Automatic Design of Robot Swarms under Concurrent Design Criteria: a Study Based on Iterated F-race

David Garzón Ramos*, Federico Pagnozzi, Thomas Stützle, and Mauro Birattari*

IRIDIA, Université libre de Bruxelles (ULB), Belgium

Abstract

Automatic design is an appealing approach to realizing robot swarms. In this approach, a designer specifies a mission that the swarm must perform, and an optimization algorithm searches for the control software that enables the robots to perform the given mission. Traditionally, research in automatic design has focused on missions specified by a single design criterion, adopting methods based on single-objective optimization algorithms. In this study, we investigate whether existing methods can be adapted to address missions specified by concurrent design criteria. We focus on the bi-criteria case. We conduct experiments with a swarm of e-puck robots that must perform sequences of two missions: each mission in the sequence is an independent design criterion that the automatic method must handle during the optimization process. We consider modular and neuroevolutionary methods that aggregate concurrent criteria via the weighted sum, hypervolume, or l^2-norm. We compare their performance with that of `Mandarina`, an original automatic modular design method. `Mandarina` integrates Iterated F-race as an optimization algorithm to conduct the design process without aggregating the design criteria. Results from realistic simulations and demonstrations with physical robots show that the best results are obtained with modular methods and when the design criteria are not aggregated.

Keywords: Swarm robotics, automatic design, bi-criteria design, Iterated F-race, evolutionary robotics.

1 Introduction

A robot swarm [1, 2] is a self-organized group of robots that can collectively perform missions beyond the capabilities of an individual robot. Researchers commonly design robot swarms by trial and error: the designer manually produces and refines the control software of the individual robots until the desired collective behavior emerges [3]. Although the manual approach has served to demonstrate the feasibility of a wide variety of collective behaviors [4, 5], it is costly, time-consuming, and hardly guarantees that the swarm can accomplish a given mission in a sufficiently satisfactory way [6, 7]. The problem is that no generally applicable method exists to tell how to produce control software for individual robots so that a desired collective behavior is obtained [4]. Optimization-based design is an alternative approach to address this design problem [8, 9]. In this approach, the design problem is restated as an optimization problem: an optimization algorithm explores a space of possible instances of control software for the individual robots, and selects the one that maximizes the collective performance of the swarm—according to a mission-specific performance measure.

Common classifications of optimization-based design methods divide them into on-line and off-line methods, and into semi-automatic and (fully) automatic methods [9]. In our research, we focus on the (fully) automatic off-line design of robot swarms [10]. Automatic off-line methods produce the control software of the robots before the swarm is deployed. Typically, the design process is conducted using simulations, and the control software obtained is then ported to the robots. The main characteristic of automatic off-line methods is that they produce control software without requiring human intervention during the design process. Recent advances

*Corresponding authors: david.garzon.ramos@ulb.be, mauro.birattari@ulb.be.

The core ideas were proposed by David Garzón Ramos and Mauro Birattari, and were then refined by the four authors. Original software was implemented by David Garzón Ramos with the assistance of Federico Pagnozzi. The experiments were conceived by David Garzón Ramos, Thomas Stützle and Mauro Birattari, and were performed by David Garzón Ramos. The manuscript was drafted by David Garzón Ramos and revised by Mauro Birattari. All authors read the manuscript and provided comments. The research was directed by Mauro Birattari.

in the automatic off-line design of robot swarms belong mainly to two approaches [8]: (i) neuroevolution [11–13]; and (ii) automatic modular design (AutoMoDe) [14, 15].

In optimization-based design, the performance measure is formally defined as part of the mission specification. The literature provides various examples in which, given a particular mission, the performance measure is formally defined as (i) a single function that measures the degree of success of the robots in the mission at hand [14, 16–20] or (ii) multiple functions that indicate whether the robots attain a set of objectives and satisfy a set of constraints [16–19, 21–32]. In the second case, the functions are ultimately concurrent design criteria to be satisfied by the robot swarm. Researchers in optimization-based design rarely adopt multi-criteria optimization methods to address problems with concurrent design criteria. The typical approach to address these problems is to aggregate the criteria into a single performance measure and then apply single-criterion optimization methods [30]. Previous studies in the automatic off-line design of robot swarms—both with neuroevolution and AutoMoDe—have indirectly addressed missions with concurrent design criteria in this way [16–18]. We believe that the current taxonomy of the possible approaches to optimization-based design [9] should be extended further to include the single/multi-criteria dichotomy. Indeed, on-line and off-line methods, as well as semi-automatic and automatic ones, face the challenge of enabling the design of robot swarms that comply with concurrent requirements specified by a designer.

The notion of *multi-criteria design* builds upon multi-criteria decision making—that is, the problem of selecting an alternative on the basis of a preference relationship [33]. Multi-criteria decision making is relevant to many different domains [34] and is commonly addressed from the perspective of single-objective and multi-objective optimization. For a classical introduction to multi-criteria decision making, see Fishburn [33]; and for a relatively recent survey on evolutionary algorithms for multi-objective optimization, see Emmerich and Deutz [35]. In single-objective approaches, the decision criteria are aggregated (e.g., by means of scalarization) into a single objective function and the space of the alternatives is totally ordered. On the other hand, multi-objective approaches do not aggregate the decision criteria, and as a result, the space of the alternatives is only partially ordered—the notion of optimal solution is replaced by the one of set of non-dominated solutions, the Pareto set. The application of these notions to the automatic design of robot swarms is rare [30].

In this paper, we contribute to the automatic generation of control software for robot swarms by introducing AutoMoDe-`Mandarina`: a modular method for the automatic design of collective behaviors for robot swarms. Given a mission specification with concurrent design criteria and a space of possible instances of control software, `Mandarina` searches for an instance that maximizes the performance of the swarm in the given mission. More precisely, it aims at finding a neutral compromise solution where all design criteria are satisfied to their best. We build `Mandarina` on the basis of `TuttiFrutti` [17], a single-criterion design method developed for the e-puck robot. Like other AutoMoDe methods [14, 16–19, 36, 37], `Mandarina` integrates Iterated F-race—an algorithm originally conceived in the framework of single-criterion optimization. `Mandarina` and `TuttiFrutti` differ only in the configuration of Iterated F-race—other characteristics, such as the control software they can produce and their target robot, are the same. Notably, they both produce probabilistic finite-state machines for e-pucks that interact with each other and with the environment by displaying and reacting to color signals. In the past, Iterated F-race [38–40] was successfully applied to the automatic off-line design of robot swarms in single-criterion design problems [14, 16–20, 36, 37, 41]. With `Mandarina`, we show that Iterated F-race is also suitable for designing robot swarms in problems with concurrent design criteria.

`Mandarina` exploits some originally unintended properties of Iterated F-race to handle more than one design criterion at a time. More precisely, it exploits its non-parametric nature and the fact that it operates on ranks. These properties were originally intended for handling problems with large scale variations and complex distributions of the objective functions. We noticed that they turn out to be appropriate and natural for handling concurrent design criteria in a multi-criteria optimization setup. Thanks to these properties, `Mandarina` allows for an automatic design process that does not require aggregating the design criteria into a single objective function. We limit the current study to demonstrating these properties in a bi-criteria problem, but we expect the methodology to be applicable to design problems with additional criteria. We contend that this is a design process that aligns better with the tenets of the automatic off-line design of robot swarms [10]: (i) an

automatic method can address a whole class of missions without undergoing any modification; and, (ii) once a mission is specified, no human intervention is allowed for in any phase of the design process. On the other hand, manually aggregating criteria unavoidably prevents practicing these tenets. Aggregating criteria requires mission-specific human intervention and the injection of domain knowledge. In the typical case, the designer is required to estimate/measure the range of scores spanned by each design criterion. Under this condition, mission-specific experimentation is necessary to understand the nature of the design criteria and to produce and refine an appropriate objective function. A method that requires aggregating design criteria via prior experimentation or expert knowledge belongs mostly to the semi-automatic approach [9]. Conversely, the fact that `Mandarina` operates without aggregating criteria makes the method a fully automatic one.

2 Related Work

Trianni and López-Ibáñez [30] described how designers often encode concurrent *mission-specific* and *mission-generic* design criteria into single objective functions—see also Doncieux and Mouret [42]. *Mission-specific* criteria are meant to express preferences on the desired outcome of the mission and/or on the behavior of the robots—for example, the number of objects collected in foraging [14, 21, 22] or the time needed by the robots to aggregate [16, 32]. On the other hand, *mission-generic* criteria are independent of the mission and are meant to express preferences on the design process itself—for example, the diversity and/or complexity of the control software that is produced [21, 22, 31] or the financial cost of realizing a certain robot swarm [36]. Many studies [16–18, 21–32, 43–45] indirectly consider the optimization-based design of robot swarms under concurrent design criteria. However, although these studies frame multi-criteria design problems, their focus is not on investigating multi-objective optimization approaches to address them. The formulation of a multi-criteria problem is often only a convenient tool that favors the materialization of a desired collective behavior.

Little research has been devoted to formally studying the advantages and limitations of addressing multi-criteria design with multi-objective optimization methods [30, 46]. Trianni and López-Ibáñez [30] made a significant contribution to these ideas. They used simulations and neuroevolution to produce the control software for robot swarms on two missions: collective motion and an abstraction of the collaborative stick-pulling experiment [47]. In each mission, the performance was measured with respect to two objective functions—i.e., the design criteria. The study compared a weighted sum approach [48] with a multi-objective approach based on the estimation of the Pareto set. In the weighted sum approach, the design criteria were mapped into terms that were subsequently aggregated into a single objective function. The authors explored various combinations of the weights associated with each term. In the multi-objective approach, the design criteria were mapped into terms that were used independently to compute the *hypervolume* [49]—i.e, the size of the design space that is dominated by the solutions obtained. In the two cases, the result of the design process was a set of solutions from which to manually select a desired one with respect to a performance preference.

Trianni and López-Ibáñez concluded that a weighted sum is appropriate when the designer is able to properly set the weights. Indeed, setting properly the weights could be possible in some cases due to the existence of prior knowledge or as a result of a trial-and-error process—for example, by testing various combinations of weights. The authors also concluded that a multi-objective method is more suitable when prior knowledge is not available or when it is not even possible to find a suitable combination of weights. For example, this is the case of criteria that vary in different proportion to each other and result in non-comparable or non-linearly related scales. The literature shows that the common approach to multi-criteria design remains the mapping of mission-specific and mission-generic design preferences into components of a single objective function—i.e., linear scalarization or weighted sum. That is, the common design methods require casting the multi-criteria design problem into a single-objective optimization one. This is the approach adopted in a large share of studies that belong in semi-automatic design and/or neuroevolution [21–30] and, more recently, those that belong in automatic modular design [16–19].

In most studies that focus on neuroevolution [21–23, 25–29], the authors do not describe the steps they

followed to conceive the objective function of the multi-criteria design problem that is considered. An example of this is the recurrent use of functions that combine a mission-specific objective with a second term that penalizes collisions [26–28]. Authors rarely state whether the objective function is (i) the formal specification of the mission in mathematical terms or it is rather (ii) a function engineered on the basis of prior/domain knowledge to guide the optimization process towards a desired solution. In studies focusing on modular design, recent work considered missions that were specified as multi-criteria problems more explicitly by defining a set of sub-missions to be achieved [16–19]. In these missions, the overall performance of the swarm is measured by a weighted sum of its performance in two sub-missions—which are executed either simultaneously [16] or sequentially [17–19]. For example, Hasselmann and Birattari [18] studied the design of a robot swarm that must change its behavior after finding a given marker in its environment. Initially, the robots must keep a steady motion; after a robot finds the marker, all robots must stop in place. In this mission, the two sub-missions must be performed sequentially and the overall performance of the swarm is measured by a weighted sum of its performance on the two.

In recent years, the family of modular methods AutoMoDe have been used to investigate various aspects of the automatic design of robot swarms [15]: optimization algorithms [14, 16, 50], software architectures [19, 37], robot capabilities [17–19, 36, 51], the robustness of the methods to the reality gap [16, 41, 52], and the efficiency of the design process itself [53]. Iterated F-race is the main optimization algorithm adopted in these studies. In this paper, we use Iterated F-race to automatically design robot swarms without aggregating concurrent design criteria into a single performance measure. In our study, we extend the experimental setup devised by Trianni and López-Ibáñez: (i) we compare various design approaches—`Mandarina`, and methods that rely on aggregating design criteria via a weighted sum, hypervolume, or l^2-norm,; (ii) we evaluate the design methods on a larger set of missions, whose specifications define varied design criteria; (iii) we consider design methods based on the modular and the neuroevolutionary approaches; (iv) we provide demonstrations with physical robots.

