Automatic Modular Design of Behavior Trees for Robot Swarms with Communication Capabilites

Jonas Kuckling, Vincent van Pelt, and Mauro Birattari(✉)

IRIDIA, Université Libre de Bruxelles, Brussels, Belgium
mbiro@ulb.ac.be

Abstract. In this work, we develop a set of behavioral and conditional modules for the use with behavior trees. We present AutoMoDe-`Cedrata`, an automatic modular design method that automatically assembles and fine-tunes these modules into behavior trees that control robot swarms. We test `Cedrata` on three missions and, to gain further insights on its effectiveness, we design control software for the same missions using AutoMoDe-`Maple`, another automatic design method, and by a group of human designers. Results show that the proposed modules allow for well-performing behavior trees. Yet, `Cedrata` had difficulties automatically generating control software that performs similarly well as the one generated by human designers, especially when involving communication.

Keywords: Swarm robotics · Automatic design · Behavior trees

1 Introduction

Swarm robotics is the combination of robotics and swarm intelligence, a field inspired by social insects such as bees or ants [3]. In swarm robotics, a set of relatively simple robotic agents are designed to solve collectively a task without any central control [4]. One difficulty of designing control software in such a setting is predicting the collective behavior emerging from the local interactions. Multiple approaches to the design of robot swarms have been studied and can be classified in two categories: manual and automatic design. In this work, we are interested in *fully automatic off-line design* [1,2]. In automatic off-line design, the problem of designing control software for a robot swarm is transformed into an optimization problem. The optimization problem is then solved without any human intervention. The software, generated by the optimization process in simulation, is then uploaded to the robots that operate in the real world.

JK and VvP contributed equally to this work and should be considered as co-first authors. The experiments were designed by JK and VvP and performed by 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.

P. A. Castillo and J. L. Jiménez Laredo (Eds.): EvoApplications 2021, LNCS 12694, pp. 130–145, 2021.
https://doi.org/10.1007/978-3-030-72699-7_9

Francesca et al. [6] proposed AutoMoDe (automatic modular design), a promising automatic off-line design approach, that assembles and fine-tunes pre-defined modules into control software. Different versions of AutoMoDe have been proposed, that investigate different aspects of the design process, such as new modules based on communication [8] or colors [7], different optimization algorithms, such as Iterated F-race [5], iterated improvement [17], or simulated annealing [18], or different control architectures, such as behavior trees [21].

AutoMoDe flavors traditionally assemble their pre-defined modules into probabilistic finite-state machines. AutoMoDe-`Maple` [15,21] introduced behavior trees [23] as a control software structure. Behavior trees have been originally developed for video games [11] but recently found application in other fields of research, such as swarm robotics [12,13,24]. A behavior tree is a tree structure that contains multiple types of nodes. The root node generates a *tick* with a fixed frequency. This tick is propagated through the tree. A node that receives the tick activates, and either distributes the tick to one or multiple of its children, or it can return the tick to its parent, along with a return value out of *success*, *failure*, or *running*. The inner nodes of the tree are called *control-flow nodes*. These nodes determine how the tick is propagated through the tree. Leaf nodes are either *condition nodes* or *actions nodes*, that respectively test sensor input or execute a unitary task. For a formal definition of the node types, see Marzinotto et al. [23]. Behavior trees offer several advantages over finite-state machines, such as modularity, two-way control transfers and improved human understandability. In one-way control transfer systems control can only be transferred in one direction, making it akin to the "goto" statement in programming [26]. In two-way control transfer systems, control can be transferred in both directions, that is the receiver can return the control to its predecessor along with information about the execution, similar to functions and their return values. `Maple` makes use of modules that have been originally designed for finite-state machines and that therefore do not allow the use of return values. As a result, the behavior trees created by `Maple` could not use the two-way control transfers.

