is calculated. The question is then how to define the maximum numbers for the perception areas.

The neural signal from the `Central` region is saturated at 1 (is maximized) when the number of pucks in this region, assuming that they are uniformly

Fig. 1 Perception areas of an agent (blue). 1. The `Central` area is directly in front of the agent. 2. The `Exploration` area stretches ahead.

distributed, is large enough to form a single cluster[1]. In our setup, this number (further referred to as `MaxCentral`) is equal to 16. Experiments have shown that if the `MaxCentral` parameter is chosen to be significantly lower, for example, 8, performance of the swarm is unsatisfactory[2].

For the `Exploration` area the interpretation of the maximum number is different. This area is intended to serve for searching pucks, and not for detecting clusters. Hence, it is only important whether there is at least one puck in this area or not. The maximum number for the `Exploration` area is therefore 1.

A saturating linear function is used to normalize the number of pucks for both perception regions. Thus, if the number of pucks in either of the regions is larger than the corresponding maximum number, the input to the associated neuron will be saturated at 1.

3.2 Basic Behaviors

Three behaviors are available for agents. `BackUpAndTurn` behavior is inspired by work in [4], where it has been proved to be efficient and the most important for the clustering behavior. `Turn` behavior is used for locating pucks. Finally, `MoveStraightAhead` behavior, as the name implies, moves an agent forward. The description of the behaviors is given in Table 1.

Table 1 Description of the behaviors

`BackUpAndTurn`	Back up for N time units, then turn at random angle*.
`Turn`	Turn at a small angle.
`MoveStraightAhead`	Move straight ahead.

*To implement this behavior, the procedure `Update`, which is called each time step and which is responsible for collecting data, feeding it into the neural network, getting the output and activating behaviors, needs an additional condition. If `BackUpAndTurn` behavior is currently being executed, then `Update` immediately returns without referring to the neural network. This is repeated until N times units have passed. After that, the `Update` procedure is executed normally.

The output of the network is an array of three floating point numbers. The methodology "winner-takes-all" is used to convert this array to an array of three

[1] We define clusters as follows. Suppose that at time t we have a graph with P vertices, where P is the number of pucks in the experiment. There is a one-to-one correspondence between the set of vertices and the set of pucks; i.e., each vertex i is a mathematical representation of a corresponding puck p_i. An *edge* between vertices a and b in this graph exists if and only if the $d_t(p_a, p_b) < h$, where $d_t(x,y)$ is the distance between x and y at time t and h is a distance threshold. In our work, we define h as one diameter of a puck. Each connected component in this graph will then represent a cluster of pucks.

[2] Further discussion of this question is given in Section 4.

integers: the maximum element of the array is rounded to 1, while two other elements are rounded to 0. The interpretation of the resulting integer array is given in Table 2.

Table 2 Interpretation of the output.

`[1, 0, 0]`	Activate `BackUpAndTurn` behavior.
`[0, 1, 0]`	Activate `Turn` behavior.
`[0, 0, 1]`	Activate `MoveStraightAhead` behavior.

3.2.1 Obstacle Avoidance

The obstacle avoidance behavior, which is used in many works (for example, [5, 9, 17, 19]; see Section 2) is not explicitly present here. Rather, to avoid collisions between robots we use `BackUpAndTurn` behavior: in the same manner as we avoid disturbing large clusters, we avoid collisions between robots. This way, we do not require an additional behavior for obstacle avoidance; thus, we reduce the complexity of learning.

Therefore, another robot should be perceived in a similar way as a large cluster. To achieve that, we maximize (make it equal to 1) the input from the `Central` area if a robot is detected within this region.

3.3 Network Training

The sketch of the neural network used in this paper is presented in Fig. 3.

It may be hard to develop a probabilistic behavioral model even for such a simple task as clustering. However, it is relatively simple to define qualitative rules. The short summary of these rules, partially inspired by works discussed in Section 2, is as follows.