3 AutoMoDe-`Mandarina`

`Mandarina` generates control software by fine-tuning and assembling predefined software modules into probabilistic finite-state machines. In the following, we describe the robot platform for which `Mandarina` designs control software, the software modules it can combine, and how it integrates Iterated F-race to address concurrent design criteria.

3.1 Robot platform

`Mandarina` produces control software for an extended version of the e-puck robot [54, 55]—see Figure 1. The e-puck is a two-wheel differential-drive robot largely used in swarm robotics research [56]. In `Mandarina`, we consider a version of the e-puck whose functional capabilities are formally defined by the reference model RM 3 [57]. A reference model [14] is a formal specification of the robot platform for which a design method can produce control software. In practice, the reference model defines the inputs and outputs of the control architecture and associates them with the hardware of the robot. The reference model RM 3 was originally introduced with `TuttiFrutti` [17], and we adopted it in `Mandarina` without any modification.

RM 3 describes an e-puck endowed with: 8 proximity sensors ($prox_i$) that can detect nearby obstacles; 3 ground sensors (gnd_j) that differentiate between gray, black, and white floor; a range-and-bearing board [58] that estimates the number of neighboring peers (n) and their relative aggregate position (V_n); an omnidirectional vision turret that can perceive color signals emitted by neighboring objects and/or robots (cam_c), and estimates their relative aggregate direction (V_c); RGB LEDs to emit color signals; and left and right wheels (v_k), whose velocity can be set independently. Table 1 defines the possible input and output values of the reference model RM 3. For a further description of the reference model RM 3, see [17, 57].

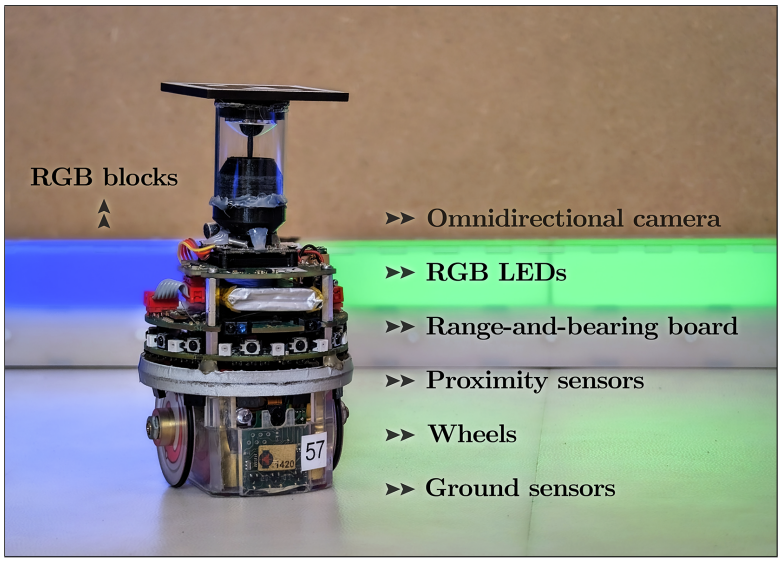

Figure 1: The e-puck robot. The picture shows an e-puck robot and the set of sensors and actuators we consider in our experiments. The picture also shows the modular RGB blocks we use to build the experimental arena for the robots.

Table 1: Reference model RM 3 [17, 57]. Robots can perceive: red (R), green (G), blue (B), cyan (C), magenta (M), and yellow (Y). Robots can display no color ($\varnothing$), cyan (C), magenta (M), and yellow (Y).

Input	**Value**	**Description**
$prox_{i\in\{1,\ldots,8\}}$	$[0,1]$	reading of proximity sensor i
$gnd_{j\in\{1,\ldots,3\}}$	$\{black, gray, white\}$	reading of ground sensor j
n	$\{0,\ldots,20\}$	number of neighboring robots detected
V_n	$([0.5,20];[0,2]\,\pi\,\text{rad})$	their relative aggregate position
$cam_{c\in\{R,G,B,C,M,Y\}}$	$\{yes, no\}$	colors perceived
$V_{c\in\{R,G,B,C,M,Y\}}$	$(1.0;[0,2]\,\pi\,\text{rad})$	their relative aggregate direction
Output	**Value**	**Description**
$v_{k\in\{l,r\}}$	$[-0.12,0.12]\,\text{m/s}$	target linear wheel velocity
$LEDs$	$\{\varnothing, C, M, Y\}$	color displayed by the LEDs

Period of the control cycle: 0.1 s.

3.2 Set of pre-defined software modules

`Mandarina` operates on a set of 13 parametric software modules that were originally conceived for `TuttiFrutti` [17]. These modules allow a robot to interact with its peers and with objects in the environment by perceiving and displaying color signals—according to the reference model RM 3. The set of modules comprises 6 low-level behaviors—the actions that a robot can execute, and 7 transition conditions—the events that trigger the transition between low-level behaviors. `Mandarina` combines low-level behaviors and transition conditions to generate the control software of the robots. Table 2 lists the set of modules on which `Mandarina` operates.

The six low-level behaviors are EXPLORATION, STOP, ATTRACTION, REPULSION, COLOR-DETECTION, and COLOR-ELUSION. In EXPLORATION, the robot moves according to a ballistic motion and avoids obstacles by rotating on itself for a number of control cycles that is defined by the parameter τ. STOP sets the robot to a standstill behavior. ATTRACTION and REPULSION are physics-based behaviors that drive the robot according to artificial forces that originate at the position of neighboring peers. In both cases, the magnitude of the force is determined by the parameter α. If no other robot is perceived, the robot moves ballistically. COLOR-DETECTION and COLOR-ELUSION steadily drive the robot respectivelly toward or away from peers and objects that display a color $\delta \in \{R, G, B, C, M, Y\}$. If no color is perceived—that is, if $\delta = \varnothing$—the robot moves ballistically. EXPLORATION, ATTRACTION, REPULSION, COLOR-DETECTION, and COLOR-ELUSION embed a physics-based [59] obstacle avoidance behavior. In all low-level behaviors, the robot can set its LEDs to display $\gamma \in \{\varnothing, C, M, Y\}$.

Table 2: Set of `Mandarina`'s modules. `Mandarina` operates on the same modules that were originally conceived for `TuttiFrutti` [17]. The modules are defined on the basis of reference model RM 3, see Table 1.

Low-level behavior *	**Parameters**	**Description**
EXPLORATION	$\{\tau, \gamma\}$	movement by random walk
STOP	$\{\gamma\}$	standstill state
ATTRACTION	$\{\alpha, \gamma\}$	physics-based attraction to neighboring robots
REPULSION	$\{\alpha, \gamma\}$	physics-based repulsion from neighboring robots
COLOR-FOLLOWING	$\{\delta, \gamma\}$	steady movement towards robots/objects of color δ
COLOR-ELUSION	$\{\delta, \gamma\}$	steady movement away from robots/objects of color δ
Transition condition	**Parameters**	**Description**
BLACK-FLOOR	$\{\beta\}$	black floor beneath the robot
GRAY-FLOOR	$\{\beta\}$	gray floor beneath the robot
WHITE-FLOOR	$\{\beta\}$	white floor beneath the robot
NEIGHBOR-COUNT	$\{\xi, \eta\}$	number of neighboring robots greater than ξ
INVERTED-NEIGHBOR-COUNT	$\{\xi, \eta\}$	number of neighboring robots less than ξ
FIXED-PROBABILITY	$\{\beta\}$	transition with a fixed probability
COLOR-DETECTION	$\{\delta, \beta\}$	robots/objects of color δ perceived

* All low-level behaviors display a color $\gamma \in \{\varnothing, C, M, Y\}$ alongside the action described.

The seven transition conditions are BLACK-FLOOR, GRAY-FLOOR, WHITE-FLOOR, NEIGHBOR-COUNT, INVERTED-NEIGHBOR-COUNT, FIXED-PROBABILITY, and COLOR-DETECTION. In BLACK-FLOOR, GRAY-FLOOR, WHITE-FLOOR, the robot transitions between low-level behaviors with a probability β if it steps into regions with black, gray, and white floor, respectively. NEIGHBOR-COUNT and INVERTED-NEIGHBOR-COUNT are transitions that are respectively triggered when the robot perceives a number of neighboring peers that is greater or less than η. FIXED-PROBABILITY is triggered with probability β. In COLOR-DETECTION, the robot switches between low-level behaviors with probability β if it perceives another robot or an object that displays a color $\delta \in \{R, G, B, C, M, Y\}$.

The parameters of the software modules are automatically tuned during the design process. For a detailed description of the implementation of the software modules, we refer the reader to the original work in which `TuttiFrutti` was introduced [17]. Figure 2 shows a simplified illustration of `Mandarina`'s software modules assembled into a probabilistic finite-state machine and the resulting behavior on an e-puck.

3.3 Bi-criteria design process

In `Mandarina`, the bi-criteria design process is cast into an optimization problem that is addressed with Iterated F-race [38, 39]. We use the implementation of Iterated F-race provided by the `irace` package [60] version 2.2. Iterated F-race maximizes the performance of the robot swarm with respect to a set of mission-specific metrics: the objective functions associated to the design criteria. Starting from the specifications of a mission, Iterated F-race selects software modules, fine-tunes its parameters, and combines them into probabilistic finite-state machines—the control software of the robots. The finite-state machines that `Mandarina` produces can have up to four states—i.e., low-level behaviors, and up to four outgoing transitions for each state—i.e., transition conditions. A transition condition will always originate and end in different states.

During the design process, Iterated F-race searches the design space looking for probabilistic finite-state machines that perform well in the mission at hand. Each finite-state machine is assessed with simulations executed in ARGoS3 [61], a multi-robot simulator specialized in robot swarms. We use ARGoS3 version beta 48, together with the argos3-epuck plugin [55]. Before starting the optimization process, Iterated F-race is given a maximum number of simulations to find a candidate solution (i.e., a finite-state machine to address the mission). The maximum number of simulations determines the duration of the optimization process. When Iterated F-race exhausts the maximum number of simulations, the optimization process ends and `Mandarina` returns the best candidate solution found so far. The control software produced by `Mandarina` is then deployed

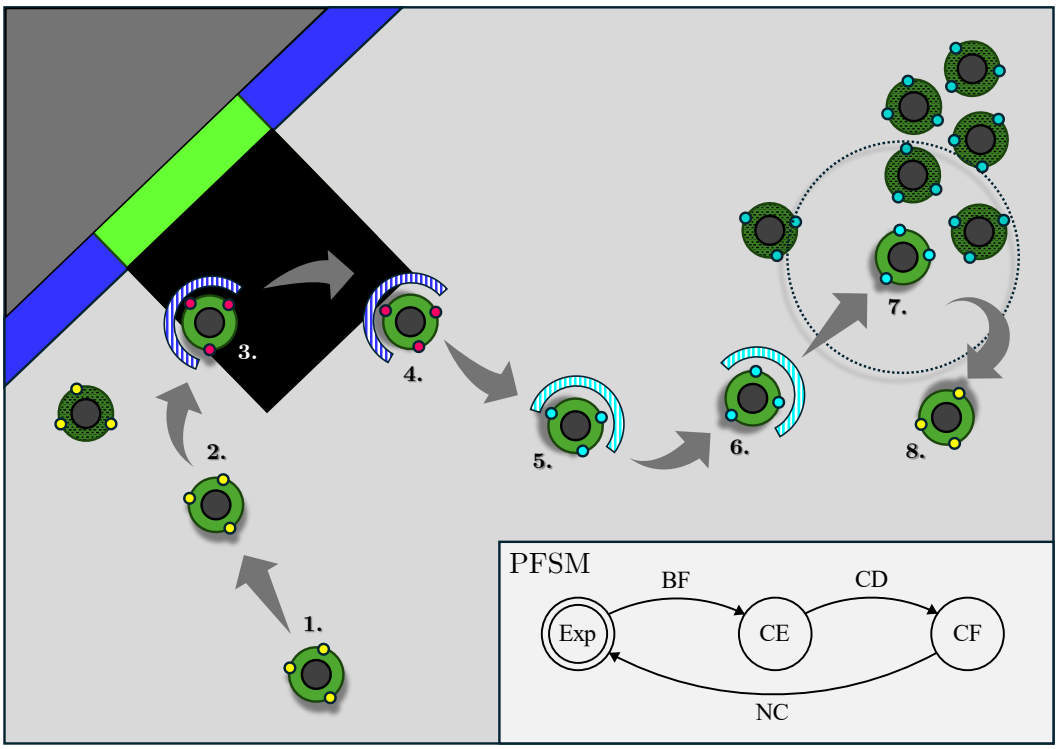

Figure 2: Simplified illustration of `Mandarina`'s software modules assembled into a probabilistic finite-state machine (PFSM) and the resulting behavior on an e-puck. 1.) The finite-state machine starts with the behavior EXPLORATION (Exp), which in this case sets the e-puck's LEDs to display yellow while the robot moves randomly in the arena. 2.) The e-puck detects a robot on its left and turns right to avoid a collision. 3.) When the e-puck detects a region with a black floor, the transition BLACK-FLOOR (BF) is activated, and the e-puck switches to the behavior COLOR-ELUSION (CE). 4.) The e-puck executes COLOR-ELUSION (CE), driving the robot away from the blue walls and changing its LEDs to display magenta. 5.) The e-puck detects that other robots are displaying cyan with their LEDs, activating the transition COLOR-DETECTION (CD) and switching to the behavior COLOR-FOLLOWING (CF). 6.) The e-puck executes COLOR-FOLLOWING (CF), moving toward other robots displaying cyan and changing its own LEDs to cyan as well. 7.) The e-puck detects two neighboring robots within its perception range, activating the transition NEIGHBOR-COUNT (NC) and switching back to EXPLORATION (CE). 8.) The finite-state machine continues to operate until the mission ends. For more information on the modules and their parameters, see Table 2.

to the robots without further modification.