Here, we propose a new set of modules that explicitly provide these return values and therefore enable two-way control transfers. We present AutoMoDe-`Cedrata`, an automatic modular design method that assembles these modules into behavior trees. We test `Cedrata` on three missions. To better appraise the effectiveness of `Cedrata`, we asked human designers in swarm robotics to perform manual designs within the constraints of `Cedrata`. No automatic design method exists for designing behavior trees for robot swarms that is based on the same reference model of `Cedrata`. The closest alternative, in terms of reference model, is AutoMoDe-`Maple` [21]. We therefore include `Maple` in our study. However, as `Maple` and `Cedrata` do not share the same reference model, any direct comparison of performance is meaningless. Instead, we will use the behavior trees generated by `Maple` to understand better the quality of those of `Cedrata`.

2 Related Work

Behavior trees have received little attention in swarm robotics so far. Jones et al. [12] used genetic programming to evolve behavior trees in a foraging mission for a swarm of kilobots. The authors were able to generate control software that performed satisfactorily in the mission. They could show that the generated control software was easily human readable. In another work, Jones et al. evolved behavior trees onboard a swarm of Xpucks in a cooperative transportation mission [13]. The authors showed that the generated control software performs satisfactorily, and that even though the evolved behavior trees may contain many modules, they can easily be reduced into a concise representation.

Ligot et al. have investigated the use of behavior trees in automatic modular design [21]. They proposed AutoMoDe-`Maple`, which assembles modules into a restricted behavior tree architecture. However, the modules for `Maple` were originally conceived to be used in finite-state machines. These behavioral modules could not provide any return values, instead they were conceived to run indefinitely. The authors restricted the allowed behavior tree structures to make use of these modules. The results of their experiments show, that for smaller design budgets, the restricted behavior trees perform similar as finite-state machines. For higher budgets, the restricted behavior tree architecture proved to be too limiting, and finite-state machines could generate control software that could not be represented within the restricted behavior tree architecture.

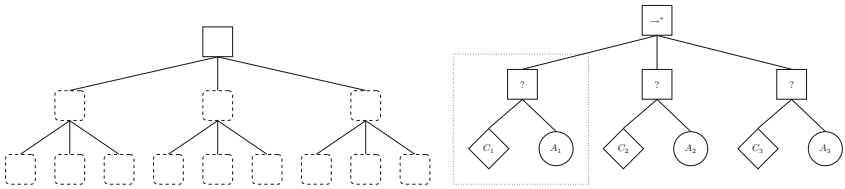
Hasselmann and Birattari proposed AutoMoDe-`Gianduja`, an automatic modular design method able to design control software for a swarm of e-puck robots that have local communication abilities [8,10]. The modules of `Gianduja` could send and receive different messages that had no prior assigned semantic. The authors showed that the design process could generate control software that meaningfully used the communication capabilities of the modules.

3 AutoMoDe-Cedrata

3.1 Behavior Tree Structure

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 needs to be a control-flow node. Nodes of the second level can be either control-flow nodes, action nodes or condition nodes. If it is an action node or a condition node, then it 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 Fig. 1a. 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. [23]. 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) Behavior tree structure for `Cedrata`. (b) Behavior tree structure for `Maple`.

Fig. 1. The possible behavior tree structures for `Cedrata` and `Maple`. 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. In `Maple`, the top-level node is fixed to sequence* node. Underneath it the tree can have between one and four selector subtrees (highlighted by the dotted border). Each selector subtree consists of one selector node with exactly two children, a condition node (with associated condition C_i) and an action node (with associated behavior A_i).

3.2 Reference Model RM2.2

Table 1. The E-puck reference model RM2.2 used by `Cedrata` [9].

Sensors	Variables
Proximity	$prox_i \in [0, 1]$, $\angle q_i$, with $i \in \{1, 2, ..., 8\}$
Ground	$gnd_i \in \{0, 0.5, 1\}$, with $i \in \{1, 2, 3\}$
Range-and-bearing	$n \in \mathbb{N}$
	r_m, $\angle b_m$, $s_m \in \{0, 1, ..., 6\})$, for $m \in \{1, 2, ..., n\}$
Actuators	Variables
Signal broadcast	$s \in \{0, 1, ..., 6\}$
Wheels	$v_l, v_r \in [-v, v]$, with $v = 0.16$m/s
Control cycle period: 100 ms	

The reference model RM2.2 is shown in Table 1 [9]. The robot has access to eight proximity sensors, three ground sensors and one range-and-bearing board for sensing. It has access to two sets of actuators: the range-and-bearing board to send messages and two wheels with differential drive. A robot 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 $\{1, ..., 6\}$. Signal values do not have a particular semantic, instead it is the role of the design process to assign semantics to the signals. The reference model also provides access to the number of neighboring robots n and for each neighboring robot m, it provides a three-tuple of the estimated distance r_m, the angle $\angle_m$ and the received signal s_m. The control cycle period is 100 ms, that is, every 100 ms the sensors and the control software are updated.

