ECOLE
POLYTECHNIQUE
DE BRUXELLES

ULB

UNIVERSITÉ LIBRE DE BRUXELLES

Design of robot swarms by optimization:

An instance of AutoMoDe based on simulated annealing

Mémoire présenté en vue de l'obtention du diplôme
d'Ingénieur Civil en informatique à finalité Intelligence Artificielle

Keneth Efrén Ubeda Arriaza

Director
Professor Mauro Birattari

Supervisor
Jonas Kuckling

Service
IRIDIA - Institut de Recherches Interdisciplinaires et de Developpements en Intelligence Artificielle

DEMIURGE
Automatic Design of Robot Swarms
An ERC Consolidator Project

Année académique
2018 - 2019

This 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

Preface

I would like to thank my thesis promotor, Mauro Birattari, for let me be part of this amazing project. I would also thank my supervisor, Jonas Kuckling, for all the help during the development of this thesis. And for his guidance and advises that allowed me to finish this work. I want to thank David Garzón Ramos, for his encouragement and help in the last stage of the development of this work. My sincere gratitude also goes to my wife, Yamina, who was there in every moment to support me in all she could. Finally I thank my parents Luis Ubeda and Berta Arriaza for their effort, that allowed me to extend my academic career.

To Katerin Elei Ubeda Arriaza, who will be always my little sister.
Para Katerin Elei Ubeda Arriaza, quien siempre será mi hermanita.

Bienaventurado el hombre que tiene en ti sus fuerzas,
En cuyo corazón están tus caminos.
Salmo 84:5

Keneth Efrén Ubeda Arriaza

Contents

Abstract

Robot swarms are groups of simple robots that can collectively accomplish missions that can not be accomplished by individual robots. The collective behavior of a robot swarm results from the individual interactions of the robots with their peers and with the environment. Despite of their simple nature, designing control software for robot swarms is a challenging endeavor—no methodology exists to determine how the individual robots should be programmed to obtain a desired collective behavior. The local interactions are difficult to predict, and in most cases, the design of the control software is manual and trial-and-error process. Still, robot swarms are becoming a prominent field in robotics because they are fully distributed, highly redundant and flexible systems.

In recent years, an effort has been devoted to the design of control software for robot swarms by automatic methods. AutoMoDe—Automatic Modular Design—is an automatic off-line approach to this purpose. In AutoMoDe, the design process is cast into an optimization problem that is solved off-line. In this thesis, I present AutoMoDe-Annealing: a new instance of AutoMoDe that implements simulated annealing as optimization algorithm. In particular, I study the influence of the components of simulated annealing when they operate with different optimization budgets and over different control architectures. I conduct my research with an empirical assessment of the control software produced for standard swarm missions: Foraging and Aggregation. Alongside, I compare the performance of AutoMoDe-Annealing with respect to the formerly published AutoMoDe-Chocolate and AutoMoDe-Maple, and the still-under-development AutoMoDe-Iterated. Results show that the control software architecture has the major influence in the performance of the produced control software. AutoMoDe-Annealing is capable of producing appropriate control software in the form of probabilistic finite state machines and behavior trees. Robot swarms designed with AutoMoDe-Annealing and probabilistic finite state machines outperformed the robot swarms designed with AutoMoDe-Chocolate and AutoMoDe-Maple. In the case of AutoMoDe-Annealing and behavior trees, the robot swarms performed similarly to AutoMoDe-Chocolate and AutoMoDe-Maple.

Résumé

Les essaims de robots sont des groupes de robots simples pouvant accomplir collectivement des missions impossibles à accomplir par des robots individuels. Le comportement collectif d'un essaim de robots résulte des interactions individuelles des robots avec leurs pairs et avec l'environnement. Malgré leur nature simple, la conception d'un logiciel de contrôle pour les essaims de robots est une tâche ardue. Il n'existe aucune méthodologie permettant de déterminer la manière dont les robots individuels devraient être programmés pour obtenir le comportement collectif souhaité. Les interactions locales sont difficiles à prévoir et, dans la plupart des cas, la conception du logiciel de contrôle est un processus manuel et un processus d'essai et d'erreur. Néanmoins, les essaims de robots sont en train de devenir un domaine de premier plan en robotique car ce sont des systèmes entièrement distribués, hautement redondants et flexibles.

Ces dernières années, un effort a été consacré à la conception de logiciels de contrôle pour essaims de robots utilisant des méthodes automatiques. AutoMoDe — Conception modulaire automatique — est une approche automatique en différé à cette fin. Dans AutoMoDe, le processus de conception est transformé en un problème d'optimisation qui est résolu en différé. Dans cette thèse, je présente AutoMoDe-Annealing: une nouvelle instance d'AutoMoDe qui implémente le *simulated annealing* en tant qu'algorithme d'optimisation. J'étudie en particulier l'influence des composants du *simulated annealing* lorsqu'ils fonctionnent avec différents budgets d'optimisation et sur différentes architectures de contrôle. Je mène mes recherches avec une évaluation empirique du logiciel de contrôle produit pour les missions d'essaim standard: *Foraging* et *Aggregation*. En parallèle, je compare les performances d'AutoMoDe-Annealing à celles d'AutoMoDe-Chocolate et AutoMoDe-Maple, précédemment publiées, et d'AutoMoDe-Iterated, encore en cours de développement. Les résultats montrent que l'architecture du logiciel de contrôle a une influence majeure sur les performances du logiciel de contrôle produit. AutoMoDe-Annealing est capable de produire un logiciel de contrôle approprié sous forme de machines probabilistes à états finis et d'arbres de comportement. Les essaims de robots conçus avec AutoMoDe-Annealing et avec des machines à états finis et probabilistes ont surpassé les essaims de robots conçus avec AutoMoDe-Chocolate et AutoMoDe-Maple. Dans le cas d'AutoMoDe-Annealing et des arbres de comportement, les essaims de robots ont fonctionné de manière similaire à AutoMoDe-Chocolate et AutoMoDe-Maple.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

AutoMoDe	Automatic Modular Design
ANN	Artificial Neural Networks
BT	Behavior Trees
ER	Evolutionary Robotics
FSM	Finite State Machines
OA	Optimization Algorithm
PFSM	Probabilistic Finite State Machine
SA	Simulated Annealing
SLS	Stochastic Local Search
SPA	Plan-Sense-Act

Symbols

T	Temperature
T_0	Initial Temperature
α	Cooling factor
s	Solution
s^*	Optimal solution
$\hat{s}$	Incumbent Solution
π	Optimization problem instance

Chapter 1

Introduction

Swarm intelligence can be defined as a discipline of artificial intelligence that uses the collective behavior of groups of social insects and animal societies as a source of inspiration to design algorithms and multi-agent systems. Some of these societies are ants, termites, bees and wasps, flocks of birds and schools of fish. Individuals of such societies individually can do only very simple tasks but collectively they can solve very complex tasks [7]. For instance, if we take the example of ants, in which we see that a single ant is not able to construct by itself a whole nest. However, collectively a group of ants can cooperate to build sophisticated nests. An ant colony can also, efficiently, find a food source by means of leaving pheromones in the way and thus identify the best path to the food source, by following the path with the stronger trail of pheromones. Beni and colleagues [4] used the term swarm intelligence in the 1980s, in a context of cellular robotic systems where simple agents organize themselves by means of nearest-neighbor interactions. After that, the term moved on to cover a wide range of studies from optimization to social insects studies as documented in the seminal book Swarm intelligence, From Natural to Artificial Systems by Dorigo, Theraulaz, and Bonabeau, published by Oxford University Press [8].

In the field of computer engineering, swarm intelligence has become a very important research field thanks to its various applications, for example in robotics [78], operational research [25], continuous optimization problems [50], and networking protocols [19, 20]. For instance, in optimization one of the most successful algorithms based on swarm intelligence is Ant Colony Optimization [26] which is inspired in the collective behavior of groups of ants that interact by leaving pheromones to communicate to other ants which path is the best one to reach the food source.

1.1 Swarm robotics

Swarm robotics is the application of swarm intelligence principles in robotics. It represents an approach to the coordination of a large number of robots whose main source of inspiration is the collective behavior of social insects such as ants, wasps, and termites. The feature in common among these insect societies is that individually the insects are able only of performing simple tasks but by cooperating they can

Figure 1.1: Social insects and animals [40].

accomplish relatively complex tasks. The emergence of collective behavior when applying swarm intelligence principles on multi-robot systems is rather impressive since it emerges despite the fact that individually, they are relatively incapable, despite the lack of centralized coordination and despite the simplicity of local interactions.

1.1.1 Definition

E. Shahin has defined swarm robotics as follows: "Swarm robotics is the study of how a large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among the agents and between the agents and the environment" [79]. In this definition, we can observe three key concepts: simplicity, collective behavior, and local interactions.

Simplicity promotes cooperation, since it means that the robots have limitations in their individual capabilities relative to the task they are trying to solve. The collective behavior emerges from the interaction among individuals, the environment and neighbors. There is no leader, which means that the swarm is self-organized and no external infrastructure interferes which means that the swarm is autonomous. Locality meaning that the limitation of interactions to a restricted local neighborhood allows the robot to perform independently from the actual swarm size. This in return makes the system scalable (w.r.t the swarm size). Additionally, robots can operate in a multi-task and parallel manner when they switch from task to task according to their environment or neighbors (conditions). In summary autonomy, self-organization, redundancy, locality, and parallel execution promote robustness, scalability, and flexibility in the system, which are system-level properties exhibited by a swarm robotic system. They are observed in natural swarms and remain as desirable properties of multi-robot systems.

- **Robustness**: Is the ability of the robot swarm system to operate despite environment disturbances and malfunctioning individuals. This can be done because swarms are redundant and no single-point-of-failure systems, the loss of an individual can be compensated by another one. Also, coordination among the robots is decentralized, so if part of the swarm is destroyed the other part could still operate and sensing is distributed making the system robust against local perturbations.

- **Flexibility**: Is the ability of the system to use different coordination strategies to tackle tasks of a different nature.

- **Scalability**: Is the ability of the system to increase the number of robots in the swarm impacting the performance positively or without impacting performance considerably. For this property it is important, for the coordination mechanisms and strategies developed, to take into consideration the operation of the swarm under varying swarm sizes.