`Mandarina` operates with a design process similar to that conceived by Francesca *et al.* [16] for `Chocolate`, which is also used in `TuttiFrutti`. The main difference is that, as mentioned in the introduction, `Mandarina` uses Iterated F-race to address missions whose specifications define multiple objective functions to be optimized. In this paper, we focus on the bi-objective case: missions are characterized by two objective functions. To achieve this, we did not need to modify Iterated F-race. We simply set it up in a way that differs from previous implementations of AutoMoDe. In `Chocolate` and `TuttiFrutti`, as in the other existing implementations of AutoMoDe, Iterated F-race evaluates candidate solutions with respect to the single objective function that characterizes the mission at hand. In `Mandarina`, we set up Iterated F-race so as to evaluate each candidate solution on the two objective functions defined in the mission specification.

Figure 3 illustrates the operation of Iterated F-race, the original optimization algorithm as used in `Chocolate` and `TuttiFrutti`, and the optimization algorithm as used in `Mandarina`. Iterated F-race operates in three main steps [40]: (i) sampling candidate instances of control software according to a particular distribution; (ii) selecting the best instance(s) from the newly sampled ones by racing; and (iii) updating and refining the sampling distribution in order to bias the sampling towards the best instance(s) found. In each race, Iterated F-race iteratively evaluates candidate finite-state machines on a number of *problem instances*. Each problem instance is a specific realization of the mission at hand—for example, a specific realization of initial conditions such as position/heading of the robots. While in `Chocolate` and `TuttiFrutti`, which address single-objective problems, the evaluation of a candidate solution on a problem instance produces a single number (the score in the mission); in `Mandarina`, it produces two numbers: the corresponding scores of the two objective functions.

Iterated F-race uses statistical tests—i.e., Friedman tests [62]—to discard candidate solutions that significantly perform worse than at least one other candidate solution. The candidate solutions are selected with respect to the average rank of their observed performance and not with respect to the numerical values. In `Mandarina`, Iterated F-race conducts the statistical tests on the two scores that are returned, and discards candidate solutions that significantly perform worse than at least one other candidate solution in the two of them. Eliminating candidate instances of control software under concurrent design criteria is only possible because Iterated F-race operates on the ranks, and not on the score obtained from the objective functions. It is not always possible to directly compare the score of an instance of control software across two objective functions. However, it is indeed possible to compare the average rank of its observed performance across them.

A popular approach to tackle multi-objective optimization problems is to first identify a set of solutions that represent various performance compromises across the objectives—an approximation of the Pareto set. From this set, an a posteriori decision-making process is applied to select a preferred solution with respect to the observed performance—as in Trianni and Lopez-Ibañez [30]. Conversely, `Mandarina` returns a single solution: the one that statistically does not perform worst than any other solution in the two design criteria. In other words, it returns a solution that consistently proved to be a good performing neutral compromise among the criteria considered.

From a practical point of view, the modifications we made to the design process originally conceived for AutoMoDe methods are minor. Indeed, we made only very few technical adjustments in the way in which Iterated F-race is typically used to evaluate each candidate control software with respect to two objective functions. However, the conceptual implications of making these modifications in `Mandarina` are important. Iterated F-race can address bi-criteria design problems if the objective functions to be optimized concurrently are presented as independent problem instances to be considered during the optimization process. This property was originally unintended in the conception of Iterated F-race. In this paper, we do not explore the extent to which this property generalizes and/or whether Iterated F-race could address design problems that consider more than two objective functions. However, we find reasonable to think that Iterated F-race could turn viable to tackle the design of robot swarms with respect to more than two criteria—provided that they can be presented as independent problem instances as well.

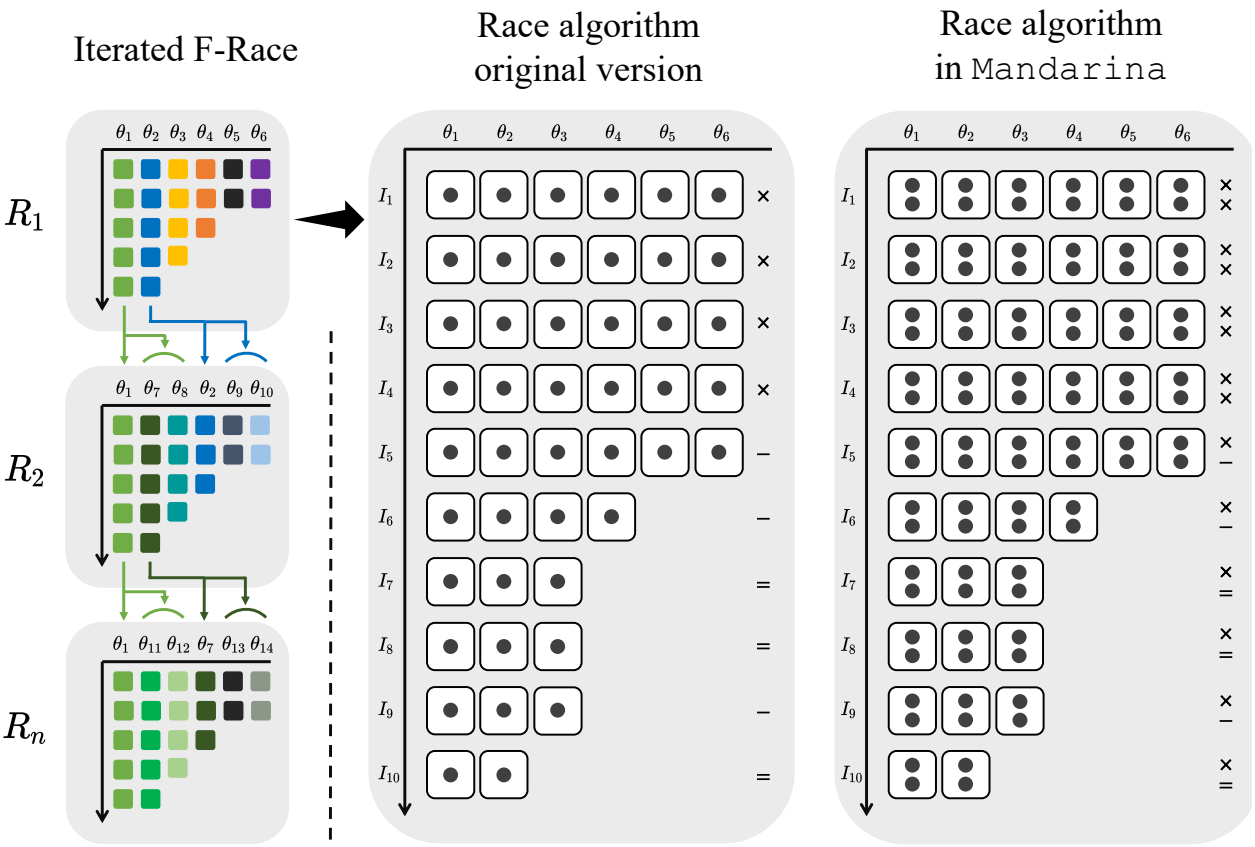

Figure 3: Illustration of Iterated F-race, the original race algorithm, and of how it is used in `Mandarina`. Iterated F-race conducts iterative races (R_i) in which candidate solutions (θ_j)—i.e., instances of the control software being developed—are evaluated with respect to a number of problem instances (I_k). Each box is the evaluation of one candidate solution on one problem instance. Dots (•) in a box represent the number of objective functions evaluated. In the first race (R_1), Iterated F-race uniformly samples the parameter space to generate candidate solutions ($\theta_{\{1,..,6\}}$). In the subsequent races ($R_{\{2,..,n\}}$), Iterated F-race refines and updates the sampling distribution to bias the sampling towards the best instance(s) found so far. For example, in R_2, $\theta_{\{7,..,8\}}$ and $\theta_{\{9,..,10\}}$ result from biased sampling towards θ_1 and θ_2, respectively. The refinement of the sampling process is illustrated by the color gradient of the boxes. The original race algorithm evaluates one single objective function. In `Mandarina`, Iterated F-race evaluates two objective functions. In a race, candidate solutions are discarded after conducting statistical tests: '×' indicates that no statistical test is performed; '−' indicates that the test discarded at least one candidate solution; and '=' indicates that the test did not discard any candidate solution. Statistical tests are conducted only after sufficient statistical evidence is gathered on the performance of the candidate solutions in an initial set of problem instances ($I_{1,..,5}$). `Mandarina` requires evaluating both objective functions on an instance I_k before conducting statistical tests to discard configurations. As no configuration can be discarded based on just one of the two objective functions, no statistical test is needed after evaluating only the first objective function of instance I_k.

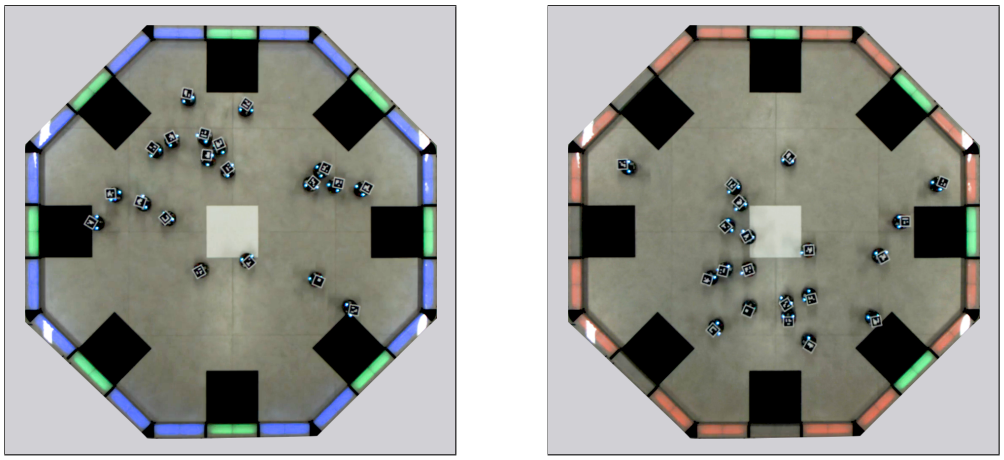

Figure 4: The experimental arena. The picture shows examples of the two possible states of the arena. On the left, the RGB blocks of the walls display blue and all RGB blocks adjacent to a black patch are switched on, displaying green. On the right, the RGB blocks of the walls display red and some RGB blocks adjacent to a black patch have switched off, displaying no color. The robots are randomly positioned.

4 Experimental setup

We assess `Mandarina` on bi-criteria missions that are specified as bi-objective optimization problems. In the absence of well-defined benchmarks, we devise a framework in which we combine instances of aggregation, foraging, and coverage to create missions. The missions included in this framework draw inspiration from missions used in previous automatic design studies [14, 16, 17, 30, 51, 52, 63]. Alongside `Mandarina`, we also conduct experiments with baseline modular and neuroevolutionary design methods.

4.1 Mission framework

We experiment with a swarm of twenty e-puck robots that must perform missions composed of two sequential parts. Each part is a sub-mission to be accomplished and is evaluated by an independent objective function, a design criterion. The mission framework comprises a set of fifteen missions, which result from combining the six sub-missions into sets of two: $\binom{6}{2} = 15$. In these missions, the swarm must perform one sub-mission for a given amount of time, then switch its behavior to perform the second one for the same amount of time.

The robots operate in an octagonal arena of $2.75\,\mathrm{m}^2$ surrounded by RGB blocks [64]—see Figure 4. The robots are randomly positioned at the beginning of each experimental run. The RGB blocks are arranged in walls and each of them can possibly display a different color. The floor of the arena is gray with nine square patches, each measuring 25 cm on each side. One of the patches is white, and the other eight are black. In every mission, RGB blocks adjacent to black patches initially turn green, and afterward, they will randomly switch off with uniform probability. The remaining RGB blocks turn red or blue to inform the robots about the sub-mission to be executed. Typically, in automatic design studies, the arena is modified on a per-mission basis to allow for varying mission specifications [14, 16, 17, 51, 52, 63]. Here, we specify a diverse set of bi-criteria missions (and their sub-missions) in a single experimental arena by changing the color of the RGB blocks at run-time.

