Optimization of swarm behavior assisted by an automatic local proof for a pattern formation task

Coppola, Mario; de Croon, Guido C.H.E.

DOI
10.1007/978-3-030-00533-7_10

Publication date
2018

Document Version
Accepted author manuscript

Published in
Swarm Intelligence - 11th International Conference, ANTS 2018, Proceedings

Optimization of Swarm Behavior Assisted by an Automatic Local Proof for a Pattern Formation Task

Mario Coppola[0000−0003−4694−2960] and Guido C.H.E. de Croon[0000−0001−8265−1496]

Faculty of Aerospace Engineering,
Delft University of Technology, Delft, The Netherlands
{m.coppola, g.c.h.e.decroon}@tudelft.nl

Abstract. In this work, we optimize the behavior of swarm agents in a pattern formation task. We start with a local behavior, expressed as a local state-action map, that has been formally proven to lead the swarm to always eventually form the desired pattern. We seek to optimize this for performance while keeping the formal proof. First, the state-action map is pruned to remove unnecessary state-action pairs, reducing the solution space. Then, the probabilities of executing the remaining actions are tuned with a genetic algorithm. The final controllers allow the swarm to form the patterns up to orders of magnitude faster than with the original behavior. The optimization is found to suffer from scalability issues. These may be tackled in future work by automatically minimizing the size of the local state-action map with a further direct focus on performance.

1 Introduction

Collaboration between autonomous agents, while already a difficult task in itself, becomes increasingly challenging when dealing with swarm of robots with limited on-board sensing and computing capacity. In recent work, detailed in [1], we introduced a method to extract local behaviors with which very limited agents could arrange into a desired shape. The agents were: homogeneous (identical and without hierarchy), anonymous (did not have identities), reactive (memoryless), could not communicate, did not have global position information, did not (explicitly) know the goal of the swarm, and operated asynchronously in an unbounded space. The only knowledge available to the agents in the decision making was: 1) a common heading direction (i.e., North), 2) the relative location of their neighbors within a maximum range (e.g., similar to the robotic system in [2]). Despite such limited agents, it was possible to define the local agent behavior such that a desired pattern would always emerge, with a formal proof that this would be reached from any initial configuration.

Simulations in [1] further showed that the swarms indeed always eventually reached the desired formation by assuming random (but feasible) actions on the

part of the agents. However, as the agents moved randomly and asynchronously, even simple patterns with a few agents were found to take hundreds of actions before completion, and this number appeared to grow exponentially with the size of the swarm and, in turn, with the complexity of the pattern. This becomes an issue if the algorithm is to be used on real robots with limited battery life. In this work, we thus explore how the behavior of the agents can be optimized so that they do not just "eventually" form the pattern, but do so efficiently. In doing so, we also explore a novel use of evolutionary algorithms in the context of swarm intelligence, as we perform an optimization procedure while maintaining the conditions for the formal proof that the goal will always eventually be achieved.

This paper is organized as follows. In Sect. 2 we review relevant literature and introduce the context of this research. Then, Sect. 3 summarizes the framework used to enable the swarm to form a desired pattern. The optimization methodology is detailed in Sect. 4, followed by an assessment in Sect. 5. In Sect. 6, we summarize the findings and discuss future work.

2 Related Work and Research Context

Evolutionary algorithms can search through vast solution spaces and discover solutions to complex problems, and are thus a popular approach to dealing with the intricacies of swarm robotics and extracting valid local behaviors [6, 14]. They have been used for numerous architectures, including: neural networks [3, 11], state machines [7], behavior trees [12], and grammar rules [5]. When applied to swarms, the following issues typically arise:

1. As the number of agents grows, the complexity of the solution and the size of the potential solution space grow [16, 10].
2. The evolutionary algorithm is likely to drift into undesired local optima. This may happen due to deceptive fitness functions or bootstrap issues [17, 9].
3. As the size of the swarm grows, the iteration time needed to find a solution grows. This can be due to, for instance:
   (a) The computational requirements needed to evaluate the fitness of a controller are higher because of the need to simulate a larger swarm.
   (b) Depending on the task, it might take longer for the desired behavior to emerge, requiring a longer simulation time upon each evaluation trial.
   (c) Each controller may have to be simulated multiple times in order to accurately assess its expected average fitness [18].