1. *If there is a cluster directly in front of an agent, it should back up and turn (to avoid disturbing the cluster).*
2. *If there is no cluster directly in front of an agent, it should always move to the puck in the* `Exploration` *area (this would help to bring some of the pucks the agent is plowing, if any, to that puck).*
3. *If there are no pucks in the* `Exploration` *area, an agent should turn around until it finds at least one. (Extrusions on the sides of the plow will help the agent keep the pucks that it has plowed.)*

Given this summary, we define characteristic situations that an agent may encounter (i.e., a training set) and provide a ”solution” for each of these situations (see Fig. 2 and Fig. 4). We then use the backpropagation learning method to train the

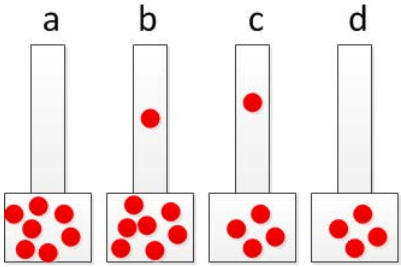

Fig. 2 The sketch of the training set. The lower rectangle is the `Central` area, the rectangle above is the `Exploration` area. Red circles are pucks. In situations ***a*** and ***b***, `BackUpAndTurn` behavior should be activated, because the relative density of pucks in the `Central` area is high. In ***c***, `MoveStraightAhead` should be activated: this will increase the number of pucks in the `Central` area; therefore, a larger cluster will be created. Finally, in ***d*** the agent should activate `Turn` behavior in attempt to find a puck in the `Exploration` area.

neural network until it starts producing correct output for all training samples[3]. It appears that although we considered only 4 possible situations from more than 3,200 possible combinations (0.00125%) of the numbers of pucks in the perception areas[4], the neural network interpolates to recognize all other situations correctly. The proposed method is somewhat similar to linear support vector machines (SVMs) used in supervised learning and statistical analysis [22], though we do not follow this approach directly.

4 Experiments

Experiments have been conducted using a custom simulator written in C#. The simulator uses the MOGRE engine[5] for 3D visualization and the MogreNewt engine[6] for modeling realistic physics in the simulation. The process of clustering observed in the simulator can be seen at Fig. 5.

[3] Since we are using a single-layer perceptron, the backpropagation is effectively reduced to the delta rule. However, in a more general case, neural networks with hidden layers can be used; therefore, the backpropagation method will be needed.

[4] In theory, the `Central` area can accommodate as many as 32 pucks (not 36, although its size is 6x6 puck diameters; this is because the plow extrusions are located within the `Central` area, leaving less space for pucks), provided that they are clustered in an extremely tight manner; the `Exploration` area in a similar way provides space for about 100 pucks. Thefore, the total number of combinations of pucks in the perception areas is $N = 32 \times 100 = 3200$.

[5] http://www.ogre3d.org/tikiwiki/MOGRE

[6] http://www.ogre3d.org/tikiwiki/MogreNewt

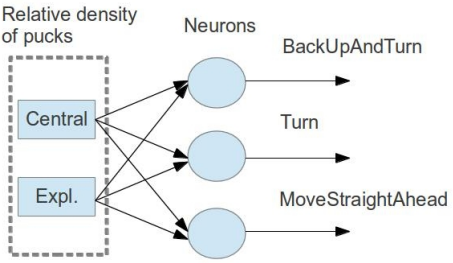

Fig. 3 The neural network. The inputs (in picture, bounded by a dashed line) are normalized relative densities of pucks in perception areas. (The normalization is made using saturating linear function, see Sect. 3.1.) The output is an array of floating-point numbers; we then apply the 'winner-takes-all' methodology to obtain one of the codes from Table 2. The operation of the neural network is described by the following equation: $\mathbf{a} = logsig(\mathbf{Wp} + \mathbf{b})$, where $logsig(n) = \frac{1}{1+e^{-n}}$, $\mathbf{a}$ is the output vector $(a_1, a_2, a_3)^T$, $\mathbf{W}$ is the weight matrix 3×2, $\mathbf{p}$ is the input vector $(p_1, p_2)^T$, and b is the bias vector $(b_1, b_2)^T$.