1.1.2 Applications

Swarm robotics has multiple potential applications such as rescue missions, surveillance, exploration in risky areas (e.g. space missions), path-finding. For instance, environmental robots can be used as garbage collectors and pest eradication. Some other industrial applications include waste disposal and micro cleaners. Ship maintenance and ocean cleaning could be performed by swarms of underwater robots. In aerospace technology, teams of flying robots may perform satellite maintenance. In the aircraft industry, a swarm of robots can perform engine maintenance eliminating the need for costly disassembly for routine preventive maintenance. Some researchers envision microsurgical robots designed to perform specific manipulation tasks without the need for conventional surgical techniques. It is important to notice that small robots, micro, and nanorobots may need to operate in very large groups

Figure 1.2: Swarm robotics system [69]

or swarms to affect the macroworld. Approaches directly inspired or derived from swarm intelligence may be the only way to control and manage such groups of small robots. These applications are benefited from swarm robotics since all of them imply group of agents trying to solve missions and which have to cooperate together in order to achieve the objective (cleaning, maintenance, explore dangerous places, explore unreachable places). Also, when working with multi-agent systems, swarm robotics helps to solve redundancy and autonomy problems.

1.2 Control Software design for robot swarms

Control Software is the piece of software that allows the robot to use the input information to perceive its environment and allows it to perform actions (actuate). For instance, if we think of a cleaning robot within a house, the robot must identify what is garbage and then take it to the recycle bin. So, the robot physically has motors, wheels, and a mechanism to take objects, however, the robot needs a piece of software that allows him to decide when to use each of his parts. This is when the control software is necessary, so we can say that the control software is a piece of software that allows the robot to decide when to use its parts depending on the situation or environment. In the case of the cleaning robot, the controller has to identify that an object is a garbage, this can be done by means of a camera. Then it has to perform an action, which could be taking or not the object. If it detects that

it is garbage, the next action will be taking it to the recycle bin. To summarize the previous example, we can say that a robot has sensors (i.e. cameras) that feed the controller of the robot to make a decision and actuators (i.e. motors) that allow the controller sending signals for the robot to perform actions.

1.2.1 Control software representation

This piece of software can be logically represented in different architectures such as: sense-plan-act (SPA) [71], subsumption (Reactive) [13], Artificial Neural Networks (ANN) [27], probabilistic Finite State Machines (PFSM)[brooks1986robust] and Behavior Trees (BT) [63]. In swarm robotics for manual design it is common to use SPA and Reactive architectures while for automatic design of control software designers often use ANN, PFSM and BT. In ANN architectures, sensor's lectures are used as inputs and the outputs are mapped to actuators. In PFSM, behaviors are represented as states that contain actions, and robots can move from one state to another by means of conditions which comes from sensors lectures (environment and interaction with neighbors). In BT architectures a behavior tree is a tree structure that contains one root node, control nodes, and execution nodes (actions or conditions).

1.2.2 A challenging task

The control software design for a robot swarm is a challenging task since it is difficult to determine how these simple robots should be programmed to perform user-designed tasks from local interactions only. To tackle this problem there are two approaches: ad-hoc and principled. In ad-hoc approaches, behaviors of individuals are designed manually to achieve a desired swarm-level behavior. In principled approaches instead of designing a specific swarm-level behavior, a general methodology is proposed. An example of a general methodology is the use of artificial evolution, for example, feedforward or recurrent multi-layer perceptrons to encode behaviors. On the other hand, Francesca and Birattari proposed a novel general methodology in 2013 called AutoMoDe that stands for Automatic Modular Design [32]. This new approach is inspired in the technique used in machine learning for dealing with the bias-variance tradeoff and aims to automatically design control software for robot swarms from existing independent modules.

1.3 Stochastic local search

Nowadays stochastic local search is a well-known technique used in different fields, machine learning, operations research in general to solve hard-computing combinatorial problems. It is common to think of an artificial intelligence problem as a combinatorial optimization problem. For instance, we can think of a salesman who has to visit multiple cities only once and he wants to visit those in the most efficient way, where efficient means minimizing the traveled distance. This is a popular mathematics combinatorial problem called Traveling Salesman Problem (TSP) where

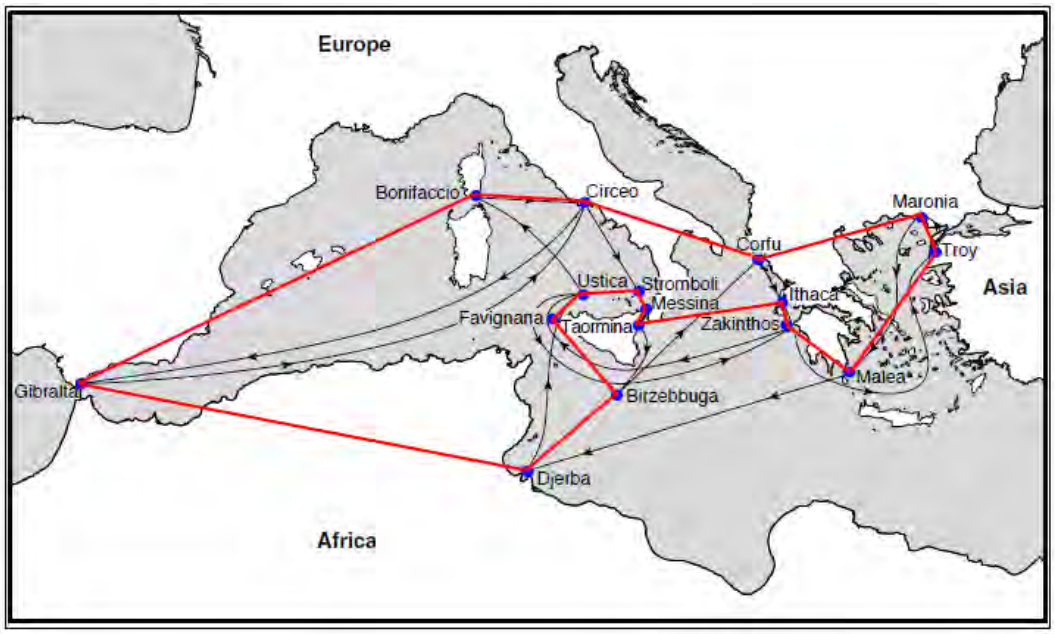


Figure 1.3: Traveling Salesman Problem [45]

the expected behavior of the computer, in order to solve the problem, is to explore all the possibilities, by combining groups of cities, and optimize the outcome. To tackle these kinds of problems, there are mathematical models based on constraints and objective functions that guarantees to find the optimal solution. However, these called "exact methods" are hard to solve for computers, in terms of time complexity, and when the search spaces are too big (in TSP when the number of cities increases [48]) it just becomes infeasible to find the optimal solution in a reasonable amount of time.

1.3.1 SLS for combinatorial optimization problems

Stochastic local search (SLS) uses the same model of constraint and objective functions but it applies probabilistic techniques to approximate good solutions. Meaning that a local search technique will try to find the optimal solution by interacting within a neighborhood (other feasible solutions) and move through the search space by selecting the most probable path to the optimal solution. It is important to mention that, in SLS algorithms moving into the neighborhood and the search initialization can be randomized.

When trying to tackle hard combinatorial optimization problems, stochastic local search offers from very simple techniques such as first improvement , best improvement to more sophisticated ones such as Simulated Annealing [52] Tabu search [36], Lin-Kernighan Algorithm [60], among others. In contrast to exact methods, local search technique can converge rather fast, even in big search spaces and approaches very closely to the global optimum solution. This fact makes local

search robust techniques, that can be adapted to different needs. However, one of the weaknesses of local search techniques is that, in general, they are incomplete, which means that they don't ensure to find the global optimal solution if there is one, instead they usually get trapped in a local optimum. However, some local search techniques (i.e. simulated annealing), are probabilistically complete meaning that if they run for enough time, they will find the global optimal solution, if there is one.

1.4 Overview

In this work, I explore a SLS algorithm as an optimization algorithm within AutoMoDe. I test the performance of this technique and compare it with the AutoMoDe Chocolate and Maple versions. The SLS technique implemented is Simulated Annealing which is a probabilistic complete algorithm that has the ability of escaping local optima. So, my hypothesis consists in observing if a SLS technique within an automatic off-line modular design approach can perform as good as current techniques for automatic design of control software for robot swarms. I compare the algorithm with other techniques and observe its robustness w.r.t the reality gap by testing the generated control software, in simulation and pseudo-reality. I also study the influence of different parameters on the performance of the optimization process.

Chapter 2. I start by deepening into state-of-the-art techniques to design control software for robot swarms, covering two different approaches, automatic modular design, and evolutionary techniques. I also present some works of Simulated Annealing in the context of robotics.

Chapter 3. I present AutoMoDe-Annealing, which is a method of automatic design based on AutoMoDe. The difference with other instances of AutoMoDe is that it uses simulated Annealing as optimization algorithm. I give details over the framework used to develop the method and how the automatic design is translated to an optimization problem that can be tackled by stochastic search techniques.

Chapter 4. I explain the settings of the experimental setup that includes a description of the robots, simulation environment, the missions and protocol used to assess the method.

Chapter 5. I present the results of the experiments I performed to assess the AutoMoDe-Annealing in two parts. The first part studies some parameters influence over the SA and the second part shows a comparison between AutoMoDe-Annealing, AutoMoDe-Chocolate, AutoMoDe-Maple and AutoMoDe-Iterated—Another instance of AutoMoDe with iterative improvement as optimization algorithm.

Chapter 6. Finally, I present the conclusions and future work in chapter 6.

Chapter 2

State-of-the-art

Swarm robotics has been mainly inspired by the observation of groups of social animals such as ant and bee colonies, flocks of birds and school of fish. These groups of animals exhibit a swarm intelligence: the individual actions of each animal leads swarm to exhibit a rather complex behavior. The distributed nature of the swarm that endows it with properties that are also desirable in artificial systems: swarms are robust, scalable and flexible [22]. In particular, those properties are relevant to the design of multi-robot systems. The study of robot systems that display properties of robot swarms has been defined as swarm robotics [23]. Despite the fact that swarm robotics has been largely studied in the past years [31], no well-defined methodology exists for the development of robot swarms. The need for engineering-based methods for the development of robot swarms was first introduced by Kazadi *et al.* [49] and further explored by Winfield *et al.* [88]. Brambilla *et al.* [10] give a complete definition of the concept of swarm engineering: it is the systematic application of scientific and technical knowledge in the requirement specification, modeling, design, realization, verification, operation and maintenance of swarm systems. My research is closely related to the concept of swarm engineering: I investigate the effectiveness of state-of-the-art optimization algorithms in the automatic design and realization of robot swarms. In this chapter, I present related work on the automatic design of control software for robot swarms and, in addition, relevant literature and applications in robotics of the optimization methods I consider in my thesis.

