Noname manuscript No.
(will be inserted by the editor)

AutoMoDe-Cedrata: Automatic design of behavior trees for controlling a swarm of robots with communication capabilities

Jonas Kuckling* · **Vincent van Pelt** · **Mauro Birattari**

Received: date / Accepted: date

Abstract Behavior trees are a control architecture that has gained recent attention in AI and robotics. Previous research on the use of behavior trees in swarm robotics has shown the necessity for the behaviors to have proper return values, instead of running indefinitely. This work extends our previous work in which we defined AutoMoDe-Cedrata, an automatic modular design that makes use of modules that have been explicitly defined for behavior trees. In this work, we extend Cedrata by introducing Cedrata-GP and Cedrata-GE which are based on genetic programming and grammatical evolution, respectively. We test these design methods on two missions and compare the performance of the automatic design methods against the performance of solutions created by human designers. The results show that the structure of Cedrata allows for well-performing solutions that are reliably found by the human designers. However, the automatic design methods fail to discover the same communication strategies as the human designers.

Keywords Swarm Robotics · Design by Optimization · AutoMoDe · Genetic Programming · Grammatical Evolution

The experiments were designed and performed by by JK and VvP. The paper was drafted by JK and edited by MB; all authors read and commented the final version. The research was directed by MB. The project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (DEMIURGE Project, grant agreement No 681872); from Belgium's Wallonia-Brussels Federation through the ARC Advanced Project GbO–Guaranteed by Optimization; and from the Belgian Fonds de la Recherche Scientifique–FNRS via the crédit d'équippement SwarmSim. JK and MB acknowledge support from the Belgian Fonds de la Recherche Scientifique–FNRS.

Jonas Kuckling
IRIDIA, Université libre de Bruxelles, Brussels, Belgium
E-mail: jonas.kuckling@ulb.be

Mauro Birattari
IRIDIA, Université libre de Bruxelles, Brussels, Belgium
E-mail: mbiro@ulb.ac.be

1 Introduction

Swarm robotics is a research area that combines robotics and swarm intelligence, and that is recognized as a promising approach for controlling large groups of robots [1–11]. Robot swarms are self-organizing decentralized systems, consisting of relatively simple robots that cooperate to achieve a goal that would not be achievable for each individual robot alone. The collective behavior of the robot swarm emerges from the interactions between the robots themselves and between the robots and the environment [12]. One challenge of swarm robotics is the difficulty of designing control software for the individual robots, so that the desired collective behavior emerges [13].

One approach to the design of control software for robot swarms is manual design, in which a human designer creates the control software. However, only few and limited principled approaches to manual design exist [14–22] and no general methodology has yet been proposed. As a result, most manual design approaches rely on trial and error, a time-consuming, costly, and often error-prone strategy [23, 24].

Other approaches rely on the use of an optimization algorithm and can be broadly categorized in two categories: semi-automatic design and fully automatic design (although hybrid approaches exist) [10]. In semi-automatic design, a human designer uses an optimization algorithm as a tool to design the control software. The designer specifies the problem and defines the parameters of the optimization algorithm. They observe the optimization process and adjust the problem specification or the parameters of the optimization algorithm until the result is satisfactory. While the semi-automatic approach alleviates some drawbacks of manual design, the involvement of a human designer still entails similar challenges: as long as no general principled approach exists, much of the performance depends on the experience and domain knowledge of the human designer.

In contrast, in fully automatic design, the role of the human designer is reduced to the problem specification. After receiving the problem specification, the fully automatic design process searches for a satisfactory solution without any further human intervention [10]. This lack of human intervention also implies that no mission-specific domain knowledge can be incorporated into the design process. Indeed, any fully automatic design method needs to be able to address not only a single mission, but a class of missions [13].

Fully automatic design often produces the control software off-line, i.e., the software is designed using simulations and only the final resulting control software is uploaded onto the real robots for evaluation. While this approach offers many advantages, like speeding up the design process through faster-than-real-time simulations and parallelization of simulation processes and no need for hardware availability for the design process, it suffers from one major drawback, the reality gap. The reality gap is the inherent difference between the simulation and the real environment and often manifests itself in the form of a performance drop [**?**]. Not all methods are affected equally by the reality gap [**?**].

Francesca et al. proposed to look at the reality gap problem akin to the bias-variance trade-off [25]. They hypothesized that design methods with a very large and fine-grained action space (“low bias”) are more prone to overfit the simulation context

("high variance"). By restricting the space of possible behaviors ("introducing bias"), it should be possible to produce software that is more robust to the reality gap. Based on this hypothesis, they proposed AutoMoDe, a class of automatic modular design methods. In automatic modular design, a set of pre-defined modules is assembled and fine-tuned into more complex control software by an optimization algorithm. The first method of this class is `Vanilla`, an automatic modular design approach that crosses the reality gap satisfactorily [25]. `Chocolate` extends `Vanilla` by using Iterated F-race [26] as the optimization algorithm to assemble a finite-state machine with up to four states and sixteen transitions from a set of six behavioral modules (mapped to the states of the finite-state machine) and six conditions (mapped to the transitions of the finite-state machine) [27]. Other AutoMoDe methods vary or extend the capabilities of `Chocolate`. `Gianduja` [28], `TuttiFrutti` [29], or `Arlequin` [30] introduce new software modules, that extend the capabilities of the robotic platform, e.g., by enabling direct communication, color detection or the use of artificial pheromones. `Waffle` [31] allows the design process not only to control aspects of the control software but also of the hardware capabilities of the robot. `IcePop` [32] investigates the use of local search-based optimization algorithms. `Maple` [33] and `Cedrata` [34] are design methods that use behavior trees [35] as the target architecture.

The work presented in this paper extends [34], which introduced `Cedrata`. Our previous work showed that the modules and architecture of `Cedrata` allowed for well-performing solutions, but the optimization algorithm (Iterated F-race) had problems finding these solutions. In this work, we investigate additionally the use of two other optimization algorithms, namely genetic programming [36] and grammatical evolution [37].