3.3 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.

Conditions. The set of conditions is shown below. 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.

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.

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.

Behaviors. The new set of behaviors is shown below. 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 to be a success or a failure. Otherwise, they return *running*.

Exploration. The robot performs a random walk strategy. It moves straight until it perceives an obstacle in front of itself. Then the robot turns on the spot for a random number of ticks 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 return *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 $w' = w_{r\&b} = \sum_{m=1}^{n}(\frac{\alpha}{r_m}, \angle b_m)$, 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 $w_0 = \sum_{i=1}^{8}(prox_i, \angle q_i)$.

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 use the same embedded collision avoidance than 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 behaviors 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: $w' = w_{r\&b} = \sum_{m \in S_r^*}(\frac{\alpha}{r_m}, \angle b_m)$ if robots are perceived, where S_r^* is the set of robots that emit the signal s. 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 $s \in S \cup \{0\}$ for the current tick, where s is a tunable parameter. This behavior always returns *success*. This behavior also sets the wheel velocity to zero.

3.4 Optimization Algorithm

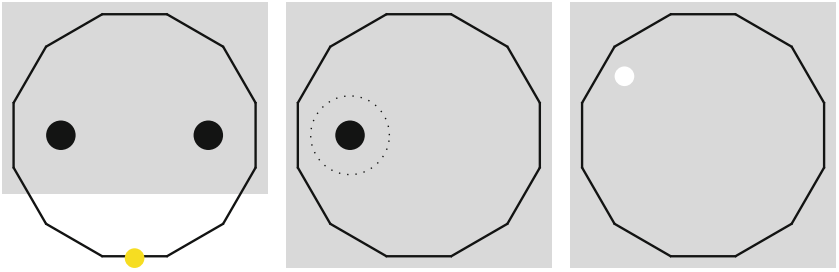
`Cedrata` uses Iterated F-race [22] as the optimization algorithm. Iterated F-race works over multiple iterations, each of them reminiscent of a race. In each iteration, a set of candidate solutions is sampled. The candidate solutions are compared over an increasing number of instances. Once a candidate solution

performs significantly worse than another, it is eliminated from the race, freeing up the budget for evaluations of more promising candidates. If only one candidate remains, or the budget for this iteration has been exhausted, new candidate solutions will be generated by sampling around the surviving candidates. These new candidates form the set of candidate solutions for the next iteration of the algorithm. For `Cedrata`, a candidate solution is a behavior tree, with the structure and modules as defined above. The optimization algorithm is free to choose any combinations of nodes, modules, and tunable parameters within these constraints. These behavior trees will be evaluated on the same mission, but with different initial starting positions and headings.

4 Experimental Setup

4.1 Missions

We consider three missions: Foraging, Marker Aggregation and Stop. All missions take place in a dodecagonal arena (see Fig. 2) and last 250 s.



(a) Foraging (b) Marker Aggregation (c) Stop

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

In Foraging (see Fig. 2a), the robots are tasked to perform an abstracted foraging task. That is, the the robots must bring as many items from the black sources inside the white nest. As the e-puck robot does not have gripping capabilities, we assume it picks up an item when it enters a black area and that it deposits a carried item when it enters the white area. The objective function for this mission is the number of recovered items: $F_{For} = \#items$.

In Marker Aggregation (see Fig. 2b), 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 Fig. 2c), the robots must find a white spot and then stop as quickly as possible. A robot is considered moving, if it has travelled more than

5 mm in 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

In this work, we study the effectiveness of `Cedrata`. To gain further insights on the quality of the generated control software, we also design control software using `Maple`. Additionally, we asked a set of human designers to manually design control software using the set of modules and the tree structure of `Cedrata`.

