Observing the Effects of Overdesign in the Automatic Design of Control Software for Robot Swarms

Mauro Birattari[1(✉)], Brian Delhaisse[1], Gianpiero Francesca[1], and Yvon Kerdoncuff[1,2]

[1] IRIDIA, Université Libre de Bruxelles, Brussels, Belgium
mbiro@ulb.ac.be
[2] ENSTA ParisTech, Palaiseau, France

Abstract. We present the results of an experiment in the automatic design of control software for robot swarms. We conceived the experiment to corroborate a hypothesis that we proposed in a previous publication: the reality gap problem bears strong resemblance to the generalization problem faced in supervised learning. In particular, thanks to this experiment we observe for the first time a phenomenon that we shall call *overdesign*. Overdesign is the automatic design counterpart of the well known overfitting problem encountered in machine learning. Past an optimal level of the design effort, the longer the design process is protracted, the better the performance of the swarm becomes in simulation and the worst in reality. Our results show that some sort of early stopping mechanism could be beneficial.

Keywords: Swarm robotics · Automatic design · Evolutionary robotics · Reality gap · Generalization · Overdesign · Early stopping

1 Introduction

Designing the control software of the individual robots so that the swarm performs a given task is a difficult problem. A number of interesting approaches have been proposed to address specific cases—e.g., [3,7,28,32,34,45,56]. Nonetheless, there is no ultimate and generally applicable method on the horizon.

Automatic design is a viable alternative. To date, the automatic design of control software for robot swarms has been mostly studied in the framework of evolutionary swarm robotics [52], which is the application of evolutionary robotics [40] in the context of swarm robotics. In the classical evolutionary swarm

This research was conceived by MB and GF and was directed by MB. The experiment was performed by YK using automatic design software developed by BD on the basis of a previous version by GF. The article was drafted by MB and GF. All authors read the manuscript and provided feedback. BD is currently with the Department of Advanced Robotics, Istituto Italiano di Tecnologia (IIT), Genova, Italy.

M. Dorigo et al. (Eds.): ANTS 2016, LNCS 9882, pp. 149–160, 2016.
DOI: 10.1007/978-3-319-44427-7_13

robotics, the control software of each individual robot is a neural network that takes sensor readings as an input and returns actuation commands as an output. The parameters of the neural network are obtained via an evolutionary algorithm that optimizes a task-specific objective function. The optimization process relies on computer-based simulation. Once simulation shows that the swarm is able to perform the given task, the neural network is uploaded to the robots and the actual real-world performance of the swarm is assessed.

The *reality gap* [9,30] is one of the major issues to be faced in evolutionary swarm robotics—and in all automatic design methods that rely on simulation. The reality gap is the intrinsic difference between reality and simulation. As a consequence of the reality gap, differences should be expected between how an instance of control software behaves in simulation and in reality. Indeed, as pointed out by Floreano et al. [18], the control software is optimized "to match the specificities of the simulation, which differ from the real world."

A number of ideas have been proposed to reduce the impact of the reality gap, including methods to increase the realism of simulation [31,36] and design protocols that alternate simulation with runs in reality [5,33]. In a recent article, Francesca et al. [22] argued that the reality gap problem is reminiscent of the generalization problem faced in supervised learning. In particular, the authors conjectured that the inability to overcome the reality gap satisfactorily might result from an excessive representational power of the control software architecture adopted. Taking inspiration from a practice that is traditionally advocated in the supervised learning literature [13], the authors explored the idea of injecting bias in the process as a means to reduce the representational power.

In this article, we elaborate further on the relationship between the reality gap problem and the generalization problem faced in supervised learning. Understanding this relationship can enable the development of new approaches to handle the reality gap. We present an experiment whose goal is to highlight, in context of the automatic design of control software for robot swarms, a phenomenon similar to *overfitting*. Indeed, if the reality gap problem is similar to the generalization problem of machine learning, one should observe that, past an optimal level of the design effort, the further the control software is optimized in simulation, the worse the performance in reality gets. In the context of the automatic design of control software, we shall call this phenomenon *overdesign*.

2 Related Work

The automatic generation of control software is a promising approach to the design of robot swarms [8,19]. Most of the published research belongs in evolutionary swarm robotics [52], which is the application of the principles of evolutionary robotics [40] in the context of swarm robotics. Evolutionary robotics has been covered by several recent reviews [6,14,48,53]. In the following, we briefly sketch some of its notable applications in swarm robotics.