4.1.1 Sub-missions

We consider a set of six sub-missions ($\mathrm{S}\{1, \cdots, 6\}$), which can be performed by e-pucks that comply with the reference model RM 3. These sub-missions are a sample of the class of missions that can be addressed with `Mandarina` and other methods that adopt the same reference model. Each sub-mission is specified by a description of a task to be executed and a corresponding objective function.

Sub-mission 1 (S1): the robots must occupy the black patches whose adjacent RGB blocks display green. The swarm is given 1 point for every 100 cumulative timesteps that the robots spend on each suitable patch.

For example, a single robot in a patch will be given one point after 100 timesteps, but 10 robots in a patch will be given 1 point after 10 timesteps. The score of the swarm is the number of points it obtains in the allotted time:

$$f_{\mathrm{S1}} = \sum_{t=1}^{T'} \sum_{i=1}^{H} I_i(t), \tag{1}$$

which must be maximized. H is the number of patches and T' the time available to the robots to perform the sub-mission. The indicator $I_i(t)$ is defined as:

$$I_i(t) = \begin{cases} 1, \text{ if at time } t \text{ the robots accumulate 100 timesteps in the patch } i; \\ 0, \text{ otherwise.} \end{cases}$$

Sub-mission S1 is inspired by aggregation missions in which the robots must gather at an indicated place [14, 16, 51].

Sub-mission 2 (S2): the robots must iteratively travel from any black patch to the white one. The swarm is given 1 point every time a robot completes a trip. The score of the swarm is the number of points it obtains in the allotted time:

$$f_{\mathrm{S2}} = I_{T'}\,, \tag{2}$$

which must be maximized. $I_{T'}$ is the number of trips executed in the time T' available to the robots to perform the sub-mission. Sub-mission S2 is inspired by foraging missions in which the robots must travel between two locations: a food source and a nest [14, 17].

Sub-mission 3 (S3): the robots must occupy the black patches adjacent to RGB blocks that are switched off. The swarm is given one point if at least two robots spend 50 timesteps in the corresponding black patch. The count of timesteps starts as soon as the two robots step into the patch, and it will continue as long as they both remain on it. The count is not affected if more than two robots occupy the patch. The score of the swarm is the number of points it obtains in the allotted time:

$$f_{\mathrm{S3}} = \sum_{t=1}^{T'} \sum_{i=1}^{H} I_i(t), \tag{3}$$

which must be maximized. H represents the number of patches and T' the time available to the robots to perform the sub-mission. The indicator $I_i(t)$ is defined as:

$$I_i(t) = \begin{cases} 1, \text{ if at time } t \text{ two robots accumulate 50 timesteps in the patch } i;; \\ 0, \text{ otherwise.} \end{cases}$$

Sub-mission S3 is inspired by strictly cooperative missions in which the robots must jointly perform a single task [30, 47].

Sub-mission 4 (S4): the robots must iteratively enter and leave the white patch. The swarm is awarded 1 point every time a robot performs these two actions. The score of the swarm is the number of points it obtains in the allotted time:

$$f_{\mathrm{S4}} = I_{T'}\,, \tag{4}$$

which must be maximized. $I_{T'}$ is the number of times a robot entered and left the white patch in the time T' that is available to the robots to perform the sub-mission. Also sub-mission S4 is inspired by foraging missions, as sub-mission S2, but here, the robots start and end at a single location.

Sub-mission 5 (S5): the robots must disperse and cover the arena. We consider the coverage to be the most effective when the minimum distance between any two pair of robots is maximized. The score of the

swarm is the cumulative sum of the minimum inter-robot distance, over time:

$$f_{\mathrm{S5}} = \sum_{t=1}^{T'} \min\left(d_{ij}(t)\right), \tag{5}$$

which must be maximized. Here, d_{ij} is the minimum distance between any pair of robots (i, j) at time t, and T' is the time available to the robots to perform the sub-mission. Sub-mission S5 is inspired by dispersion and coverage missions in which the robots must maintain a fixed inter-robot distance to achieve a specific spatial distribution [16, 52, 63].

Sub-mission 6 (S6): the robots must remain within a 25 cm distance from the walls of the arena, without entering the black patches. The score of the swarm is the aggregate time that the robots spend in the suitable areas:

$$f_{\mathrm{S6}} = \sum_{t=1}^{T'} \sum_{i=1}^{N} I_i(t), \tag{6}$$

which must be maximized. N is the number of robots and T' is the time available to the robots to perform the sub-mission. The indicator $I_i(t)$ is defined as:

$$I_i(t) = \begin{cases} 1, \text{ if the robot } i \text{ is in gray floor and in a 25 cm distance from a wall at time } t; \\ 0, \text{ otherwise.} \end{cases}$$

Sub-mission S6 is inspired by missions in which the robots must display a specific spatial distribution, like sub-mission S5 [16, 51, 63]. However, the robots here must maintain a specific distance from an element in their environment, irrespective of the distance to their peers.

We selected the sub-missions S$\{1, \cdots, 6\}$ because they pose specific challenges to a multi-objective optimization process. First, the performance of candidate solutions is stochastic and depends on the robots' initial positions (e.g., in S5 and S6) and on the way the RGB blocks turn off (e.g., in S1 and S3). Second, estimating realistic bounds for the objective functions is not straightforward without conducting preliminary experiments (e.g., in S2 and S4). Indeed, it is challenging to anticipate to what degree the interactions between the robots will hinder a candidate solution from reaching a theoretical upper performance bound. Third, the objective functions can take discrete or continuous variables (e.g., in S4 and S5), and their range can span over different orders of magnitude (e.g., in S3 and S6). Finally, the objective functions can represent conflicting goals (e.g., in S1 and S3).

4.1.2 Specification of bi-criteria missions

We define the set M of bi-criteria missions ($m_{\mathrm{S}p.\mathrm{S}q}$) by pairing sub-missions (Sp, Sq) in fifteen combinations—see Table 3. In all cases, the robots must execute the two sub-missions, one after the other. The time T available to the robots to execute a mission is 120 s. The time T' available to execute each sub-mission is 60 s. Accordingly, the swarm's performance in a mission is assessed for the initial 60 s with regard to one sub-mission and for the remaining 60 s with regard to the other. The two scores are returned after each experimental run.

We expect the swarm to be able to perform a mission $m_{\mathrm{S}p.\mathrm{S}q}$ regardless of the order of Sp and Sq. Therefore, in our experiments, the order in which the sub-missions must be executed is randomly defined at the beginning of the evaluation of the mission (i.e., for each problem instance). For example, the execution of $m_{\mathrm{S}p.\mathrm{S}q}$ can result uniformly in the sequence S$p\rightarrow$ Sq or S$q\rightarrow$ Sp. We anticipate that using a fixed predefined order of sub-missions could lead to control software instances specialized for that specific order, making it likely to fail when the order of Sp and Sq changes. On the contrary, our stochastic approach helps minimize the risk of biasing the design process toward a specific order.

In every mission, we use a fixed color coding (blue and red) to characterize the associated sub-missions—see Table 3. At every moment in time, the walls of the arena display blue or red to indicate the sub-mission to be executed. For example, in a sequence S$q\rightarrow$ Sp, the walls will initially display the color blue, indicating that the

Table 3: Set of missions (M). The missions are paired combinations ($m_{Sp.Sq}$) of the six sub-missions (S$\{1, \cdots, 6\}$). In each combination, the colors blue and red characterize the sub-missions Sp (■) and Sq (■). In a mission ($m_{Sp.Sq}$), the sub-missions Sp and Sq must be executed in sequence, but the order of the sequence (Sp ■ ⇄ ■ Sq) is randomly defined in every experimental run. Sp and Sq are executed during an equivalent period of time. The execution of a mission $m_{Sp.Sq}$ returns the score of the swarm with respect to Sp and Sq, regardless the order of the sequence.

No.	Mission	Combination of sub-missions
1	$m_{S1.S2}$	S1 ■ ⇄ ■ S2
2	$m_{S1.S3}$	S1 ■ ⇄ ■ S3
3	$m_{S1.S4}$	S1 ■ ⇄ ■ S4
4	$m_{S1.S5}$	S1 ■ ⇄ ■ S5
5	$m_{S1.S6}$	S1 ■ ⇄ ■ S6
6	$m_{S2.S3}$	S2 ■ ⇄ ■ S3
7	$m_{S2.S4}$	S2 ■ ⇄ ■ S4
8	$m_{S2.S5}$	S2 ■ ⇄ ■ S5
9	$m_{S2.S6}$	S2 ■ ⇄ ■ S6
10	$m_{S3.S4}$	S3 ■ ⇄ ■ S4
11	$m_{S3.S5}$	S3 ■ ⇄ ■ S5
12	$m_{S3.S6}$	S3 ■ ⇄ ■ S6
13	$m_{S4.S5}$	S4 ■ ⇄ ■ S5
14	$m_{S4.S6}$	S4 ■ ⇄ ■ S6
15	$m_{S5.S6}$	S5 ■ ⇄ ■ S6

swarm must execute the sub-mission Sq. After 60 seconds, the walls switch to red, indicating the end of Sq and the start of the sub-mission Sp. The color of the walls is an environmental cue that the robots can perceive. We anticipate that the design methods will design robot swarms that, although not pre-programmed to do so, will rely on the color of the walls to transition between behaviors and accomplish the two sub-missions—as in previous studies [17].

4.2 Baseline methods

We conduct experiments with three automatic design methods other than `Mandarina`: `TuttiFrutti` [17], which belong in the modular approach, and `EvoColor` [17] and `NEAT-Color`, which belong in the neuroevolutionary one. Like `Mandarina`, these three methods produce control software for e-pucks defined by the reference model RM 3, and are specialized in the design of collective behaviors for robots that can perceive and react to color signals. However, unlike `Mandarina`, they do not naturally work with concurrent design criteria. `TuttiFrutti`, `EvoColor`, and `NEAT-Color` are limited to using a single score as an evaluation criterion during the design process. We consider these methods as our baseline because, to address a bi-criteria design problem, they need to aggregate the concurrent design criteria into a single performance measure—the standard approach in the literature, see Section 2.

`TuttiFrutti` is identical to `Mandarina` in all aspects but in the way in which Iterated F-race is configured to conduct the optimization process. In `TuttiFrutti`, Iterated F-race is used with its default configuration [60]. `EvoColor` is a straightforward neuroevolutionary method that produces control software in the form of a fully-connected feed-forward artificial neural network. The input and output nodes are defined according to the inputs and outputs defined by the reference model RM 3–see Table 1. `EvoColor` fine-tunes the synaptic weights of the neural network via an evolutionary optimization process based on elitism and mutation. `NEAT-Color` is a neuroevolutionary method in which the evolutionary process selects both the network topology and the synaptic weights of the artificial neural network. `NEAT-Color` is based on `NEAT`—an algorithm introduced by Stanley and Miikkulainen [65] and tested in the context of the automatic design of robot swarms by Hasselman et al. [20]. We include `NEAT-Color` as a more sophisticated alternative method to `EvoColor`. The input and output nodes in `NEAT-Color` are the same as for `EvoColor`. In `NEAT-Color`, the topology of the network is initialized with a disconnected network. We conduct the design process using `TuttiFrutti` and `EvoColor` as

originally introduced [17]. The parameters for conducting the design process with `NEAT-Color` are the ones defined by Hasselmann *et al.* [20] for `NEAT`.

4.2.1 Aggregating design criteria into a single performance measure

We use `TuttiFrutti`, `EvoColor`, and `NEAT-Color` alongside unary metrics that aggregate the two scores into a single measure. We consider the weighted sum (`WS`), the hypervolume (`HV`), and the l^2-norm (`L2`). These unary metrics provide the single performance value that `TuttiFrutti`, `EvoColor`, and `NEAT-Color` require to conduct the design process.