2.1 Control software design

The design of robot swarms refers to the planning and development of swarm systems on the basis of *a priori* known initial specifications. For example, a common idealized application for robot swarms is the planetary exploration [80]. In such case, a swarm might be required to explore the surface of the planet while taking samples of ground. Those specifications provide to the designer the basis on which to select the number and type of robots that are needed, as well as the appropriate exploratory collective behavior to achieve the mission. However, there is no mature or precise method to design the control software for the individual robots so that it will produce the desired

Figure 2.1: The design challenge in swarm robotics [24].

collective behavior in the swarm [11]. In most cases, the intuition and experience of the human designer is the key to the development of efficient robot swarms [31]. As an alternative to the manual design, recent work has been devoted to investigate the automatic design of robot swarms. Brambilla *et al.* [10] discuss literature on the approaches to the design of robot swarms, and present a taxonomy that divides the design methods into behavior-based design and automatic design.

2.1.1 Behavior-based design

The behavior-based design of robot swarms is mainly an iterative process of trial and error. The individual behavior of each robot is implemented, studied and improved until the desired collective behavior is obtained. In some cases, the design process benefits from previous studies on social animals that provide a mathematical model from which to predict the relationship between individual and collective behaviors [82, 35, 55]. As mentioned before, a promising alternative to behavior-based design is the automatic design of robot swarms. The control software for the individual robots is produced without the explicit intervention of the developer.

2.1.2 Automatic design

In automatic design, the problem of designing the control software is translated into an optimization problem: there is a search space comprising different design choices, and an optimization algorithm is used to explore it [6]. Another classification for automatic methods refers to the differences in the design and deployments environments. Under this criterion, they are classified into on-line and off-line methods. There are two main approaches to the automatic design of robot swarms: evolutionary robotics and modular design [31].

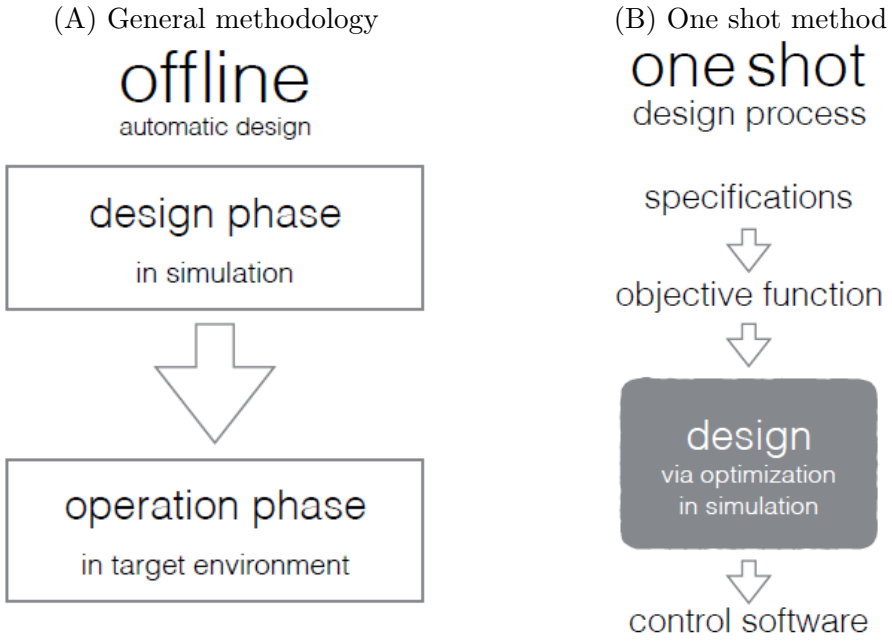


FIGURE 2.2: Off-line automatic design.

On-line methods

In on-line methods, the design process takes place when the robot swarm already has been deployed in its operational environment. On-line methods are commonly used under mission-specific studies and their main advantage is that they facilitate the adaptation of robot swarms to changes in their work-space [12, 39]. A relevant study that proposes an on-line method for multi-robot systems was published by Parker [74]. The author proposed L-ALLIANCE, a behavior-based control architecture that allows the on-line optimization of internal parameters of control software. Matarić [65] surveyed various behavior-based control architectures that allow the on-line optimization of parameters. Lee and Arkin [56] applied learning momentum [18] to a multi-robot system.

Off-line methods

On the contrary, off-line methods design robot swarms before they are deployed in their operational environment. The design process iteratively considers, evaluates and discards different instances of control software. In order to support this process, the evaluation of the control software is performed in computer-based simulations—achieving then faster evaluations and preventing damage of the robots. This work is related to off-line methods which is why I will describe relevant off-line methods in the following section. On-line and off-line methods have different characteristics which make them suitable in different situations. First, on-line methods profit from the information on the actual operational environment and can be fine-tuned to

operate there. Off-line methods rely on the representation of the environment that often is provided by a simulator. Second, on-line methods are constrained to design control software over a reduced design space. The design process is performed by the robots themselves and often the computational resources are limited [87]. Moreover, designers must discard potentially dangerous control software from the design space to reduce damage risk. On the other hand, off-line methods operate *a priori* and under fast simulations [6]. Hence, they can afford larger design spaces. Finally, on-line methods are fully distributed processes. The robots cannot count on any centralized entity to measure performance and guide the search—they rely on their inner and collective experience to evaluate the control software [76]. In off-line methods, the search can be guided by global information acquired within the computer-based simulations [75].

2.2 Evolutionary robotics

Evolutionary robotics [72] is an approach to the automatic design, that applies evolutionary computation—Artificial principle of natural selection and evolution—techniques to single [37, 44] and multi-robot systems [9, 21, 85, 81]. Neuro-evolution is the most studied automatic design approach in swarm robotics. It uses an evolutionary algorithm to optimize the parameters of an artificial neural network that takes as an input sensor readings, and returns actuation commands. The application of the evolutionary robotics approach in swarm robotics is known by the name of evolutionary swarm robotics [84]. Some of the important characteristics shared in works related to evolutionary swarm robotics are: (i) All the robots of the swarm execute identical copies of the same control software; (ii) During the design process, the objective function is globally and centrally evaluated. It is computed, evaluating the performance of the swarm as a whole; (iii) The optimization algorithm used within the design process is an evolutionary algorithm that features elitism, recombination and mutation operators. Most of the research is aimed to prove the feasibility of the approach, if a particular collective behavior can be obtained via artificial evolution.

2.2.1 Evolutionary swarm robotics

The firsts in adopting an evolutionary approach in the context of swarm robotics were Quinn *et al.* [77], they obtained a coordinated motion behavior by using an evolutionary algorithm to optimize control software based on a neural network. After this work, several robot swarms designed via evolutionary robotics have been described in the literature. For example, Christensen and Dorigo [17] showed that using evolution is possible to obtain a swarm of robots that is able to perform simultaneously, hole-avoidance and photo-taxis. Under similar settings, Baldassarre *et al.* [3] obtained coordinated motion with a swarm of four physically connected robots. Hauert *et al.* [43] used evolutionary robotics to obtain the control software for a swarm of aerial robots that are required to establish a wireless communication network. Trianni and Nolfi [85] studied the design of self-organizing behaviors via evolution.

Waibel *et al.* [86] investigated the influence on the performance of different selective pressures (individual and collective) and different team compositions (homogeneous and heterogeneous swarms). Ferrante *et al.* [30] used a method based on grammatical evolution to evolve task specialization in a robot swarm. In particular, the authors highlighted the environmental conditions that are necessary for task specialization to emerge.

The research in evolutionary swarm robotics has been influenced by the current trends in evolutionary computation, for example, studies on novelty search [57] promote diversity alongside performance. Results show that novelty search is robust to issues that impact the classical evolutionary formulation, including early convergence and stagnation. Gomes *et al.* [38] introduced novelty search in the context of swarm robotics. They used novelty search to automatically develop control software for aggregation and resource sharing.

2.3 Modular design

In recent years, modular control architectures have been proposed as alternative to neuro-evolution. The control software is obtained by combining modules either via manual or automatic design. Duarte *et al.* [28] presented an approach based on the hierarchical decomposition of complex behaviors into simpler behaviors. The simple behaviors are generated manually or via evolutionary robotics to later on be combined using high-level neural networks. The results—simulation only—indicate that the approach outperforms the classical, monolithic evolutionary approach. Similarly, Duarte *et al.* [29] used hierarchical decomposition to obtain control software for a patrolling task. The control software comprises low-level behaviors and one module that arbitrates the low-level behaviors. The low-level behaviors were implemented as neural networks and were obtained via evolution. The arbitrator was manually written. The downside of the approach is that hierarchical decomposition is task-dependent and has to be done manually by the designer.

2.3.1 AutoMoDe

Francesca and Birattari proposed a novel general methodology [32] in 2013 named AutoMoDe: Automatic Modular Design. This new approach is inspired in the technique used in machine learning for dealing with the bias-variance trade-off, which consists of injecting an appropriate bias to obtain suitable general solutions with low variance. AutoMoDe produces control software for robot swarms by selecting and combining preexisting modules (the bias) in the form of a probabilistic finite state machine (PFSM). What AutoMoDe-Vanilla—the first release of AutoMoDe—uses the optimization algorithm F-RACE [5] to build control software in the form of finite state machines. Th PFSMs is indeed, the architecture of the control software.

The performance of AutoMoDe-Vanilla, was assessed against three other different techniques methods. Evostick [59], a method based on neuro-evolution and considered state-of-the-art for control software design; U-human, that is unconstrained humans that can use a programming language to develop the control software ; and C-human,

that is constrained humans that could only use some predefined modules and combine them to develop a control software. The results of this study showed that AutoMoDe performed better than U-humans and Evostick but not yet than C-humans. The conclusion was, that F-Race was not powerful enough, and it could be improved to get better results with AutoMoDe. In consequence, AutoMoDe-Chocolate—The second instance of AutoMoDe—replaced F-Race [61] with Iterated F-Race [61] (I-Race). I-Race is an improved version of F-Race , that when used in AutoMoDe, allowed *Chocolate* to outperform *Vanilla*, U-humans and C-humas.