A number of authors adopted the classical evolutionary robotics approach: robots are controlled by neural networks optimized via an evolutionary algorithm. Quinn et al. [43] developed a coordinated motion behavior and tested it

on three Kheperas. Christensen and Dorigo [11] developed a simultaneous hole-avoidance and phototaxis behavior and tested it on three s-bots. Baldassarre et al. [1] developed a coordinated motion behavior for physically connected robots and tested it on four s-bots. Trianni and Nolfi [51] developed a self-organizing synchronization behavior and tested it on two and three s-bots. Waibel et al. [54] developed an idealized foraging behavior and tested it on two Alices.

For completeness, we mention a number of studies in the automatic design of control software for robot swarms that departed from the classical evolutionary swarm robotics. Hecker et al. [29] developed a foraging behavior by optimizing the parameters of a finite state machine via artificial evolution. They tested the behavior on three custom-made robots. Gauci et al. [24,25] developed object clustering and self-organized aggregation by optimizing the six parameters of a simple control architecture using evolutionary strategy and exhaustive search, respectively. Experiments were performed with five and forty e-pucks, respectively. Duarte et al. [15,16] proposed an approach based on the hierarchical decomposition of complex behaviors into basic behaviors, which are then developed via artificial evolution or implemented manually. The authors obtained behaviors for object retrieval and patrolling. In a successive study [17], the authors used artificial evolution to produce control software for a swarm of ten aquatic robots and solve four different sub-tasks: homing, dispersion, clustering and area monitoring. The control software for the four sub-tasks was then combined in a sequential way to accomplish a complex mission. The authors performed experiments in a 330 m $\times$ 190 m waterbody next to the Tagus river in Lisbon, Portugal. The results show that the control software produced crosses the reality gap nicely. Francesca et al. [20–22] proposed AutoMoDe: an approach that automatically assembles and fine tunes robot control software starting from predefined modules. The authors developed behaviors for seven tasks: aggregation, foraging, shelter with constrained access, largest covering network, coverage with forbidden areas, surface and perimeter coverage, and aggregation with ambient cues. The developed behaviors were tested with swarms of twenty e-pucks.

3 Facts and Hypotheses

Neural networks have been studied for over seven decades, with alternating fortune—e.g., [12,35,37,46,55]. Around the year 2000, neural networks appeared to be superseded by other learning methods. They regained the general attention of researchers and practitioner in the last decade, thanks to the major success of deep learning—e.g., see [47]. In the context of our reasoning, we are interested in scientific facts about neural networks and their generalization capabilities that where established mostly in the 1990's. In particular, we are interested in the relationship between prediction error and two characteristics: (1) the complexity of the neural network; and (2) the amount of training effort.

A fundamental result for understanding the relationship between error and complexity is the so called *bias/variance decomposition* [26].[1] It has been proved that the prediction error can be decomposed into a bias and a variance component. Low-complexity neural networks—i.e., those with a small number of hidden neurons and therefore low representational power—present a high bias and a low variance. Conversely, high-complexity neural networks—i.e., those with a large number of hidden neurons and therefore a high representational power—present a low bias and a high variance.