Weighted sum (`WS`): we scalarize the two objective functions ($f_{\mathrm{S}p}$ and $f_{\mathrm{S}q}$) of a mission ($m_{\mathrm{S}p.\mathrm{S}q}$) through a weighted sum ($f_{m_{\mathrm{S}p.\mathrm{S}q}} = (\alpha).f_{\mathrm{S}p} + (1-\alpha).f_{\mathrm{S}q}$) regulated by a parameter $\alpha \in [0, 1]$. This metric can be used alongside `TuttiFrutti`, `EvoColor`, and `NEAT-Color`, but it requires defining a suitable α value for the mission before the design process begins. Given that we consider the absence of a performance range for $f_{\mathrm{S}p}$ and $f_{\mathrm{S}q}$, it is hardly possible to predefine such suitable α values in our experiments. Therefore, we conduct experiments with a set of five α values in every mission ($\alpha \in \{0.2, 0.4, 0.5, 0.6, 0.8\}$)—likewise Trianni and Lopez Ibáñez [30]. These experiments yield a large set of instances of control software for each baseline method and mission, and a subsequent decision-making process is necessary to select solutions from this set. We conduct this decision-making process manually. After all experiments are completed, we evaluate the instances produced with the five α values and we rank them according to their score. For each method, we identify the best, median, and worst combination of α values for every mission. We then use this α values to assemble three sets of instances of control software for each method: the best (B), median (M), and worst (W) instances produced. It should be noted that this a posteriori manual selection process does not align with the tenets of a fully automatic design process [10]. However, we include it in our study to provide an estimate of the best, median, and worst performance one can expect when using a weighted sum in these missions.

Hypervolume (`HV`): the hypervolume quantifies the area in the objective space that a candidate solution dominates. In our experiments, the sub-missions and their associated objective functions define the objective space for each mission. To calculate the hypervolume, the objective space must be normalized using a reference point, indicating the upper performance bounds of the two objective functions under consideration. However, in our experiments, we cannot pre-establish this normalization due to the absence of a score range for $f_{\mathrm{S}p}$ and $f_{\mathrm{S}q}$. Iterated F-race has a way to address this problem: it dynamically computes these bounds for every problem instance based on the observed scores of all candidate solutions on that specific instance. Therefore, in the case of `TuttiFrutti`, the hypervolume can be computed using the implementation provided by the maintainers of the `irace` software package [60], originally proposed by Fonseca *et al.* [66]. In the case of `EvoColor` and `NEAT-Color`, there is no straightforward way to dynamically compute the performance bounds during the evolutionary optimization process. Consequently, we do not apply the hypervolume to the neuroevolutionary methods.

In our experiments, we use the hypervolume in a way that differs from the typical approach found in the multi-objective optimization literature [67]. The hypervolume is often utilized to identify a set of solutions with different performance trade-offs—i.e., an aproximation of the Pareto front—and then select a preferred solution through an informed decision-making process. This was the approach followed by Trianni and Lopez Ibáñez [30]. Conversely, we use the hypervolume in `TuttiFrutti` to find a single no-preference solution to the design problem. `TuttiFrutti` will look for a solution that, on an individual basis, covers the largest area in the objective space across all tested problem instances. We expect that an instance that maximizes its coverage of the objective space will yield a neutral compromise solution, where the two sub-missions are performed at their best. We consider this application of the hypervolume a fully automatic design method because it does not require prior experimentation, or manual intervention and decision making after the design process.

l^2-norm (`L2`): we compute the Euclidean distance in the objective space between a reference point and the point defined by the two scores of a candidate solution. Like the hypervolume, `TuttiFrutti` utilizes Iterated F-race to dynamically compute performance bounds for each problem instance, normalizing the objective space

Table 4: Design methods under evaluation. We consider three versions of `NEAT-Color`, `EvoColor` and `TuttiFrutti` that adopt the weighted sum. Each of these versions comprises a set of control software instances produced using the best, median, and worst combinations of α values identified for each mission. We also consider two additional versions of `TuttiFrutti`, one that adopts the hypervolume and other that adopts the l^2-norm. The last method method under evaluation is `Mandarina`.

No.	Label	Method	Unary metric	Set of instances	Approach
1	`NC-WS-`W	`NEAT-Color`	weighted sum	worst	neuroevolution
2	`NC-WS-`M	`NEAT-Color`	weighted sum	median	neuroevolution
3	`NC-WS-`B	`NEAT-Color`	weighted sum	best	neuroevolution
4	`EC-WS-`W	`EvoColor`	weighted sum	worst	neuroevolution
5	`EC-WS-`M	`EvoColor`	weighted sum	median	neuroevolution
6	`EC-WS-`B	`EvoColor`	weighted sum	best	neuroevolution
7	`TF-WS-`W	`TuttiFrutti`	weighted sum	worst	AutoMoDe
8	`TF-WS-`M	`TuttiFrutti`	weighted sum	median	AutoMoDe
9	`TF-WS-`B	`TuttiFrutti`	weighted sum	best	AutoMoDe
10	`TF-HV`	`TuttiFrutti`	hypervolume	n.a.	AutoMoDe
11	`TF-L2`	`TuttiFrutti`	l^2-norm	n.a.	AutoMoDe
12	`Mandarina`	`Mandarina`	n.a.	n.a.	AutoMoDe

and establishing an appropriate reference point. Then it aggregates the two scores of the candidate solution using the l^2-norm. In this case, we also do not apply the l^2-norm to `EvoColor` and `NEAT-Color` as they do not provide a straightforward way to dynamically compute the performance bounds.

The l^2-norm is a metric used in no-preference multi-objective optimization methods [68]. These methods aim at finding a single solution that satisfies a decision maker who has no special expectations or preference on performance trade-offs. In essence, all objective functions are treated as equally important, and the decision maker is satisfied with finding a single solution that optimizes them. This reasoning aligns well with our bi-criteria design problem: our goal is to find instances of control software that enable the swarm to perform its mission ($m_{\mathrm{S}p.\mathrm{S}q}$) by doing well in the two sub-missions comprised (Sp, Sq)—no other information is provided to the design process. We expect that `TuttiFrutti` will produce neutral compromise solutions by incorporating the l^2-norm. We consider this application of the l^2-norm also a fully automatic design method.

4.3 Protocol

Table 4 lists the design methods considered in our experiments. We use `Mandarina` and the baseline methods to produce control software for the fifteen missions. We generate 150 instances of control software with `Mandarina`—10 per mission. In the experiments involving the weighted sum, we produce a total of 2250 instances of control software using `NEAT-Color`, `EvoColor`, and `TuttiFrutti`—10 per method, mission, and α value. For each method, we then select sets of the best, median, and worst instances, each comprising 150 instances—10 per mission. We also produce 150 instances of control software using `TuttiFrutti` and the hypervolume, and other 150 using `TuttiFrutti` and the l^2-norm—10 per mission in each case. All design methods are given a budget of 100 000 simulation runs to produce each instance of control software.

The instances of control software produced by a design method are evaluated twice—once for each possible order of the sub-missions. We do this to assess the swarm's capability to perform the mission regardless of the order of the sub-missions. In all cases, the design process and the evaluation of the control software is conducted using the ARGoS3 simulator [61].

We also provide 180 demonstration videos of the behavior of the robots when the control software produced in simulation is ported to physical robots. We present 30 videos for each method among `NC-WS-`B, `EC-WS-`B, `TF-WS-`B, `TF-HV`, `TF-L2`, and `Mandarina`. They demonstrate the behavior of instances of control software for every mission and for each possible order of the sub-missions.

4.3.1 Statistics

We use a Friedman rank sum test [62] to compare the relative performance of the methods across all 15 missions, taking into account the evaluations conducted in each sub-mission. Specifically, we conduct a Friedman two-way analysis of variance with repeated measures. This rank-based non-parametric test uses a block design. In our protocol, the treatment factor is the method under analysis and the blocking factor is the sub-missions. We present the results of the Friedman test with the average rank of the methods along with its 95% confidence interval. The average rank indicates the relative performance of a method with respect to the others across the complete set of experimental results. We selected the Friedman test to compare methods because it is invariant to the magnitude of the objective functions of the sub-missions and can be applied with no assumptions about the distribution of the performance data. The Friedman test has proven to be a useful tool for assessing the relative expected performance of automatic design methods [16, 20, 51, 52].

In our experiments, we use different performance measures to conduct the optimization process (scores, hypervolume, and the l^2-norm). A priori, focusing solely on the aggregated results based on one of these measures could unfairly favor the design methods that use that same measure. Therefore, to make fair comparisons, we report the relative aggregate performance of the methods when evaluated according to the three: the scores, hypervolume, and the l^2-norm. We presents results of three Friedman tests computed according to the scores, hypervolume, and the l^2-norm. By considering and presenting them all, we compensate for possible bias.

A method is considered significantly better than another if it has a lower average rank, and there is no overlap in the confidence intervals of the two methods.

The per-mission results are presented through scatter plots that show the performance of control software instances in the objective space—the scores. These scatter plots are provided for each mission. For the weighted sum approach, we only include the best sets from `NEAT-Color`, `EvoColor`, and `TuttiFrutti` as they illustrate the best performance one could expect.

5 Results

We present results on a per-method basis. Figure 5 shows three statistical comparisons between the methods. Figure 5.A shows the results of a Friedman test that is applied directly to the scores obtained from the evaluation of the generated control software. Figure 5.B also shows the results of a Friedman test, but in this case the test is applied after aggregating the scores using the hypervolume. Figure 5.C shows the results of a Friedman test that is applied after aggregating the scores using the l^2-norm. Figure 6 shows scatter plots with the scores obtained for each mission. The code, the control software produced, the experimental results, and the demonstration videos are available as supplementary material [69].

`Mandarina`. In all three Friedman tests (Figure 5), the average rank of `Mandarina` is significantly lower than the average rank of the baseline methods. `Mandarina` outperformed the baselines, whether the comparison is conducted directly using scores, the hypervolume, or the l^2-norm. This indicates that `Mandarina` was more effective than the other methods in producing neutral compromise instances of control software that satisfied the design criteria. In the per-mission results (Figure 6), the score of `Mandarina`'s instances is mostly concentrated in the top-right region of the scatter plots. This shows that, in most cases, the collective behaviors designed by `Mandarina` aim at maximizing the performance of the swarm in the two sub-missions.

`TF-WS-`B, `TF-WS-`M, `TF-WS-`W. By exploring a wide range of weight combinations, we generated a repertoire of control software instances that satisfy the design criteria in varying degrees. From this repertoire, we identified the best (`TF-WS-`B), median (`TF-WS-`M), and worst (`TF-WS-`W) sets using the post hoc decision-making process described in Section 4. In the three Friedman tests (Figure 5), the average rank of the best, median, and worst sets of instances of `TuttiFrutti` is significantly lower than the average rank of the sets of `EvoColor` and `NEAT-Color`. `TuttiFrutti` generated sets of instances that generally outperform those produced by `EvoColor` and `NEAT-Color` when using the weighted sum approach. On the other hand, the relative performance between the selected sets (`TF-WS-`B, `TF-WS-`M, `TF-WS-`W) and the other variants of `TuttiFrutti` (`TF-HV` and `TF-L2`) differs

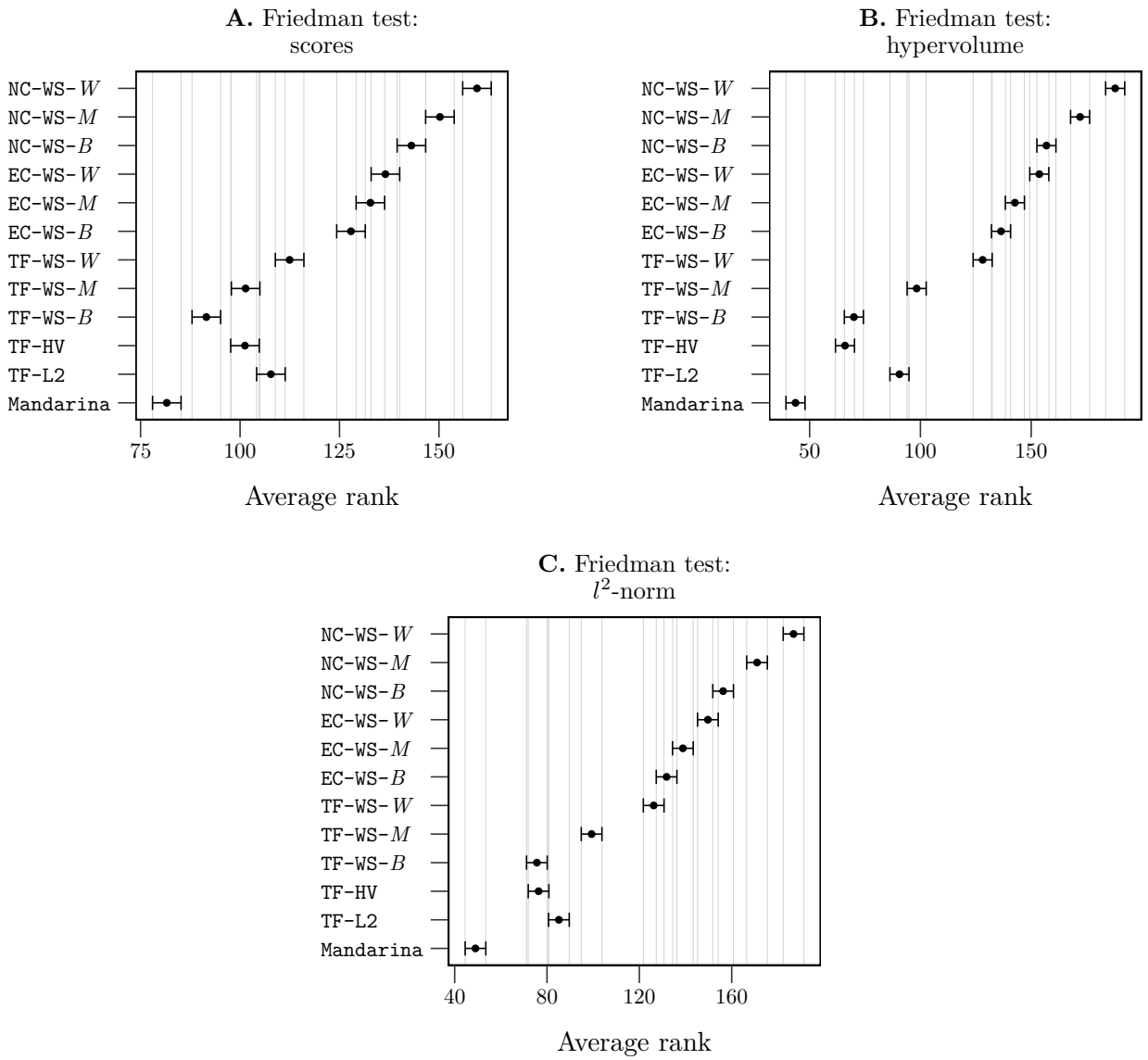

Figure 5: Friedman tests. A. The test is directly applied to the scores. B. The test is applied after aggregating the scores using the hypervolume. C. The test is applied after aggregating the scores using the l^2-norm. In all cases, the test considers the results obtained across all missions and provides a relative ranking between the methods. The plot shows the average rank of each method and its 95 % confidence interval. The lower the rank, the better.