Other researchers have explored further the capabilities of AutoMoDe. Kuckling *et al.* [53] proposed AutoMoDe-Maple which is a variant of AutoMoDe where they explore the possibility of adopting behavior trees as an architecture for the control software of robot swarms. The results of their experiments show that behavior trees are viable and promising architecture to automatically generate control software. Hasselmann *et al.* [42] present an automatic design method that generates communication-based behaviors for robot swarms. AutoMoDe-Gianduja extends AutoMoDe Chocolate. Endowing the robots with the capability of transmitting messages.

2.4 Simulated Annealing

The Simulated Annealing search is an analogy [66] to the state of physical systems composed of particles in statistical mechanics. At high temperatures, the particles are less unbounded to move, and the structure is subject to significant changes. The temperature decreases over time, and with it the probability for a particle to move, until the system reaches a state of lowest energy, its ground state. These ideas were turned into heuristics methods for tackling combinatorial optimization problems by Kirkpatrick *et al.* [51] , and independently V Černý (1985) [15]. The physical temperature is translated into a "temperature" parameter, the state of the physical system corresponds to a candidate solution, the ground state corresponds to the globally optimal solution, and a change of state corresponds to a move to a neighbouring candidate solution.

In few words, Simulated Annealing is a stochastic local search algorithm, inspired by the work of Metropolis *et al.* [66], that starts from an initial solution and it iteratively explores the neighbourhood of the current solution. The main characteristic of Simulated Annealing is that it always accepts improving solutions and also worsening solutions probabilistically depending on the decline of quality in the solution and a parameter called temperature.

2.4.1 Simulated Annealing work

Simulated Annealing is one of the oldest and most studied metaheuristics. In fact, Alberto Franzin and Thomnas Stutzle [34] they found more than 6000 articles with the keyword "Simulated Annealing" in the title. Simulated Annealing is a high performing heuristics for many problems [1, 70]. In robotics, Simulated Annealing has been mostly used to solve path planning and obstacle avoidance problems. Park *et al.* [73] implemented Simulated Annealing algorithm as a technique for escaping

local minima in local and global path planning. Martınez-Alfaro *et al.* [62] used the simulated annealing algorithm to obtain a collision-free optimal trajectory among fixed polygonal obstacles. Janabi-Sharifi *et al.* [47] proposed an integrated approach to local and global path planning of robots in stationary environments. Miao *et al.* [67] they enhanced a standard Simulated Annealing algorithm. They improved the computing performance of the Simulated Annealing implementation significantly. The Simulated Annealing implementation achieved optimal or near-optimal robot path solution, making its real-time and on-line applications possible. Tavares *et al.* [83] studied the sensitivity of each SA continuous parameter in the context of a robot path planning problem. Miao and Tian [67] propose a simulated annealing based approach to determine the optimal or near-optimal path quickly for a mobile robot in dynamic environments with static and dynamic obstacles.

To the best of my knowledge Simulated Annealing has not been used in the context of swarm robotics. In particular, there is no literature that refers to Simulated Annealing for the automatic design of control software for robot swarms.

2.5 Contribution

I presented promising methods for the automatic design of control software for robot swarms, however consolidated literature on the topic is still missing. The majority of the published studies do not present comparisons between different methods. Indeed, some fundamental questions are hardly addressed in the current literature: Which automatic method is the best under which conditions? How general is method X? How well does method X perform on different tasks? These kind of questions are fundamental for the development of a mature science. For that reason in this work, I use a different optimization algorithm—Simulated Annealing—, within AutoMoDe. Simulated Annealing has not been tested in the automatic design context so far, but has shown good results for many other combinatorial optimization problems. Simulated Annealing has been also used for the path-planning problem in the context of robotics. I use Simulated Annealing to build a new variant of AutoMoDe—AutoMoDe-Annealing—, so I can compare against AutoMoDe-Chocolate, AutoMoDe-Maple and AutoMoDe-iterated. I also study the performance of the method in two different missions, to test its generalization property. I test this, by means of an empirical study for which I give more detail in chapter 5.

Chapter 3

AutoMoDe

An automatic design method is an iterative process supported by an optimization algorithm that explores a set of control software candidates searching for the best solutions for a given task. The control software candidates are sampled from the set of all possible controllers which is the software control search space. Each control software candidate is evaluated using an objective function which gives a metric on the quality of the control software. In other words, the quality of the control software represents its efficacy in solving given tasks when instantiated in a robot swarm. Commonly, the evaluation of the objective function is performed using computer-based simulations of the chosen robotic platform and the experimental setting in which the final system will operate. The objective function guides the optimization algorithm by selecting the best candidates for the next iterations, in the search for the best control software solutions. Usually, at the end of this process, the obtained control software is instantiated and tested using real robots.

In this chapter I describe the components of the automatic design process, specifically of AutoMoDe. I focus then in the two important components for the development of this work: The control software search space and the optimization algorithm. Finally I describe the Python extension of AutoMoDe that I used to develop my solution, AutoMoDe-Annealing.

3.1 Components of the automatic design process

The process of an automatic design method can be divided in 6 components: **control software search space**, **objective function**, **optimization algorithm**, **reference model** and **scenario**.

Control software search space. It represents the space of all the possible controllers that the automatic design method can generate. A controller is composed of a pre-defined architecture chosen by the human designer, and a set of design parameters.

Objective function. It is a mathematical function used to evaluate the performance of the swarm in a specific task. It plays a critical role since it guides the search process of the optimization algorithm.

Optimization algorithm. It explores the control software search space for suitable controllers to solve the given task. It terminates the search either by meeting a threshold on the objective function metric or after exhausting a pre-defined number of evaluations.

Reference model. The reference model includes the he specific robot used in the swarm. But is not limited to the robot itself but it comprises the capabilities of the robots. That means that in the automatic design, the method designs control software for a group of robots with the same characteristics— behaviors and conditions. The capabilities of the robots influence the method since different capabilities result in different behaviors. Since the design process is done in simulation it is necessary to have a simulated version of the reference model. However, there is an important limitation when working in simulation, which is the reality gap [64, 46]. The reality gap is the difference between reality and the simulation models used in the automatic design process. There is a recent study [58] arguing that it is unnecessary to assume reality is more complex than simulation models for the effects of the reality gap to occur. they show that performance drop and rank inversion can occur if one automatically designs control software in simulation using a model and then assesses it in simulation on another model that hey call a pseudo-reality.

Scenario. It represents the environment in which the swarm operates and which has to be carefully replicated in the simulation tool. It includes the characteristics of the environment such as the size and the geometry of the environment, among others. AutoMoDe-Vanilla, AutoMoDe-Chocolate and AutoMoDe-Maple as automatic design methods, use the same tools for some elements and differs in others.

3.1.1 Common components among instances of AutoMoDe

I first describe the common elements between AutoMoDe-Vanilla, AutoMoDe-Chocolate and AutoMoDe-Maple to then point out to the differences among them. This is the list of common elements among the three aforementioned instances of AutoMoDe:

- **Reference model**: The three instances of AutoMoDe use an extended version of the e-puck [68] robots for which I give more details in chapter 5, see Table 5.1. In the three AutoMoDe instances, the robot swarm is simulated using ARGoS [75], a multi-engine simulator of swarm robotics systems I give more details in chapter 5.
- **Objective function**: This is task-dependent, no method-dependent, and it is the same function across the variants when comparing them.

- **Scenario**: These are the same when comparing the AutoMoDe variants but are independent to the method.

3.1.2 Distinct components

In section 3.2 and 3.3 I explain the elements that are different in the three aforementioned instances of AutoMoDe. The distinct components are: the search space and optimization algorithm.

3.2 Control software search space

AutoMoDe-Vanilla, AutoMoDe-Chocolate and AutoMoDe-Maple work with preexisting constituent behaviours and preexisting conditions. In fact, the three of them work with the same six preexisting low-level behaviors and six conditions. The following description is provided as it was in the original paper.

Low-level Behaviors. (i) **Exploration** is a random walk strategy. The robot goes straight until an obstacle is perceived by the front proximity sensors. Then, the robot turns on the spot for a random number of control cycles drawn in $\{0, \ldots, \tau\}$, where τ is an integer parameter $\in \{0, \ldots, 100\}$. (ii) **Stop** orders the robot to stay still. (iii) **Phototaxis** moves the robot towards a light source. If no light source is perceived, the robot goes straight. (iv) **Anti-phototaxis** moves the robot away from the light source. If no light source is perceived, the robot goes straight. (v) **Attraction** moves the robot in the direction of the neighboring peers. The speed of convergence towards the detected peers is controlled by a real parameter $\alpha \in [1, 5]$. If no peer is detected, the robot goes straight. (vi) **Repulsion** moves the robot away from the neighboring peers. The real parameter $\alpha \in [1, 5]$ controls the speed of divergence. Obstacle avoidance is embedded in all low-level behaviors, with the exception of stop.

Conditions. (i) **Black**-, (ii) **gray**- and (iii) **white- floor** are true with probability $\beta \in [0, 1]$ if the ground sensor perceives the floor as black, gray, or white, respectively. (iv) **Neighbor-count** is true with a probability computed as a function $z(n) \in [0, 1]$ of the number of robots detected via the range-and-bearing board. A real parameter $\eta \in [0, 20]$ and an integer parameter $\xi \in \{0, \ldots, 10\}$ control the steepness and the inflection point of the function, respectively. (v) **Inverted-neighbor-count** is true with probability $1 - z(n)$. (vi) **Fixed-probability** is true with probability $\beta \in [0, 1]$. The parameters β, η and ξ must be tuned by the automatic design process.

3.2.1 Control Software architecture

AutoMoDe-Vanilla and AutoMoDe-Chocolate use the same architecture for the control software which is Probabilistic Finite State Machine (PFSMs). On the other hand, AutoMoDe Maple uses behavior trees as architecture. The search space then,

Low-level behavior	Description
EXPLORATION	random walk movement
STOP	stay still state
ATTRACTION	physics-based attraction to neighboring robots
REPULSION	physics-based repulsion from neighboring robots
PHOTOTAXIS	moves the robots towards the light
ANTI-PHOTOTAXIS	moves the robots against the light

Transition condition	Description
BLACK-FLOOR	black floor under the robot
GRAY-FLOOR	gray floor under the robot
WHITE-FLOOR	white floor under the robot
NEIGHBOR-COUNT	number of neighboring robots grater than η
INVERTED-NEIGHBOR-COUNT	number of neighboring robots lower than η
FIXED-PROBABILITY	transition with a fixed probability

Table 3.1: Summary table of Behaviors and conditions.

is constituted by the behaviors, conditions and the architecture used to represent the control software for the robot swarm.