For a definition of `Cedrata` see Sect. 3. `Maple` [21] is another automatic modular design method that assembles the modules into behavior trees. It has access to six conditions (Black Floor, Grey Floor, White Floor, Neighborhood Count, Inverted Neighborhood Count, and Fixed Probability) and six behaviors (Exploration, Stop, Phototaxis, Anti-Phototaxis, Attraction, and Repulsion). As the behaviors have been originally defined for finite-state machines, they can only return *running*. To allow the use of these behaviors in behavior trees, `Maple` restricts the structure of the generated trees and can only generate trees in the shape shown in Fig. 1b. Underneath a top-level sequence* node are between one and four selector sub-trees. The first selector sub-tree in the example is highlighted using a dotted box. Each selector sub-tree consists of one selector node with a condition node as its first child and an action node as its second child. `Maple` uses Iterated F-race [22] as its optimization algorithm.

For the manual designs, a human designer builds the control software of the robot using the same constraints (modules and behavior tree structure) as `Cedrata`. For the design process, the designers have access to a visual interface that allows them to visualize and manipulate the trees and to directly launch simulations of the control software. They have access to the value of the objective function as automatic methods and to a visual representation of the arena and the behavior of the swarm for inspection. The human designers chosen have expertise in swarm robotics, but do not have prior knowledge of behavior trees or the specific module set of `Cedrata`.

4.3 Protocol

The automatic designs are conducted according to the following protocol. For each mission, `Cedrata` is executed with different budgets: 20 000, 50 000, 100 000 and 200 000 simulation runs. After this number of simulations, the automatic design process is halted and it returns the best control software produced. For each budget, 10 runs of the methods are run, leading to 10 instances of control

software. Additionally, 10 runs of `Maple` with a budget of 200 000 simulation runs will be performed. The manual design will be done by four human designers per mission, with a maximum design duration of 4 h.

Table 2. 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

Simulations are performed in the ARGoS simulator [25], a realistic and physics-based simulator. In accordance with the consensus in the literature, a realistic noise model is applied to the simulation (see Table 2). 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 [20] showed 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.

4.4 Reference Designs

Besides the behavior trees created by the manual designs method, we will define a *reference* tree for each mission. These reference designs are not part of the experimental protocol and are designed by people with knowledge of the study. They are built using the same constraints on the tree structure as the manual designs, but without time constraints. These designs serve to highlight particular strategies that we expected to be discovered in each mission. They were not known to the human designers prior to their manual designs. By comparing the results of `Cedrata` to the reference designs and the manual designs of the human designers, we can gain insights on the effectiveness of `Cedrata`. Furthermore, by comparing the results of the human designers to the reference design, we can make statements about the general usability of behavior trees, as no human designer had prior experience with behavior trees.

Foraging. This mission was initially conceived for a reference model, which had access to the light sensor. As such, a light was placed behind the nest area to guide robots in their search. However, `Cedrata` does not have any light detection capabilities, therefore preventing use of this information not only for the automatic design but also for a human designer. The reference design for this mission is available in the supplementary material [16]. In this design, robots make use of the signal framework to send indications about the location of the

food sources to their neighbors. When a robot finds a food source, it emits a signal. Robots that are in search of a food source can then receive the signal to attract them to it.

Marker Aggregation. This mission was designed to encourage the design process to make use of the signal framework. The reference design for this mission is available in the supplementary material [16]. 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.

Stop. The reference design for this mission is available in the supplementary material [16]. 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.