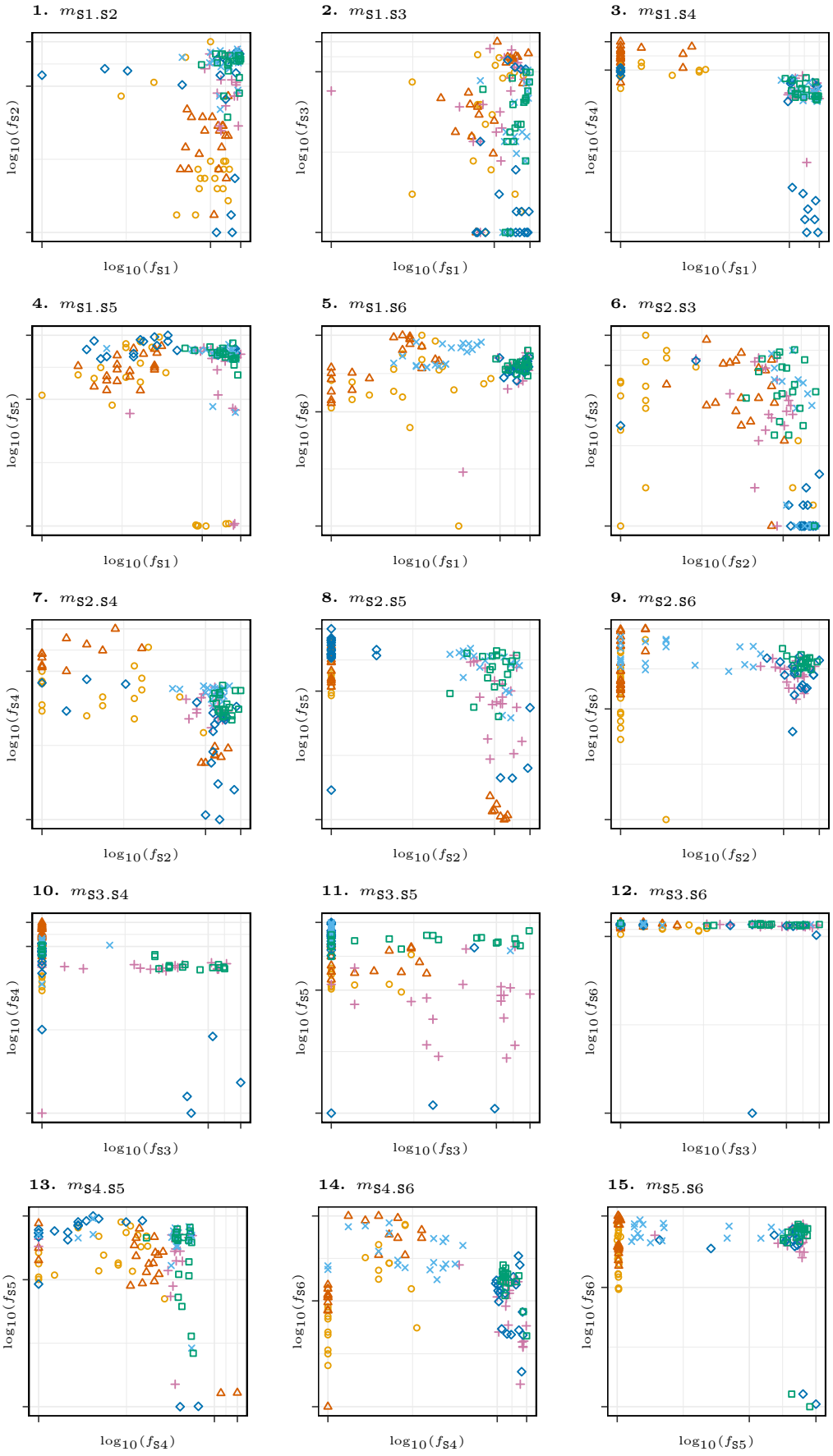

Figure 6: Per-mission results. The scatter plots show the performance (scores) of control software instances in the objective space for the 15 missions, and for the methods `NC-WS-`B (○), `EC-WS-`B (△), `TF-WS-`B (×), `TF-HV` (+), `TF-L2` (◇), and `Mandarina` (□). The results are displayed using a logarithmic scale to accommodate the performance ranges of the sub-missions, which span varying orders of magnitude. In all plots, the higher, the better.

depending on the metrics used to conduct the test. TF-WS-B has a significantly lower rank than TF-HV and TF-L2 when the instances are directly evaluated with respect to the scores (Figure 5.A). However, the rank difference fades if the instances are evaluated with respect to the hypervolume and the l^2-norm—see Figure 5.B and 5.C, respectively. In the per-mission results (Figure 6), the score of TF-WS-B's instances is spread in the top-right region of the scatter plots. This indicates that the collective behaviors designed by TF-WS-B aim at maximizing the performance of the swarm in the two sub-missions, in most cases.

TF-HV. In all Friedman tests (Figure 5), TF-HV has a significantly lower rank than the methods based on EvoColor and NEAT-Color. However, compared to other methods based on TuttiFrutti, the average rank of TF-HV also varied depending on the metrics used to conduct the tests. If the instances are evaluated with respect to the hypervolume, TF-HV ranks similarly to TF-WS-B (the best set)—see Figure 5.B. However, if the instances are directly evaluated using the scores, the average rank of TF-HV is similar to that observed in TF-WS-M (the median set)—see Figure 5.A. A similar phenomenon appears when comparing TF-HV and TF-L2. If the instances are evaluated with respect to the hypervolume, the rank of TF-HV is significantly lower than that of TF-L2—see Figure 5.B. However, when the instances are evaluated based on the l^2-norm, which is the measure used in the optimization process of TF-L2, our experimental setup was unable to detect significant differences between TF-HV and TF-L2—see Figure 5.C. In the per-mission results (Figure 6), the score of TF-HV's instances is spread near the top-right region of the scatter plots. This shows that most of the designed collective behaviors also aim at maximizing the performance of the swarm in the two sub-missions.

TF-L2. The average rank of TF-L2 is significantly lower than that of the methods based on EvoColor and NEAT-Color in the three Friedman tests (Figure 5). However, as discussed previously, the relative average rank of TF-L2 compared to other methods based on TuttiFrutti varies depending on the metric used to conduct the tests. If the instances are directly evaluated with respect to the scores, the average rank of TF-L2 is between TF-WS-M and TF-WS-W—see Figure 5.A. This rank shows a sub-par performance for the method. If the instances are evaluated with respect to the hypervolume, the average rank of TF-L2 is situated between TF-WS-B and TF-WS-M—see Figure 5.B. Unlike the previous case, this second rank shows that the method has moderate performance. TF-L2 ranks the best if the instances are evaluated with respect to the l^2-norm—see Figure 5.C. In this third case, we did not observe a significant difference between the average rank of TF-L2 and that of TF-WS-B and TF-HV, and the average rank of TF-L2 is significantly lower than that of TF-WS-M. In the per-mission results (Figure 6), TF-L2 instances scored with mixed performance. In a minor share of missions, the instances are distributed close to the top-right region of the scatter plots. However, in a larger share, they concentrate near the axes of the scatter plots. This shows that, in most cases, TF-L2 failed to consistently design collective behaviors that aim at maximizing the performance of the swarm in the two sub-missions.

EC-WS-B, EC-WS-M, EC-WS-W. Likewise with TuttiFrutti, we also used EvoColor to generate a repertoire of behaviors and we identified the best (EC-WS-B), median (EC-WS-M), and worst (EC-WS-W) sets. In the three Friedman tests (Figure 5), the average rank of the sets of EvoColor is only significantly lower than that of the sets of NEAT-Color. All other alternative methods outperformed EvoColor, except for NEAT-Color. The per-mission results (Figure 6) show that the score of EC-WS-B's instances is concentrated near the axes of the scatter plots, attaining high values in a single sub-mission but not in the other. Instead of generating neutral compromise solutions that maximize the performance in both sub-missions, EvoColor primarily designed collective behaviors that prioritize maximizing the performance of one of them. By only addressing a single design criterion, EvoColor instances are able to achieve the highest performance values in single sub-missions—compared to the neutral compromise solutions produced by the modular methods. However, doing so, EvoColor did not produce instances of control software that fully satisfy all the design criteria.

NC-WS-B, NC-WS-M, NC-WS-W. The experiments with NEAT-Color showed results that are qualitatively similar to those obtained with EvoColor. Also in this case, we generated a repertoire of behaviors and we identified the best (NC-WS-B), median (NC-WS-M), and worst (NC-WS-W) sets. In the three Friedman tests, all alternative methods outperformed the sets of NEAT-Color—see Figure 5. The per-mission results (Figure 6) show that, like EC-WS-B, the score of NC-WS-B's instances concentrates near the axes of the scatter plots. However, in general, NEAT-Color's instances often achieve lower scores compared to EvoColor's instances. The collective

behaviors designed by NEAT-Color also prioritize maximizing the performance of a single design criterion. In this sense, NEAT-Color also failed to produce instances of control software that fully satisfy the design criteria.

We assessed the control software produced by NC-WS-B, EC-WS-B, TF-WS-B, TF-HV, TF-L2, and Mandarina in physical e-pucks. Video demonstrations are provided as supplementary material [69]. The demonstrations show that modular methods—TF-WS-B, TF-HV, TF-L2, and Mandarina—transfer better from simulation to reality than those based on neuroevolution—NC-WS-B, EC-WS-B. In the case of modular methods, the behavior observed in physical e-pucks qualitatively resembles that observed in simulation. Conversely, in the case of neuroevolutionary methods, the e-pucks do not show any meaningful behavior that resembles the one observed in simulation. This is a phenomenon that has been observed in previous automatic design studies [17, 20, 51]. Neuroevolutionary methods are known to overfit the simulation environment, which prevents their control software from porting well to physical robots. Our results show that this phenomenon also seems to occur in the design problem we consider.

6 Discussion

The optimization-based design of robot swarms (fully or semi-automatic) has been traditionally studied on missions specified as single-criterion design problems, whether they are inherently single-criterion or adapted from multi-criteria ones. Consequently, there is currently no established experimental framework for specifying multi-criteria missions and studying how automatic design methods perform when addressing them. We focused on studying a bi-criteria design problem. To conduct our experiments, we specified bi-criteria missions by combining instances of well-known swarm robotics problems—foraging, aggregation, and coverage—into missions consisting of sequential parts. More precisely, (i) we specified a diverse set of sub-missions to be combined in a set of missions, (ii) we systematically performed experiments on the resultant combinations, and (ii) we compared the methods on their aggregated performance across all missions in the set. These sequential missions allowed us to estimate the expected performance of the methods under consideration and demonstrated to be an appropriate experimental framework for bi-criteria design studies. The missions we devised here are intended to be performed by e-puck robots that comply with the reference model RM 3. However, we contend that the experimental protocol in this paper can be used to develop new experimental setups for other robot platforms or environments.