Probabilistic Finite State Machines

PFSMs are composed of atomic behaviors linked by conditional state transitions. Both the atomic behaviors and the conditional state transitions use the sensors and actuators of the e-puck. In figure 3.1 we can observe an example of PSFM obtained with AutoMoDe. For more details I refer the reader to the original paper [33].

Behavior Trees

The BT architecture is a tree structure that contains one root node, control nodes and execution nodes which are the actions and conditions. This architecture aims to enhance expressiveness and structural modularity. Modularity can simplify the implementation of optimization algorithms based on local manipulations. It could also allow pruning unused part of the tree to increment readability. One more advantage is that a sub-tree could be optimized independently of each other and used then used to form another tree that represents a more complex behavior. In figure 3.2 We can observe an example of BT obtained with AutoMoDe Maple. For more details I refer the reader to the original paper [53].

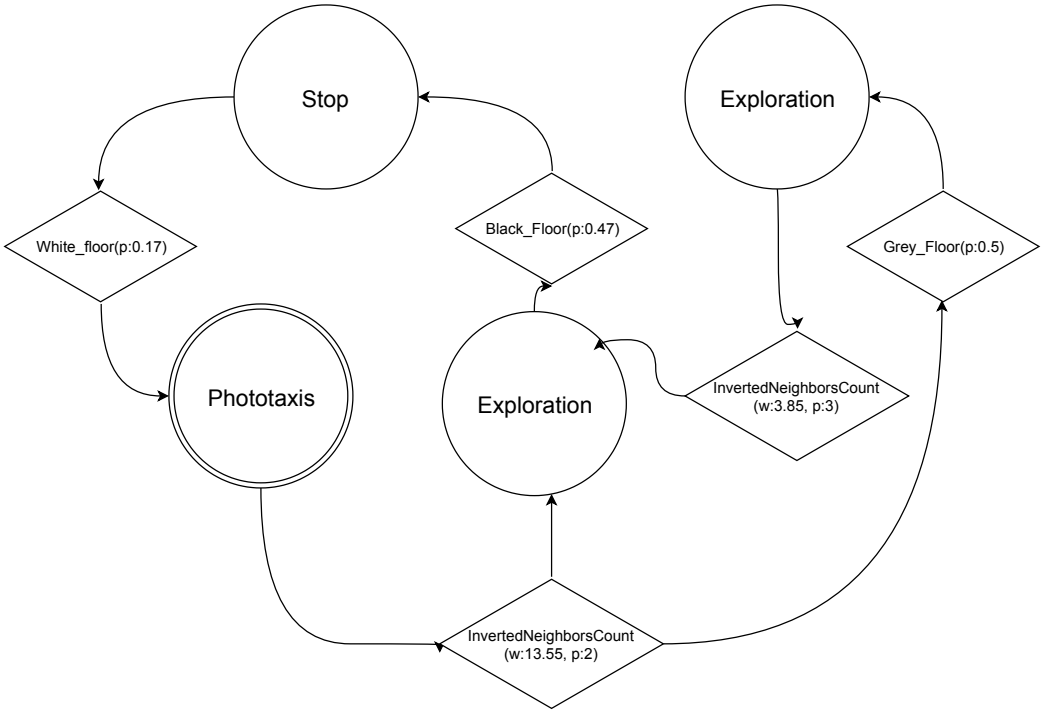


FIGURE 3.1: Probabilistic Finite State Machine

Constraints

Both PFSMs and BT are limited in the number of states and nodes that can be used in a single controller. PFSMs is limited to up to four states, and up to four outgoing transitions per state. BT is restricted to a fixed top-level sequence* node that has up to four subtrees as children. Each subtree contains a selector node with exactly two children: one condition node and one action node. Kuckling *et al.* [54] shows that the search space for PFSMs in Chocolate is bigger than the search space for BT, even though the behaviors and conditions are exactly the same.

3.3 Optimization algorithm

As I explain in subsection 3.2 the control software design consists of a set of behaviors and conditions Table 3.1, and the control architecture (PFSMs-Chocolate and BTs-Maple). First, I describe the optimization problem as follows.

For AutoMoDe-Vanilla and Chocolate, the problem consists in finding the best PFSM that meets the constraints in the number of states and transitions, and maximizes/minimizes an objective function. Where the PFSM represents the control software, therefore a solution. Given the constraints, the PFSM should meet them to be a feasible solution. The objective function is the metric used to measure the robot swarm in a given mission. For AutoMoDe Maple, the problem consists in

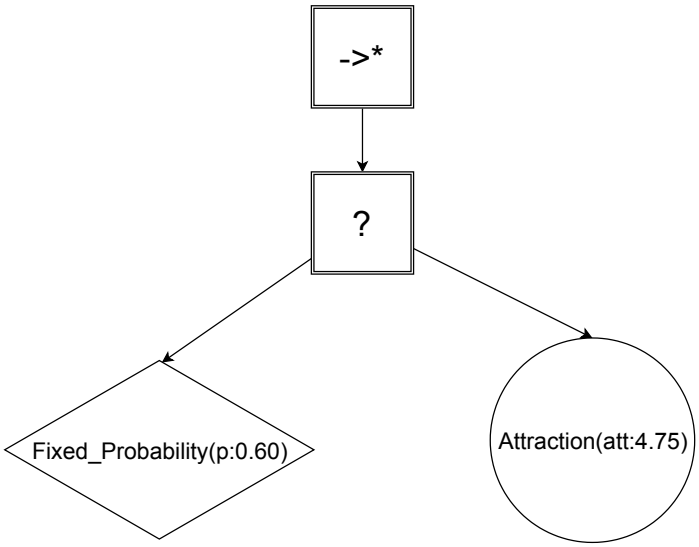


Figure 3.2: Behavior tree architecture

finding the best BT that meets the constraints in the number of nodes and levels, and maximizes/minimizes an objective function. Where the BT represents the control software, therefore a solution. Given the constraints, the BT should meet them to be a feasible solution. The objective function is the metric used to measure the robot swarm in a given mission.

AutoMoDe Vanilla was implemented with F-Race [5] as optimization algorithm, which is a racing algorithm for tuning meta-heuristics. F-Race seemed to be appropriate in the context of control software for tobot swarms for its ability to handle stochacity in the evaluation candidates. Later on, Francesca *et al.* [32]. realized that the optimization could be improved to obtain better control software. In consequence, they substituted F-Race with Iterated F-Race in Chocolate [2].

Maple only defers from Chocolate in the control software architecture, thus it also uses Iterated F-Race as optimization algorithm. In Iterated F-Race, the optimization procedure consists of a series of iterations. Each iteration is an execution of the F-Race algorithm. Firstly, an initial set of candidate solutions - control software designs - is generated by sampling the space of feasible solutions in a uniformly random way. The initial control software candidates are evaluated in first execution of the F-Race algorithm. Once the F-Race algorithm is finished, the surviving controllers - the controllers that were not discarded in the optimization process - are used as an initial point to generate a new set of control software candidates on

which the following iteration will operate. The new set of candidates is obtained by sampling the space of feasible solutions according to a distribution that gives a higher probability of being selected to solutions that are close to the surviving solutions. The new set of control software candidates is evaluated in a further execution of F-Race. The process is iterated and stops when a budget of evaluations, set by the designer, has been performed.

3.4 AutoMoDe Python extension

For implementing AutoMoDe-Annealing I worked on top of a framework built by Kuckling. He has participated in the development of AutoMoDe Maple [53] and is currently investigating local search strategies to improve the optimization process in automatic design of control software for robot swarms. He implemented an iterative improvement algorithm to test AutoMoDe with a local search technique as optimization algorithm. For that, he used Python as programming language. However, all the simulation environment is simulated with ARGOS [75], which is implemented in C++. ARGOS works with plugins so, the e-puck robots [68], AutoMoDe-Chocolate and AutoMoDE-Maple are implemented as ARGoS plugins. What he did is isolate the local search implementation into Python and he wrapped the ARGoS modules to use them for the local search process. In the first instance of AutoMoDe the authors also isolated the optimization process in the programming language R.

3.4.1 ARGoS plugins

First of all, it is better to know how AutoMoDe is implemented and how it works. AutoMoDe is implemented as plugins and it uses the ARGoS version 3 as simulation engine. Specifically, we use ARGoS3-beta48 to simulate the swarm of robots. The ARGoS3-epuck v48, is a plugin for ARGoS that implements the e-puck robot. The epuck robot is the robot platform in used in AutoMoDE. Together these two programs form the environment that we use to perform the simulations. The focus of them is to provide the performance of a robot swarm in a specific mission, as quick as possible but accurately. Basically, this is where the reality is simulated. The results should be as close to reality as possible. AutoMoDe-FSM (Chocolate) and AutoMoDe-BT (Maple) are the ARGoS plugins that contain the control architecture for AutoMoDe. These plugins use the epuck-DAO, which is an abstraction layer that gives limited but focused access to the sensors and actuators of the e-puck robot. The AutoMoDe plugins also use other module called loopfunctions. The loopfunctions plugin is in charge of setup and control the simulated missions. Together these implementations form what the developing environment to develop or extend AutoMoDe.

3.4.2 Python setup

Originally the optimization process in AutoMoDe was executed in R. For the instance of AutoMoDe presented in this work I use a Python extension that wraps the

Table 3.2: Framework components. This is the software with its versions and description, used as a framework to develop or extend AutoMoDe.