In state of the art, the problems above have mostly been tackled in two ways. First, there are methods that try to deal with the broad solution space. For example, Gomes et al. [8] used novelty search to encourage a broader exploration of the solution space. The second way is to use global knowledge and insights to aid the evolutionary process. For example, Duarte et al. [3] partitioned complex swarm behavior into simpler sub-behaviors. Hüttenrauch et al. [10], with a focus on deep reinforcement learning, used global information to guide the learning process towards a solution. Alternatively, Trianni et al. [18] and Ericksen et al.

[4] explored whether evolved behaviors for smaller swarms could generalize to larger swarms.

Evolutionary approaches are thus typically used to establish the behavior needed to achieve the global goal, but it is not known whether the final behavior generalizes to all initial conditions. In this work we present the first steps to an alternate approach towards achieving an optimum swarm behavior: optimizing a behavior that is formally proven to always eventually lead to the emergent global goal. In this approach, the proof that the goal will always eventually be achieved remains preserved throughout. The focus of the optimization procedure is not on figuring out *how* to solve the problem, but on how to do it more efficiently while ensuring that the resulting behavior still guarantees that any initial condition will always eventually lead to the goal. Using the framework from [1], and limited agents as introduced therein, we attempt to optimize the local behavior of the agents in finite pattern formation tasks of increasing complexity. We begin from a local state-action map given to the agents. This state-action map can be verified to always eventually lead to the goal, but is not optimized for performance, as any agent in a given state can select its action randomly from several options, with equal probability. We then tune this state-action map with the goal of simplifying the behavior and minimizing the number of actions needed, on average, to achieve the final pattern when starting from an arbitrary initial configuration. More specifically, we do the following:

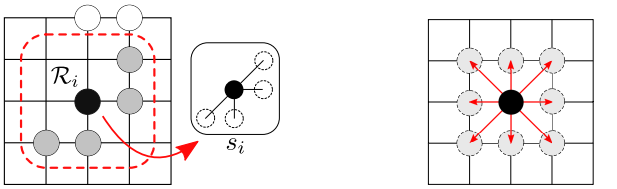
1. Restrict the possible actions that an agent can take when in a given state, subject to the constraint that it must still be provable that the global goal will emerge. This minimizes the size of the local state-action map of the agents, and in turn the size of the possible solution space.
2. We take the minimized state-action map and we apply an evolutionary algorithm to optimize the probability of executing each action.

The desired final outcome is a probabilistic local state-action map that enables the agents to arrange into the desired pattern most efficiently when starting from a random initial configuration.

3 Framework and Approach to Pattern Formation

This section summarizes the pattern formation methodology, which can be found in more detail in [1]. For the sake of brevity, in this work we will assume that the swarm operates in a grid world and in discrete time. However, as demonstrated in [1], the behavior can also be used in continuous time and space with asynchronous agents.

Consider N robots that exist in an unbounded discrete grid world and operate in discrete time. In the case studied in this paper, each robot $\mathcal{R}_i$ can sense the location of its neighbors in the 8 grid points that surround it, as depicted in Fig. 1a. This is the local state s_i of the agent (which is all the information that it has). The local state space $\mathcal{S}$ consists of all combinations of neighbors that it could sense, such that $|\mathcal{S}| = 2^8$. At time step $k = 0$, we assume the swarm begins



(a) Example local state **(b)** Possible actions

Fig. 1: Depictions of local state and the actions that an agent can take

in a connected topology forming an arbitrary pattern P_0. At each time step, one random robot in the swarm takes an action, whereby it can move to any of the 8 grid points surrounding it, as depicted in Fig. 1b. This is the action space of the agents, denoted $\mathcal{A}$. Moreover, if a robot takes an action, then it will not take an action at the next time step (unless no other robot can take an action).

The goal of the swarm is to rearrange from its initial arbitrary pattern P_0 into a desired pattern P_{des}. This is achieved using the following principle. The local states that the agents are in when P_{des} is formed are extracted, this forms a set of local desired states $\mathcal{S}_{des} \in \mathcal{S}$, as depicted by the examples in Fig. 2. If robot $\mathcal{R}_i$ finds itself in any state $s_i \in \mathcal{S}_{des}$ then it is instructed to not move, because, from its perspective, the goal has been achieved. In [1], it is shown that, given a P_{des} and the corresponding $\mathcal{S}_{des}$, it can be automatically verified whether the local desired states will uniquely form P_{des}, or whether they can also can give rise to spurious global patterns. In the following, we assume that set of local desired states has passed this verification. Therefore, until P_{des} is formed, at least one agent will be in a state $s \notin \mathcal{S}_{des}$ and will seek to amend the situation. The swarm will then keep reshuffling until P_{des} forms.

