AutoMoDe-IcePop: Automatic Modular Design of Control Software for Robot Swarms Using Simulated Annealing

Jonas Kuckling(✉), Keneth Ubeda Arriaza, and Mauro Birattari

IRIDIA, Université Libre de Bruxelles, Brussels, Belgium
{jonas.kuckling,mbiro}@ulb.ac.be

Abstract. Prior research has shown that the optimization algorithm is an integral part of the automatic modular off-line design of control software for robot swarms and can have great influence on the quality of the control software produced. In this paper we investigate, whether a stochastic local search metaheuristic—simulated annealing—can be used as the optimization algorithm in the automatic modular design of robot swarms. The results indicate that simulated annealing is indeed a viable candidate. Additionally, we investigate the influence of some obvious variations of simulated annealing on the performance of the automatic modular design.

Keywords: Swarm robotics · Automatic design · Simulated annealing

1 Introduction

Designing control software for a robot swarm is a challenging task, as the global desired behavior usually emerges from the interactions of the robots between each other and the environment [10,37]. Manual software design therefore often relies on trial-and-error [4] and a general methodology for designing control software for robot swarms is still missing [12].

Automatic design offers a promising alternative, by transforming the design problem into an optimization problem. Instead of writing control software that performs a specific mission, a target architecture is optimized with regard to a mission-dependent objective function. A popular automatic design approach is neuro-evolutionary swarm robotics which uses evolutionary algorithms to design artificial neural networks. While this approach has successfully been applied to many missions [8,11,21,33,35,36], multiple challenges remain to be solved [5,31,34]. The most important is the weak transferability of the generated control software, resulting in performance drops when deployed in reality. This drop

JK and KUA contributed equally to this work and should be considered as co-first authors. The experiments were designed by JK and performed by KUA. The paper was drafted by JK and edited by MB; all authors read and commented the final version. The research was directed by MB.

B. Bogaerts et al. (Eds.): BNAIC 2019/BENELEARN 2019, CCIS 1196, pp. 3–17, 2020.
https://doi.org/10.1007/978-3-030-65154-1_1

in performance is often associated with the reality gap—inherent differences between the design context of the simulation and the real world.

Francesca et al. [14] see in this phenomenon a resemblance to the problem of over-fitting in machine learning. Analogous to the bias-variance trade-off [9,17], they propose to introduce a bias to the automatic design process. Their proposed bias is a restriction of possible control software, by defining a control architecture which can be composed through the combination of predefined modules. As a proof of concept, Francesca et al. implemented AutoMoDe-`Vanilla`, an automatic modular design approach that generates finite-state machines with up to four states. Such generated finite-state machines are composed of states, which will execute an associated behavior as long as they are active, and transitions, that have an associated probabilistic condition which can trigger the transition from one state to another. `Vanilla` uses F-race [2] to combine the finite-state machines out of a set of predefined modules (behaviors and conditions) and to fine tune their parameters.

With AutoMoDe-`Chocolate` [13], Francesca et al. implemented a variant of `Vanilla` that differs only in the optimization algorithm employed. `Chocolate` uses Iterated F-race [3], instead of F-race. The results of their experiments show that `Chocolate` performs significantly better than `Vanilla` on many missions. Given that the only difference between the two methods is the optimization algorithm it seems apparent that the optimization algorithm is an important part of the automatic modular design approach and can have a great influence on the performance of generated control software. Following up on this observation, we create `IcePop`, another instance of AutoMoDe. It is functionally similar to `Chocolate` and `Vanilla` but it uses simulated annealing as an optimization algorithm. We choose simulated annealing because it is a well-studied algorithm [6,19,26,29,32] that has found many applications (for surveys see for example [1] and [32]).

Simulated annealing is a metaheuristic inspired by the thermodynamical process of annealing [23]. At higher temperatures the particles in a crystal are more excited and can move more freely than at lower temperatures. Similarly, the simulated annealing algorithm has a "temperature" parameter. When it is high, the algorithm has a chance to accept worsening solutions, mimicking the free movement of the particles. At lower temperatures, the algorithm will select worsening solutions less likely, thus constraining the movement of the solution candidate. Simulated annealing has shown properties that are desirable for the automatic design of control software. It has been shown to effectively traverse the search space and to converge quickly towards promising solutions [22]. This allows an efficient use of the allocated budget. Furthermore, simulated annealing contains mechanisms to escape local optima—e.g., by accepting worsening moves at higher temperatures. Without any a priori knowledge of the shape of the search space, this is an important property as it reduces the risk of premature convergence to suboptimal solutions.

The rest of this paper is structured as follows: In Sect. 2 we present the experimental setup that we used—the robotic platform, the design methods

and the experimental protocol. In Sect. 3 we present four experiments and their results. In Sect. 4 we summarize our findings and give an outlook to future work.

2 Experimental Setup

In this section we describe the experimental setup and protocol that was used to obtain the results described in Sect. 3.