Software name	Description	Version
ARGoS	Physics-based simulator designed to simulate large-scale robot swarms.	3-beta48
ARGoS-epuck	Plugin that enables the use of the E-puck robot in ARGoS.	3-epuckv48
epuck-DAO	An abstraction layer that gives access to the sensors and actuators of the epuck.	2019-Thesis-DesignByOptimization
AutoMoDe-FSM	AutoMoDe Chocolate module that contains PFSMs as control architecture.	2019-Thesis-DesignByOptimization
AutoMoDe-BT	AutoMoDe Maple module that contains PFSMs as control architecture.	2019-Thesis-DesignByOptimization
loopfunctions	Implementations that set up and control the missions (i.e. final scores).	2019-Thesis-DesignByOptimization
Python	Programming language used to implement local search strategies.	3.4

environment described in subsection 3.4.1. The Python program, for local search, creates a representation for a controller, PFSMs for Chocolate and BT for Maple. It has no knowledge about the implementation of the individual ARGoS modules and is working blindly on the design of the control software. It works blindly because the local search has no insight on how the modules work, the mission or anything else. That, allows to simply import a different Python file with modules or set a different mission and the local search will behave the same. Yet, the local search needs some information about the quality of the control software. For evaluating the control software quality, it passes the control software to the implementation of AutoMoDe in ARGoS and AutoMoDe returns a single value—given by the loopfunctions module. The control software is actually a representation that in this case depends on the version of AutoMoDe. The value returned is the performance metric that corresponds to the value obtained by evaluating the objective function.

Chapter 4

AutoMoDe-Annealing Implementation

AutoMoDe-Annealing is a new instance of AutoMoDe that uses as simulated annealing as optimization algorithm. The idea behind using simulated annealing is to validate if a local search technique can perform as good as state-of-the-art methods of control software design for robot swarms. This instance of AutoMoDe is able to handle finite state machines and behavior trees as control software architecture. In this chapter I explain the algorithm used for the implementation of simulated annealing as optimization algorithm for AutoMoDe. I give an explanation of each the simulated annealing components. Later, I explain how the algorithm works in the context of automatic off-line design. Finally I retake the components of simulated annealing to describe their implementation in this work.

4.1 Simulated annealing

As stated before in chapter 2, simulated annealing is a stochastic local search algorithm with the ability to escape local optima. SA is characterized by the possibility of probabilistically accepting worsening moves. The acceptance criterion used by simulated annealing is often the Metropolis condition (Kirkpatric *et al.* [52], Metropolis *et al.* [66]), which always accepts a neighboring candidate solution when it is better or equal to the current solution.This criterion also accepts a worse neighboring candidate solution with the following probability:

$$\exp\left(-\Delta(s', s)/T\right) \tag{4.1}$$

Where $s \in S$, s is a candidate solution in the set S of all possible candidate solutions, and $f : S \to \mathbb{R}$ is the objective function. $f(s)$ is the objective function value of candidate solution s. $\Delta(s', s)$ is the objective function difference of two candidate solutions s and s', and T is the parameter temperature. A worsening solution is accepted with a probability that depends on both the amount of worsening $\Delta(s', s)$ and T. If $\Delta(s', s)$ is small, there is no big difference in solution quality and a solution is more likely to be accepted when the temperature is high - which typically happens

at the beginning of the search. When the temperature is low, which usually happens towards the end of the search, improving candidate solutions are prioritized. However, if $\Delta(s', s)$ is big enough and close to the temperature value, accepting a worse solution will be less likely to happen.

The temperature controls the transition from an initial exploratory behaviour to a final exploitative one. Selecting high enough temperature values will end with, the algorithm failing to converge towards good solutions, and for too low temperature values, the probability of getting trapped in sub-optimal regions is higher. It loses the missing chance to escape from local optima solutions.

4.2 Component-based simulated annealing

Franzin and Stützle [34] decompose the simulated annealing algorithm in nine different components: Initial Solution, Neighborhood, Initial Temperature, Stopping Criterion, Acceptance criterion, Exploration criterion, Temperature Length, Cooling Scheme and Temperature Restart. The algorithm is shown in Algorithm 1. The Initial

Algorithm 1: Component-based simulated annealing algorithm

```
Data: Problem instance π
Input: Neighborhood N for the solutions, Initial Solution s_0
Output: Best solution s* found during the search
best solution s* := incumbent solution ŝ := s_0;
i := 0;
T_0 := initialize temperature according to Initial Temperature;
while Stopping Criterion is not met do
    choose a solution s_{i+1} in the Neighborhood of ŝ according to Exploration
     Criterion;
    if s_{i+1} meets Acceptance Criterion then
        ŝ := s_{i+1};
        if ŝ improves over s* then
            s* := ŝ;
    if Temperature Length is met then
        update temperature according to Cooling Scheme;
    reset temperature according to Cooling Scheme;
    i := i + 1
return s*
```

Solution and the Neighborhood are problem-specific components, and are taken as the input of the algorithm. The other seven components, which are directly related to the algorithm, are described in the following list. The decomposition of the SA algorithm in these nine components is a key representation to better understand the algorithm, and for identifying which components are potentially more relevant to the automatic modular design. Therefore, I used this structure to implement my own

Table 4.1: Simulated annealing components

Component	Description
Initial solution	Problem-specific
Neighborhood	Problem-specific
Initial temperature	Starting value of the temperature parameter.
Stopping criterion	Determines when the execution is finished.
Exploration criterion	Chooses a solution in the neighborhood.
Acceptance criterion	Determines if a new solution replaces the incumbent solution.
Temperature lentgth	Indicates if the temperature is updated.
Cooling scheme	Updates the temperature value in each iteration.
Temperature restart	Resets the temperature to its initial value, or another higher value

version of Simulated Annealing that works as optimization algorithm for AutoMoDe Chocolate and Maple. I give more details of the components in the section 4.4.

4.3 Optimization process

Inspired by Algorithm 1 the execution of SA for AutoMoDe works as follows. SA starts by taking as input the Initial Solution, the Neighborhood, a problem instance π. The initial solution is a BT or FSM that models the stop behavior. The neighborhood is defined by perturbation operators and the instance is represented by the mission that the robot swarm is attempting to solve. Then SA proceeds by initializing its internal status, in particular setting a value for the initial temperature. In this implementation is a fixed value.

Starting from the initial solution, SA iteratively selects one candidate solution in the Neighborhood according to the Exploration Criterion. A neighbor would be the current controller after randomly applying a perturbation operator. The new candidate solution is evaluated against the incumbent candidate solution using the acceptance criterion—Metropolis condition; if it also improves over the best solution found so far—global-best controller—, it becomes the new global-best candidate solution. The Temperature Length determines whether the temperature parameter has to be updated. In this case by default is 1, which means that the temperature will change in every iteration. If the temperature length indicates that the temperature should be updated, the Cooling Scheme sets the temperature to its new value. For this implementation, the Cooling Scheme is set by the experimenter before start the run and it stays constant along the optimization process.

To favour a new phase of exploration, the Temperature Restart scheme controls whether the temperature should be reset to a higher value. At each iteration, the Stopping Criterion is checked. The algorithm will stop after have exhausted a budget of simulations. If the criterion is met the algorithm terminates returning the best candidate solution. The best candidate solution is the best control software obtained in the design process.

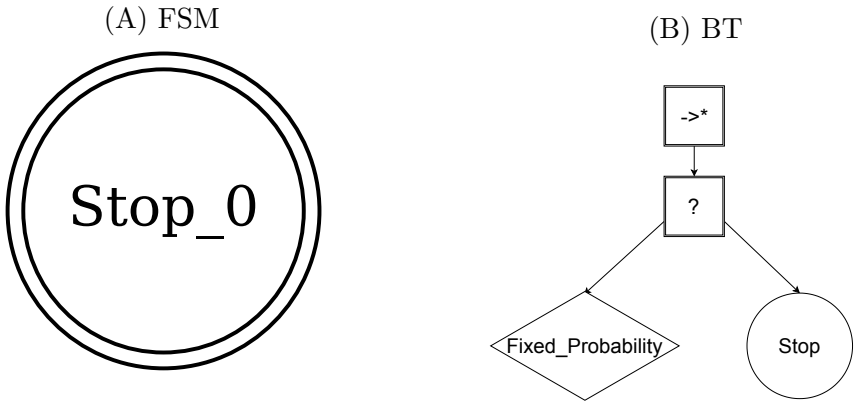


Figure 4.1: Initial solutions for AutoMoDe Chocolate (A) and Maple (B).

4.4 Implementation

I used the simulated annealing component-based algorithm, so I could try some variants of simulated annealing with different components and observe from which one, automatic design could get more benefit. Concretely, what I implemented is a simulated annealing algorithm that aims to study the optimization process of automatic design of control software for robot swarms. In the following subsection I describe the implementation of each simulated annealing component. See Table 4.1 to remember the components.

4.4.1 Initial Solution

The initial solution to start the algorithm is the minimal controller. The minimal controller is the representation of the stop behavior with a fixed probability in PFSM for Chocolate and BT for Maple as it is shown in Figure 4.1.

4.4.2 Neighborhood

There is no an explicit neighborhood, instead the neighborhood is defined by perturbation operators. The perturbation operators for PFSMs are listed in Table 4.2 and for BTs in Table 4.3.

Just to recall, there are some restrictions when constricting either a BT or PFSM. These restrictions has to be met in order to have a feasible solution in the neighborhood. For behavior trees: Each behavior tree that is composed of a sequence*-node with 1-4 children. Each of these children is a selector node with exactly two children, a condition node and an action node (in that order). For PFSMs: Each finite state machine with up to 4 states and 16 transitions such that no state has more than 4 outgoing transitions, but also that every state has at least 1 outgoing transition.

Table 4.2: PFSM perturbation operators.

Operator	Description
Change initial state	Changes the initial state of the PFSM.
Add state	Add a new state to the PFSM.
Remove state	Removes an state from the PFSM.
Change state behavior	Swap a randomly-selected behavior with other behavior
Change state behavior parameter	Change a randomly-selected parameter of a state
Add transition	Adds a new transition to the PFSM.
Remove Transition	Removes a new transition to the PFSM.
Change transition begin	Change the starting state of a randomly selected transition.
Change transition end	Change the ending state of a randomly selected transition.
Change transition condition	Swaps the condition of a randomly selected transition
Change trans/condition parameter	Changes a parameter of the condition of a transition.

Table 4.3: BT perturbation operators.

Operator	Description
Add sub-tree	Add a new action/condition sub-tree.
Remove sub-tree	Removes a sub-tree randomly selected.
Change sub-tree order	Randomly select and moves a sub-tree to a new position.
Change action-node behavior	Changes an action-node behavior randomly.
Change action-node parameters	Changes an action-node parameter.
Change condition-node condition	Swaps the condition of a random condition-node.
Change condition-node parameters	Changes a single parameter for one condition.

Additionally, no transition is allowed to be pointing back into the same state it originated from. This leads to the special case of a PFSM with only one state. In this case, it will have no transitions. The graph described by the states and transitions needs to be at least weakly connected.