When an agent $\mathcal{R}_i$ is in a state $s_i \notin \mathcal{S}_{des}$, it can execute an action. From the state space and action space, we can extract a state-action map $\mathcal{Q} = (\mathcal{S} \setminus \mathcal{S}_{des}) \times \mathcal{A}$. However, not all actions should be allowed. The actions that: a) cause collisions and b) cause local separation of the swarm are eliminated from $\mathcal{Q}$, because they are not safe. From this, we extract a *safe* state-action map $\mathcal{Q}_{safe}$, where $\mathcal{Q}_{safe} \subseteq \mathcal{Q}$. From this process, there will also emerge some local states that cannot take any safe actions. An agent in such a state will not be able to move or else it will either collide with other agents or possibly cause separation of the swarm. We refer to such states as *blocked* states. The set of blocked states is denoted $\mathcal{S}_{blocked}$. By contrast, there are states where an agent will be capable of moving away from its neighborhood without issues. We call these states *simplicial*. The set of simplicial states is denoted $\mathcal{S}_{simplicial}$. Fig. 3 shows examples of blocked states and simplicial states.

Now consider a graph $G_{\mathcal{S}} = (V, E)$. Let the nodes of $G_{\mathcal{S}}$ be all states that the agents can be in, such that $V = \mathcal{S}$. The edges of $G_{\mathcal{S}}$ are all local transitions between states. These are all the state transitions that an agent can locally experience as a result of the changing environment when it, or any other agent in the swarm, moves. More specifically, $G_{\mathcal{S}}$ is the union of three subgraphs:

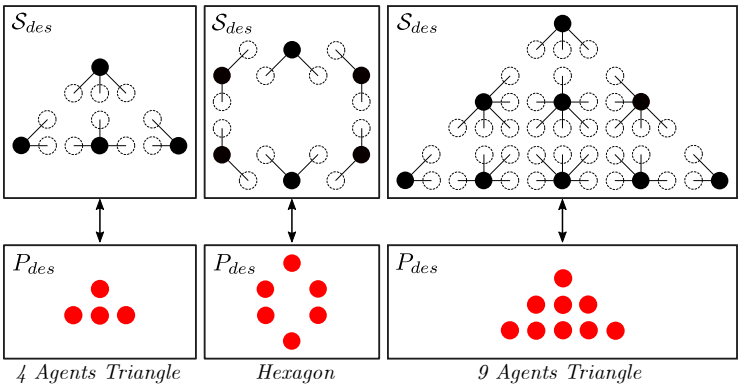


Fig. 2: Set of desired states $\mathcal{S}_{des}$ for the exemplary patterns treated in this paper, featuring patterns of increasing complexity and size (from left to right)

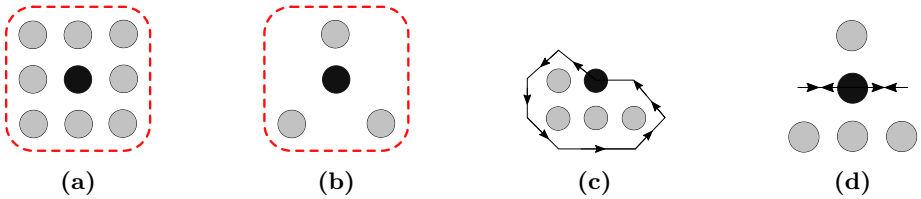


Fig. 3: Examples of: **(a)** a state $s \in \mathcal{S}_{blocked}$, due to it being surrounded; **(b)** a state $s \in \mathcal{S}_{blocked}$, because any motion will cause the swarm to locally disconnect; **(c)** a state $s \in \mathcal{S}_{active} \cap \mathcal{S}_{simplicial}$, because it can travel around all its neighbors; **(d)** a state $s \in \mathcal{S}_{active}$ but $s \notin \mathcal{S}_{simplicial}$, because it can move but it cannot travel around all its neighbors or else it might disconnect the swarm

$G^1_{\mathcal{S}}$ indicates all state transitions that an agent could go through by an action of its own, based on $\mathcal{Q}_{safe}$. $G^2_{\mathcal{S}}$ indicates all state transitions that an agent could go through by an action of its neighbors (which could also move out of view). $G^3_{\mathcal{S}}$ indicates all state transitions that an agent could go through if another agent, previously out of view, were to move into view and become a new neighbor. Furthermore, let $G^{2r}_{\mathcal{S}}$ be a subgraph of $G^2_{\mathcal{S}}$. $G^{2r}_{\mathcal{S}}$ only indicates the state transitions in $G^2_{\mathcal{S}}$ where a neighbor moves about the central agent, but *not* out of view. By analyzing certain properties of these graphs, it can be verified that the pattern P_{des} will eventually form starting from any initial pattern P_0. Specifically, the following conditions need to be met:

1. $G^1_{\mathcal{S}} \cup G^2_{\mathcal{S}}$ shows that each state in $\mathcal{S}$ features a path to each state in $\mathcal{S}_{des}$.