2 Related work

Behavior trees are a control architecture that originates from video games [38], but which since has found applications in fields such as artificial intelligence or robotics [39]. In this work, we follow the behavior tree definition of Marzinotto et al. [35].

In this framework, behavior trees are a control architecture whose structure can be described as a directed acyclic graph and that operate on a tick that is created with a fixed frequency f_{tick} by an implicitly defined root node. Every time a tick is generated, it traverses the tree, activating the nodes that it visits. The inner nodes of the tree are called control-flow nodes and control the way that the tick takes through the tree. The leaf nodes can be either an action node that executes a single time step of a behavior or a condition node that checks a condition of the environment.

After activating, each node in a behavior tree returns the tick to its parent along with one of three possible return values (*success*, *failure*, *running*) that determine the further way that the tick takes through the tree. Condition nodes can return *success* or *failure*, depending on whether or not their associated condition is met. Action nodes usually return *running*, *success* or *failure*, if the action takes longer than one time step or if the robot is in a state that can be classified as success or failure with regard to the associated action or behavior. The control-flow nodes receive the return value of one of their children nodes and determine if they either pass the tick to another child

or to its parent, together with a return value. In this work, we consider the following control-flow nodes: selector (?), sequence ($\rightarrow$), selector* ($?^*$), sequence* ($\rightarrow^*$). For a detailed and formal definition of all nodes, see [35].

Ligot et al. have proposed `Maple`, an automatic modular design method that assembles modules into a restricted behavior tree architecture [33]. `Maple` utilizes the same modules as `Chocolate` [27]. As these modules have originally been conceived for finite-state machine, they do not offer any states that could be characterized as *success* or *failure* and can therefore only return *running*. The authors propose a very restricted architecture that can successfully incorporate these modules. Results showed that `Maple` could produce solutions that performed adequately, but that the architecture limited the space of possible solutions and that some well-performing solutions found by `Chocolate` could not be represented within the restricted architecture of `Maple`. This work highlighted the need for modules that have proper return values.

Other works that have applied behavior trees to swarm robots include the works by Jones et al. [40,41]. In [40], Jones et al. evolved behavior trees for a swarm of kilobots. In that work, the authors defined only atomic actions that are executed for a single tick and then always return *success*. This necessitates the inclusion of a *repeat* node that can repeatedly tick its children, allowing these actions to be executed more than once consecutively.

In another work, Jones et al. evolved behavior trees onboard of a swarm of Xpucks [41]. The actions in these behavior trees are also atomic, as they perform a singular write to the blackboard. For the onboard evolution, however, each robot performs a design process using genetic programming to create behavior trees. At regular intervals, the best performing behavior tree is selected as the current control software for the robot.

Another approach to the design of control software in the context of swarm robotics was proposed by Neupane and Goodrich [42]. They used distributed grammatical evolution to design behavior trees for a swarm of simulated robots.

GESwarm is another design approach that uses grammatical evolution [43]. In that work, the authors proposed an automatic design method that can design a foraging behavior for a swarm of footbot robots. The control software is represented as a set of policies that map conditions and the current behavior to actions of the robot. The behaviors that the robot can select exhibit similar properties as the behaviors of `Chocolate`, namely that they are simple and can run potentially endlessly, without any implicit success or failure states.

Communication-based behaviors have been used in many works in swarm robotics. With `Gianduja`, Hasselmann et al. were able to show that an automatic design process can automatically assign a semantic to messages that do not have meaning a priori [28].

3 AutoMoDe-`Cedrata`

Originally, we presented `Cedrata` in [34]. For the convenience of the reader, we describe again the method here.

Table 1: The E-puck reference model RM2.2 used by Cedrata [44].

Sensors	Variables
Proximity	$prox \in [1,8]$, $\angle q \in [0,2\pi]$
Ground	$gnd \in \{0, 0.5, 1\}$
Range-and-bearing	$n \in \mathbb{N}$, $r \in [0.5, 20]$, $\angle b \in [0,2\pi]$
	n_s, r_s, $\angle b_s$, for $s \in \{1,...,6\}$

Actuators	Variables
Signal broadcast	$s \in \{0,1,...,6\}$
Wheels	$v_l, v_r \in [-v, v]$, with $v = 0.16\,\mathrm{m/s}$
Control cycle period: 100 ms	

3.1 Reference model

The reference model RM2.2, on which Cedrata is based, is shown in Table 1 [44]. The robot is equipped with eight proximity sensors, three ground sensors and one range-and-bearing board for sensing and two sets of actuators: the range-and-bearing board to send messages and two wheels with differential drive. The reference model formalizes the way that the control software has access to these sensors and actuators. The control software can set the speed of the two wheels of the robot independently. It also always sends a signal value s, that can be equal to 0, which is a special value that means *no signal* and that is sent by default, or an integer in $\{1,...,6\}$. Similar to [28], signal values do not have a particular semantic, instead, it is the role of the design process to assign semantics to the signals. For the sensors, the reference model provides an aggregated vector (in the form of magnitude and direction) over all proximity readings and a single aggregated ground reading. The reference model also provides access to the number of neighboring robots n and a vector to their center of mass. Similarly, it provides the number of messaging robots and a vector to the center of mass of the messaging robots, for each signal $s \in S$. The control cycle period is 100 ms, that is, every 100 ms the sensors are updated and the control software is invoked, generating a new tick in the behavior tree.

3.2 Modules

Based on the reference model RM2.2, we defined fourteen modules—seven behavior modules and seven condition modules. In the following descriptions of the signal-based conditions and behaviors, the set of signals $\{1,...,6\}$ will be denoted S. Some modules can use a special value *any* that is activated if any of the signals in S is received. The set $S^* = S \cup \{any\}$ will denote the sets used by these modules.

Behaviors are associated to action nodes and allow the robot to interact with the environment. The action nodes can return *success* or *failure* if the behavior ends in a state that it considers being a success or a failure. Otherwise, they return *running*. The behavior modules are defined as follows:

Exploration The robot performs a random walk. It moves straight until it perceives an obstacle in front of itself. Then the robot turns on the spot for a random number

of time steps in $\{0,...,\tau\}$, where $\tau \in \{1,...,100\}$ is a tunable parameter. This behavior always return *running*.