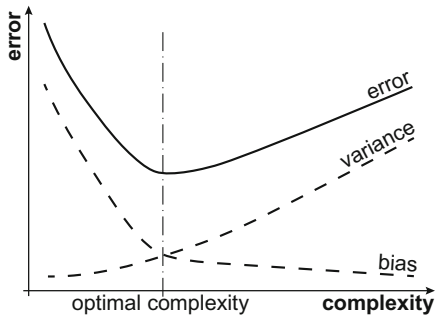


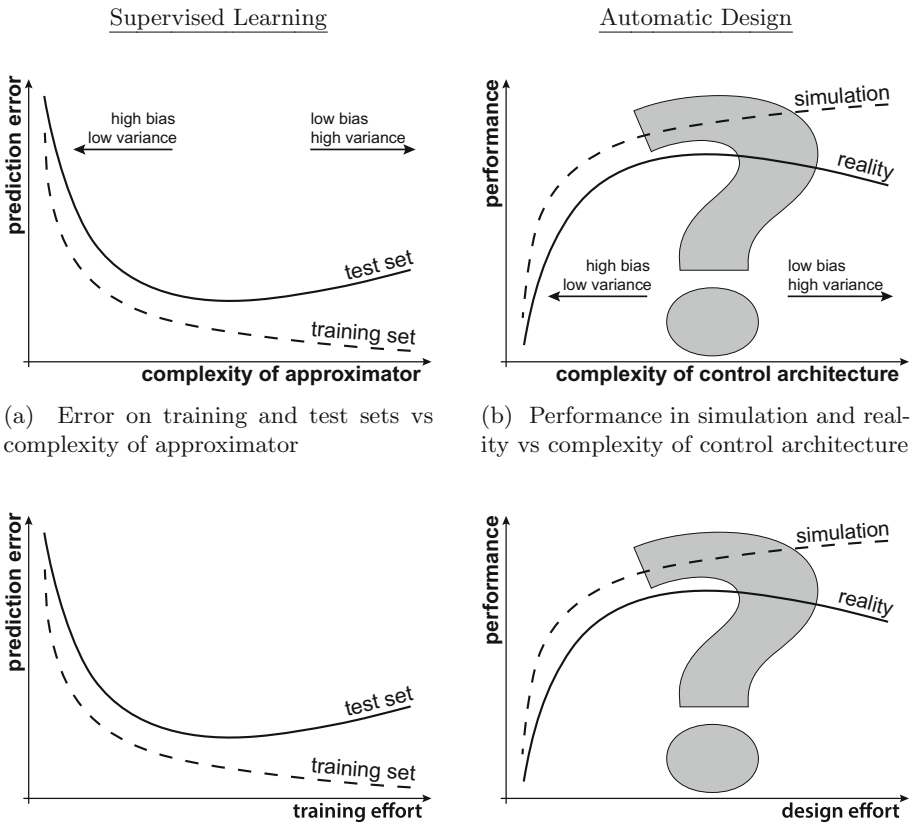
Fig. 1. Decomposition of the error into a bias and a variance component

As the bias and variance components combine additively, the error presents a U shape: for an increasingly large level of complexity, the error first decreases and then increases again. This implies that high complexity (i.e., high representational power and low bias) is not necessarily a positive characteristic: indeed an optimal value of the complexity exist. Beyond that value, prediction error increases. See Fig. 1 for a graphical illustration of the concept. In other terms, a complex network (i.e., high number of neurons and therefore high representational power) is able to learn complex functions but then generalizes poorly. Indeed, it is an established fact that the higher the complexity of a neural network (as of any functional approximator), the lower is the error on the training set and the higher is the error on a previously unseen test set—provided that we are beyond the optimal complexity. This fact is graphically represented in Fig. 2a: past the optimal level of complexity, the errors on training set and test set diverge.

Concerning the relationship between prediction error and training effort, a second important fact has been established, which goes under the name of *overfitting*—or alternatively overtraining. Overfitting is the tendency of a neural network (as of any functional approximator) to overspecialize to the examples used for training, which impairs its generalization capabilities. As a result of overfitting, one can observe that if the learning process is protracted beyond a given level, the error on the training and test sets diverge. Indeed, past an optimal level of the training effort, which is typically unknown a priori, the error on a previously unseen test set increases, while the one on the training set keeps decreasing. This fact is graphically represented in Fig. 2c.

It should be noted that the two facts illustrated in Figs. 2a and c are strictly related. The former considers the case in which the level of training effort is fixed and the complexity of the approximator is varied; the latter, considers the dual case in which the complexity of the approximator is fixed and the amount

[1] For a more advanced and general treatment of the issue, see also [57].



(a) Error on training and test sets vs complexity of approximator

(b) Performance in simulation and reality vs complexity of control architecture

(c) Error on training and test sets vs training effort

(d) Performance in simulation and reality vs design effort

Fig. 2. Conceptual relationship between the bias-variance tradeoff in supervised learning and in automatic design (a/b) and between *overfitting* in supervised learning and *overdesign* in automatic design (c/d)

of training effort is varied. In both cases, past an a priori unknown level of the independent variable, the error on the training and test sets diverge.

Several ideas have been proposed to deal with these facts and produce so called *robust* learning methods. The most notable ones are cross-validation and regularization techniques—e.g., see [2,49]. In the context of this article, it is worth mentioning a technique known as *early stopping*, which consists in halting the learning process before the error on training and test set start to diverge—e.g., see [10,39,42,44].