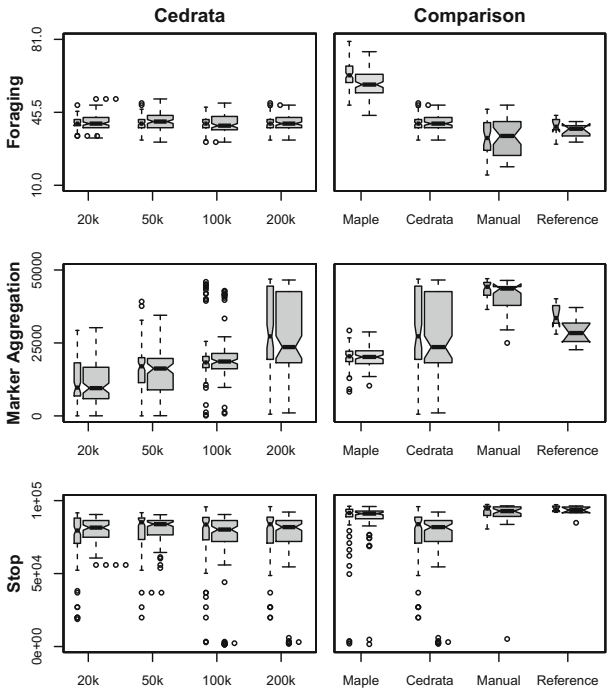


Fig. 3. Results for all conducted experiments. Each row contains all experiments for a particular mission. The first column shows the development of `Cedrata` with increasing budgets. The second column contains a comparison between `Cedrata`, `Maple` (both for a budget of 200k), the manual, and the reference designs. Results are shown both in the design context (thin boxes) and pseudo-reality context (thick boxes).

5 Results

In the following sections, methods are often claimed to "perform significantly better" or "outperform" another method. It implies that a Wilcoxon rank sum test has been performed with a confidence of 95%.

Foraging. The performance of the design methods on the FORAGING mission is shown in Fig. 3. All plots include the results in both design and pseudo-reality environments. The plots show, that for `Cedrata` despite increasing budgets the performance remains similar. A detailed inspection of the created behavior trees shows that the adopted strategies of `Cedrata` are the same regardless of the budget. For `Cedrata`, all the generated behavior trees contain an Exploration behavior which make the robot explore until the end of the mission. Eventually, the robot will pass over a food source and then over the nest. The fact that these strategies do not evolve with the budget size implies that finding them is not a difficult task for the optimization process, as it does so even with low budgets. A typical example behavior tree is available in the supplementary material [16]. In this example, we can see an Exploration behavior but also two conditions that could have been removed: the Fixed Probability condition does not trigger a particular action, and the Neighborhood Count condition is, in this case, too high to ever return *success*. As for the shown tree, a lot of generated instances of control software contain what we can call *superfluous* modules, i.e., modules that do not play a part in the strategy. We can also observe these superfluous modules in the control software generated with `Maple`, but in a more limited way. This could be explained by the constraints on the tree structure: `Maple` imposes sub-trees that have exactly one condition and one action node, leaving few possibilities to add extraneous nodes. On the other hand, a superfluous module can easily be added in `Cedrata` and, as it will not influence the performance of the swarm, be kept throughout the optimization process.

The human designers used the same strategy as `Cedrata`, which is based on the Exploration behavior, achieving similar performance as `Cedrata`. Although the strategies from `Cedrata` and the manual designs are similar, the performance of the control software generated by the human designers shows a wider variance. This is mainly caused by one human designer, whose control software does not seem as fine-tuned as the one by the other designers. This shows one of the major drawbacks of manual design, the dependance on the specific abilities of the human designer. Contrary to our hypothesis, the communication-based reference design does not outperform the exploration-based strategies found by either `Cedrata` or the human designers. `Maple`, due to its different reference model, can make use of the light source to forage more efficiently than the other designs. When assessed in pseudo-reality, `Cedrata`, the manual designs, and the reference design only suffer from small drops of the performance. This is indication that `Cedrata` could also demonstrate to be resistant against the reality gap.

Marker Aggregation. The performance of the design methods on MARKER AGGREGATION is shown in Fig. 3. All plots include the results in both design and pseudo-reality environments. The performance of `Cedrata` clear improvements