2.1 Robotic Platform

`IcePop` designs control software for a swarm of modified e-puck robots [16,30]. The e-puck robots are equipped with two wheels, whose velocity can be adjusted independently, three ground sensors that can perceive the greyscale color value of the floor, and eight IR transceivers that are spaced equally around the robot, that can perceive proximity and light values. The robot is also equipped with a range-and-bearing board [18] that comprises twelve IR emitters and twelve receivers equally distributed along the perimeter of the board and pointed radially and outwards, on the horizontal plane. The range-and-bearing board allows the e-puck to send and receive messages within a range of 0.7m. In order to abstract the actual sensor configuration, we use a reference model [20]. Specifically, we use RM1.1 (see Table 1), the reference model that was used to define the modules of `Chocolate`.

In this reference model, each robot has eight light and proximity sensors returning floating point values between 0 and 1. $prox_i$ and $light_i$ define the proximity and light reading for the ith sensor respectively. Three ground sensors ($ground_i$) return one of three values, indicating whether the ground underneath them is black, gray or white. The reference model uses the range-and-bearing board to count the number of neighbors in communication range (n) and calculates an attraction vector (V_d) towards the center of mass of all perceived robots. Additionally the robot has two wheels, whose velocity can be adjusted independently (v_l and v_r for the velocity of the left wheel and the right wheel respectively).

2.2 Automatic Design Methods

We compare two automatic modular design methods: `Chocolate` and `IcePop`. `Chocolate` [13] generates probabilistic finite-state machines with up to four states. For that it uses a set of six behaviors and six conditions that are defined on top of RM1.1 [20]. The six behaviors are: exploration, stop, phototaxis, anti-phototaxis, attraction and repulsion. The six conditions are: black-floor, gray-floor, white-floor, neighbor-count, inverted-neighbor-count and fixed-probability. For a detailed description of the modules, we refer the reader to their original definition [14]. The optimization algorithm used by `Chocolate` is Iterated F-race [27].

Table 1. Reference model RM1.1 [20]. Sensors and actuators of the e-puck robot. The period of the control cycle is 100 ms.

Sensor/Actuator	Parameters	Values
proximity	$prox_i$, with $i \in \{0, \ldots, 7\}$	$[0, 1]$
light	$light_i$, with $i \in \{0, \ldots, 7\}$	$[0, 1]$
ground	$ground_i$, with $i \in \{0, \ldots, 2\}$	$\{black, gray, white\}$
range-and-bearing	n	$\{0, \ldots, 19\}$
	V_d	$([0, 0.7]m, [0, 2\pi]$ radian)
wheels	v_l, v_r	$[-0.12, 0.12]$ m/s

Algorithm 1. Component-based simulated annealing algorithm

```
best solution s* := incumbent solution ŝ := s_0
i := 0
T_0 := initialize temperature according to initial temperature
while stopping criterion is not met do
    choose a solution s_{i+1} in the neighborhood of ŝ according to exploration criterion
    if s_{i+1} meets acceptance criterion then
        ŝ := s_{i+1}
        if ŝ improves over s* then
            s* := ŝ
        end if
    end if
    if temperature length steps since last temperature update then
        update temperature according to cooling scheme;
    end if
    reset temperature according to temperature restart mechanism;
    i := i + 1
end while
return s*
```

In this paper, we propose `IcePop`. It is based on `Chocolate`, as it uses the same modules and target architecture. The difference between the two methods is that `IcePop` adopts the component-based simulated annealing algorithm (see Algorithm 1) as the optimization algorithm. Franzin and Stützle proposed this component-based algorithm in an effort to unify many variants of the simulated annealing algorithm [15]. We choose to adopt this algorithm because it provides the flexibility to easily change components should the need arise.

The component-based simulated annealing algorithm contains placeholders for commonly used components. In Table 2, we present our choices of components that we use in the implementation of the simulated annealing for `IcePop`. The initial solution supplied to the algorithm is a minimal valid instance of control software. In our case this is a finite-state machine with exactly one state executing the stop behavior. The neighborhood function is implicitly defined through the application of a random valid perturbation operator. In `IcePop`, we

Table 2. Configuration of the simulated annealing algorithm.

Component	Type	Parameter
Initial solution	Minimal controller	Stop behavior
Neighborhood	Defined through perturbation operators	
Initial temperature	Fixed value	$T_0 = 125.0$
Stopping criterion	Budget of simulations	50000 simulations
Exploration criterion	Random exploration	Valid perturbation operators
Acceptance criterion	Metropolis condition	Mean with 10 samples
Temperature length	Fixed value	$T_{length} = 1$
Cooling scheme	Geometric cooling	$\alpha = 0.9782$
Temperature restart	Fixed value	Every 5000 simulations