Stop The robot stays still. This behavior always returns *running*.

Grouping The robot tries to get closer to its neighbors by moving in the direction of the geometric center of its neighbors. If the number of neighbors becomes greater than N_{max}, the behavior returns *success*, where N_{max} is a tunable parameter. If the number of neighbors becomes smaller than N_{min}, the behavior returns *failure*, where N_{min} is a tunable parameter. Otherwise, it returns *running*. The speed of convergence is controlled by the tunable parameter $\alpha \in [1,5]$. The robot moves in the direction $w = w' - kw_0$, where w' is the target component and kw_0 is the obstacle avoidance component. If robots are perceived, then $w' = w_{r\&b} = (\alpha \cdot r, \angle b)$, otherwise $w' = (1, \angle 0)$. kw_0 is the obstacle avoidance component, with k being a constant fixed to 5 and w_0 defined as $w_0 = (prox, \angle q)$.

Isolation The robot tries to move away from its neighbors by moving in the opposite direction of the geometric center of its neighbors. If the number of neighbors becomes smaller than N_{min}, the behavior returns *success*, where N_{min} is a tunable parameter. If the number of neighbors becomes greater than N_{max}, the behavior returns *failure*, where N_{max} is a tunable parameter. Otherwise, it returns *running*. The speed of divergence is controlled by the tunable parameter $\alpha \in [1,5]$. The Isolation behavior uses the same embedded collision avoidance as in Grouping, but with w' defined as: $w' = -w_{r\&b}$ if robots are perceived, where $w_{r\&b}$ is defined as in the Grouping behavior. Otherwise $w' = (1, \angle 0)$.

Meeting The robot listens for a signal $s \in S^*$ emitted by other robots and moves towards the geometrical centre of the emitters. The behavior returns *success* if the distance between the robot and the geometrical centre is smaller than a distance d_{min}, where d_{min} is a tunable parameter. The behavior returns *failure* if the robot does not perceive any robot sending the expected signal. Otherwise, the behavior returns *running*. The Meeting behavior uses the same embedded collision avoidance as in Grouping, but with w' defined as: $w' = w_{r\&b} = (\alpha \cdot r_s, \angle b_s)$ if robots emitting s are perceived. Otherwise $w' = (1, \angle 0)$.

Acknowledgement The robot sends a signal $s \in S$ and waits for an answer in the form of the same signal, where s is a tunable parameter. The behavior returns *success* if the signal is received or *running* if not. After t_{max} ticks, the behavior returns *failure* if the signal is still not received, where t_{max} is a tunable parameter. This behavior also sets the velocity of both wheels to zero.

Emit Signal The robot sets its emitted signal to $s \in S$ for the current tick, where s is a tunable parameter. This behavior always returns *success*. This behavior also sets the wheel velocity to zero.

Conditions are associated to condition nodes and check an aspect of the environment. The condition nodes return *success*, when their condition is met, or *failure*, otherwise. The condition modules are defined as follows:

Black Floor When all grounds sensors detect a black floor, the condition returns *success* with probability β, where β is a tunable parameter.

Grey Floor When all grounds sensors detect a grey floor, the condition returns *success* with probability β, where β is a tunable parameter.

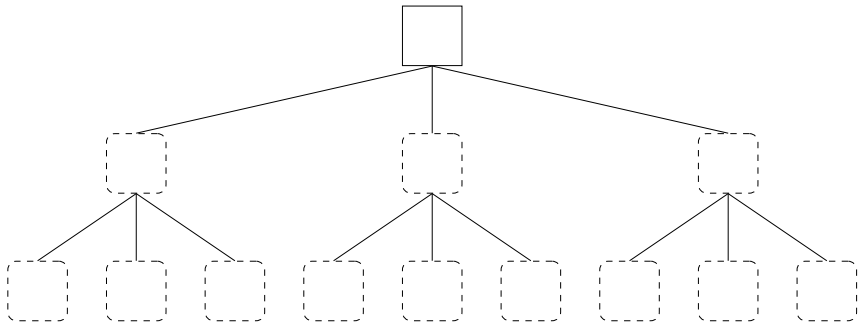


Fig. 1: The possible behavior tree structure for Cedrata. In Cedrata, the top-level node can be any control-flow node. Underneath it the tree can have between one and three nodes, chosen among control-flow nodes, action nodes and condition nodes. If a control-flow node is chosen, then it can have between one and three children, which are either action nodes or condition nodes.

White Floor When all grounds sensors detect a white floor, the transition is enabled with probability β, where β is a tunable parameter.

Neighborhood Count Returns *success* with probability $z(n) = \frac{1}{1+e^{\eta(\xi-n)}}$ where n is the number of robots in the neighborhood, $\eta \in [0,20]$ and $\xi \in \{0,1,...,10\}$ are tunable parameters.

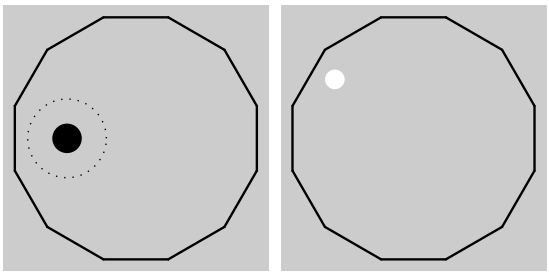
Inverted Neighborhood Count Same as Neighborhood Count but with probability $1-z(n)$.

Fixed Probability Returns *success* with probability β, where β is a tunable parameter.

Receiving Signal Returns *success* if the robot has perceived a neighbor sending $s \in S^*$ in the last 10 ticks, where s is a tunable parameter.

3.3 Architecture

In Cedrata, the optimization process can create a tree that has a maximum of three levels and a maximum of three children per node. The top-level node must be a control-flow node. Nodes of the second level can be control-flow nodes, action nodes or condition nodes. If it is an action node or a condition node, then the node can have no children itself. Not all branches are forced to have the same depth: the top-level node could have some children that are control-flow nodes and some that are action or condition nodes. Nodes on the third level can only be action nodes or condition nodes. The structure of such trees is depicted in Figure 1. The optimization process can choose any control-flow node type to be either a sequence, sequence*, selector or selector* node. For a formal definition of these nodes, see Marzinotto et al. [35]. The tree is allowed to have at most four action nodes and four condition nodes. The constraints on the depth and on the number of children implicitly impose that the tree contains no more than four control nodes. These constraints have been chosen to allow similar numbers of action and condition nodes as in Maple.