2. For all states $s \in \mathcal{S}_{blocked} \cap \mathcal{S}_{\neg des}$, none of the cliques [1] of each state can be formed uniquely by agents that are in a state $s \in \mathcal{S}_{des} \cap \mathcal{S}_{simplicial}$.
3. $G^{2r}_{\mathcal{S}}$ shows that all static states with two neighbors can directly transition to an active state.
4. $G^{1}_{\mathcal{S}}$ shows that any agent in state $s \in \mathcal{S}_{active} \cap \mathcal{S}_{simplicial}$ could move around all its local neighbors (as exemplified in Fig. 3c).
5. $G^{3}_{\mathcal{S}}$ shows that any agent in any state $s \in \mathcal{S}_{des} \cup \mathcal{S}_{blocked}$ will always, by the arrival of a new neighbor in an open position, transition into an active agent (with the exception of any agent that is, or becomes, surrounded).

The motivations behind these conditions can be found in [1]. They are not repeated here due to page restrictions. However, they essentially ensure that all agents will keep moving around with sufficient freedom for the swarm to reshuffle without deadlocks or endless loops until the pattern is achieved. These conditions are local in nature; they focus on the local perception in an agent's limited sensing range and the actions that the agent could take as a result. The advantage of this is that checking whether the conditions are met is independent of the size of the swarm, avoiding the combinatorial explosion that would otherwise ensue. This makes it possible to verify them within a heuristic optimization process. This proof, combined with the fact that it can deal with very limited agents (anonymous, homogeneous, memoryless, with limited range sensing, and without needing any communication, global knowledge, or seed agents) moving in space, sets the work in [1] apart from other works such as [13, 19, 15].

4 Optimization Methodology

Following the framework in Sect. 3, we can know whether a given pattern P_{des} will eventually form if the agents act based on its corresponding $\mathcal{Q}_{safe}$. However, this may take a significant amount of actions, due to the fact that any active agent could move at any time step and select a random action from its options in $\mathcal{Q}_{safe}$. The objective of this article is to minimize the number of actions that the agents will take, on average, to form P_{des} when starting from an arbitrary pattern P_0. In Sect. 4.1 and Sect. 4.2 we take two preliminary steps to automatically, at the local level, prune $\mathcal{Q}_{safe}$ from unnecessary actions. This will lead us to a new set $\mathcal{Q}_{reduced} \subseteq \mathcal{Q}_{safe}$ which is minimally sufficient to achieve the global goal. This reduces randomness in the system and restricts the solution space. Then, in Sect. 4.3, we use an evolutionary algorithm to tune the probability of taking each action in $\mathcal{Q}_{reduced}$, leading to a final controller. Throughout all steps, measures will be taken to ensure that the conditions of the proof (as detailed at the end of Sect. 3) remain respected. We apply this procedure to the patterns from Fig. 2.

[1] A *clique* is a connected set of an agent's neighbors. Without the central agent, the agents in each clique would remain connected with each other, but the different cliques would not be connected.

4.1 Step 1: A-Priori Local Reduction of Active States

$\mathcal{S}$ can be sub-divided in two sets: $\mathcal{S}_{active}$, in which agents take an action based on $\mathcal{Q}_{safe}$, and $\mathcal{S}_{des} \cup \mathcal{S}_{blocked}$, in which the agents do not take actions. For simplicity, the latter is grouped under the umbrella set $\mathcal{S}_{static} = \mathcal{S}_{des} \cup \mathcal{S}_{blocked}$. In this step, we aim to move states from $\mathcal{S}_{active}$ to $\mathcal{S}_{static}$. This will reduce the number of agents in the swarm that are likely to move, decreasing the size of $\mathcal{Q}_{safe}$.