In a previous article, Francesca et al. [22] argued that the reality gap problem faced in automatic design of robot control software is reminiscent of the generalization problem faced in supervised learning. If the two problems are indeed sufficiently similar, one should be able to observe in the automatic design

the counterparts of the facts illustrated in Figs. 2a and c. In particular, one should observe that the performance in simulation and reality diverge (1) for an increasing level of complexity of the control architecture—Fig. 2b; and (2) for an increasing level of the design effort—Fig. 2d. The only difference between Figs. 2a and b (and between Figs. 2c and d) is that the former concerns the *minimization* of error, while the latter the *maximization* of performance. On Figs. 2b and d, we superimposed a large question mark to signify that these plots represent hypotheses, as opposed to the plots appearing on their left, which represent established scientific facts supported by a vast literature.

Guided by the hypothesis depicted in Fig. 2b, Francesca et al. [22] proposed an automatic design method that, according to their intentions, has a lower representational power (i.e., lower complexity) than the neural network typically adopted in evolutionary swarm robotics. Experimental results confirm that, with respect to a control architecture with a higher representational power, one with lower representational power yields a lower performance in simulation but a higher one in reality [20–22]. Although these results are preliminary and insufficient to establish the hypothesis depicted in Fig. 2b as a scientific fact, they are coherent with our expectations and corroborate our reasoning.

On the other hand, the hypothesis depicted in Fig. 2d has never been subject of investigation, at least to the best of our knowledge. In Sect. 4, we present an experiment whose goal is to see whether, for a sufficiently large design effort, the performance in simulation and reality of automatically designed control software tend to diverge. As this phenomenon would be the automatic design counterpart of overfitting, we shall call it *overdesign*.

4 Experiment

In this section, we present the material adopted in the experiment, the automatic design method, the task, the protocol, and the results.

Robots. We consider a particular version of the e-puck robot [38]. This version was formally defined in [22] via a reference model that describes the set of sensors and actuators exposed to the control software. In this section, we provide a brief sketch of the reference model. We refer the reader to [22] for the details. The e-puck moves thanks to a two-wheel differential steering system. The e-puck senses the obstacles (e.g., walls and other robots) via eight infrared proximity sensors. The e-puck measures the reflectance of the floor via three ground sensors placed under the front of the body. Thanks to a range-and-bearing extension board [27], the e-puck perceives the presence of other e-pucks in a 0.7 m range. For each perceived robot, the e-puck senses its relative distance and angle.

The control cycle has a period of 100 ms. At each time step, the control software receives the readings through the variables $prox_i$, $light_i$, gnd_i, r_m, and $\angle b_m$ that abstract respectively, proximity, light, ground sensors and the readings of the range-and-bearing board. Based on these variables, the control software decides the command values v_l and v_r to be applied to the wheel motors.

Design Method. We adopt `EvoStick`, an automatic design method presented in [22]. We briefly illustrate `EvoStick` here and we refer the reader to [22] for the details. `EvoStick` is an implementation of the classical evolutionary swarm robotics approach: an evolutionary algorithm optimizes the feed-forward neural network that controls each robot. Inputs and outputs of the neural networks are defined on the basis of the reference model. In particular, the neural network has 24 inputs: 8 readings from the proximity sensors, 8 from the light sensors, 3 from the ground sensors and 5 that are obtained by aggregating the range-and-bearing readings [22]. The outputs are the commands to the two wheel motors.

The neural network has 50 real-valued parameters that are optimized by an evolutionary algorithm that features mutation and elitism. The evolutionary algorithm operates on populations of 100 neural networks. At a given iteration, each neural network is tested 10 times in simulation. The population to be tested at the subsequent iteration is created as follows: the 20 best performing neural networks (the elite), are included unchanged; 80 further neural networks are generated from the elite via mutation. Simulations are performed using ARGoS [41].

Task. A swarm of $N = 20$ e-pucks must perform the aggregation task previously studied in [22]. The environment is a dodecagonal arena of 4.91 m^2 surrounded by walls—see Fig. 3. The floor is gray, except two circular black areas, a and b. These areas have the same radius of 0.35 m and are centered at 0.60 m from the center of the arena. The swarm must aggregate on either a or b. At the beginning of the run, each robot is randomly positioned in the arena. The run lasts for $T = 240$ s during which, the robots move in the arena according to their control software. At the end of a run, the performance of the swarm is computed using the objective function

$$F = max(N_a, N_b)/N, \tag{1}$$

where N_a and N_b are the number of e-pucks that, at the end of the run, are on a and b, respectively, and N is the total number of e-pucks. The objective function ranges from 0, when no e-puck is either on a or b, to 1, when all e-pucks are either on a or b.