with the size of the budget. Cedrata develops two primary strategies: one that uses communication to indicate the position of the black spot, similar to the reference design, and one that only explores until the black spot is found, without explicitly communicating its position. The total performance of the automatic design is influenced by how many strategies of each kind are generated. In this experiment, the designs generated with the lower budget sizes 20 000 and 50 000 contain only behavior trees using the second strategy. The experiment with a budget of 100 000 simulations produced one behavior tree using a reference-like strategy. When the budget increases to 200 000 simulations, this amount increase to four designs finding that strategy. This increasing proportion of reference-like strategies seems to be linked to the budget size. As communication-based strategies usually require two matching modules sending the same signal, this could be an indication that the generation of communication-based strategies benefits from a more thorough exploration of the search space, as opposed to simple exploitation of previously found solutions. The generated behavior trees also contain superfluous modules, as already seen in Foraging. An example of a behavior tree using the pure exploration strategy is available in the supplementary material [16]. This behavior tree contains some modules that are not useful: the two Emit Signal behaviors send signals that will never be perceived, as no other signal-based modules are present; and the sequence* sub-tree will always only execute its first child because the Stop behavior will always return *running*. This can be explained using the same reasoning that we have proposed for the Foraging mission, which is that Cedrata is flexible regarding the tree structure and therefore allows more choices for placing superfluous modules.

The manual designs use a strategy similar to the one used in the reference design. Yet the human designers where able to find better fine-tuned instances of control software than the reference design. When compared to Cedrata, the designs generated by the human designers seem to outperform Cedrata. If the comparison is however restricted to the four reference-like designs that Cedrata produced for a budget of 200 000 simulation runs, then both design methods perform similarly. With no access to communication through its reference model, Maple only generates behavior trees similar to the behavior trees of the communication-less strategy found by Cedrata. When comparing the generated behavior trees, Cedrata also leads to more variety in the trees than Maple: for the same strategy, trees with different topologies can be created. Also in this mission, Cedrata, the manual designs, and the reference design successfully manage to mitigate the effects of the pseudo-reality gap.

Stop. The performance of the design methods on the Stop mission is shown in Fig. 3. All plots include the results in both design and pseudo-reality environments. As in Foraging, Cedrata shows similar performance and uses the same strategy for all budget sizes, meaning that the strategy is easily discovered by the optimization process. Cedrata uses the following strategy: robots isolate from each other. The swarm expands and covers all the arena, giving a high probability for a single robot to move over the white spot in the process. As a robot that moves slower than 5 mm per second is considered to be not moving by

the objective function, and robots in the Isolation behavior often pass under this threshold when they are far away from other robots, the resulting performance is relatively good. Some trees have an Exploration behavior for when robots do not detect neighbors, one such example is available in the supplementary material [16]. This tree contains again some superfluous modules, especially the three Receiving Signal conditions for signals that can't be sent, which is very common for control software generated with `Cedrata`. In this mission, `Cedrata` does not exploit the communication abilities of the modules. Following the discussion of the mission Marker Aggregation, this could be attributed to a lack of exploration of the search space. The simplicity of the isolation strategy might lead to the optimization process prematurely converging around these solutions. The human designers used strategies that are similar to the one used in the reference design. As we hypothesized, the correct use of communication leads to behavior trees that outperform those that do not use communication (in this case the ones generated by `Cedrata` and `Maple`). `Maple` finds a strategy where the robots randomly explores the arena until it finds a sufficient number of neighbors to stop. This strategy could have also been discovered by `Cedrata`, as it only makes use of the modules Exploration, Neighborhood Count, and Stop, which are available to both design methods. This could be another indication that the optimization process of `Cedrata` converged prematurely towards a too simplistic solution, which was not available for `Maple`. For all methods, some simulation runs, either in the design or in the pseudo-reality environment, show very low results (almost equal to zero) compared to the other simulation runs. These results correspond to some experiments where no robot finds the white spot or the spot is found within the last seconds of the experiment. As in the previous two missions, `Cedrata`, the manual designs, and the reference design showed to be resistant against the pseudo-reality gap.

6 Conclusion

In this work, a new flavour of AutoMoDe called `Cedrata` has been introduced to pursue the work started with a previous one called `Maple`, which introduced behavior trees. `Maple` uses modules from earlier flavours, which have been designed for finite state machines. This forces the behavior tree to adopt a particular structure. `Cedrata` introduces a new set of modules that are specifically designed for behavior trees and allow the tree to have a more flexible structure. We tested `Cedrata` on three different missions. For each mission we included designs by `Maple`, manual designs, done by human designers, and reference designs, created with the objective of serving as examples. Multiple observations can be extracted from the results. The modules and behavior tree structure for `Cedrata` allow for well-performing instances of control software. Indeed, in two of the three considered missions, designs following on `Cedrata` constraints are able to outperform `Maple`, which had no access to communication capabilities. In the third mission, `Maple` outperforms all design methods operating within the constraints of `Cedrata`. This is because `Maple` has access to an ambient clue (the