As explained in Sect. 3, an important axiom needed to guarantee that P_{des} will form is that, for a swarm of N agents, N instances of the local states in $\mathcal{S}_{static}$, with repetition, must uniquely rearrange into P_{des}. If this is not the case, another pattern could emerge where all agents are in a state $\mathcal{S}_{static}$ and do not move. Here, because we are already at the optimization stage, we consider the case where the original $\mathcal{S}_{static}$ already guarantees that P_{des} is unique. From this starting point, we present a method to augment $\mathcal{S}_{static}$ based only on a local analysis, while keeping P_{des} as the unique static pattern.

Consider a state $s \in \mathcal{S}_{active}$. For s, we locally check whether it could be fully surrounded by agents with a state within $\mathcal{S}_{static}$. If this is not possible, because s is such that at least one of its neighbors would be in an active state, then we add s to $\mathcal{S}_{static}$. This is because we know that, if this state s were static, there would always be one active agent somewhere next to it anyway, so P_{des} still remains the only unique pattern that can be formed by static states. Then, when this active neighbor moves, the local state of the agent will also change and it will also no longer be static. We run this process iteratively for all states until no more states from $\mathcal{S}_{active}$ can be moved to $\mathcal{S}_{static}$. As an exception, due to the importance of active simplicial states remaining active to guarantee that there is motion in the swarm, these are not included in the process.

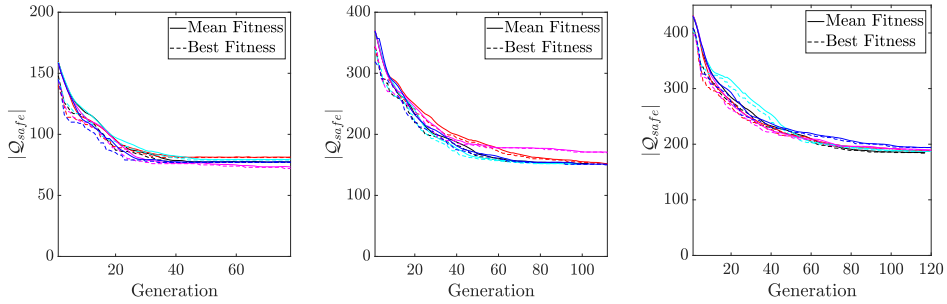
Using this approach, it is possible to significantly increase the size of $\mathcal{S}_{static}$, and in turn reduce the size of $\mathcal{Q}_{safe}$. Additionally, one can also add to $\mathcal{S}_{static}$ all states that expect more neighbors than are present in the swarm. For instance, for a swarm of 4 robots, all states with 4 or more neighbors may be discarded, because they cannot happen in the first place and we need not consider them. Tab. 1 shows the results of Step 1 for the patterns in Fig. 2.

4.2 Step 2: Local Elimination of Unnecessary Actions

In this step, individual state-action pairs that are not necessary towards achieving the final pattern, in accordance with the proof, are discarded. The objective is

Table 1: Results of Step 1 on the size of $\mathcal{S}_{static}$ (which increases) and $\mathcal{Q}_{safe}$ (which decreases)

		4 Agents Triangle	*Hexagon*	*9 Agents Triangle*
Before	$\lvert\mathcal{S}_{static}\rvert$	28	30	33
	$\lvert\mathcal{Q}_{safe}\rvert$	543	550	531
After	$\lvert\mathcal{S}_{static}\rvert$	188	128	87
	$\lvert\mathcal{Q}_{safe}\rvert$	172	381	439



(a) 4 agents triangle **(b)** Hexagon **(c)** 9 agents triangle

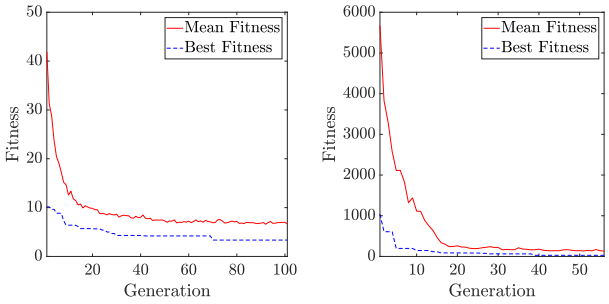
Fig. 4: Results of the evolutionary reductions of $\mathcal{Q}_{safe}$ from Step 2

to minimize $|\mathcal{Q}_{safe}|$ while keeping the conditions listed at the end of Sect. 3. The minimization was performed with a Genetic Algorithm (GA) in order to avoid local minima. The fitness function to be minimized was $f = |\mathcal{Q}_{safe}|$, subject to the following constraints:

1. $\mathcal{S}_{static}$ must not change. This is because, following Step 1, we know that all remaining states must be active, else a spurious pattern might form.
2. The conditions at the end of Sect. 3 must be respected.