The missions we devised are distinctive from the related literature in that they consist of a combination of sub-missions that must be executed sequentially. In this class of missions, the modular methods (Mandarina and TuttiFrutti) outperformed the neuroevolutionary ones (EvoColor and NEAT-Color). Mandarina and TuttiFrutti designed robot swarms that, as anticipated, use the color cues (red and blue lights) to switch between behaviors and execute each sub-mission with a tailored action. On the contrary, EvoColor and NEAT-Color designed robot swarms that do not seem to react to color cues. Instead, they typically stick to a single behavior that either addresses the two sub-missions to a very limited extent, or that is only tailored to address one of them. By observing the results of the two approaches, we argue that the modular methods performed better because they succeeded in designing robot swarms that can switch behaviors at runtime, which we consider necessary for executing the sequential missions. Switching between behaviors is a rather complex action inherently enabled on Mandarina and TuttiFrutti due to their modular control architecture—i.e., the probabilistic finite-state machine. The neuroevolutionary methods, on the contrary, must evolve the switching behavior from scratch within the artificial neural network. Neither EvoColor nor the more sophisticated NEAT-Color achieved this task successfully.

We introduced Mandarina to leverage Iterated F-race's non-parametric, rank-based nature and address challenges that arise in multi-criteria design problems. On the one hand, due to its non-parametric nature, we could use Mandarina in all missions without requiring information about the behavior/distribution of the objective functions. The objective functions of the sub-missions were ultimately treated as black-boxes during the optimization process. On the other hand, due to its rank-based nature, Mandarina could effectively handle

pairs of objective functions with different score ranges, discrete and continuous, and across various orders of magnitude. We compared Mandarina with other alternative methods that also use Iterated F-race—i.e., the baselines derived from TuttiFrutti. The aggregated results show that Mandarina achieved significantly better performance. Mandarina differs from the bi-criteria methods based on TuttiFrutti in that Iterated F-race directly operates on the scores obtained in each sub-mission. Indeed, Mandarina operates without normalization or estimation of the performance bounds of the objective functions. On the contrary, in the bi-criteria methods based on TuttiFrutti, Iterated F-race operates on performance estimations that result from aggregating the scores into single performance measures—the weighted sum, the hypervolume, or the l^2-norm. We conjecture that in Mandarina, Iterated F-race could perform more accurate statistical comparisons of the performance of the candidate solutions by directly comparing the scores of each sub-mission. This could possibly have contributed to the search for better performing control software instances. In the methods based on TuttiFrutti, aggregating the scores into a single measure may have hindered the ability of Iterated F-race to statistically distinguish between the performance of candidate solutions. We will devote future work to deepening our understanding of this issue.

We dedicated a substantial part of our research to estimating the expected performance of weighted sum methods with respect to the other approaches under evaluation. To do so, we conducted a systematic search to find suitable weight combinations for each method and mission individually. TF-WS-B, EC-WS-B, and NC-WS-B are sets of control software instances generated using the best weight combinations found in the search. In a sense, these three sets represent the expected outcome of a design process in which a designer is able to set appropriate weights to scalarize the objective functions across all missions. The results of our experiments showed that, even in this best-case scenario, the most effective weighted sum method (TF-WS-B) did not outperform other bi-criteria approaches (e.g., TF-HV and TF-L2). On the contrary, it was outperformed by one of them: Mandarina. The weighted sum is currently the common approach for framing/addressing multi-criteria problems in the optimization-based design of robot swarms. Here, we provide empirical evidence to support the idea that exploring multi-criteria design methods beyond the conventional weighted sum is worthwhile. Firstly, a designer can avoid the time-consuming and costly process of manually searching for suitable weights. Secondly, methods like Mandarina can produce solutions that perform better than those obtained with the weighted sum. We expect these results to motivate future research on approaches for addressing multiple design criteria in the fully and semi-automatic design of robot swarms.

7 Conclusions

In this paper, we investigated the automatic design of control software for robot swarms under concurrent design criteria. To conduct our study, we introduced Mandarina, an automatic design method that generates control software for robot swarms in the form of probabilistic finite-state machines. We used Mandarina to address design problems in which the mission the swarm must perform is specified as a set of independent objective functions to be optimized concurrently.

Mandarina utilizes Iterated F-race to perform a multi-objective optimization process that searches for control software instances that maximize the performance of the swarm in the mission at hand. That is, the instances that allow the swarm to satisfy all the design criteria to their best. The traditional approach to using Iterated F-race in multi-objective optimization problems is to aggregate the design criteria into a single performance measure. In Mandarina, on the contrary, we use Iterated F-race to conduct the optimization process without aggregating the design criteria. Our experiments show that this approach outperforms the traditional approach, including methods that use the weighted sum, the hypervolume, and the l^2-norm.

The related literature shows that multi-criteria design problems are common and interesting to the scientific community. In fact, researchers often unintentionally specify missions that simultaneously express varied preferences and design criteria related to mission outcomes, robot behavior, or the design process itself. However, in the related studies, the researchers neglect the multi-criteria nature of the design problem and the

possible benefit of using multi-objective optimization methods to address it. Our study on bi-criteria design presents elements that can bootstrap further research into the broader multi-criteria design of robot swarms using optimization-based techniques. More precisely, we provide here (i) an experimental framework for specifying bi-criteria missions, which can be extended to consider additional criteria; (ii) a set of baseline approaches—the weighted sum, the hypervolume, and the l^2-norm; (iii) empirical results regarding popular automatic design approaches—modular design and neuroevolution; and (iii) we demonstrate the feasibility of these automatic design processes with both simulations and physical robots.

We identified two key questions to be addressed in upcoming research. *What is the most suitable way to compare design methods effectively?* Our study considered design methods that rely on optimization processes driven by different performance measures. We noticed that the expected performance of a method compared to others can vary depending on the metric used to assess the generated control software instances. For this reason, here we decided to analyze and compare the methods with respect to all performance measures used in the optimization processes. However, a better protocol should be defined to compare the methods in a fair way. *To which extent* `Mandarina`*'s capability to address multi-criteria design problems generalize beyond the bi-objective case?* In this paper, we focused on missions specified by two concurrent design criteria. However, in practice, `Mandarina` could handle additional criteria as long as each is provided to Iterated F-race as an independent problem instance to be considered in the optimization process. Considering more concurrent criteria during the design process can lead to a greater number of performance trade-offs and potentially make Iterated F-race's statistical comparison of control software less effective. Future research should expand the experimental setup to investigate this issue.

Data availability statement

The data that supports the findings of this study are available in the supplementary material of this article, see [69].

Acknowledgments

The authors thank Dr. Leslie Pérez Cáceres for helping in setting up `irace`'s hypervolume implementation. 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 through the crédit d'équipement SwarmSim and the crédit de recherche SwarmUp. DGR acknowledges support from the Colombian Ministry of Science, Technology and Innovation–Minciencias. TS and MB acknowledge support from the Belgian Fonds de la Recherche Scientifique–FNRS, of which they are Research Directors.

References

[1] G. Beni, From swarm intelligence to swarm robotics, in: E. Şahin, W. M. Spears (Eds.), Swarm Robotics: SAB 2004 International Workshop, Vol. 3342 of Lecture Notes in Computer Science, Springer, Berlin, Germany, 2005, pp. 1–9. doi:10.1007/978-3-540-30552-1_1.

[2] E. Şahin, Swarm robotics: from sources of inspiration to domains of application, in: E. Şahin, W. M. Spears (Eds.), Swarm Robotics: SAB 2004 International Workshop, Vol. 3342 of Lecture Notes in Computer Science, Springer, Berlin, Germany, 2005, pp. 10–20. doi:10.1007/978-3-540-30552-1_2.

[3] J. Kuckling, Recent trends in robot learning and evolution for swarm robotics, Frontiers in Robotics and AI 10 (2023) 1134841. doi:10.3389/frobt.2023.1134841.

[4] M. Brambilla, E. Ferrante, M. Birattari, M. Dorigo, Swarm robotics: a review from the swarm engineering perspective, Swarm Intelligence 7 (1) (2013) 1–41. doi:10.1007/s11721-012-0075-2.

[5] M. Schranz, M. Umlauft, M. Sende, W. Elmenreich, Swarm robotic behaviors and current applications, Frontiers in Robotics and AI 7 (2020) 36. doi:10.3389/frobt.2020.00036.

[6] H. Hamann, M. Schranz, W. Elmenreich, V. Trianni, C. Pinciroli, N. Bredeche, E. Ferrante, Editorial: designing self-organization in the physical realm, Frontiers in Robotics and AI 7 (2020) 164. doi:10.3389/frobt.2020.597859.

[7] M. Dorigo, G. Theraulaz, V. Trianni, Reflections on the future of swarm robotics, Science Robotics 5 (2020) eabe4385. doi:10.1126/scirobotics.abe4385.

[8] G. Francesca, M. Birattari, Automatic design of robot swarms: achievements and challenges, Frontiers in Robotics and AI 3 (29) (2016) 1–9. doi:10.3389/frobt.2016.00029.

[9] M. Birattari, A. Ligot, K. Hasselmann, Disentangling automatic and semi-automatic approaches to the optimization-based design of control software for robot swarms, Nature Machine Intelligence 2 (9) (2020) 494–499. doi:10.1038/s42256-020-0215-0.

[10] M. Birattari, A. Ligot, D. Bozhinoski, M. Brambilla, G. Francesca, L. Garattoni, D. Garzón Ramos, K. Hasselmann, M. Kegeleirs, J. Kuckling, F. Pagnozzi, A. Roli, M. Salman, T. Stützle, Automatic off-line design of robot swarms: a manifesto, Frontiers in Robotics and AI 6 (2019) 59. doi:10.3389/frobt.2019.00059.

[11] S. Nolfi, D. Floreano, Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines, 1st Edition, MIT Press, Cambridge, MA, USA, 2000, a Bradford Book.

[12] V. Trianni, Evolutionary Swarm Robotics, Springer, Berlin, Germany, 2008. doi:10.1007/978-3-540-77612-3.

[13] S. Doncieux, N. Bredeche, J.-B. Mouret, A. Eiben, Evolutionary robotics: what, why, and where to, Frontiers in Robotics and AI 2 (2015) 4. doi:10.3389/frobt.2015.00004.

[14] G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, M. Birattari, AutoMoDe: a novel approach to the automatic design of control software for robot swarms, Swarm Intelligence 8 (2) (2014) 89–112. doi:10.1007/s11721-014-0092-4.

[15] M. Birattari, A. Ligot, G. Francesca, AutoMoDe: a modular approach to the automatic off-line design and fine-tuning of control software for robot swarms, in: N. Pillay, R. Qu (Eds.), Automated Design of Machine Learning and Search Algorithms, Natural Computing Series, Springer, Cham, Switzerland, 2021, pp. 73–90. doi:10.1007/978-3-030-72069-8_5.

[16] G. Francesca, M. Brambilla, A. Brutschy, L. Garattoni, R. Miletitch, G. Podevijn, A. Reina, T. Soleymani, M. Salvaro, C. Pinciroli, F. Mascia, V. Trianni, M. Birattari, AutoMoDe-Chocolate: automatic design of control software for robot swarms, Swarm Intelligence 9 (2–3) (2015) 125–152. doi:10.1007/s11721-015-0107-9.

[17] D. Garzón Ramos, M. Birattari, Automatic design of collective behaviors for robots that can display and perceive colors, Applied Sciences 10 (13) (2020) 4654. doi:10.3390/app10134654.

[18] K. Hasselmann, M. Birattari, Modular automatic design of collective behaviors for robots endowed with local communication capabilities, PeerJ Computer Science 6 (2020) e291. doi:10.7717/peerj-cs.291.

[19] J. Kuckling, V. van Pelt, M. Birattari, 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, Vol. 12694 of Lecture Notes in Computer Science, Springer, Cham, Switzerland, 2021, pp. 130–145. doi:10.1007/978-3-030-72699-7_9.

[20] K. Hasselmann, A. Ligot, J. Ruddick, M. Birattari, Empirical assessment and comparison of neuro-evolutionary methods for the automatic off-line design of robot swarms, Nature Communications 12 (2021) 4345. doi:10.1038/s41467-021-24642-3.

[21] S. Jones, M. Studley, S. Hauert, A. Winfield, 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, Vol. 6 of Springer Proceedings in Advanced Robotics, Springer, Cham, Switzerland, 2018, pp. 487–501. doi:10.1007/978-3-319-73008-0_34.

[22] S. Jones, A. Winfield, S. Hauert, M. Studley, Onboard evolution of understandable swarm behaviors, Advanced Intelligent Systems 1 (6) (2019) 1900031. doi:10.1002/aisy.201900031.

[23] D. Marocco, S. Nolfi, Emergence of communication in embodied agents evolved for the ability to solve a collective navigation problem, Connection Science 19 (1) (2007) 53–74. doi:10.1080/09540090601015067.