4.4.3 Initial temperature

For this implementation of SA the initial temperature will use the simple option, a *fixed value* $T_0 = k$. I made this design decision because it seems that this parameter has not such a big impact on the performance [34], so I could focus on more relevant parameters.

4.4.4 Stopping criterion

I have also implemented various stopping criteria. Even though for the experiments, I only use the **Budget** stopping criterion.

- **Time termination**. It terminates after a fixed amount of time. It should be set before starting the run.
- **Discount termination**. It terminates when the temperature has reached a given value. It should be set before starting the run.
- **Budget termination**. It terminates when a given number of simulations have been performed.

4.4.5 Exploration criterion

The exploration criterion is mostly random exploration, but is restricted by the number of perturbation operators implemented, see Table 4.2, 4.3. The exploration strategy for this implementation has two components: perturbation operators and a moving window of random seeds.

Perturbation operators. In each iteration, one perturbation operator is randomly chosen and applied to generate a neighbor solution. There is a list to keep track of the operators that have been already applied to generate neighbors. So if the operator is in the list another operator is chosen. When all the operators have been used, the list is cleaned making all the operators eligible for generating new solutions again.

Random seed window. First I need to explain that for each controller I don't make one evaluation but a set of evaluations of the controller over a set of random seeds. The point of that was to obtain more robust controllers that are not biased over one particular random seed since our search process is stochastic. What happens now is that instead of one seed the controller could be biased in a group of seeds. We use a mechanism to avoid bias over a particular group of random seeds. It consists of

keeping moving windows of random seeds over all the run. For example, the windows could have size 10 which means that each controller has to be evaluated over 10 different random seeds. If we set the window moving to 2, that means that after each iteration of the search, the first two seeds are going to be discarded and two new ones will enter the set. In summary, each controller is going to be evaluated in the previous 8 random seeds + 2 new seeds in each iteration after the first one and thus decreasing the probability to get biased. This mechanism is part of the exploration criterion because it affects the intensification process of the algorithm by changing the seeds in which a controller is evaluated. When changing the seeds we could have a decrease in the quality of the solution. But it promotes exploration by moving the window seed over the search process.

4.4.6 Acceptance criterion

I implemented various acceptance criteria, in part for extending the functionality of the Jonas' framework but also because I use some of them to test my SA implementation. In the context of local search and specifically in this work, I refer to acceptance criteria as the mechanisms to evaluate a set of results obtained from one solution against another set of results generated by a perturbed solution. This is because as I explained in the Exploration criterion, I evaluate the controllers over a set of random seeds. So if my window size is ten, I end up with a vector of 10 values each of them corresponding to the result of the objective function used to measure the solution quality. Since to measure the quality of a solution we use a vector, it makes sense to aggregate the results to build a metric and then compare the quality. It also thinks in more robust methods to compare sets as are signed-rank statistical tests. For those reasons, I implemented the following acceptance criteria.

- **Mean**.It takes the mean of the set of values returned by the evaluation of the performance of the controller in a given mission.
- **Median**. It takes the median of the set of values returned by the evaluation of the performance of the controller in a given mission.
- **Max**.It takes the maximum value of the set of values returned by the evaluation of the performance of the controller in a given mission.
- **Min**. It takes the minimum of the set of values returned by the evaluation of the performance of the controller in a given mission.
- **T-Student test**. T-student test with mean in a 0.95 confidence interval.
- **Wilcoxon test**. Wilcoxon test that compares median ranks among sets in a 0.95 confidence interval.
- **Metropolis condition**. Classical metropolis condition that can work with any other acceptance criteria listed above. It means that to get $\Delta(s', s)$ the metropolis could use the Mean, Median, Wilcoxon test, etc.

Metropolis is the acceptance criterion used for SA and the others are for local search in general.

4.4.7 Temperature length

The default implementation of SA in this work has $Tlength = 1$ but, we will see in chapter 5 that I try other variants of SA with a different Temperature Length.

4.4.8 Cooling Scheme

The cooling mechanism of this implementation follows a geometric scheme $T_{i+1} = \alpha \times T_i$ where T_i is the temperature in a given iteration i and T_{i+1} is the temperature for the next iteration $i + 1$. And $\alpha \in [0, 1]$ is a fixed value that has to be set before starting the search.

4.4.9 Restart mechanism

This SA algorithm can handle three different types of restart mechanisms.

- **Restart after a percentage** of the initial temperature remains. The temperature restarts with the original Initial temperature value. For example, when the temperature is has reached the 1 % of its initial value.
- **Restart k times** in a whole run.The temperature restarts with the original Initial temperature value. For example one restart at the 50% of the run.
- **Reheat**, which means reset the temperature value to a higher value. The new temperature value is calculated over the temperature that has generated a bigger improvement during the run so far.

4.4.10 Parallelism

One important feature we achieved in this implementation is that the controller evaluations—candidate solution evaluations—can be performed in parallel in an MPI environment. MPI stands for Message Passing Interface. It is a standardized and portable message-passing system developed for distributed and parallel computing. This is useful if the experimenter has access to a computing cluster since it can significantly reduce the experiments time. Otherwise, if there is a window of 10 random seeds and we are comparing two controllers, in the worst case the algorithm should wait until 20 simulations (10 for one controller, 10 for the other) are being executed one by one. We can distribute every simulation to a different computing unit. Therefore we should wait at most the time the slowest simulation takes to move forward into the optimization process.

Chapter 5

Experimental Setup

To assess the capabilities of AutoMoDe-Annealing, I conducted a series of experiments in which AutoMoDe-Annealing is used to automatically design the control software for robot swarms that are aimed to perform two different tasks: aggregation and foraging. I selected these two tasks because they are common benchmarks in swarm robotics. I assess AutoMoDe-Annealing for two control software architectures: probabilistic finite state machines and behavior trees. Every experiment is then performed for AutoMoDe-Annealing-FSM and AutoMoDe-Annealing-BT. The experiments consist, firstly, in studying the simulation budget— 5K, 10K, 25K, 50K and 100K simulations— influence over the design process. Additionally, I study three parameters of the optimization algorithm: the acceptance criterion, window size, and restart mechanism. This study is done to observe the influence of the parameters over the design process. Finally, I compare both instances of AutoMoDe-Annealing against AutoMoDe-Chocolate, AutoMoDe-Maple, AutoMoDe-Iterated-BT and AutoMoDe-Iterated-FSM.

5.1 Robot platform and reference model

AutoMoDe-Annealing is specialized for a swarm of e-puck robots Figure 5.1. The e-puck robot is a small wheeled robot designed for research and education [68]. The e-puck robot has eight IR transceivers used as light and proximity sensors. It is also equipped with a range-and-bearing board that comprises 12 IR emitters and 12 receivers equally distributed along the perimeter of the board and pointed radially and outwards, on the horizontal plane. The range-and-bearing board allows the e-puck to reliably send and receive messages within a range of 0.7 m. The reference model adopted in this research for the robot platform described above is given in Table 5.1.

5.2 Missions

As I explained in the previous chapters, the control software is designed for a specific mission. This mission is described by an objective function and the optimization

Figure 5.1: E-puck robot [14]

Table 5.1: Reference model RM1.1 [41]. Sensors and actuators of the e-puck robot. The period of control cycle is 100 ms.

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

algorithm optimized the outcome of the objective function. Below I describe the missions, accompanied by its objective function, I used to assess the robot swarm with the control software generated by AutoMoDe-Annealing.

5.2.1 Foraging

This mission is inspired by the behavior of ants which search for food sources distributed around their nest. The main challenge is to find the optimum search strategies that maximize the collection of food. The foraging scenario is shown in Figure 5.2. The arena contains two source areas represented by black circles and a nest represented by white area. A light is placed behind the nest to help the robots

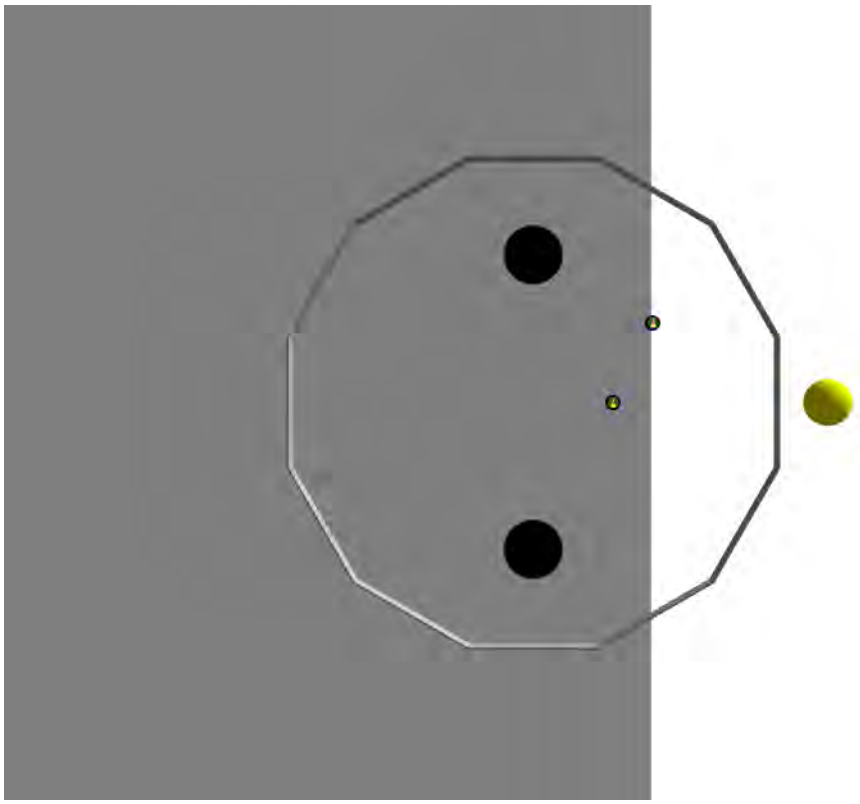


FIGURE 5.2: Foraging scenario.

to navigate. In this version of foraging, a robot gets an object when it enters in a source. The robot leaves the object in the nest, when it arrives to the white zone. The goal of the swarm is to retrieve as many objects as possible. The objective function is:

$$F_{foraging} = N_i \tag{5.1}$$

where N_i is the number of objects retrieved.