(a) MARKER AGGREGATION (b) STOP

Fig. 2: Layouts of the arena for the missions considered.

3.4 Optimization algorithm

The optimization algorithm of `Cedrata` is Iterated F-race [26]. Iterated F-race works over several iterations, each reminiscent of a race. In each iteration, Iterated F-race samples an initial population of candidate solutions. These candidate solutions are evaluated over an increasing number of instances, in this case different starting positions and orientations of the robots in the mission. If at one point a candidate solution is statistically worse than another one (determined by a Friedman test), it is discarded. By discarding inferior solutions, Iterated F-race frees up the design budget for more promising solutions. The iteration ends if either the allocated budget for this iteration is exhausted or all but a fixed number of candidate solutions are discarded. The next iteration then samples its initial population around the elites of the previous iteration.

4 Experimental setup

In this work, we test `Cedrata` and related design methods on a set of two missions. The experimental setup is equivalent to the one described in [34], but we describe it again for the convenience of the reader. All code and data is available from the supplementary material [45].

4.1 Missions

We consider two missions: MARKER AGGREGATION and STOP. These missions must be performed in a dodecagonal arena (see Figure 2) and last 250 s.

In MARKER AGGREGATION (see Figure 2a), the robots must aggregate within the dotted area. The area itself is not perceivable to the robots. Instead, a black spot is placed in the middle of the aggregation area that can serve as a marker. The objective function for this mission is the cumulative time that the robots spend within the aggregation area: $F_{MA} = \sum_{i=0}^{2500} N_A^i$, where N_A^i is the number of robots in the aggregation area at time step i.

In STOP (see Figure 2b), the robots must find a white spot and then stop as soon as possible. A robot is considered moving, if it has travelled more than 5 mm in

Table 2: Parameters for genetic programming and grammatical evolution. Parameters for genetic programming are adapted from [40] and parameters for grammatical evolution are adapted from [42].

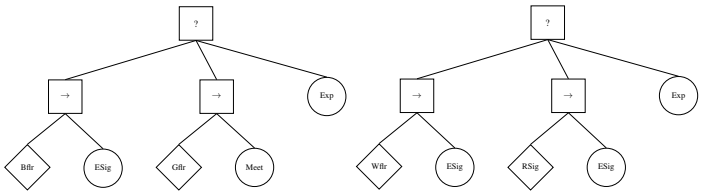
Parameter	Genetic Programming	Grammatical Evolution
Initialization	half-and-half	uniform_tree
Selection strategy	tournament selection	truncation
Tournament size	3	–
Selection proportion	–	50%
Crossover	one-point crossover	one-point crossover
Population size	25	100
Number of elites	3	1
Crossover probability	0.8	0.9
Uniform mutation probability	0.05	–
Shrink mutation probability	0.1	–
Node replacement mutation probability	0.5	–
Ephemeral mutation probability	0.5	–
Flip per codon mutation probability	–	0.01
Codon size	–	1000

the last time step. The objective function for this mission is reduced for each robot that is not moving at any given time step before the white spot has been found and for each robot that is moving after the white spot has been found and additionally for the time that the swarm needed to discover the white spot: $F_{Stop} = 100000 - \left(\bar{t}N + \sum_{t=1}^{\bar{t}} \sum_{i=1}^{N} \bar{I}_i(t) + \sum_{\bar{t}}^{2500} \sum_{i=1}^{N} I_i(t)\right)$, where $\bar{t}$ is the time step during which the white spot was discovered, $I_i(t)$ is an indicator that a robot i has moved in time step t and $\bar{I}_i(t)$ is an indicator that a robot i has not moved in time step t.

4.2 Design methods

We consider Cedrata, as described in Section 3. As Cedrata had problems in producing communication-based strategies for the mission considered, we performed experiments with additional design methods. The design method Cedrata-GP uses the same constraints as Cedrata, but uses genetic programming [36] as the optimization algorithm. The parameters of this design method are adapted from the work of Jones et al. [40] and summarized in Table 2. We use the genetic programming implementation of the DEAP library [46]. The design method Cedrata-GE is also based on Cedrata, but uses grammatical evolution [37] as the optimization algorithm. We use the grammatical evolution implementation of PonyGE2 [47].

We also performed a number of manual designs. For the manual designs, we asked human designers—with prior experience in swarm robotics, but not with behavior trees—to design control software within the same constraints as Cedrata, that is with the same modules and architecture. The human designers had access to the AutoMoDe Editor [48], a tool that allows the designers to visualize and manipulate the behavior trees and to launch simulations of the designed behavior tree. The human designers received feedback about their designed behavior tree through the objective function and a visual representation of the arena and the behavior of the swarm.



(a) MARKER AGGREGATION (b) STOP

Fig. 3: The reference designs for the two missions. The conditions and actions names have been abbreviated in the following way: Exp: Exploration; Meet: Meeting; ESig: Emit Signal; Bflr: Black Floor; Gflr: Grey Floor; Wflr: White Floor; RSig: Receiving Signal.

Lastly, we include a reference design as an additional point of reference for the reader. These reference designs are not part of the experimental protocol and have been designed by us. They do not aim to be the best performing solutions for each mission but simply to provide a sensible solution. These designs serve to highlight particular strategies that we expected to be discovered in each mission.

4.3 Reference designs

The reference design for the mission MARKER AGGREGATION is shown in Figure 3a. In this design, robots explore the arena until they find the marker. Then, using the signal framework, they will attract their neighbors to the aggregation area.

The reference design for the mission STOP is shown in Figure 3b. In this design, robots will send and forward signals to their neighbors to transmit the information that the white spot has been discovered. If a robot received a signal, it stops; if it does not receive any signal, it explores the arena to find the white spot.