have defined eleven perturbation operators: adding a state, removing a state, adding a transition, removing a transition, changing the initial state, changing the starting point of a transition, changing the end point of a transition, changing the behavior associated with a state, changing the condition associated with a transition, changing the parameters of a behavior, and changing the parameters of a condition. The initial temperature is set to 125.0. The stopping criterion is defined as a maximum budget of simulation runs. That is, after the allocated budget of simulation runs is exhausted, the algorithm should return the final instance of control software. The exploration criterion selects a random valid perturbation operator and applies it on the incumbent solution. The acceptance criterion is the Metropolis condition [23,28] that accepts or rejects new solutions based on their performance. The Metropolis condition will always accept an improving solution, and will accept a worsening solution with probability $\exp(-(e - e')/T)$ where T is the current temperature, e is quality of the currently best solution and e' is the quality of the perturbed solution. Because the performance of each instance of control software is stochastic, e and e' will be computed as the mean of a sample of 10 runs of the respective instance of control software. The temperature length determines the number of steps before the temperature cools down again. We set the value to 1, so that the cooling happens in every step. The cooling scheme that is then applied is the geometric cooling [23]. In geometric cooling, the updated temperature is computed as $T * \alpha$, where T is the current temperature and α is the cooling coefficient, which we set as $\alpha = 0.9782$. Additionally, the temperature resets to the initial value every 5000 simulations.

The source code of our implementation of `IcePop` is available at: https://github.com/keua/design-of-robot-swarms-by-optimization

2.3 Missions

All experiments were conducted with 20 robots on two missions Aggregation with Ambient Cues (AAC) and Foraging.

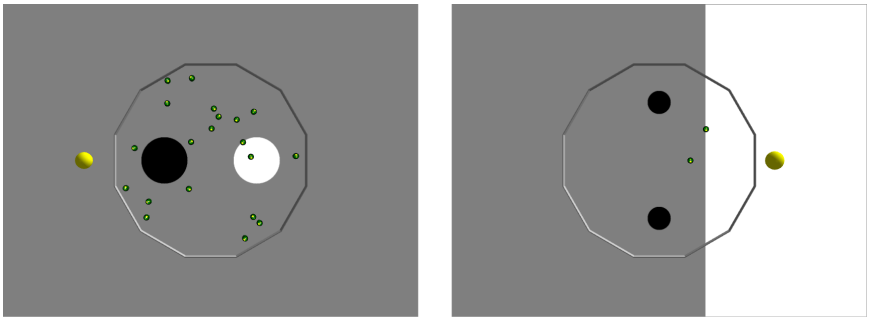

Fig. 1. The two missions: AAC (*left*) and FORAGING (*right*).

AAC. The arena contains two circles, one black, one white. A light source is placed on the side of the arena that contains the black circle (Fig. 1, left). The robots are tasked to aggregate on the black spot. The objective function $F_{\text{AAC}} = \sum_{t=0}^{T} N_t$ where N_t is the number of robots on the black circle at time step N_t.

Foraging. The arena contains two source areas in the form of black circles and a nest, as a white area. A light source is placed behind the nest to help the robots to navigate (Fig. 1, right). As the robots have no gripping capabilities, we consider an idealized version of foraging, where a robot is deemed to retrieve an object when it enters a source and then the nest. The goal of the swarm is to retrieve as many objects as possible. The objective function is $F_{\text{F}} = N_i$, where N_i is the number of retrieved objects.

2.4 Protocol

As each design process is stochastic, we run 20 independent designs for each design method, resulting in 20 instances of control software. The so obtained instances are then each assessed 10 times in the design context (what we call simulation) and another 10 times in a different simulation setting (what we call pseudo-reality). Pseudo-reality is a concept to evaluate the transferability of control software [25]. Instead of assessing the performance directly in reality, a different simulation context is used. Research has shown that control software that transfers well into reality also transfers well into pseudo-reality, while control software that transfers badly into reality also transfers badly into pseudo-reality.

The results are presented in notched box-and-whisker boxplots, giving a visual representation of the samples. In such a notched box-and-whisker boxplot, the horizontal thick line denotes the median of the sample. The lower and upper sides of the box are called upper and lower hinges and represent the 25th and 75th percentile of the observations, respectively. The upper whisker extends either up to the largest observation or up to 1.5 times the difference between upper hinge and median—whichever is smaller. The lower whisker is defined

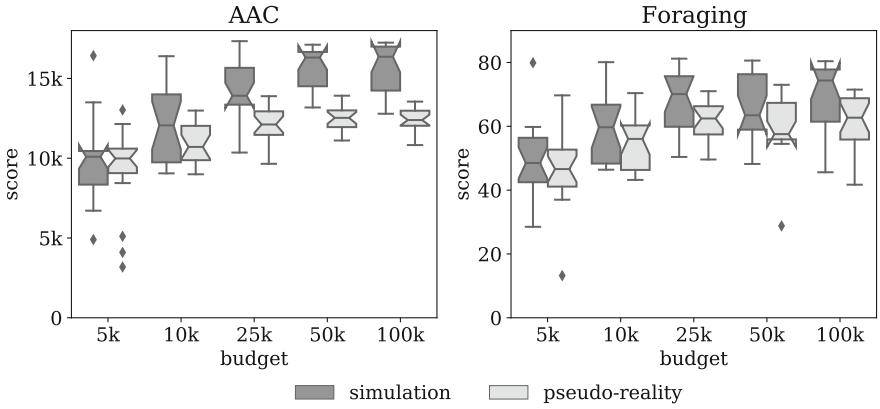

Fig. 2. Performance of control software created by `IcePop` for different budgets.

analogously. Small circles represent outliers (if any), that are observations that fall beyond the whiskers. Notches extend to $\pm 1.58 IQR/\sqrt{n}$, where IQR is the interquartile range and n = 20 is the number of observations. Notches indicate the 95% confidence interval on the position of the median. If the notches of two boxes do not overlap, the observed difference between the respective medians is significant [7].