The population of the GA was formed by 100 binary genomes. Each gene in a genome represented a state-action pair in $\mathcal{Q}_{safe}$, with a 1 indicating that the state-action pair is kept and a 0 indicating that it is eliminated. All genomes in the initial population were such that the constraints were respected. Then, the new generation consisted of: elite members (30%), new offspring (40%), and mutated members (30%). Offspring genomes were the result of an `AND` operation between two parent genomes. This automatically meant that any offspring would be at least as fit as its parents, because the `AND` operator natively either reduced or kept the quantity of activated bits. Offspring were only kept if they complied with the constraints, else the parents were forced to look for new mates to perform the `AND` operation with. This also made for a convenient stopping criterion, which is when all children are equally as fit as the parents or when all parents are unable to find any mate that will result in a valid offspring. On each generation round, mutation was applied to a random portion of the population, for which the `NOT` operator was randomly applied to 10% of each selected member's genome (thus changing 1s to 0s and viceversa). Similarly as to the offspring, a mutation was kept only if it returned a genome for which the constraints were met, else it was discarded and a new mutation was attempted. This way there was a guarantee that the population always consisted of valid genomes.

We executed 5 evolutionary runs for each pattern from Fig. 2, with similar results. The results of the runs are shown in Fig. 4. Thanks to the local nature of the proof, the evolution time was not dependent on the number of agents in the swarm.



(a) 4 Agents Triangle **(b)** Hexagon

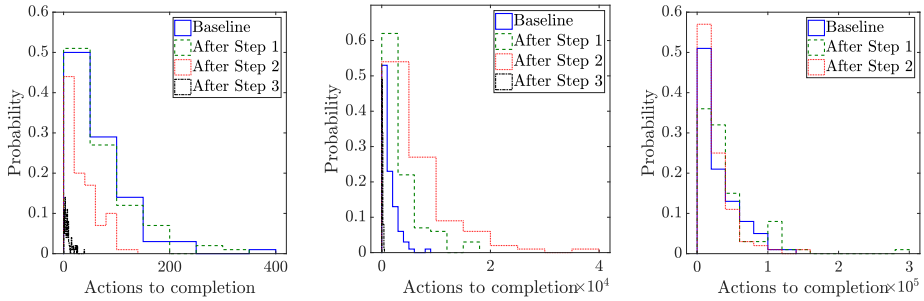
Fig. 5: Optimization results of Step 3 from the best evolutionary runs

4.3 Step 3: Behavior Optimization

Steps 1 and 2 lead to reduced state-action maps $\mathcal{Q}_{reduced}$ that are minimally sufficient to guarantee that the patterns will be achieved. In Step 3, we tune the probability of executing each action in $\mathcal{Q}_{reduced}$. This is done with a more classical evolutionary robotics approach to swarm robotics: the swarm is simulated and evaluated based on its statistical performance, and this information is used in the fitness function of a GA.

The fitness function to be minimized is the expected number of actions needed to achieve the goal. This was evaluated by the mean over 10 trials. The GA used a population of 100 scalar genomes. Each gene in a genome held a value $0 < p \leq 1$, indicating the probability of taking the corresponding action from $\mathcal{Q}_{reduced}$. By means of the inequality, it is not possible to bring the probability of a state-action pair down to 0 and deactivate it (keeping the proof intact). Each new generation was produced by elite members (30%), offspring (40%), and mutated members (30%), as in Step 2. Offspring resulted from mixing two parent's genomes via a uniform crossover strategy, where each gene of an offspring's genome is randomly selected from the genes of either parent with equal probability. Mutation was applied to random genomes, for which 10% of their genes were replaced by random values from a uniform distribution. The members of the initial population were produced randomly from uniform distributions.

Using this scheme, we optimized the behavior for the 4 agents triangle and the hexagon, running 5 evolutionary runs each. The best evolutionary runs are shown in Fig. 5. For the triangle, 3 out of 5 runs converged. For the hexagon, 2 out of 5 runs converged, one considerably lower than the other. We associate the convergence issues to bootstrap and noise issues during evaluation, which grow with the size of the swarm. In light of this, we were unable to establish an optimal solution for the triangle with 9 agents. This is due to two problems: 1) the controllers in early generations took a very long time to evaluate, which made executing the GA troublesome, 2) the fitness metric was subject to considerable variance, leading to inaccurate controller evalations. These problems and their implications are discussed further in Sect. 5.