4.4 Protocol

For each mission, `Cedrata` is executed with different budgets: 20 000, 50 000, 100 000 and 200 000 simulation runs. The budget specifies the number of simulations that the design process is allowed to perform before it returns the best control software produced. Additionally, 10 runs of `Cedrata-GP` and `Cedrata-GE`, with a budget of 200 000 simulation runs are performed. For each combination of method, mission and budget, 10 independent runs of the methods are performed, leading to 10 instances of control software. The manual design are done by four human designers per mission, with a maximum design duration of 4 hours.

Simulations are performed in a realistic and physics-based simulation environment, based on the ARGoS simulator [49]. The simulated robots have a real world counter-part, and the simulator has been used in the past with this robotic platform and comparison between simulated performance and real-world performance have been made. In accordance with the consensus in the literature, a realistic noise model

Table 3: Design and pseudo-reality noise models

Sensor/actuator	design model	pseudo-reality model
Proximity	0.05	0.05
Light	0.05	0.90
Ground	0.05	0.05
Range-and-bearing	0.85	0.90
Wheels	0.05	0.15

is applied to the simulation (see Table 3). The generated instances of control software of all designs methods are assessed in pseudo-reality to investigate the impact of the reality gap. Ligot and Birattari [**?**] have shown that the effect of the reality gap can be mimicked in simulation-only environments, by testing the control software with a different noise model than it was originally designed for.

5 Results

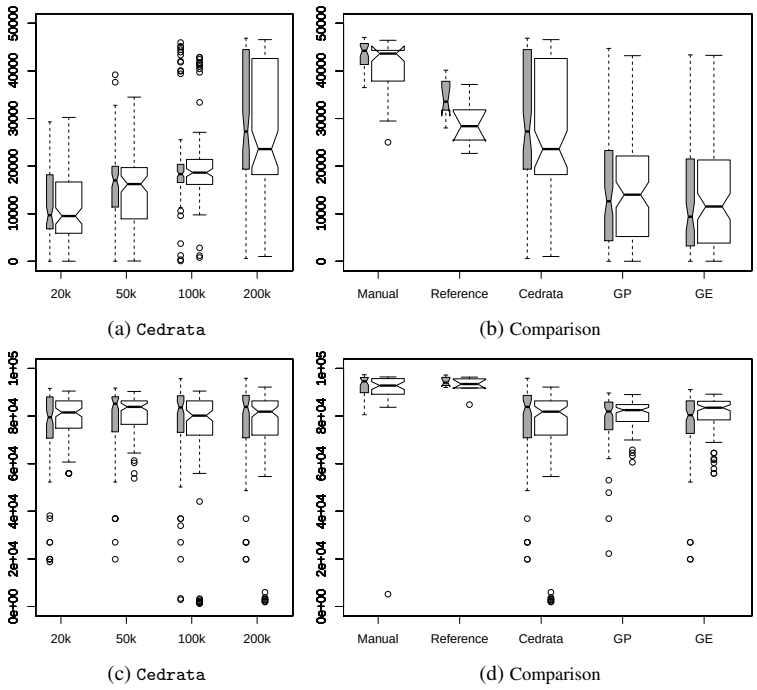
In this section, we describe the results obtained by the experimental setup described in Section 4. In the supplementary material [45], we also include an extended study, where we include another method that is based on another reference model which provides us with some additional insights tangential to the work presented here.

Figure 4 shows the results for the missions STOP and MARKER AGGREGATION. Results are shown for both the performance in simulation and pseudo-reality.

Figure 4a shows the development of the performance of `Cedrata` in the mission MARKER AGGREGATION. There is a clear trend of increasing performance with increasing budget. A detailed investigation of the generated control software reveals that `Cedrata` develops two general solution strategies: one strategy is based on the communication framework, while the other is not.

In the communication-less strategy (for an example see Figure 5a), the robots explore the arena until they discover the black spot, at which point they usually stop. In the communication-based strategy (see Figure 5b), however, the robots make use of the communication behaviors to quickly aggregate within the target area. The communication-based designs are similar in that regard to the reference design. The performance of `Cedrata` for each budget then seems to primarily depend on the ratio of the two strategies. Indeed, for design budgets of 20 000 and 50 000 simulation runs, Cedrata only produces control software that uses the communication-less strategy. For a budget of 100 000 simulation runs, `Cedrata` produces a single solution that follows the communication-based strategy and for a budget of 200 000 simulation runs, four designs make use of that strategy. It appears that the ratio of communication-less to communication-based strategies depends on the available budget. Indeed, as the communication-based strategy requires at least two modules to interact correctly, Iterated F-race is more likely to discover such a combination the more often it samples new solutions, which depends on the number of iterations and therefore the budget.

In Figure 4b, we can see the comparison of performances across all considered design methods. All manual designers found solutions that make use of communication. Their control software performs similar well as the communication-based be-



(a) `Cedrata` (b) Comparison

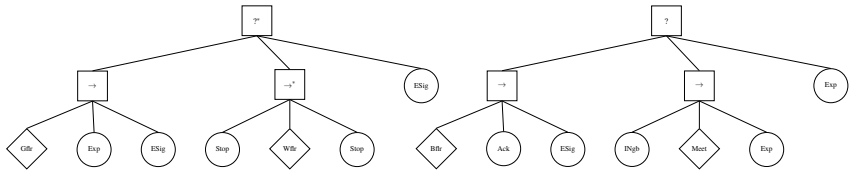
(c) `Cedrata` (d) Comparison

Fig. 4: Results for the mission MARKER AGGREGATION (top) and STOP (bottom). The left plots show the development of the performance over increasing budget for `Cedrata`. The right plots show the comparison of all design methods under consideration for a budget of 200 000 simulation runs. The thin plots present the results in simulation, the thick plots the results in pseudo-reality.

havior trees generated by `Cedrata` and better than the reference design, which was not meant to be the best performing solution, but just to highlight the general strategy. The human designers were therefore not only able to discover the strategy but also to find a reasonable tuning for the parameter.