3 Results

In this section we describe four experiments we conducted and the results we obtained. The instances of control software produced, the details of their performances, and videos of their execution on the robots are available as online supplementary material [24]. We also discuss possible reasons for the results.

3.1 Influence of the Budget

We conduct one experiment to investigate the influence of the budget on the performance of the generated control software. Designs with a smaller budget need less time to finish but usually produce results that perform less well in simulation. The higher the time the better usually the performance in simulation, but an overdesigning effect might be observed, where the improvement in simulation does not carry over to reality. We tested `IcePop` with five different budgets (5000, 10000, 25000, 50000 and 100000 simulations respectively).

The results displayed in Fig. 2 show the influence of the budget on the performance of the control software generated by `IcePop`. One trend that is apparent from the data, is that, as expected, a larger design budgets leads to control software that performs better in simulation. However the relative improvement diminishes and the performance seems to reach a peak around a budget of 50000 simulations.

Furthermore the performance in pseudo-reality improves initially with an increased budget. Here, however, the performance levels after the budget of 25000 simulations is reached and does not improve any further. This could be an indicator that the design reached the peak performance that is still transferable. Further designs might improve the performance in simulation but the transferability will suffer in return.

3.2 Influence of the Sample Size

We chose the Metropolis condition as the acceptance criterion in the component-based simulated annealing for `IcePop`. In its original definition it was defined to compare two single performance measures. As the evaluation of the performance of an instance of control software is stochastic, we sample several simulation runs. The mean of this sample is then supplied to the Metropolis condition.

In a second experiment, we investigate the influence of the sample size on the performance of the generated control software. Smaller sample sizes use less of the budget to evaluate one solution, allowing more solution candidates to be investigated. On the other hand, outliers will have a greater impact on the mean of the samples and thus the perceived performance. Larger sample sizes lead to the inverse effect. Fewer total solution candidates would be investigated but the performance of each individual solution candidate is more robust to outliers. We study the influence of the sample size on the performance of the generated control software by evaluating the performance in simulation and in pseudo-reality for three sample sizes: 5, 10, and 15. Additionally we test every variant on the three budgets that showed peak performance in the previous experiment (25000, 50000, and 100000 simulations).

Figure 3 shows the results for the three different variants of the sample size over the three investigated budgets. For a budget of 25000 simulations, all variants perform similar and no differences can be seen, both in simulation and pseudo-reality. In the case of a budget of 50000, the variant with a sample size of 10 samples performs slightly better than the other two variants, in the mission FORAGING when assessed in simulation. In pseudo-reality, this difference however is not present anymore. It could therefore very well be that this is simply a statistical artifact of the stochastic design process. For 100000 simulation runs, the three variants achieve a comparable performance again and only minor differences can be observed. All in all, the three different sample sizes that we compared show no noticeable differences.

3.3 Influence of the Restarting Mechanism

We conduct a third experiment, to investigate the influence of the restarting mechanism. Restarting resets the temperature to a higher value, allowing the design process to make bigger movements in the search space again. We investigate four different restarting mechanisms: fixed length (restarts after a fixed number of simulations, in this case every 5000 simulations), no restart (the temperature cools over the whole design process and is never restarted), reheat (the