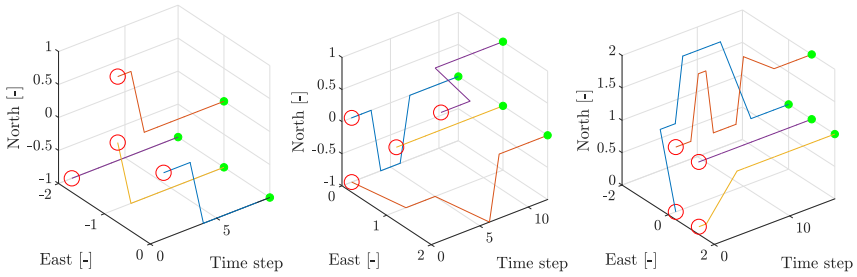
(a) 4 Agents Triangle **(b)** Hexagon **(c)** 9 Agents Triangle

Fig. 6: Normalized histograms of the performance of the system through all steps of the optimization

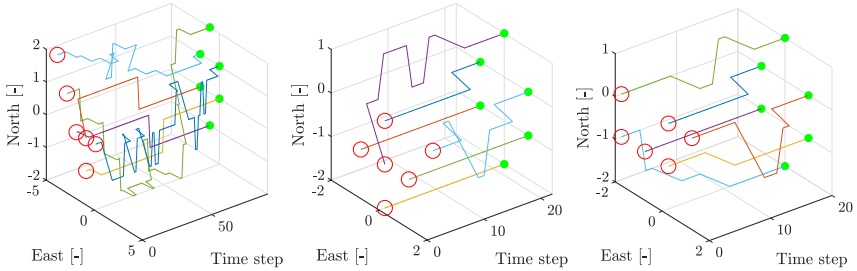
5 Results and Discussion

We tested the performance of the original baseline controller against the performance of the controllers after Step 1, Step 2, and Step 3. Each controller was tested 100 times. The normalized distributions for the number of actions to completion are shown in Fig. 6. Exemplary simulations of swarms as they create a pattern using the evolved behaviors from Step 3 are shown in Fig. 7. In all tests, the desired pattern was eventually achieved. Reducing the size of $\mathcal{Q}_{safe}$ in Steps 1 and 2 simplified the state-action map and did not have an impact on whether the goal could be achieved. This result serves as empirical evidence for the proofs in [1], which were apt guards to ensure that the swarm could form the pattern. Then, when the minimized state-action map $\mathcal{Q}_{reduced}$ was optimized, we were able to significantly improve the performance of the system for the triangle with 4 agents and the hexagon, achieving a fast average performance while also respecting the proof.

There remain issues to be investigated. The first issue is that Step 1 and 2 only modified $\mathcal{Q}_{safe}$ with the goal to minimize its size and simplify the agent's behavior. As seen in Fig. 7, this in itself does not necessarily aid performance. In future work, there should be efforts to understand how to reduce $\mathcal{Q}_{safe}$ while also improving performance. The second issue is scalability, as encountered in Step 3. Most notably, this prevented us from completing Step 3 for the triangle with 9 agents. It is possible that these problems can be mitigated by improving Steps 1 and 2 to minimize $|Q_{safe}|$ while also assessing performance. Another option could be to stop simulations before completion and use a fitness measure that favors global patterns closer to P_{des} over other less similar patterns. However, it might also be possible that the scalability issue is intrinsic to the system. As the size of the swarm grows, then the relative information that each agent has of the whole swarm decreases, and it become increasingly difficult for an agent to predict whether an action is the best for the good of the whole swarm. It would be interesting to explore this limitation in future work.



(a) 4 Agents Triangle



(b) Hexagon

Fig. 7: Exemplary simulations showing pattern formation of the triangle with 4 agents and the hexagon after optimization

6 Conclusion and Future Work

The approach presented in this paper is a first step towards optimizing swarm behavior of severely limited agents by aid of an automatic proof, where a local proof allows for the fast verification of certain properties, and can thus be included within the optimization process. The focus was not on how to achieve goal, but on how to achieve it more efficiently. This led to efficient controllers where the number of actions needed to achieve the patterns were significantly lower than the original controllers, making them more suitable for use in the real world. In the meanwhile, the controllers remained such that eventual success by the swarm is guaranteed.