light) that was not available for Cedrata. We hypothesized that communication-based strategies might offset this disadvantaged, but our results showed that there was no gain in performance for this particular mission. The manual designs performed, in average, as well as the reference ones. This indicates that, under the experimental conditions, designers with no prior knowledge of behavior trees are able to understand and use them to solve missions efficiently. This highlights the human understandability, one of the often claimed advantages of behavior trees.

While behavior trees are convenient to design for human designers, it seems to be more difficult for automatic design processes. In this work, Cedrata was unable to reach as good performances as the manual or reference designs on some missions. Based on the Marker Aggregation mission results, we could presume that it is only a matter of budget, and that Cedrata should provide better results as soon as we allocate enough budget. However, the results in the other two missions do not support this hypothesis, since we cannot observe any significant improvement of performance over the budget. An important reason, why higher budgets still may lead to improved control software is the size of the search space, which is larger in Cedrata than it is for example in Maple. Furthermore, due to the flexible structure of the trees, the search space contains a lot of control software with superfluous modules. As the optimization algorithm cannot distinguish between necessary and superfluous modules, parts of the budget will be spent on trying to tune these superfluous modules, even though they have no influence on the performance. Additionally, communication-based strategies seem to be difficult to automatically design. This could be explained by the fact, that most communication-based strategies require at least two modules that are tuned to the same signal value. If Iterated F-race does not randomly sample a solution that already is tuned to the correct signals, the signal-based modules are without worth for the performance, and the design process might converge on solutions that do not make use of communication. In order to counteract this convergence, the optimization algorithm might need to allow for more exploration.

Cedrata introduces both a new set of modules and a new tree structure and the results provided make it impossible to attribute specific observations to one of the changes. For example, in the Stop mission, Cedrata had the possibility to outperform Maple, as evidenced by the manual and reference designs; however Cedrata led to lower performing results than Maple. The problem may reside in the set of modules, the flexibility of the tree structure, or both of them. Without further experiments, it is difficult to attribute the results to one of the proposed causes. Another observation drawn from the results is that Cedrata includes more superfluous modules in its architecture than Maple. This leads to the hypothesis that the number of such modules is related to the freedom given on the control structure. Further experiments could try to verify this hypothesis, by either investigating architectures of different freedom or by actively pruning unused modules during the design process. More generally, Iterated F-Race, the optimization algorithm used by Cedrata, seems to be unable to efficiently explore

the control software space. There are different parameters, such as the budget or the number of iterations, that could influence the strategies discovered by the design process. Further experiments into these parameters and their influence might provide better insights on how control software in the form of behavior trees can effectively be designed. Another idea would be to assess the use of other optimization algorithms, like or simulated annealing [14] or novelty search [19] that is a divergent algorithm that promotes exploration.

Acknowledgements. JK and MB acknowledge support by the FNRS. 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) and from Belgium's Wallonia-Brussels Federation through the ARC Advanced Project GbO–Guaranteed by Optimization.