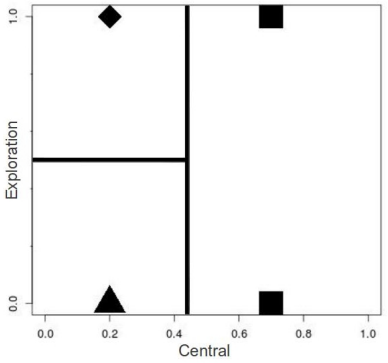

Fig. 4 Target outputs for the training set. The inputs for the neural network are coordinates of the points; the output is denoted by symbols, which are interpreted as follows: ■ - `BackUpAndTurn`, ▲ - `Turn`, ♦ - `MoveStraightAhead`. Black solid lines are decision boundaries obtained by the application of the delta rule. It can be seen that the given patterns are linearly separable. Note that this particular problem could be solved by using only two neurons (one for each decision boundary) with saturating linear transfer functions. The output of the network will then be the binary code of a behavior. In this paper, however, we prefer to use three neurons (one for each behavior) with log-sigmoid transfer functions; therefore, the output of the network is an array of floating-point numbers which we interpret as the 'confidence' of the network in a given behavior.

Initially, 100 pucks are scattered randomly (using the uniform distribution) over a squared area with sides of 40 puck diameters. Five agents[7] start in random positions

[7] The number of agents has been chosen more or less arbitrarily; most works in the clustering problem, including [4], [5], [19], and [23] tend to choose the number of robots in the range from 1 to 10, so we concluded 5 to be the most typical value.

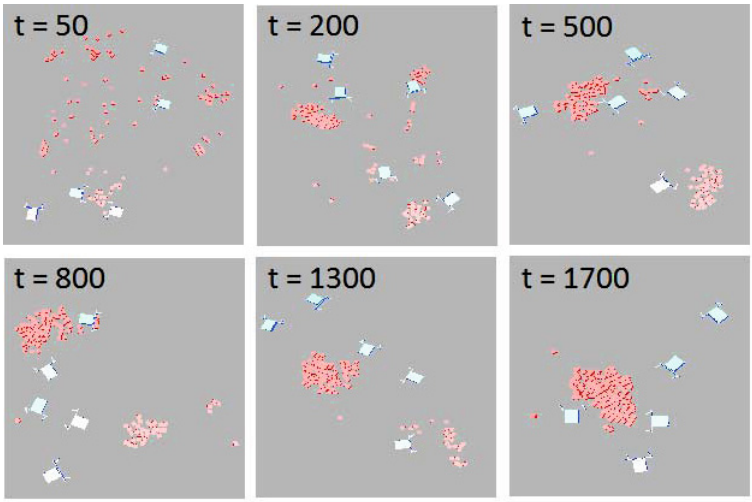

Fig. 5 Simulation of the clustering problem in process. Several phases of the process can be distinguished. In the first stage ($0 < t < 500$, t is time), the number of clusters is quickly decreasing and a few large clusters (two in this case; sometimes it may be three or even four clusters) are formed. In the second stage ($500 < t < 1700$), the smaller cluster is gradually destroyed; pucks from the second cluster are delivered to the bigger cluster. (This process, however, is statistical; we have observed situations when, on the contrary, the bigger cluster was destroyed.) In the final phase ($t > 1700$), a single cluster is formed. A few pucks may be removed from the cluster from time to time, but they are then returned back. (All time intervals are given in simulation time units and may vary from trial to trial.)

within this area. There are no boundaries (walls) around the simulation area. Note, however, that agents tend to keep pucks within the initial area, due to the `Turn` behavior.