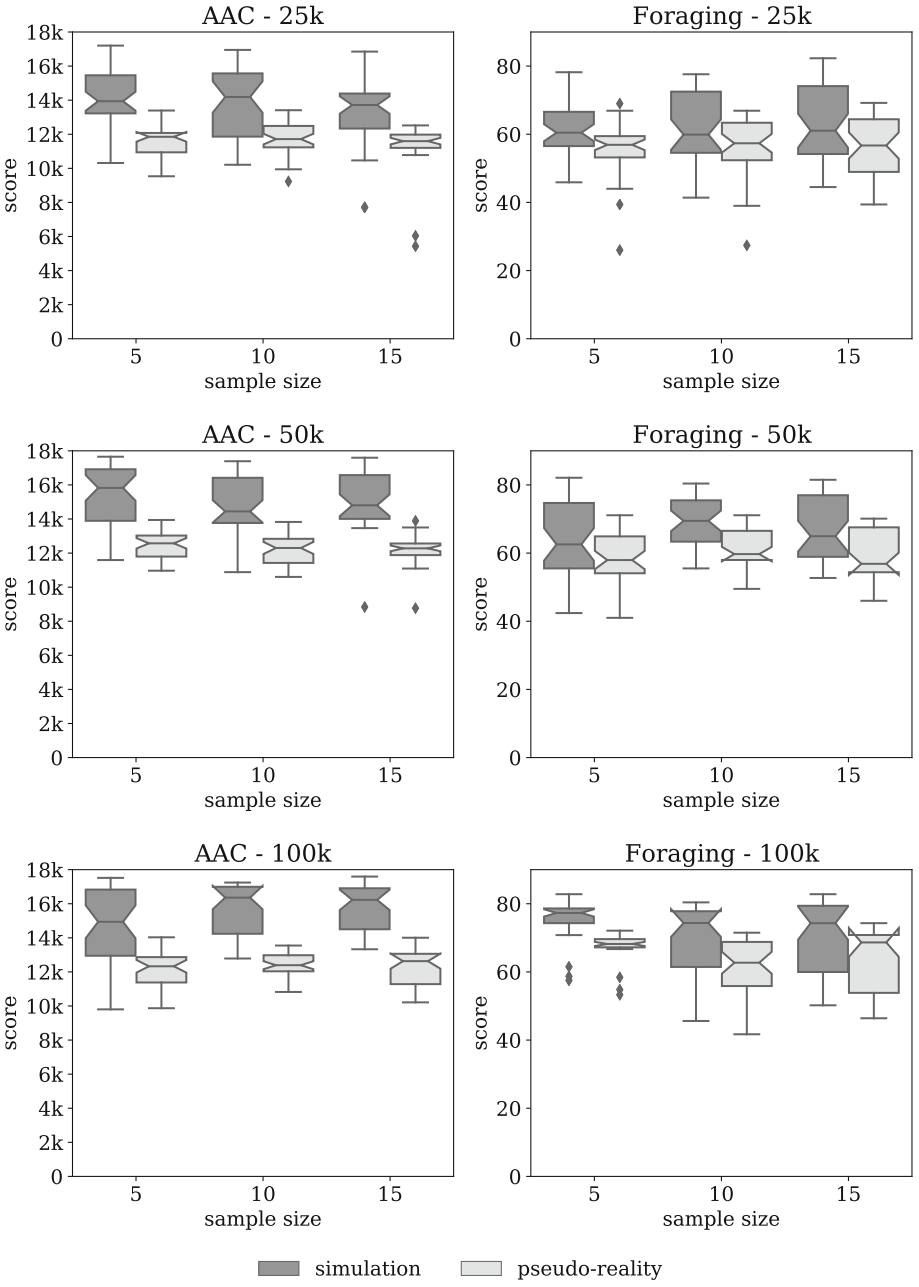

Fig. 3. Influence of the sample size.