Protocol. The experiment comprises two phases. In Phase 1, `EvoStick` is run 30 times for 256 iterations each. In order to evaluate the performance of the swarm at different levels of the design effort, for each run of `EvoStick` we collect the best neural network produced at four different stages: iteration 4, 16, 64, and 256. In Phase 2, we evaluate the neural networks collected in Phase 1. Each neural network is evaluated once in simulation and once in reality. The evaluation is performed under the same experimental conditions of Phase 1. Concerning the evaluation in reality, we tried to reduce human intervention as much as possible to avoid biasing the results: (1) the control software is automatically uploaded to each e-puck via the infrastructure described in [23]; (2) the performance of the swarm is formally evaluated using the objective function defined in Eq. 1

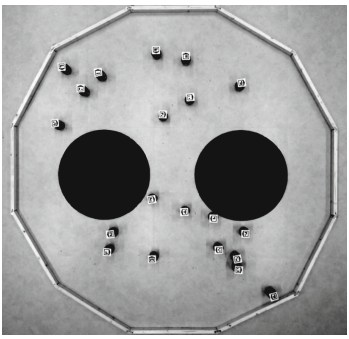


Fig. 3. Arena and twenty e-pucks

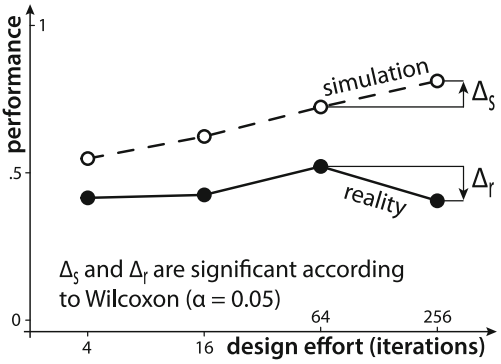


Fig. 4. Results of the experiment

and is computed automatically via the tracking system described in [50]; (3) the tracking system is also used to automatically drive the robots to random initial positions at the beginning of each evaluation.

Results. Figure 4 summarizes the results. Visually, the two curves representing the average performance in simulation and reality closely resemble the hypothetical ones that we sketched in Fig. 2d. In particular, between iteration 64 and 256 of the evolutionary algorithm, the performance in simulation increased while the one in reality decreased. To confirm that the observed trends are a genuine phenomenon rather than simply random fluctuations, we used the paired Wilcoxon signed rank test (with 95 % confidence level) to analyze the performance difference between iteration 64 and 256. We did this for both curves. In both cases, the null hypothesis we tested is that the performance at iteration 64 and 256 is the same and that the observed differences are the result of random fluctuations. As alternative hypotheses we used those suggested by Fig. 2d: from iteration 64 to 256, the performance in simulation increases while the one in reality decreases. In both cases, the observations reject the null hypothesis in favor of the alternative.

5 Conclusions

In the article, we presented results that corroborate a previously formulated hypothesis: the reality gap problem bears strong resemblance to the generalization problem faced in supervised learning. In particular, we presented an experiment that highlights a phenomenon that we shall call *overdesign*: as the training effort increases, past an optimal value, the performance that an automatically designed swarm obtains in reality diverges from the one it obtains in simulation.

The results presented in this article are preliminary, as they concern a single automatic design method and a single task. To establish overdesign as a scientific fact, further experimental work is needed and should involve a sufficiently large number of automatic design methods and tasks. Nonetheless, the results

presented here are in line with our expectations and corroborate our hypothesis. Moreover, they are in line also with similar results previously obtained in the automatic fine-tuning of the parameters of metaheuristics. Within that context, Birattari [4] devised an experiment in which an iterated local search algorithm is fine-tuned on an instance of the quadratic assignment problem and is then tested on another instance of the same problem. The author recorded the cost of the best solution found by the algorithm on the two instances as a function of the tuning effort. The results show that, past an optimal value of the tuning effort, the costs diverge: on the tuning instance the cost keeps decreasing, while on the test instance it starts increasing. In the context of the automatic fine-tuning of metaheuristics, the phenomenon observed has been named *overtuning*.