To measure the performance, we use the metrics from [5]: the average size of a cluster and the size of the largest cluster. Statistics are collected every 10 time units of the simulation. In Section 3.1 we have mentioned that we predefine the maximum number of pucks for perception areas (for `Central` area this parameter is named `MaxCentral`). These numbers are the only tunable parameters for the learning mechanism. We can change these parameters by effectively reprogramming a robot (i.e., modifying the ”software”). All other parameters, such as robot and puck sizes, or the number and the sizes of perception areas, most likely cannot be adjusted without affecting the ”hardware” (we consider the ”hardware” parameters to be given as is). The efficiency of the proposed ”software” can be estimated by using trials with different values for the ”software” parameters. Thus, we conduct experiments with different values for the `MaxCentral` parameter to determine the performance of the proposed learning mechanism. Results, averaged for 10 runs for each value of the `MaxCentral` parameter, can be seen in Fig. 6 and Fig. 7.

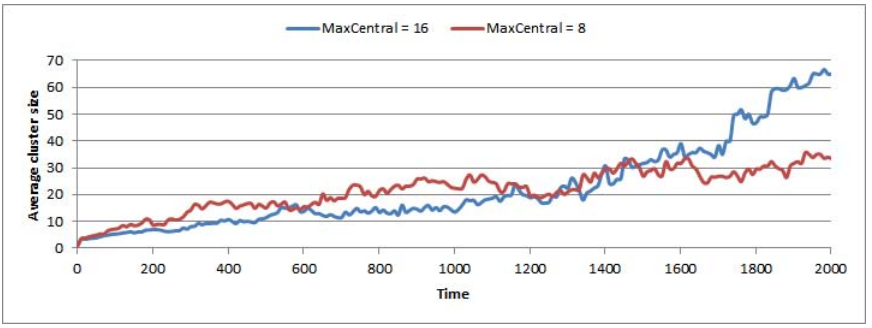

Fig. 6 Average cluster size for different values of the `MaxCentral` parameter. With `MaxCentral` = 16, the average of 33 (1 main cluster with 98 pucks and 2 clusters with a single puck in each), 50 (1 main cluster with 99 pucks and a separated puck) or 100 is typical in the final stage of the process, since pucks are occasionally removed from the cluster (but are quickly returned back). With `MaxCentral` = 8, the average cluster increases slower.

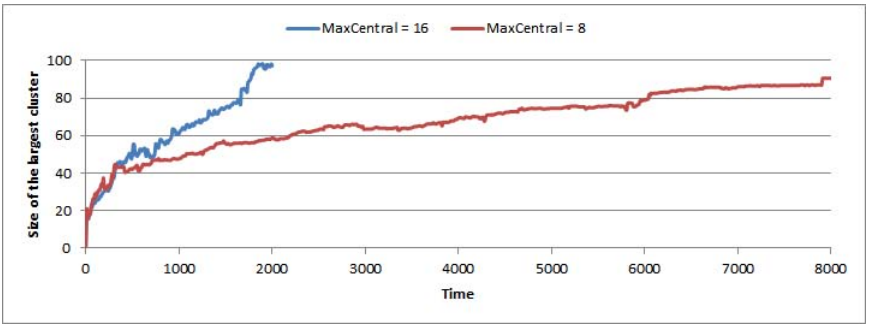

Fig. 7 Size of the largest cluster for the different values of the `MaxCentral` parameter. The horizontal axis (time) has been extended in this figure. Experiments with `MaxCentral` = 16 were stopped once a cluster of 100 pucks was formed. Experiments with `MaxCentral` = 8 took approximately 4 times more time to converge to a single cluster. However, in a few experiments with `MaxCentral` = 8 the convergence to a single cluster was not achieved even after 8000 time units.

The `MaxCentral` parameter has proved to be critical for performance. If this parameter is chosen to be too low (for example, 8), the `BackUpAndTurn` behavior is often triggered prematurely. In this case, agents are highly unlikely to destroy smaller clusters in order to push a few pucks to bigger clusters; thus, the size of the largest cluster and the average cluster size grow slowly (see Fig. 7; the size of the largest cluster is growing approximately 4 times slower than for the value for `MaxCentral` equal to 16). When the `MaxCentral` parameter is high enough (for example, 16), agents are allowed to ”steal” pucks from one cluster to deliver them to another one; therefore, the size of the largest cluster and the average cluster size grow faster.