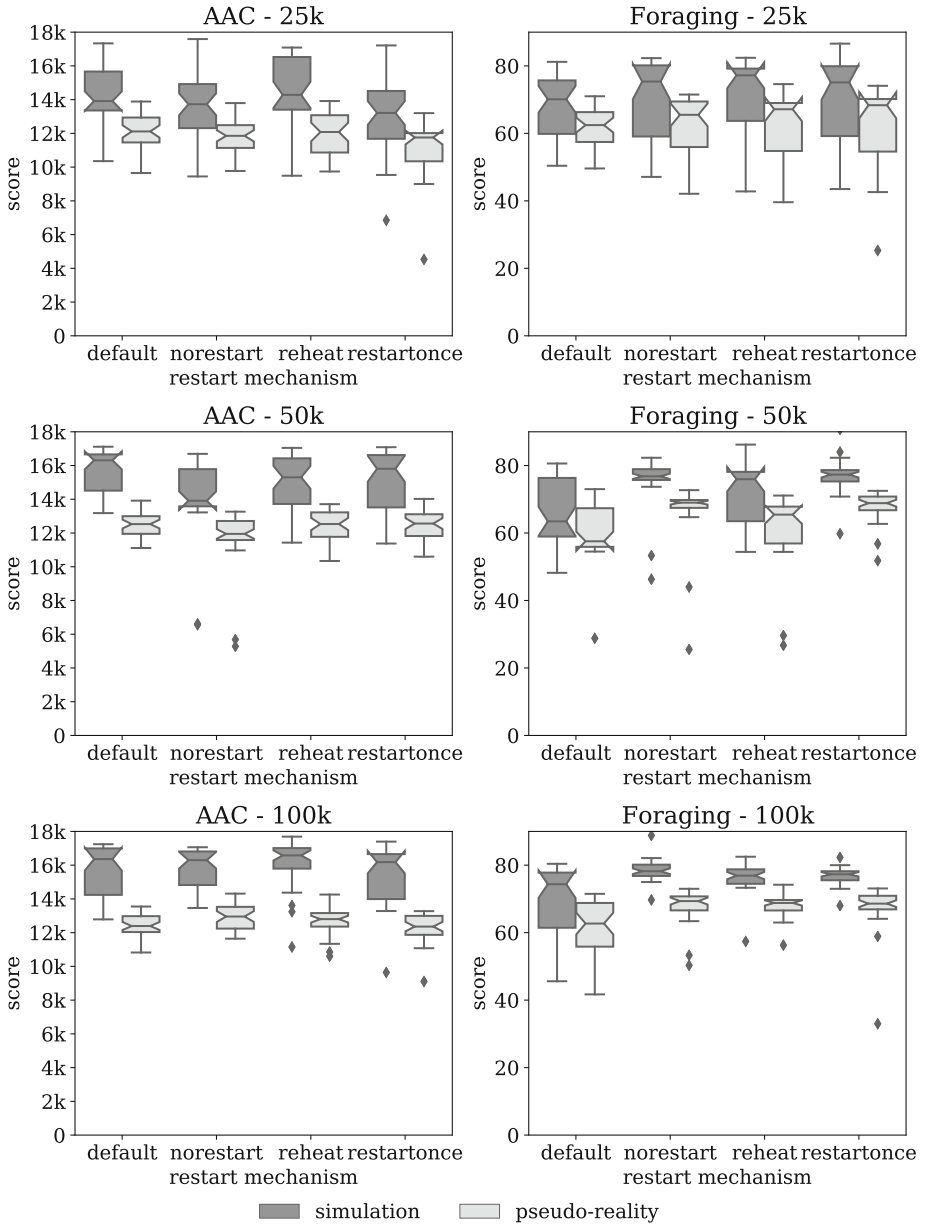

Fig. 4. Influence of the restart mechanism.

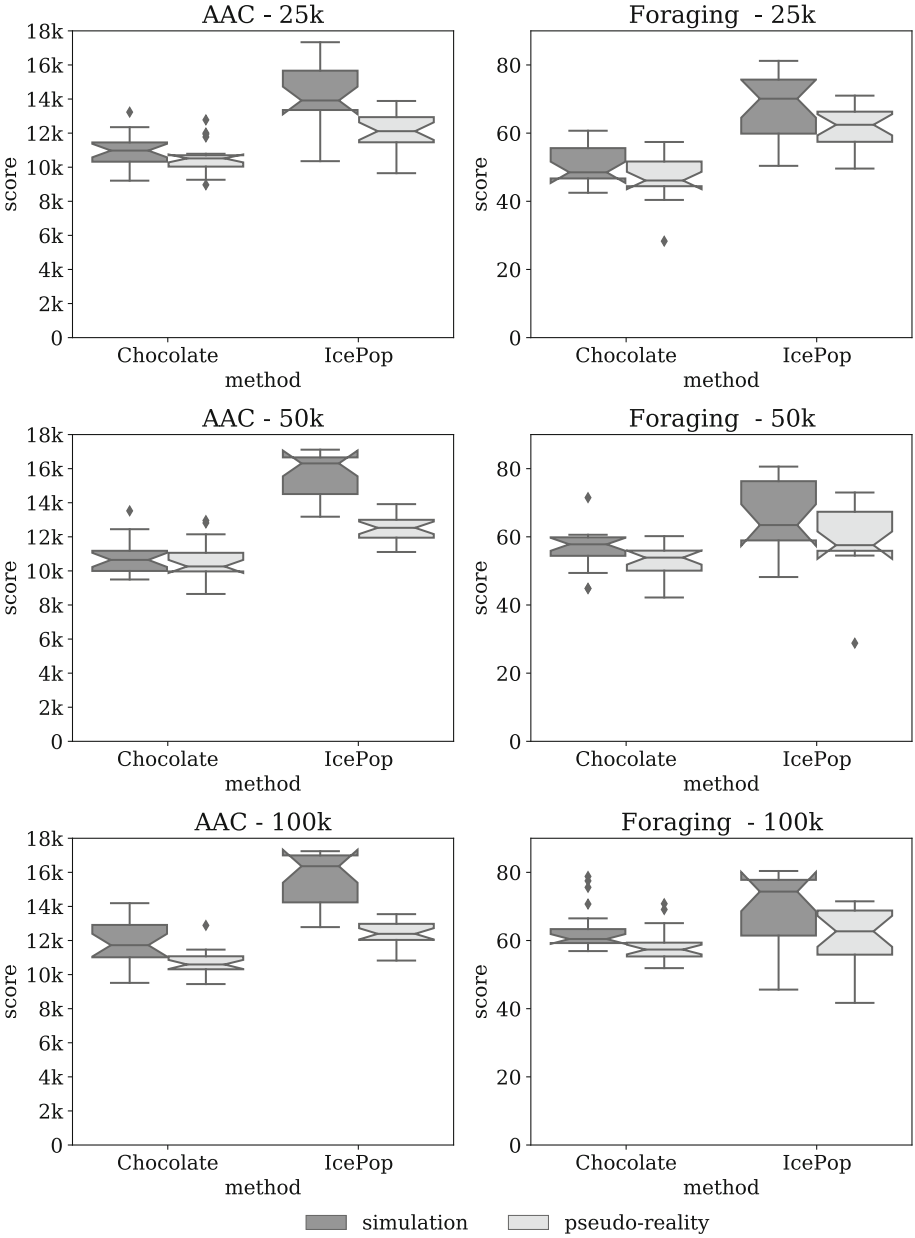

Fig. 5. Comparison between `Chocolate` and `IcePop`.

temperature is reset every 5000 simulations, the new temperature is set to the one that generated the biggest improvement so far), restart once (after the half of the budget is exhausted the temperature resets). We test all restarting mechanisms on budgets of 25000, 50000 and 10000 simulations.