In the article, we have developed our reasoning and conducted our experiment within the domain of the automatic design of control software for robot swarms. The choice was dictated simply by the fact that this is our research domain and it is within this domain that we wish to investigate the effects of the reality gap. Moreover, we have focused on a classical evolutionary swarm robotics setting because a relatively large number of studies have been developed under this setting. Indeed, there is an established research community that operates in the framework of evolutionary swarm robotics and that could be potentially affected by our contribution. Nonetheless, we do not have any reason to doubt that the phenomenon of overdesign could be observed in other approaches to the automatic design of control software for robot swarms and for single robots, as well. The experimental methodology we adopted in the study presented is sufficiently general and straightforward to be applicable in further studies. Yet, it should be noticed that an experiment to observe overdesign can be time consuming. The experiment presented in this article comprises 120 runs with 20 e-pucks and $30 \times 100 \times 10 \times 256 + 120 = 7,680,120$ runs in simulation.

Besides shedding new light on the reality gap problem, the concepts discussed in this article and the results presented could suggest improvements to the current practice in evolutionary (swarm) robotics and more generally in the automatic design of robot control software. In particular, the results suggest that one should check whether the control software obtained upon convergence of the design process is indeed the one that perform the best in reality. Moreover, these results suggest that a form of early stopping could be beneficial.

To summarize, future work should produce further evidence that the risk of overdesign is concrete in the automatic design of control software for robot swarms. Moreover, future research could be devoted to the development of early stopping mechanisms or similar overdesign-aware techniques that could contribute to mitigate the reality gap problem.

Acknowledgments. Mauro Birattari acknowledges support from the Belgian F.R.S.–FNRS, of which he is a Senior Research Associate.

References

1. Baldassarre, G., Trianni, V., Bonani, M., Mondada, F., Dorigo, M., Nolfi, S.: Self-organised coordinated motion in groups of physically connected robots. IEEE Trans. Syst. Man Cybern. Part B **37**(1), 224–239 (2007)
2. Bauer, F., Pereverzev, S., Rosasco, L.: On regularization algorithms in learning theory. J. Complex. **23**, 52–72 (2007)
3. Berman, S., Kumar, V., Nagpal, R.: Design of control policies for spatially inhomogeneous robot swarms with application to commercial pollination. In: International Conference on Robotics and Automation, ICRA 2011, pp. 378–385. IEEE Press, Piscataway (2011)
4. Birattari, M.: Tuning Metaheuristics: A Machine Learning Perspective. Springer, Germany (2009)
5. Bongard, J., Zykov, V., Lipson, H.: Resilient machines through continuous self-modeling. Science **314**(5802), 1118–1121 (2006)
6. Bongard, J.C.: Evolutionary robotics. Commun. ACM **56**(8), 74–83 (2013)
7. Brambilla, M., Brutschy, A., Dorigo, M., Birattari, M.: Property-driven design for swarm robotics: a design method based on prescriptive modeling and model checking. ACM Trans. Auton. Adapt. Syst. **9**(4), 17.1–17.28 (2015)
8. Brambilla, M., Ferrante, E., Birattari, M., Dorigo, M.: Swarm robotics: a review from the swarm engineering perspective. Swarm Intell. **7**(1), 1–41 (2013)
9. Brooks, R.A.: Artificial life and real robots. In: Varela, F.J., Bourgine, P. (eds.) Toward a Practice of Autonomous Systems. Proceedings of the First European Conference on Artificial Life, pp. 3–10. MIT Press, Cambridge (1992)
10. Caruana, R., Lawrence, S., Giles, L.: Overfitting in neural nets: backpropagation, conjugate gradient, and early stopping. In: Leen, T., Dietterich, T., Tresp, V. (eds.) Advances in Neural Information Processing Systems 13, NIPS 2000, pp. 402–408. MIT Press (2001)
11. Christensen, A.L., Dorigo, M.: Evolving an integrated phototaxis and hole-avoidance behavior for a swarm-bot. In: Artificial Life, ALIFE 2006, pp. 248–254. MIT Press, Cambridge (2006)
12. Cybenko, G.: Approximations by superpositions of a sigmoidal function. Math. Control Signals Syst. **2**(4), 303–314 (1989)
13. Dietterich, T., Kong, E.B.: Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Department of Computer Science, Oregon State University (1995)
14. Doncieux, S., Mouret, J.B.: Beyond black-box optimization: a review of selective pressures for evolutionary robotics. Evol. Intell. **7**(2), 71–93 (2014)
15. Duarte, M., Oliveira, S.M., Christensen, A.L.: Evolution of hierarchical controllers for multirobot systems. In: Artificial Life, ALIFE 2014, pp. 657–664. MIT Press, Cambridge (2014)
16. Duarte, M., Oliveira, S.M., Christensen, A.L.: Hybrid control for large swarms of aquatic drones. In: Artificial Life, ALIFE 2014, pp. 785–792. MIT Press, Cambridge (2014)
17. Duarte, M., Costa, V., Gomes, J.C., Rodrigues, T., Silva, F., Oliveira, S.M., Christensen, A.L.: Evolution of collective behaviors for a real swarm of aquatic surface robots. arXiv-CoRR abs/1511.03154 (2015)
18. Floreano, D., Husbands, P., Nolfi, S.: Evolutionary robotics. In: Siciliano, B., Khatib, O. (eds.) Handbook of Robotics, pp. 1423–1451. Springer, Germany (2008)