It is interesting that in the first 1000 times steps the performance yielded by the smaller parameter value is better (see Fig. 6). This is because agents guided by a smaller `MaxCentral` parameter tend to form many clusters of a relatively small size (10-20 pucks), whereas "greedier" agents (with `MaxCentral` = 16) start creating bigger clusters from the very beginning, and smaller clusters which are occasionally formed are likely to be destroyed shortly. However, once several big clusters have been formed, the average cluster size starts to grow relatively fast, whereas in the first case it is growing more slowly. We conclude that the tradeoff here is between convergence to smaller, bigger clusters at the expense of increased time.

It appears that the probability of removing pucks from a cluster is a function of its size. The only way to remove pucks from a big cluster is to follow a tangent line to the cluster and to plow some pucks from its skirt. Due to the `Turn` behavior, which is likely to be activated afterwards, the pucks are then either returned back to the cluster (if there are no other piles of pucks outside), or pushed to another cluster. If the cluster is relatively large, then the chance that the first puck that will appear in the `Exploration` area will belong to the same cluster is large; the pucks will thus be returned to the cluster. If the cluster is relatively small, there is a high chance that a significant part of it (or even the entire cluster) would be removed without activating `BackUpAndTurn` behavior. Hence, the smaller the cluster is, the bigger is the probability to remove pucks from it. This probabilistic process is functionally similar to what has been developed by Deneubourg et al. [4]. However, no explicit probabilistic rules are present in our system. The global probabilistic behavior emerges based on the geometrical shapes of the robots and pucks, and the individual behaviors provided by neural-based controllers. More detailed theoretical derivations related to this subject can be found in [24].

5 Conclusions

Emergent behavior is a key to using a swarm of simple cheap robots for solving complex tasks without centralized control. Different approaches have been proposed to designing the agents' behavior in such a way that the desired global swarm behavior emerges. However, in some situations these approaches may turn out to be inapplicable or inefficient. In this work, we have presented a simple method to design a swarm behavior based on supervised learning of a small number of samples representing situations that an agent may encounter in the world of clustering.

While obtaining global probabilistic behavior which is functionally similar to what has been described in [4] and subsequent works, we avoid creating explicit probabilistic models of the individual behavior of the agents. Our approach is extreme in its simplicity. We use no specialized grippers for the agents; the number of behaviors is limited to three; the neural network is a single-layer perceptron. Yet, such a simple approach yields the desired result: the swarm of agents accomplishes the clustering task.

With increasing complexity of sensory input and the number of behaviors the training will most probably become more complicated. However, the basic idea of

swarm robotics implies an extreme minimalism of robots [23]; the number of sensory inputs and behaviors is expected to be relatively small. If it is difficult to distinguish "characteristic", or "boundary" situations from the set of possible sensory inputs, the process of supervised learning can be made iterative: add one sample after another, until the produced behavior becomes acceptable. With this assumption, we think that the proposed approach may be efficient for other tasks studied in swarm robotics.

Acknowledgements. W.B. acknowledges funding from NSERC under the Discovery Grant Program RGPIN 283304-07 and RGPIN 283304-12.