Figure 4 shows the results for the different restarting mechanisms. The results for a budget of 25000 simulation runs show no difference between the four variants. In case of a budget of 50000 simulation runs all variants perform similarly in the mission AAC. In the mission FORAGING, the restarting mechanism that restarts every 5000 simulation runs performs worse than the other three variants. For a budget of 100000 simulation runs, all four variants perform similarly again. In the mission FORAGING, however, the fixed length restarting mechanism (default) shows a larger distribution than the other three variants.

In conclusion, the four different variants fail to produce noticeable differences in the performance of the generated control software.

3.4 Comparison with `Chocolate`

In the last experiment, we compare the performance of `IcePop` with `Chocolate` across three different budgets (25000, 50000 and 100000 simulations).

Figure 5 shows the comparison results of `IcePop` with `Chocolate` for budgets of 25000, 50000, and 100000 simulations respectively. Throughout all three budgets, it is apparent that `IcePop` performs better in simulation than `Chocolate` in both missions. In the mission AAC, the difference in performance is statistically significant.

Unfortunately the drop of performance when assessed in pseudo-reality is slightly larger for `IcePop` than for `Chocolate`. This could indicate that `IcePop` might be less transferable to real robots than `Chocolate`. Despite the larger performance drop, `IcePop` still performs better in pseudo-reality, and in AAC this difference in performance is also statistically significant.

Additionally, we have taken the best performing instance of control software of `IcePop` and `Chocolate` (with a design budget of 100k simulations) for each mission and directly applied it to a swarm of twenty real e-pucks. Videos of the performance of the control software on real robots can be found online [24].

4 Conclusions

In this work we have investigated a default configuration for simulated annealing in the context of automatic modular design. The results indicate that simulated annealing can be a viable candidate for the automatic modular design of robot swarms. Additionally, we have investigated the influence of some obvious variations to the simulated annealing on the performance of the automatic modular design. The component-based simulated annealing approach allowed us to easily implement these variants.

Simulated annealing is a well studied optimization algorithm with many proposed extensions, improvements and variants. A next step could be finding a

suitable configuration of components that satisfies best the demands of the automatic modular design. Also, it would be interesting to apply `IcePop` to a broader range of missions.

Acknowledgements. The project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681872). Jonas Kuckling and Mauro Birattari acknowledge support from the Belgian *Fonds de la Recherche Scientifique* – FNRS.

References

1. Aarts, E., Korst, J., Michiels, W.: Simulated annealing. In: Burke, E.K., Kendall, G. (eds.) Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, pp. 187–210. Springer, Boston (2005). https://doi.org/10.1007/0-387-28356-0_7
2. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: Langdon, W.B., et al. (eds.) GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 11–18. Morgan Kaufmann Publishers, San Francisco (2002)
3. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: `F-Race` and iterated `F-Race`: an overview. In: Bartz-Beielstein, T., Chiarandini, M., Paquete, L., Preuss, M. (eds.) Experimental Methods for the Analysis of Optimization Algorithms, pp. 311–336. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-02538-9_13
4. Brambilla, M., Ferrante, E., Birattari, M., Dorigo, M.: Swarm robotics: a review from the swarm engineering perspective. Swarm Intell. **7**(1), 1–41 (2013). https://doi.org/10.1007/s11721-012-0075-2
5. Bredeche, N., Haasdijk, E., Prieto, A.: Embodied evolution in collective robotics: a review. Front. Robot. AI **5**, 12 (2018). https://doi.org/10.3389/frobt.2018.00012
6. Burke, E.K., Bykov, Y.: The late acceptance hill-climbing heuristic. Eur. J. Oper. Res. **258**(1), 70–78 (2017). https://doi.org/10.1016/j.ejor.2016.07.012
7. Chambers, J.M., Cleveland, W.S., Kleiner, B., Tukey, P.A.: Graphical Methods For Data Analysis. CRC Press, Belmont (1983)
8. Christensen, A.L., Dorigo, M.: Evolving an integrated phototaxis and hole-avoidance behavior for a swarm-bot. In: Rocha, L.M., Yaeger, L.S., Bedau, M.A., Floreano, D., Goldstone, R.L., Vespignani, A. (eds.) Artificial Life - ALIFE, pp. 248–254. MIT Press, Cambridge (2006). A Bradford Book
9. Dietterich, T.G., Kong, E.B.: Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Department of Computer Science, Oregon State University, Corvallis, OR, USA (1995)
10. Dorigo, M., Birattari, M., Brambilla, M.: Swarm robotics. Scholarpedia **9**(1), 1463 (2014). https://doi.org/10.4249/scholarpedia.1463
11. Ferrante, E., Turgut, A.E., Duéñez-Guzmán, E.A., Dorigo, M., Wenseleers, T.: Evolution of self-organized task specialization in robot swarms. PLoS Comput. Biol. **11**(8), e1004273 (2015). https://doi.org/10.1371/journal.pcbi.1004273
12. Francesca, G., Birattari, M.: Automatic design of robot swarms: achievements and challenges. Front. Robot. AI **3**(29), 1–9 (2016). https://doi.org/10.3389/frobt.2016.00029