The approach encountered problems with scalability in the final step. This could be tackled by using the automatic minimization steps, prior to the final optimization, to reduce the solution space in a way that is more favorable for performance. However, there remains the issue that, as the size of the swarm grows, each agent becomes less empowered to take an optimal action, given that it has relatively less information on the state of the swarm. For this reason, it would also be valuable to explore how scalability improves when the agents have more information of their surroundings (e.g., they can sense further away), or some limitations are lifted (e.g. memory).

Bibliography

[1] Coppola, M., Guo, J., Gill, E.K., de Croon, G.C.H.E.: Provable emergent pattern formation by a swarm of anonymous, homogeneous, non-communicating, reactive robots with limited relative sensing and no global knowledge or positioning. ArXiv Preprint arXiv:1804.06827(Submitted to Swarm Intelligence, Springer) (2018)

[2] Coppola, M., McGuire, K.N., Scheper, K.Y.W., de Croon, G.C.H.E.: On-board communication-based relative localization for collision avoidance in micro air vehicle teams. Autonomous Robots (2018)

[3] Duarte, M., Costa, V., Gomes, J., Rodrigues, T., Silva, F., Oliveira, S.M., Christensen, A.L.: Evolution of collective behaviors for a real swarm of aquatic surface robots. PloS one **11**(3), e0151834 (2016)

[4] Ericksen, J., Moses, M., Forrest, S.: Automatically evolving a general controller for robot swarms. In: 2017 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 1–8 (2017)

[5] Ferrante, E., Duéñez Guzmán, E., Turgut, A.E., Wenseleers, T.: Geswarm: Grammatical evolution for the automatic synthesis of collective behaviors in swarm robotics. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation. pp. 17–24. GECCO '13, ACM, New York, NY, USA (2013)

[6] Francesca, G., Birattari, M.: Automatic design of robot swarms: Achievements and challenges. Frontiers in Robotics and AI **3**, 29 (2016)

[7] 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 Intelligence **9**(2), 125–152 (Sep 2015)

[8] Gomes, J., Urbano, P., Christensen, A.L.: Introducing novelty search in evolutionary swarm robotics. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Engelbrecht, A.P., Groß, R., Stützle, T. (eds.) Swarm Intelligence. ANTS 2012. pp. 85–96. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[9] Gomes, J., Urbano, P., Christensen, A.L.: Evolution of swarm robotics systems with novelty search. Swarm Intelligence **7**(2), 115–144 (2013)

[10] Hüttenrauch, M., Šošić, A., Neumann, G.: Guided deep reinforcement learning for swarm systems. ArXiv Preprint (2017), arXiv:1709.06011

[11] Izzo, D., Simões, L.F., de Croon, G.C.H.E.: An evolutionary robotics approach for the distributed control of satellite formations. Evolutionary Intelligence **7**(2), 107–118 (2014)

[12] Jones, S., Studley, M., Hauert, S., Winfield, A.: Evolving behaviour trees for swarm robotics. Distributed Autonomous Robotic Systems: The 13th International Symposium. Springer Proceedings in advanced Robotics. pp. 487–501 (2018)

[13] Klavins, E.: Programmable self-assembly. IEEE Control Systems **27**(4), 43–56 (Aug 2007)

[14] Nolfi, S.: Power and the limits of reactive agents. Neurocomputing **42**(1–4), 119 – 145 (2002)
[15] Rubenstein, M., Cornejo, A., Nagpal, R.: Programmable self-assembly in a thousand-robot swarm. Science **345**(6198), 795–799 (2014)
[16] Saska, M., Vonásek, V., Chudoba, J., Thomas, J., Loianno, G., Kumar, V.: Swarm distribution and deployment for cooperative surveillance by micro-aerial vehicles. Journal of Intelligent & Robotic Systems **84**(1), 469–492 (Dec 2016)
[17] Silva, F., Duarte, M., Correia, L., Oliveira, S.M., Christensen, A.L.: Open issues in evolutionary robotics. Evol. Comput. **24**(2), 205–236 (Jun 2016)
[18] Trianni, V., Nolfi, S., Dorigo, M.: Cooperative hole avoidance in a swarm-bot. Robotics and Autonomous Systems **54**(2), 97 – 103 (2006)
[19] Yamins, D., Nagpal, R.: Automated global-to-local programming in 1-d spatial multi-agent systems. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2. pp. 615–622. AAMAS '08, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2008)