References

1. Bayindir, L., Sahin, E.: A Review of Studies in Swarm Robotics. Turkish Journal of Electrical Engineering 15, 115–147 (2007)
2. Yogeswaran, M.: Swarm Robotics: An Extensive Research Review. In: Advanced Knowledge Application in Practice, pp. 259–278 (2010)
3. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press (1999)
4. Deneubourg, J.L., Goss, S., Franks, N., Sendova-Franks, A., Detrain, C., Chretien, L.: The dynamics of collective sorting robot-like ants and ant-like robots. In: First International Conference on the Simulation of Adaptive Behaviour, pp. 356–363 (1991)
5. Beckers, R., Holland, O.: From local actions to global tasks: Stigmergy and collective robotics. In: Artificial Life, vol. IV, pp. 181–189 (1994)
6. Holldobler, B., Wilson, E.O.: Journey to the Ants: A Story of Scientific Exploration. Belknap Press of Harvard University Press (1994)
7. Seeley, T.D.: Honeybee democracy. Princeton University Press (2010)
8. Holldobler, B., Wilson, E.O.: The Superorganism: The Beauty, Elegance and Strangeness of Insect Societies. W. W. Norton & Company, Inc (2009)
9. Mataric, M.: Designing emergent behaviors: From Local Interactions to Collective Intelligence. In: From Animals to Animats 2. Proceedings of the Second International Conference of Simulation Adaptive Behavior, pp. 432–441 (1992)
10. Martinoli, A., Ijspeert, A.: A Probabilistic Model For Understanding And Comparing Collective Aggregation Mechanisms. In: Floreano, D., Mondada, F. (eds.) ECAL 1999. LNCS, vol. 1674, pp. 575–584. Springer, Heidelberg (1999)
11. Bahceci, E., Sahin, E.: Evolving Aggregation Behaviors For Swarm Robotic Systems: A Systematic Case Study. Technical report METU-CENG-TR-2005-03, Middle East Technical University, Turkey (2005)
12. Trianni, V., Groß, R., Labella, T.H., Şahin, E., Dorigo, M.: Evolving Aggregation Behaviors in a Swarm of Robots. In: Banzhaf, W., Ziegler, J., Christaller, T., Dittrich, P., Kim, J.T. (eds.) ECAL 2003. LNCS (LNAI), vol. 2801, pp. 865–874. Springer, Heidelberg (2003)
13. Trianni, V., Nolfi, S.: Engineering The Evolution of Self-Organizing Behaviors In Swarm Robotics: A Case Study. Artificial Life 17(3), 183–202 (2011)
14. Trianni, V.: Evolutionary Swarm Robotics: Evolving Self-Organising Behaviours in Groups of Autonomous Robots. SCI, vol. 108. Springer, Heidelberg (2008)
15. Jeanson, R., Rivault, C., Deneubourg, J., Blancos, S., Fourniers, R., Jost, C., Theraulaz, G.: Self-Organized Aggregation in Cockroaches. Animal Behaviour 69, 169–180 (2005)

16. Garnier, S., Jost, C., Jeanson, R., Gautrais, J., Asadpour, M., Caprari, G., Theraulaz, G.: Collective Decision-Making by a Group of Cockroach-Like Robots. In: Proceedings of the IEEE Swarm Intelligence Symposium, pp. 233–240 (2005)
17. Soysal, O., Sahin, E.: Probabilistic Aggregation Strategies in Swarm Robotic Systems. In: Proceedings of the IEEE Swarm Intelligence Symposium, pp. 325–332 (2005)
18. Nouyan, S., Dorigo, M.: Chain Formation in a Swarm of Robots. Technical report TR/IRIDIA/2004-18, IRIDIA - University Libre de Bruxelles, Belgium (2004)
19. Martinoli, A., Mondada, F.: Collective and cooperative group behaviours: Biologically inspired experiments in robotics. In: Proceedings of the Fourth International Symposium on Experimental Robotics ISER-95. LNCIS, vol. 223, pp. 1–10. Springer, Heidelberg (1997)
20. Gross, R., Bonani, M., Mondada, F., Dorigo, M.: Autonomous Self-assembly in a Swarmbot. In: Proceedings of the Third International Symposium on Autonomous Minirobots for Research and Edutainment, pp. 314–322 (2006)
21. Parker, C., Zhang, H.: Blind bulldozing: multiple robot nest construction. In: Proceedings of the International Conference on Robots and Systems, pp. 2010–2015 (2003)
22. Christianini, N., Shawe-Taylor, J.: An Introduction To Support Vector Machines and Other Kernel-Based Learning Methods. Cambridge University Press (2003)
23. Vardy, A.: Accelerated patch sorting by a robotic swarm. In: Canadian Conference on Robot Vision (2012)
24. Kazadi, S., Abdul-Khaliq, A., Goodman, R.: On the convergence of puck clustering systems. Robotics and Autonomous Systems 38, 93–117 (2002)