19. Francesca, G., Birattari, M.: Automatic design of robot swarms: achievements and challenges. Front. Robot. AI **3**(29), 1–9 (2016)
20. Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., Reina, A., Soleymani, T., Salvaro, M., Pinciroli, C., Mascia, F., Trianni, V., Birattari, M.: AutoMoDe-Chocolate: automatic design of control software for robot swarms. Swarm Intell. **9**(2/3), 125–152 (2015)
21. Francesca, G., et al.: An experiment in automatic design of robot swarms. In: Dorigo, M., Birattari, M., Garnier, S., Hamann, H., Montes de Oca, M., Solnon, C., Stützle, T. (eds.) ANTS 2014. LNCS, vol. 8667, pp. 25–37. Springer, Heidelberg (2014)
22. Francesca, G., Brambilla, M., Brutschy, A., Trianni, V., Birattari, M.: AutoMoDe: a novel approach to the automatic design of control software for robot swarms. Swarm Intell. **8**(2), 89–112 (2014)
23. Garattoni, L., Francesca, G., Brutschy, A., Pinciroli, C., Birattari, M.: Software infrastructure for e-puck (and TAM). Technical report 2015–004, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (2015)
24. Gauci, M., Chen, J., Li, W., Dodd, T.J., Groß, R.: Self-organized aggregation without computation. Int. J. Robot. Res. **33**(8), 1145–1161 (2014)
25. Gauci, M., Chen, J., Li, W., Dodd, T.J., Groß, R.: Clustering objects with robots that do not compute. In: Lomuscio, A., et al. (eds.) Autonomous Agents and Multiagent Systems, AAMAS 2014, pp. 421–428. IFAAMAS, Richland (2014)
26. Geman, S., Bienenstock, E., Doursat, R.: Neural networks and the bias/variance dilemma. Neural Comput. **4**(1), 1–58 (1992)
27. Gutiérrez, Á., Campo, A., Dorigo, M., Donate, J., Monasterio-Huelin, F., Magdalena, L.: Open e-puck range & bearing miniaturized board for local communication in swarm robotics. In: International Conference on Robotics and Automation, ICRA 2009, pp. 3111–3116. IEEE Press, Piscataway (2009)
28. Hamann, H., Wörn, H.: A framework of space-time continuous models for algorithm design in swarm robotics. Swarm Intell. **2**(2), 209–239 (2008)
29. Hecker, J.P., Letendre, K., Stolleis, K., Washington, D., Moses, M.E.: Formica ex machina: ant swarm foraging from physical to virtual and back again. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Engelbrecht, A.P., Groß, R., Stützle, T. (eds.) ANTS 2012. LNCS, vol. 7461, pp. 252–259. Springer, Heidelberg (2012)
30. Jacobi, N., Husbands, P., Harvey, I.: Noise and the reality gap: the use of simulation in evolutionary robotics. In: Morán, F., et al. (eds.) Advances in Artificial Life. LNCS (LNAI), vol. 929, pp. 704–720. Springer, London (1995)
31. Jakobi, N.: Evolutionary robotics and the radical envelope-of-noise hypothesis. Adapt. Behav. **6**(2), 325–368 (1997)
32. Kazadi, S., Lee, J.R., Lee, J.: Model independence in swarm robotics. Int. J. Intell. Comput. Cybern. **2**(4), 672–694 (2009)
33. Koos, S., Mouret, J., Doncieux, S.: The transferability approach: crossing the reality gap in evolutionary robotics. IEEE Trans. Evol. Comput. **17**(1), 122–145 (2013)
34. Lopes, Y.K., Trenkwalder, S.M., Leal, A.B., Dodd, T.J., Groß, R.: Supervisory control theory applied to swarm robotics. Swarm Intell. **10**(1), 65–97 (2016)
35. McCulloch, W., Pitts, W.: A logical calculus of ideas immanent in nervous activity. Bull. Math. Biophys. **5**(4), 115–133 (1943)
36. Miglino, O., Lund, H.H., Nolfi, S.: Evolving mobile robots in simulated and real environments. Artif. Life **2**(4), 417–434 (1995)
37. Minsky, M., Papert, S.: Perceptrons: An Introduction to Computational Geometry. MIT Press, Cambridge (1969)