References

1. Birattari, M., et al.: Automatic off-line design of robot swarms: a manifesto. Front. Robot. AI **6**, 59 (2019). https://doi.org/10.3389/frobt.2019.00059
2. Birattari, M., Ligot, A., Hasselmann, K.: Disentangling automatic and semi-automatic approaches to the optimization-based design of control software for robot swarms. Nature Mach. Intell. **2**(9), 494–499 (2020). https://doi.org/10.1038/s42256-020-0215-0
3. Dorigo, M., Birattari, M.: Swarm intelligence. Scholarpedia **2**(9), 1462 (2007). https://doi.org/10.4249/scholarpedia.1462
4. Dorigo, M., Birattari, M., Brambilla, M.: Swarm robotics. Scholarpedia **9**(1), 1463 (2014). https://doi.org/10.4249/scholarpedia.1463
5. 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
6. 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
7. Garzón Ramos, D., Birattari, M.: Automatic design of collective behaviors for robots that can display and perceive colors **10**(13), 4654 (2020). https://doi.org/10.3390/app10134654
8. Hasselmann, K., Birattari, M.: Modular automatic design of collective behaviors for robots endowed with local communication capabilities. PeerJ Comput. Sci. **6**, e291 (2020). https://doi.org/10.7717/peerj-cs.291
9. Hasselmann, K., et al.: Reference models for AutoMoDe. Technical report, TR/IRIDIA/2018-002, IRIDIA, Université libre de Bruxelles, Belgium (2018)
10. Hasselmann, K., Robert, F., Birattari, M.: Automatic design of communication-based behaviors for robot swarms. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Reina, A., Trianni, V. (eds.) ANTS 2018. LNCS, vol. 11172, pp. 16–29. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00533-7_2
11. Isla, D.: Handling complexity in the Halo 2 AI. In: Game Developers Conference. vol. 12 (2005)
12. Jones, S., Studley, M., Hauert, S., Winfield, A.: Evolving behaviour trees for swarm robotics. In: Groß, R. (ed.) Distributed Autonomous Robotic Systems. SPAR, vol. 6, pp. 487–501. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-73008-0_34

13. Jones, S., Winfield, A., Hauert, S., Studley, M.: Onboard evolution of understandable swarm behaviors. Adv. Intell. Syst. **1**(6), 1900031 (2019). https://doi.org/10.1002/aisy.201900031
14. 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
15. Kuckling, J., Ligot, A., Bozhinoski, D., Birattari, M.: Behavior trees as a control architecture in the automatic modular design of robot swarms. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Reina, A., Trianni, V. (eds.) ANTS 2018. LNCS, vol. 11172, pp. 30–43. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00533-7_3
16. Kuckling, J., van Pelt, V., Birattari, M.: Automatic modular design of behavior trees with communication capabilities: supplementary material. http://iridia.ulb.ac.be/supp/IridiaSupp2020-011/ (2020)
17. Kuckling, J., Stützle, T., Birattari, M.: Iterative improvement in the automatic modular design of robot swarms. PeerJ Comput. Sci. **6**, e322 (2020). https://doi.org/10.7717/peerj-cs.322
18. Kuckling, J., Ubeda Arriaza, K., Birattari, M.: Simulated annealing as an optimization algorithm in the automatic modular design of robot swarms. In: Beuls, K., (eds.) Proceedings of the Reference AI & ML Conference for Belgium, Netherlands & Luxemburg, BNAIC/BENELEARN 2019. CEUR Workshop Proceedings, vol. 2491, CEUR-WS.org, Aachen, Germany (2019)
19. Lehman, J., Stanley, K.O.: Abandoning objectives: evolution through the search for novelty alone. Evol. Comput. **19**(2), 189–223 (2011). https://doi.org/10.1162/EVCO_a_00025
20. 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
21. Ligot, A., Kuckling, J., Bozhinoski, D., Birattari, M.: Automatic modular design of robot swarms using behavior trees as a control architecture. PeerJ Comput. Sci. **6**, e314 (2020). https://doi.org/10.7717/peerj-cs.314
22. 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
23. Marzinotto, A., Colledanchise, M., Smith, C., Ögren, P.: Towards a unified behavior trees framework for robot control. In: IEEE International Conference on Robotics and Automation, ICRA, pp. 5420–5427. IEEE, Piscataway, NJ, USA (2014). https://doi.org/10.1109/ICRA.2014.6907656
24. Neupane, A., Goodrich, M.: Learning swarm behaviors using grammatical evolution and behavior trees. In: Kraus, S. (ed.) Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19), pp. 513–520. IJCAI (2019). https://doi.org/10.24963/ijcai.2019/73
25. Pinciroli, C., et al.: ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intell. **6**(4), 271–295 (2012). https://doi.org/10.1007/s11721-012-0072-5
26. Ögren, P.: Increasing modularity of UAV control systems using computer game behavior trees. In: Thienel, J., et al. (eds.) AIAA guidance, navigation, and control conference 2012, pp. 358–393. AIAA Meeting Papers (2012). https://doi.org/10.2514/6.2012-4458