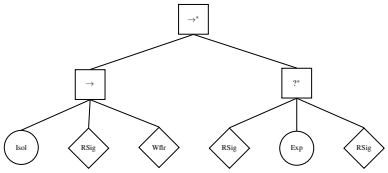
`Cedrata-GP` and `Cedrata-GE` both fail to generate any solution making use of the communication modules, even for a budget of 200 000 simulation runs. Interestingly, both design methods generate solutions that, under the right circumstances, perform nearly as good as the best instances of control software generated by `Cedrata`. However, this appear to be mostly due to the initial starting position favoring quick aggregation within the target zone and in total both `Cedrata-GP` and `Cedrata-GE` perform worse than `Cedrata`.

Figure 4c shows the development of performance over budget for `Cedrata` in the mission STOP. Unlike in MARKER AGGREGATION, there is no improvement for increasing budgets. Instead, the performance remains relatively stable. Investigation of the generated behavior trees reveals that `Cedrata` fails to make use of the communication modules for this mission. All generated behavior trees employ a strategy, where the robots are using the Isolation behavior (for an example, see Figure 5c). As



(a) MARKER AGGREGATION, without communication

(b) MARKER AGGREGATION, communication-based



(c) STOP

Fig. 5: Typical behavior trees generated by Cedrata.

a result, the swarm expands and with high probability a robot passes over the white spot. At the end of the expansion phase, the robots slow down and move relatively little, often falling below the threshold of 5 mm per time step. Some behavior trees also include an Exploration module for cases when no neighbors are detected.

Figure 4d displays a comparison of the performances of all design methods in the mission STOP. The manual designs, just like the reference design, make use of the communication framework and show the best performance. Both Cedrata-GP and Cedrata-GE find solutions that follow the same Isolation-based strategy as Cedrata and achieve similar performances. For all design methods, there are some runs where the performance is relatively close to 0. Often in these runs, the control software fails to find the white spot.

We made some observations that hold for all considered missions: The first observation is that all design methods show a relatively pseudo-reality gap. That is they experience only a small drop in performance, when assessing the control software in pseudo-reality. We believe that this is a fist indicator that Cedrata and the design methods based on it might transfer well into reality as well. A second observation is that all behavior trees generated by Cedrata, Cedrata-GP and Cedrata-GE contain a large number of modules that will never be ticked by the behavior tree. We believe this to be due to the reduced restriction in the architecture, which allows modules to be easily placed in the tree in a way that ensures they will never receive a tick. The design process has no explicit way of distinguishing necessary and superfluous modules and all techniques that aim at generating new behavior trees (random sampling around elites, cross-overs, mutations) are therefore highly likely to transfer some of the superfluous modules into the newly generated behavior tree. This poses a challenges to the automatic design process. Namely that the design process will spend some resources on tuning these superfluous modules, which have no influence on the behavior of the swarm, thus effectively wasting a part of the allocated budget. Lastly, we observed that the automatic design process had difficulties generating

communication-based behaviors. In both MARKER AGGREGATION and STOP, the human designers found well performing solutions that made use of the communication framework. Only in MARKER AGGREGATION was `Cedrata` able to generate at least a few solutions following a similar strategy. Our initial hypothesis was that this might have been caused by some properties of the underlying optimization algorithm, Iterated F-race. We have therefore replaced Iterated F-race with two different optimization algorithms, whose parametrization we have taken from other works of the swarm robotics literature. Unfortunately, both `Cedrata-GP` and `Cedrata-GP` appeared to have even greater difficulties generating communication-based behaviors than `Cedrata`. We believe that this could be due to the fact that communication requires two corresponding modules, a sender and a receiver, while all other strategies can rely on a single module.

6 Conclusion

In this work, we have extended AutoMoDe-`Cedrata`, by implementing two variants `Cedrata-GP` and `Cedrata-GE`, based on genetic programming and grammatical evolution. We have investigated the performance of these automatic design methods over a set of two missions and compared them to solutions found by human designers, following the same constraints. The results generated by the human designers show that the modules and constraints of `Cedrata` are sensible, as the human designers were able to design control software that performed satisfactorily. Furthermore, as the human designers had no prior knowledge of behavior trees, this seems to be an indicator that behavior trees are an intuitive control architecture to design for. The automatic design method `Cedrata`, on the other hand, was not able to generate communication-based behaviors. We hypothesized that this might have been due to some property of the optimization algorithm Iterated F-race, and therefore we created `Cedrata-GP` and `Cedrata-GE`, two variants of `Cedrata` that are based on genetic programming and grammatical evolution, respectively. Neither of these two variants was able to generate communication-based strategies either.

For future work, we would like to investigate in more detail how an automatic design process can discover meaningful communication-based strategies. The results of this work indicate that simply tuning the parameters of an optimization algorithm would probably not be enough. One possible approach could be to create an interleaved optimization process. Starting from a minimal communicating solution, we alternate between fixing the sending or the receiving part of the behavior tree and optimizing the remaining part of the tree. Another approach to solve this problem could be cooperative co-evolution. We could possibly create two distinct populations that are given a sending or receiving module respectively. This ensures the existence of communication from the starting population. Subsequent generations could then refine the communication protocol and integrate it with the other modules. Additionally, the results presented here showed that `Cedrata` and its variants were able to perform satisfactorily also in pseudo-reality. While this is an indicator that the design approaches might cross the reality gap well, we would like to confirm this hypothesis by performing real robot experiments.