[24] A. L. Christensen, M. Dorigo, Evolving an integrated phototaxis and hole-avoidance behavior for a swarm-bot, in: L. M. Rocha, L. S. Yaeger, M. A. Bedau, D. Floreano, R. L. Goldstone, A. Vespignani (Eds.), Artificial Life X: Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems, MIT Press, Cambridge, MA, USA, 2006, pp. 248–254, a Bradford Book.

[25] M. Quinn, L. Smith, G. Mayley, P. Husbands, Evolving controllers for a homogeneous system of physical robots: structured cooperation with minimal sensors, Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences 361 (1811) (2003) 2321–2343. doi:10.1098/rsta.2003.1258.

[26] M. Duarte, V. Costa, J. Gomes, T. Rodrigues, F. Silva, S. M. Oliveira, A. L. Christensen, Evolution of collective behaviors for a real swarm of aquatic surface robots, PLOS ONE 11 (3) (2016) e0151834. doi:10.1371/journal.pone.0151834.

[27] C. Ampatzis, E. Tuci, V. Trianni, A. L. Christensen, M. Dorigo, Evolving self-assembly in autonomous homogeneous robots: experiments with two physical robots, Artificial Life 15 (4) (2009) 465–484. doi:10.1162/artl.2009.Ampatzis.013.

[28] M. Duarte, S. M. Oliveira, A. L. Christensen, Hybrid control for large swarms of aquatic drones, in: H. Sayama, J. Rieffel, S. Risi, R. Doursat, H. Lipson (Eds.), ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems, MIT Press, Cambridge, MA, USA, 2014, pp. 785–792. doi:10.7551/978-0-262-32621-6-ch105.

[29] V. Trianni, S. Nolfi, Self-organizing sync in a robotic swarm: a dynamical system view, IEEE Transactions on Evolutionary Computation 13 (4) (2009) 722–741. doi:10.1109/TEVC.2009.2015577.

[30] V. Trianni, M. López-Ibáñez, Advantages of task-specific multi-objective optimisation in evolutionary robotics, PLOS ONE 10 (8) (2015) e0136406. doi:10.1371/journal.pone.0136406.

[31] J. Gomes, P. Urbano, A. L. Christensen, Evolution of swarm robotics systems with novelty search, Swarm Intelligence 7 (2–3) (2013) 115–144. doi:10.1007/s11721-013-0081-z.

[32] J. Gomes, A. L. Christensen, Task-agnostic evolution of diverse repertoires of swarm behaviours, in: M. Dorigo, M. Birattari, C. Blum, A. L. Christensen, A. Reina, V. Trianni (Eds.), Swarm Intelligence: 11th International Conference, ANTS 2018, Vol. 11172 of Lecture Notes in Computer Science, Springer, Cham, Switzerland, 2018, pp. 225–238. doi:10.1007/978-3-030-00533-7_18.

[33] P. C. Fishburn, Utility Theory for Decision Making, Publications in Operations Research, John Wiley & Sons, New York, NY, USA, 1970.

[34] C. Zopounidis, M. Doumpos (Eds.), Multiple Criteria Decision Making: Applications in Management and Engineering, Multiple Criteria Decision Making, Springer, Cham, Switzerland, 2017. doi:10.1007/978-3-319-39292-9.

[35] M. T. M. Emmerich, A. H. Deutz, A tutorial on multiobjective optimization: fundamentals and evolutionary methods, Natural Computing 17 (3) (2018) 585–609. doi:10.1007/s11047-018-9685-y.

[36] M. Salman, A. Ligot, M. Birattari, Concurrent design of control software and configuration of hardware for robot swarms under economic constraints, PeerJ Computer Science 5 (2019) e221. doi:10.7717/peerj-cs.221.

[37] A. Ligot, J. Kuckling, D. Bozhinoski, M. Birattari, Automatic modular design of robot swarms using behavior trees as a control architecture, PeerJ Computer Science 6 (2020) e314. doi:10.7717/peerj-cs.314.

[38] P. Balaprakash, M. Birattari, T. Stützle, Improvement strategies for the F-Race algorithm: sampling design and iterative refinement, in: T. Bartz-Beielstein, M. J. Blesa, C. Blum, B. Naujoks, A. Roli, G. Rudolph, M. Sampels (Eds.), Hybrid Metaheuristics: 4th International Workshop, HM 2007, Vol. 4771 of Lecture Notes in Computer Science, Springer, Berlin, Germany, 2007, pp. 108–122. doi:10.1007/978-3-540-75514-2_9.

[39] M. Birattari, Z. Yuan, P. Balaprakash, T. Stützle, F-Race and Iterated F-Race: an overview, in: T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss (Eds.), Experimental Methods for the Analysis of Optimization Algorithms, Springer, Berlin, Germany, 2010, pp. 311–336. doi:10.1007/978-3-642-02538-9_13.

[40] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, T. Stützle, The irace package: iterated racing for automatic algorithm configuration, Operations Research Perspectives 3 (2016) 43–58. doi:10.1016/j.orp.2016.09.002.

[41] A. Ligot, M. Birattari, Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms, Swarm Intelligence 14 (2020) 1–24. doi:10.1007/s11721-019-00175-w.

[42] S. Doncieux, J.-B. Mouret, Beyond black-box optimization: a review of selective pressures for evolutionary robotics, Evolutionary Intelligence 7 (2) (2014) 71–93. doi:10.1007/s12065-014-0110-x.

[43] S. A. Engebråten, J. Moen, O. A. Yakimenko, K. Glette, A framework for automatic behavior generation in multi-function swarms, Frontiers in Robotics and AI 7 (2020) 175. doi:10.3389/frobt.2020.579403.

[44] D. S. Brown, R. Turner, O. Hennigh, S. Loscalzo, Discovery and exploration of novel swarm behaviors given limited robot capabilities, in: R. Groß, A. Kolling, S. Berman, E. Frazzoli, A. Martinoli, F. Matsuno, M. Gauci (Eds.), Distributed Autonomous Robotic Systems: The 13th International Symposium, Vol. 6 of Springer Proceedings in Advanced Robotics, Springer, Cham, Switzerland, 2018, pp. 447–460. doi:10.1007/978-3-319-73008-0_31.

[45] S. A. Engebråten, J. Moen, O. A. Yakimenko, K. Glette, Evolving a repertoire of controllers for a multi-function swarm, in: K. Sim, P. Kaufmann (Eds.), Applications of Evolutionary Computation: 21st International Conference, EvoApplications 2018, Vol. 10784 of Lecture Notes in Computer Science, Springer, Cham, Switzerland, 2021, pp. 734–749. doi:10.1007/978-3-319-77538-8_49.

[46] K. Mason, S. Hauert, Evolving multi-objective neural network controllers for robot swarms, in: N. Basilico, M. Sridharan, N. Agmon, F. Amigoni, J. Biswas, A. Farinelli, M. Gini, G. A. Kaminka, D. Nardi (Eds.), 2023 Autonomous Robots and Multirobot Systems Workshop, 2023.

[47] A. J. Ijspeert, A. Martinoli, A. Billard, L. M. Gambardella, Collaboration through the exploitation of local interactions in autonomous collective robotics: the stick pulling experiment, Autonomous Robots 11 (2) (2001) 149–171. doi:10.1023/A:1011227210047.

[48] R. T. Marler, J. S. Arora, The weighted sum method for multi-objective optimization: new insights, Structural and Multidisciplinary Optimization 41 (6) (2010) 853–862. doi:10.1007/s00158-009-0460-7.

[49] E. Zitzler, L. Thiele, Multiobjective optimization using evolutionary algorithms – a comparative case study, in: A. Eiben, T. Bäck, M. Schoenauer, H.-P. Schwefel (Eds.), Parallel Problem Solving from Nature – PPSN V: 5th International Conference, Vol. 1498 of Lecture Notes in Computer Science, Springer, Cham, Switzerland, 1998, pp. 292–301. doi:10.1007/BFb0056872.

[50] J. Kuckling, T. Stützle, M. Birattari, Iterative improvement in the automatic modular design of robot swarms, PeerJ Computer Science 6 (2020) e322. doi:10.7717/peerj-cs.322.

[51] M. Salman, D. Garzón Ramos, M. Birattari, Automatic design of stigmergy-based behaviours for robot swarms, Communications Engineering 3 (2024) 30. doi:10.1038/s44172-024-00175-7.

[52] M. Kegeleirs, D. Garzón Ramos, K. Hasselmann, L. Garattoni, G. Francesca, M. Birattari, Transferability in the automatic off-line design of robot swarms: from sim-to-real to embodiment and design-method transfer across different platforms, IEEE Robotics and Automation Letters 9 (3) (2024) 2758–2765. doi:10.1109/LRA.2024.3360013.

[53] F. Pagnozzi, M. Birattari, Off-policy evaluation of the performance of a robot swarm: importance sampling to assess potential modifications to the finite-state machine that controls the robots, Frontiers in Robotics and AI 8 (2021) 55. doi:10.3389/frobt.2021.625125.

[54] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, A. Martinoli, The e-puck, a robot designed for education in engineering, in: P. Gonçalves, P. Torres, C. Alves (Eds.), ROBOTICA 2009: Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, Instituto Politécnico de Castelo Branco, Castelo Branco, Portugal, 2009, pp. 59–65.

[55] L. Garattoni, G. Francesca, A. Brutschy, C. Pinciroli, M. Birattari, Software infrastructure for e-puck (and TAM), Tech. Rep. TR/IRIDIA/2015-004, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (2015).

[56] M. Dorigo, G. Theraulaz, V. Trianni, Swarm robotics: past, present, and future [point of view], Proceedings of the IEEE 109 (7) (2021) 1152–1165. doi:10.1109/JPROC.2021.3072740.

[57] K. Hasselmann, A. Ligot, G. Francesca, D. Garzón Ramos, M. Salman, J. Kuckling, F. J. Mendiburu, M. Birattari, Reference models for AutoMoDe, Tech. Rep. TR/IRIDIA/2018-002, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (2018).

[58] Á. Gutiérrez, A. Campo, M. Dorigo, J. Donate, F. Monasterio-Huelin, L. Magdalena, Open e-puck range & bearing miniaturized board for local communication in swarm robotics, in: K. Kosuge (Ed.), 2009 IEEE International Conference on Robotics and Automation (ICRA), IEEE, Piscataway, NJ, USA, 2009, pp. 3111–3116. doi:10.1109/ROBOT.2009.5152456.

[59] J. Borenstein, Y. Koren, Real-time obstacle avoidance for fast mobile robots, IEEE Transactions on Systems, Man, and Cybernetics 19 (5) (1989) 1179–1187. doi:10.1109/21.44033.

[60] M. López-Ibáñez, L. Pérez Cáceres, J. Dubois-Lacoste, T. Stützle, M. Birattari, The irace package: user guide, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, version 3.4.1 (2020).

[61] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. A. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, M. Dorigo, ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems, Swarm Intelligence 6 (4) (2012) 271–295. doi:10.1007/s11721-012-0072-5.

[62] W. J. Conover, Practical Nonparametric Statistics, 3rd Edition, Wiley Series in Probability and Statistics, John Wiley & Sons, New York, NY, USA, 1999.

[63] F. J. Mendiburu, D. Garzón Ramos, M. R. A. Morais, A. M. N. Lima, M. Birattari, AutoMoDe-Mate: automatic off-line design of spatially-organizing behaviors for robot swarms, Swarm and Evolutionary Computation 74 (2022) 101118. doi:10.1016/j.swevo.2022.101118.

[64] D. Garzón Ramos, M. Salman, K. Ubeda Arriaza, K. Hasselmann, M. Birattari, MoCA: a modular RGB color arena for swarm robotics experiments, Tech. Rep. TR/IRIDIA/2022-014, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (2022).

[65] K. O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, Evolutionary Computation 10 (2) (2002) 99–127. doi:10.1162/106365602320169811.

[66] C. M. Fonseca, L. Paquete, M. López-Ibáñez, An improved dimension-sweep algorithm for the hypervolume indicator, in: 2006 IEEE Congress on Evolutionary Computation, IEEE, Piscataway, NJ, USA, 2006, pp. 1157–1163. doi:10.1109/CEC.2006.1688440.

[67] A. P. Guerreiro, C. M. Fonseca, L. Paquete, The hypervolume indicator: computational problems and algorithms, ACM Computing Surveys 54 (6) (2021). doi:10.1145/3453474.

[68] K. Miettinen, Nonlinear Multiobjective Optimization, Springer, New York, NY, USA, 1998. doi:10.1007/978-1-4615-5563-6.

[69] D. Garzón Ramos, F. Pagnozzi, T. Stützle, M. Birattari, Automatic design of robot swarms under concurrent design criteria: a study based on Iterated F-Race: supplementary material, `https://iridia.ulb.ac.be/supp/IridiaSupp2024-002` (2024).