13. Francesca, G., et al.: AutoMoDe-Chocolate: automatic design of control software for robot swarms. Swarm Intell. **9**(2–3), 125–152 (2015). https://doi.org/10.1007/s11721-015-0107-9
14. 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). https://doi.org/10.1007/s11721-014-0092-4
15. Franzin, A., Stützle, T.: Revisiting simulated annealing: a component-based analysis. Comput. Oper. Res. **104**, 191–206 (2019). https://doi.org/10.1016/j.cor.2018.12.015
16. Garattoni, L., Francesca, G., Brutschy, A., Pinciroli, C., Birattari, M.: Software infrastructure for e-puck (and TAM). Technical report TR/IRIDIA/2015-004, IRIDIA, Université libre de Bruxelles, Belgium (2015)
17. Geman, S., Bienenstock, E., Doursat, R.: Neural networks and the bias/variance dilemma. Neural Comput. **4**(1), 1–58 (1992). https://doi.org/10.1162/neco.1992.4.1.1
18. 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: Kosuge, K. (ed.) IEEE International Conference on Robotics and Automation, ICRA, pp. 3111–3116. IEEE, Piscataway (2009). https://doi.org/10.1109/ROBOT.2009.5152456
19. Hajek, B.: Cooling schedules for optimal annealing. Math. Oper. Res. **13**(2), 311–329 (1988). https://doi.org/10.1287/moor.13.2.311
20. Hasselmann, K., et al.: Reference models for AutoMoDe. Technical report TR/IRIDIA/2018-002, IRIDIA, Université libre de Bruxelles, Belgium (2018)
21. Hauert, S., Zufferey, J.C., Floreano, D.: Evolved swarming without positioning information: an application in aerial communication relay. Auton. Robots **26**(1), 21–32 (2009). https://doi.org/10.1007/s10514-008-9104-9
22. Hoos, H., Stützle, T.: Stochastic Local Search: Foundations & Applications, 1st edn. Morgan Kaufmann Publishers, San Francisco (2005). https://doi.org/10.1016/B978-1-55860-872-6.X5016-1
23. Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983). https://doi.org/10.1126/science.220.4598.671
24. Kuckling, J., Ubeda Arriaza, K., Birattari, M.: AutoMoDe-IcePop: automatic modular design of control software for robot swarms using simulated annealing (2020). Supplementary material. http://iridia.ulb.ac.be/supp/IridiaSupp2020-003/
25. Ligot, A., Birattari, M.: Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms. Swarm Intell. **14**(1), 1–24 (2019). https://doi.org/10.1007/s11721-019-00175-w
26. Lundy, M., Alistair, M.: Convergence of an annealing algorithm. Math. Program. **34**(1), 111–124 (1986). https://doi.org/10.1007/BF01582166
27. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T.: The irace package: iterated racing for automatic algorithm configuration. Oper. Res. Perspect. **3**, 43–58 (2016). https://doi.org/10.1016/j.orp.2016.09.002
28. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of state calculations by fast computing machines. J. Chem. Phys. **21**(6), 1087–1092 (1953). https://doi.org/10.1063/1.1699114

29. Mitra, D., Romeo, F., Sangiovanni-Vincentelli, A.: Convergence and finite-time behavior of simulated annealing. In: 1985 24th IEEE Conference on Decision and Control, pp. 761–767. IEEE Press, Piscataway (1985). https://doi.org/10.1109/CDC.1985.268600
30. Mondada, F., et al.: The e-puck, a robot designed for education in engineering. In: Gonçalves, P., Torres, P., Alves, C. (eds.) Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, pp. 59–65. Instituto Politécnico de Castelo Branco, Castelo Branco (2009)
31. Nedjah, N., Silva Junior, L.: Review of methodologies and tasks in swarm robotics towards standardization. Swarm Evol. Comput. **50**, 100565 (2019). https://doi.org/10.1016/j.swevo.2019.100565
32. Nikolaev, A.G., Jacobson, S.H.: Simulated annealing. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol. 146, pp. 1–39. Springer, Boston (2010). https://doi.org/10.1007/978-1-4419-1665-5_1
33. Quinn, M., Smith, L., Mayley, G., Husbands, P.: Evolving controllers for a homogeneous system of physical robots: structured cooperation with minimal sensors. Philos. Trans. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. **361**(1811), 2321–2343 (2003). https://doi.org/10.1098/rsta.2003.1258
34. Silva, F., Duarte, M., Correia, L., Oliveira, S.M., Christensen, A.L.: Open issues in evolutionary robotics. Evol. Comput. **24**(2), 205–236 (2016). https://doi.org/10.1162/EVCO_a_00172
35. Trianni, V., López-Ibáñez, M.: Advantages of task-specific multi-objective optimisation in evolutionary robotics. PLoS One **10**(8), e0136406 (2015). https://doi.org/10.1371/journal.pone.0136406
36. Trianni, V., Nolfi, S.: Self-organizing sync in a robotic swarm: a dynamical system view. IEEE Trans. Evol. Comput. **13**(4), 722–741 (2009). https://doi.org/10.1109/TEVC.2009.2015577
37. Yang, G.Z., et al.: The grand challenges of Science Robotics. Sci. Robot. **3**(14), eaar7650 (2018). https://doi.org/10.1126/scirobotics.aar7650