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. Rubenstein, M., Cornejo, A., Nagpal, R.: Programmable self-assembly in a thousand-robot swarm. Science **345**(6198), 795–799 (2014). DOI 10.1126/science.1254295
2. Werfel, J., Petersen, K., Nagpal, R.: Designing collective behavior in a termite-inspired robot construction team. Science **343**(6172), 754–758 (2014). DOI 10.1126/science.1245842
3. Yang, G.Z., Bellingham, J., Dupont, P.E., Fischer, P., Floridi, L., Full, R., Jacobstein, N., Kumar, V., McNutt, M., Merrifield, R., Nelson, B.J., Scassellati, B., Taddeo, M., Taylor, R., Veloso, M., Wang, Z.L., Wood, R.: The grand challenges of Science Robotics. Science Robotics **3**(14), eaar7650 (2018). DOI 10.1126/scirobotics.aar7650
4. Garattoni, L., Birattari, M.: Autonomous task sequencing in a robot swarm. Science Robotics **3**(20), eaat0430 (2018). DOI 10.1126/scirobotics.aat0430
5. Slavkov, I., Carrillo-Zapata, D., Carranza, N., Diego, X., Jansson, F., Kaandorp, J., Hauert, S., Sharpe, J.: Morphogenesis in robot swarms. Science Robotics **3**(25), eaau9178 (2018). DOI 10.1126/scirobotics.aau9178
6. Yu, J., Wang, B., Du, X., Wang, Q., Zhang, L.: Ultra-extensible ribbon-like magnetic microswarm. Nature Communications **9**(1), 3260 (2018). DOI 10.1038/s41467-018-05749-6
7. Li, S., Batra, R., Brown, D., Chang, H.D., Ranganathan, N., Hoberman, C., Rus, D., Lipson, H.: Particle robotics based on statistical mechanics of loosely coupled components. Nature **567**(7748), 361–365 (2019). DOI 10.1038/s41586-019-1022-9
8. Xie, H., Sun, M., Fan, X., Lin, Z., Chen, W., Wang, L., Dong, L., He, Q.: Reconfigurable magnetic microrobot swarm: multimode transformation, locomotion, and manipulation. Science Robotics **4**(28), eaav8006 (2019). DOI 10.1126/scirobotics.aav8006
9. Dorigo, M., Theraulaz, G., Trianni, V.: Reflections on the future of swarm robotics. Science Robotics **5**, eabe4385 (2020). DOI 10.1126/scirobotics.abe4385
10. Birattari, M., Ligot, A., Hasselmann, K.: Disentangling automatic and semi-automatic approaches to the optimization-based design of control software for robot swarms. Nature Machine Intelligence **2**(9), 494–499 (2020). DOI 10.1038/s42256-020-0215-0
11. Hasselmann, K., Ligot, A., Ruddick, J., Birattari, M.: Empirical assessment and comparison of neuro-evolutionary methods for the automatic off-line design of robot swarms. Nature Communications **12**, 4345 (2021). DOI 10.1038/s41467-021-24642-3
12. Dorigo, M., Birattari, M., Brambilla, M.: Swarm robotics. Scholarpedia **9**(1), 1463 (2014). DOI 10.4249/scholarpedia.1463
13. Birattari, M., Ligot, A., Bozhinoski, D., Brambilla, M., Francesca, G., Garattoni, L., Garzón Ramos, D., Hasselmann, K., Kegeleirs, M., Kuckling, J., Pagnozzi, F., Roli, A., Salman, M., Stützle, T.: Automatic off-line design of robot swarms: a manifesto. Frontiers in Robotics and AI **6**, 59 (2019). DOI 10.3389/frobt.2019.00059
14. Hamann, H., Wörn, H.: A framework of space–time continuous models for algorithm design in swarm robotics. Swarm Intelligence **2**(2–4), 209–239 (2008). DOI 10.1007/s11721-008-0015-3
15. Kazadi, S.: Model independence in swarm robotics. International Journal of Intelligent Computing and Cybernetics **2**(4), 672–694 (2009). DOI 10.1108/17563780911005836
16. Berman, S., Kumar, V., Nagpal, R.: Design of control policies for spatially inhomogeneous robot swarms with application to commercial pollination. In: 2011 IEEE International Conference on Robotics and Automation (ICRA), pp. 378–385. IEEE, Piscataway, NJ, USA (2011). DOI 10.1109/ICRA.2011.5980440
17. Beal, J., Dulman, S., Usbeck, K., Viroli, M., Correll, N.: Organizing the aggregate: languages for spatial computing. In: M. Marjan (ed.) Formal and Practical Aspects of Domain-Specific Languages: Recent Developments, pp. 436–501. IGI Global, Hershey, PA, USA (2012). DOI 10.4018/978-1-4666-2092-6.ch016
18. 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 Transactions on Autonomous Adaptive Systems **9**(4), 17:1–17:28 (2014). DOI 10.1145/2700318