38. Mondada, F., et al.: The e-puck, a robot designed for education in engineering. In: 9th Conference on Autonomous Robot Systems and Competitions, pp. 59–65. Instituto Politécnico de Castelo Branco, Portugal (2009)
39. Morgan, N., Bourlard, H.: Generalization and parameter estimation in feedforward nets: some experiments. In: Touretzky, D. (ed.) Advances in Neural Information Processing Systems 2, NIPS 1990, pp. 630–637. Morgan Kaufman, San Mateo (1990)
40. Nolfi, S., Floreano, D.: Evolutionary Robotics. MIT Press, Cambridge (2000)
41. Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L.M., Dorigo, M.: ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intell. **6**(4), 271–295 (2012)
42. Prechelt, L.: Early stopping-but when? In: Orr, G.B., Müller, K.-R. (eds.) NIPS-WS 1996. LNCS, vol. 1524, pp. 55–59. Springer, Heidelberg (1998)
43. Quinn, M., Smith, L., Mayley, G., Husbands, P.: Evolving controllers for a homogeneous system of physical robots: structured cooperation with minimal sensors. Philos. Trans. Royal Soc. London A Math. Phys. Eng. Sci. **361**(1811), 2321–2343 (2003)
44. Raskutti, G., Wainwright, M.J., Yu, B.: Early stopping and non-parametric regression: an optimal data-dependent stopping rule. J. Mach. Learn. Res. **15**, 335–366 (2014)
45. Reina, A., Valentini, G., Fernández-Oto, C., Dorigo, M., Trianni, V.: a design pattern for decentralised decision making. PLoS ONE **10**(10), e0140950 (2015)
46. Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain. Psychol. Rev. **65**(6), 386–408 (1958)
47. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
48. Silva, F., Duarte, M., Correia, L., Oliveira, S.M., Christensen, A.L.: Open issues in evolutionary robotics. Evol. Comput. (2016, in press)
49. Stone, M.: Cross-validatory choice and assessment of statistical predictions. J. Royal Stat. Soc. Ser. B (Methodol.) **36**(2), 111–147 (1974)
50. Stranieri, A., Turgut, A., Salvaro, M., Garattoni, L., Francesca, G., Reina, A., Dorigo, M., Birattari, M.: IRIDIA's arena tracking system. Technical report 2013–013, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (2013)
51. Trianni, V., Nolfi, S.: Self-organising sync in a robotic swarm. A dynamical system view. IEEE Trans. Evol. Comput. **13**(4), 722–741 (2009)
52. Trianni, V.: Evolutionary Swarm Robotics. Springer, Germany (2008)
53. Trianni, V.: Evolutionary robotics: model or design? Front. Robot. AI **1**(13), 1–6 (2014)
54. Waibel, M., Keller, L., Floreano, D.: Genetic team composition and level of selection in the evolution of cooperation. IEEE Trans. Evol. Comput. **13**(3), 648–660 (2009)
55. Werbos, P.: Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Ph.D. thesis, Harvard University, Cambridge (1974)
56. Werfel, J., Petersen, K., Nagpal, R.: Designing collective behavior in a termite-inspired robot construction team. Science **343**(6172), 754–758 (2014)
57. Wolpert, D.: On bias plus variance. Neural Comput. **9**(6), 1211–1243 (1997)