19. Reina, A., Valentini, G., Fernández-Oto, C., Dorigo, M., Trianni, V.: A design pattern for decentralised decision making. PLOS ONE **10**(10), e0140950 (2015). DOI 10.1371/journal.pone.0140950
20. Lopes, Y.K., Trenkwalder, S.M., Leal, A.B., Dodd, T.J., Groß, R.: Supervisory control theory applied to swarm robotics. Swarm Intelligence **10**(1), 65–97 (2016). DOI 10.1007/s11721-016-0119-0
21. Pinciroli, C., Beltrame, G.: Buzz: a programming language for robot swarms. IEEE Software **33**(4), 97–100 (2016). DOI 10.1109/MS.2016.95
22. Hamann, H.: Swarm robotics: a formal approach. Springer, Cham, Switzerland (2018). DOI 10.1007/978-3-319-74528-2
23. Brambilla, M., Ferrante, E., Birattari, M., Dorigo, M.: Swarm robotics: a review from the swarm engineering perspective. Swarm Intelligence **7**(1), 1–41 (2013). DOI 10.1007/s11721-012-0075-2
24. Francesca, G., Birattari, M.: Automatic design of robot swarms: achievements and challenges. Frontiers in Robotics and AI **3**(29), 1–9 (2016). DOI 10.3389/frobt.2016.00029
25. 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 Intelligence **8**(2), 89–112 (2014). DOI 10.1007/s11721-014-0092-4
26. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T.: The irace package: iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016). DOI 10.1016/j.orp.2016.09.002
27. 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**(2–3), 125–152 (2015). DOI 10.1007/s11721-015-0107-9
28. Hasselmann, K., Birattari, M.: Modular automatic design of collective behaviors for robots endowed with local communication capabilities. PeerJ Computer Science **6**, e291 (2020). DOI 10.7717/peerj-cs.291
29. Garzón Ramos, D., Birattari, M.: Automatic design of collective behaviors for robots that can display and perceive colors. Applied Sciences **10**(13), 4654 (2020). DOI 10.3390/app10134654
30. Ligot, A., Hasselmann, K., Birattari, M.: AutoMoDe-Arlequin: neural networks as behavioral modules for the automatic design of probabilistic finite state machines. In: M. Dorigo, T. Stützle, M.J. Blesa, C. Blum, H. Hamann, M.K. Heinrich, V. Strobel (eds.) Swarm Intelligence: 12th International Conference, ANTS 2020, *Lecture Notes in Computer Science*, vol. 12421, pp. 109–122. Springer, Cham, Switzerland (2020). DOI 10.1007/978-3-030-60376-2_21
31. Salman, M., Ligot, A., Birattari, M.: Concurrent design of control software and configuration of hardware for robot swarms under economic constraints. PeerJ Computer Science **5**, e221 (2019). DOI 10.7717/peerj-cs.221
32. Kuckling, J., Ubeda Arriaza, K., Birattari, M.: AutoMoDe-IcePop: automatic modular design of control software for robot swarms using simulated annealing. In: B. Bogaerts, G. Bontempi, P. Geurts, N. Harley, B. Lebichot, T. Lenaerts, G. Louppe (eds.) Artificial Intelligence and Machine Learning: BNAIC 2019, BENELEARN 2019, *Communications in Computer and Information Science*, vol. 1196, pp. 3–17. Springer, Cham, Switzerland (2020)
33. Ligot, A., Kuckling, J., Bozhinoski, D., Birattari, M.: Automatic modular design of robot swarms using behavior trees as a control architecture. PeerJ Computer Science **6**, e314 (2020). DOI 10.7717/peerj-cs.314
34. Kuckling, J., van Pelt, V., Birattari, M.: Automatic modular design of behavior trees for robot swarms with communication capabilities. In: P.A. Castillo, J.L. Jiménez Laredo (eds.) Applications of Evolutionary Computation: 24th International Conference, EvoApplications 2021, *Lecture Notes in Computer Science*, vol. 12694, pp. 130–145. Springer, Cham, Switzerland (2021). DOI 10.1007/978-3-030-72699-7_9
35. Marzinotto, A., Colledanchise, M., Smith, C., Ögren, P.: Towards a unified behavior trees framework for robot control. In: 2014 IEEE International Conference on Robotics and Automation (ICRA), pp. 5420–5427. IEEE, Piscataway, NJ, USA (2014). DOI 10.1109/ICRA.2014.6907656
36. Koza, J.R.: Genetic programming: on the programming of computers by means of natural selection, first edn. MIT Press, Cambridge, MA, USA (1992). A Bradford Book
37. O'Neill, M., Ryan, C.: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language, first edn. Genetic Programming Series. Springer, Boston, MA, USA (2003). DOI 10.1007/978-1-4615-0447-4
38. Isla, D.: Handling complexity in the Halo 2 AI. In: Game Developers Conference, GDC 2005, vol. 12. Game Developers Conference (GDC), London, United Kingdom (2005)

39. Colledanchise, M., Ögren, P.: Behavior Trees in Robotics and AI: An Introduction, 1st edn. Chapman & Hall/CRC Artificial Intelligence and Robotics Series. CRC Press, Boca Raton, FL, USA (2018). DOI 10.1201/9780429489105
40. Jones, S., Studley, M., Hauert, S., Winfield, A.: Evolving behaviour trees for swarm robotics. In: R. Groß, A. Kolling, S. Berman, E. Frazzoli, A. Martinoli, F. Matsuno, M. Gauci (eds.) Distributed Autonomous Robotic Systems: The 13th International Symposium, *Springer Proceedings in Advanced Robotics*, vol. 6, pp. 487–501. Springer, Cham, Switzerland (2018). DOI 10.1007/978-3-319-73008-0_34
41. Jones, S., Winfield, A., Hauert, S., Studley, M.: Onboard evolution of understandable swarm behaviors. Advanced Intelligent Systems **1**(6), 1900031 (2019). DOI 10.1002/aisy.201900031
42. Neupane, A., Goodrich, M.: Learning swarm behaviors using grammatical evolution and behavior trees. In: S. Kraus (ed.) Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19, pp. 513–520. IJCAI Organization, CA, USA (2019). DOI 10.24963/ijcai.2019/73
43. Ferrante, E., Duéñez-Guzmán, E.A., Turgut, A.E., Wenseleers, T.: GESwarm: grammatical evolution for the automatic synthesis of collective behaviors in swarm robotics. In: C. Blum (ed.) GECCO'13: Proceedings of the 15th annual conference on Genetic and evolutionary computation, pp. 17–24. ACM, New York, NY, USA (2013). DOI 10.1145/2463372.2463385
44. Hasselmann, K., Ligot, A., Francesca, G., Garzón Ramos, D., Salman, M., Kuckling, J., Mendiburu, F.J., Birattari, M.: Reference models for AutoMoDe. Tech. Rep. TR/IRIDIA/2018-002, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (2018)
45. Kuckling, J., van Pelt, V., Birattari, M.: AutoMoDe-Cedrata: automatic design of behavior trees for controlling a swarm of robots with communication capabilities: supplementary material. http://iridia.ulb.ac.be/supp/IridiaSupp2021-004/ (2021)
46. Fortin, F.A., De Rainville, F.M., Gardner, M.A., Parizeau, M., Gagné, C.: DEAP: evolutionary algorithms made easy. Journal of Machine Learning Research **13**, 2171–2175 (2012)
47. Fenton, M., McDermott, J., Fagan, D., Forstenlechner, S., Hemberg, E., O'Neill, M.: PonyGE2: grammatical evolution in Python. https://arxiv.org/abs/1703.08535 (2017)
48. Kuckling, J., Hasselmann, K., van Pelt, V., Kiere, C., Birattari, M.: AutoMoDe Editor: a visualization tool for AutoMoDe. Tech. Rep. TR/IRIDIA/2021-009, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (2021)
49. Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G.A., Ducatelle, F., Birattari, M., Gambardella, L.M., Dorigo, M.: ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intelligence **6**(4), 271–295 (2012). DOI 10.1007/s11721-012-0072-5