5.2.2 Aggregation with Ambient Cues

The main objective of aggregation is to group all the robots of a swarm in a region of the environment. Aggregation is a very useful building block, as it allows a swarm of robots to get sufficiently close one to another so that they can interact. In swarm robotic systems the aggregation mission can be considered as one of the fundamental behaviors. It can act as a precursor to other behaviors such as flocking and self-assembly. The aggregation scenario is shown in Figure 5.3. The arena contains two circular regions, one black and one white, each of them with the same diameter. The black region is located closer to the light source, which is on the left side of the arena. The robots have to aggregate on the black region and can use the light and the white region to orientate themselves. The performance measure is

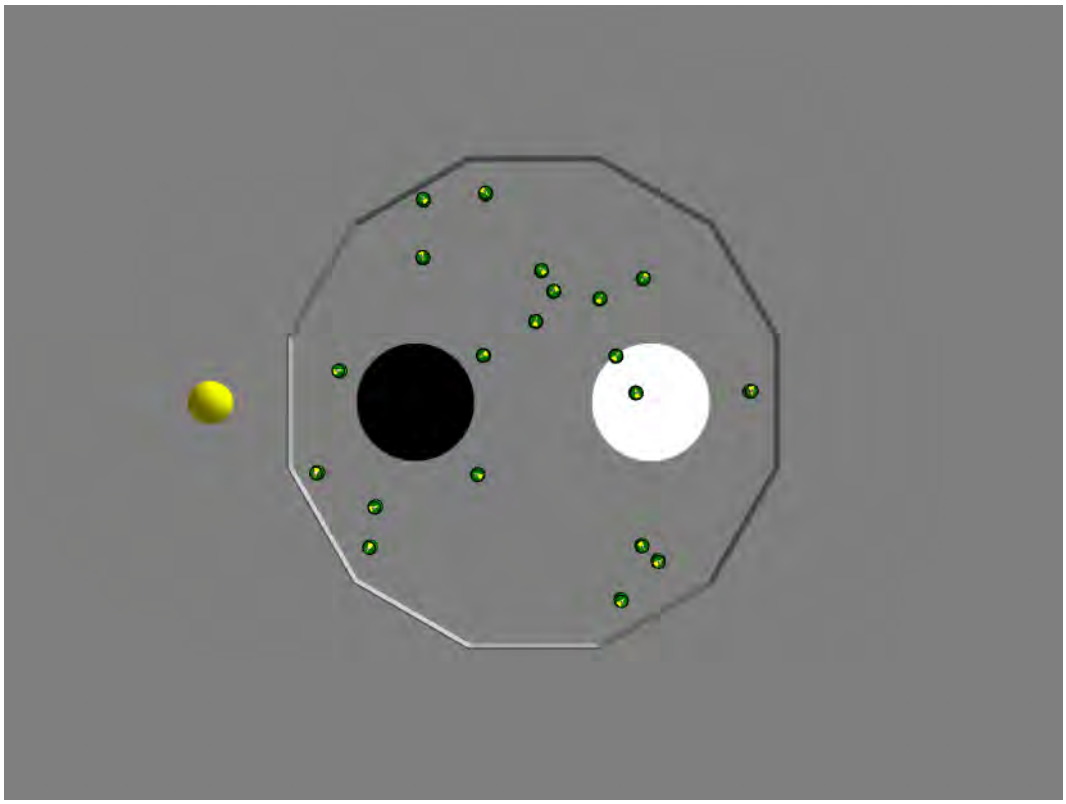


Figure 5.3: Aggregation with Ambient Cues scenario.

defined in terms of an objective function to maximize:

$$F_{aggregation} = \sum_{t=1}^{T} N(t) \tag{5.2}$$

where $N(t)$ is the number of robots on the black region at time t.

5.3 Simulated annealing default configuration

I have set default values for the simulated annealing that works as optimization algorithm in AutoMoDe-Annealing. The default values for the components and parameters are shown in Table 5.3. This configuration was used to study the budget impact and comparison. It was also the starting point for the parameter analysis. In each parameter study I only change the parameter of study and the rest of the parameters remains with the same values.

Table 5.2: Default configuration simulated annealing

Component	Type	Value
Initial solution	Stop behavior	Fixed probability
Neighborhood		Based on perturbation operators
Initial temperature	Fixed value	125.0
Stopping criterion	Budget of simulations	5K
Acceptance criterion	Metropolis condition	Mean
Exploration criterion	Random exploration	Perturbation operators and window
Temperature length	Fixed value	1
Cooling scheme	Geometric cooling	0.9782
Temperature restart	fixed value	Every 5000 simulations

5.4 Experiments protocol

The experiments presented in this work respect the following protocol: there is no human intervention in the automatic design process. The main objective of the experiments is to assess the expected performance of AutoMoDe-Annealing in designing control software for a robot swarm. I run three sets of experiments: budget analysis, parameter analysis and comparison. In the budget analysis, I run AutoMoDe-Annealing in five different design budgets. A design budget is the total number of simulation runs that AutoMoDe-Annealing can use to design the control software. The five design budgets are: 5000, 10000, 25000, 50000 and 100000 simulation runs. For each design budget, I execute 20 independent runs of AutoMoDe-Annealing. I obtain 20 instances of control software; I then assess the performance of these instances on simulation and pseudo-reality by performing 10 runs of each of them. By assessing in simulation and pseudo-reality I quantify the effects of the pseudo-reality gap. The initial position and orientation of the robots are obtained by running the constituent behavior exploration for a random number of seconds in $\{1, 2, \ldots, 20\}$. In the parameter analysis, I study 3 parameters that I consider relevant in the simulated annealing algorithm for this problem: acceptance criterion, window size and the restart mechanism. The same assessment protocol as for the budget analysis applies for this study. Finally, AutoMoDe-Annealing is compared with AutoMoDe-Chocolate, AutoMoDe-Maple, and AutoMoDe-Iterated. This last instance of AutoMoDe, AutoMoDe-Iterated is still in development, but it is interesting to compare it with the others because it is also based on a stochastic local search algorithm, the iterative improvement.

5.4.1 Budget influence

The automatic design of control software for robot swarms is a heavy task for the computer given that it has to perform lots of simulations. It is then interesting to study the design method for various budgets of simulation. I study this so I can find how AutoMoDe-Annealing performs for relatively low budgets–5K and 10K—and bigger ones-25K,50K and 100K. This is also interesting because we can observe if

for bigger budgets the method can still take advantage or it converges after some number of simulations.

5.4.2 Acceptance criterion influence

Simulated annealing is a highly configurable algorithm. One of the most important parameters is the acceptance criterion. The study of this parameter aims to observe the performance of different acceptances criterion for 25K, 50K and 100K budget of simulations.

5.4.3 Window size influence

In a stochastic local search algorithm, the exploration criterion is quite important. It is important because too much exploration can end up in a no converging algorithm. However, if there isn't enough exploration the algorithm can easily get stuck in a locally optimal solution. Finding a good balance is then important for the algorithm to converge and get closer to the global optima solutions. I study this by changing the window size for three budgets of simulation: 25K, 50K and 100K.

5.4.4 Restart mechanism influence

Restarting a stochastic local search algorithm is a mechanism to escape from stagnation. In this case simulated annealing, every time it restarts, is more likely to explore and accept worse solutions. It is then important to find the best mechanism in this problem to observe its influence over the control software performance that has been generated.

5.4.5 Comparison

A comparison with other design methods is very important because I can immediately know if my method can compete against state-of-the-art methods. I perform a comparison against AutoMoDe-Chocolate and AutoMoDe-Maple—two state-of-the-art methods—and another instance of AutoMoDe based on a local search algorithm. I make the comparison for three budgets of simulation: 25K, 50K, and 100K.

5.5 Statistical analysis

Concerning the results of the design method, I present notched box-and-whisker boxplots. A notched box-and-whisker boxplot gives a visual representation of a sample. The horizontal thick line denotes the median. The lower and upper sides of the box are called upper and lower hinges and represent the 25th and 75th percentile of the observations, respectively. The upper whisker extends either up to the largest observation or up to 1.5 times the difference between upper hinge and median—whichever is smaller. The lower whisker is defined analogously. Small circles represent outliers (if any), that are observations that fall beyond the whiskers.

Notches extend to $\pm 1.58 IQR/\sqrt{n}$, where IQR is the interquartile range and n = 20 is the number of observations. Notches indicate the 95% confidence interval on the position of the median. If the notches of two boxes do not overlap, the observed difference between the respective medians is significant [16]. In the boxplots, we include also the results obtained in simulation in order to appraise the impact of the pseudo-reality gap on the design methods. Results obtained in pseudo-reality are represented by wide light-gray boxes and those obtained in simulation by narrow dark-gray boxes.

Chapter 6

Experiments and Results

In this chapter, I present the results of assessing control software generated by my implementation of AutoMoDe, AutoMoDe-Annealing. I assess the method in a simulation environment and pseudo-reality. In the first section, I present a study of the budget influence over the method. In the second section, I present the influence of the acceptance criterion over three different simulation budgets. In the third section, I present the influence of the window size parameter over the method, through three different simulations budgets. In the fourth section, I present the results of the restart mechanism influence in the algorithm. I try four different mechanisms. Finally, in the last section I compare AutoMoDe-Annealing against AutoMoDe-Iterated, AutoMoDe-Maple and AutoMoDe-Chocolate. In every experiment, AutoMoDe-Annealing is assessed for behavior trees and finite state machines. I refer to them as AutoMoDe-Annealing-BT and AutoMoDe-Annealing-FSM respectively. It is important to mention that, in every study the method is evaluated for two missions: aggregation with ambient cues—I refer to it as aggregation—and foraging. These missions are explained in more detail in Chapter 5.

6.1 Budget influence

The automatic design process uses a budget of simulations that helps to explore the control software search space. This is an important subject of study because simulating swarm of robots could be a hard-computing process. In Figure 6.1 the performance is shown through five different increasing budgets. We can observe that for Annealing-BT in aggregation, the budget appears not to have any influence. Even if we compare 5K and 100K there is no significant difference. For foraging, Annealing-BT seems to be affected by the budget, since the results look less disperse when increasing the budget. An overall observation for Annealing-BT is that we can not find a significant difference between simulation and pseudo-reality.

On the other hand, Annealing-FSM shows a modest improvement when increasing the simulation budget. However, we cannot appreciate the increment in performance when going from 5K to 10K but from 5K to 25K, which means that we should consider bigger differences in the budget to observe a significant improvement. We can also

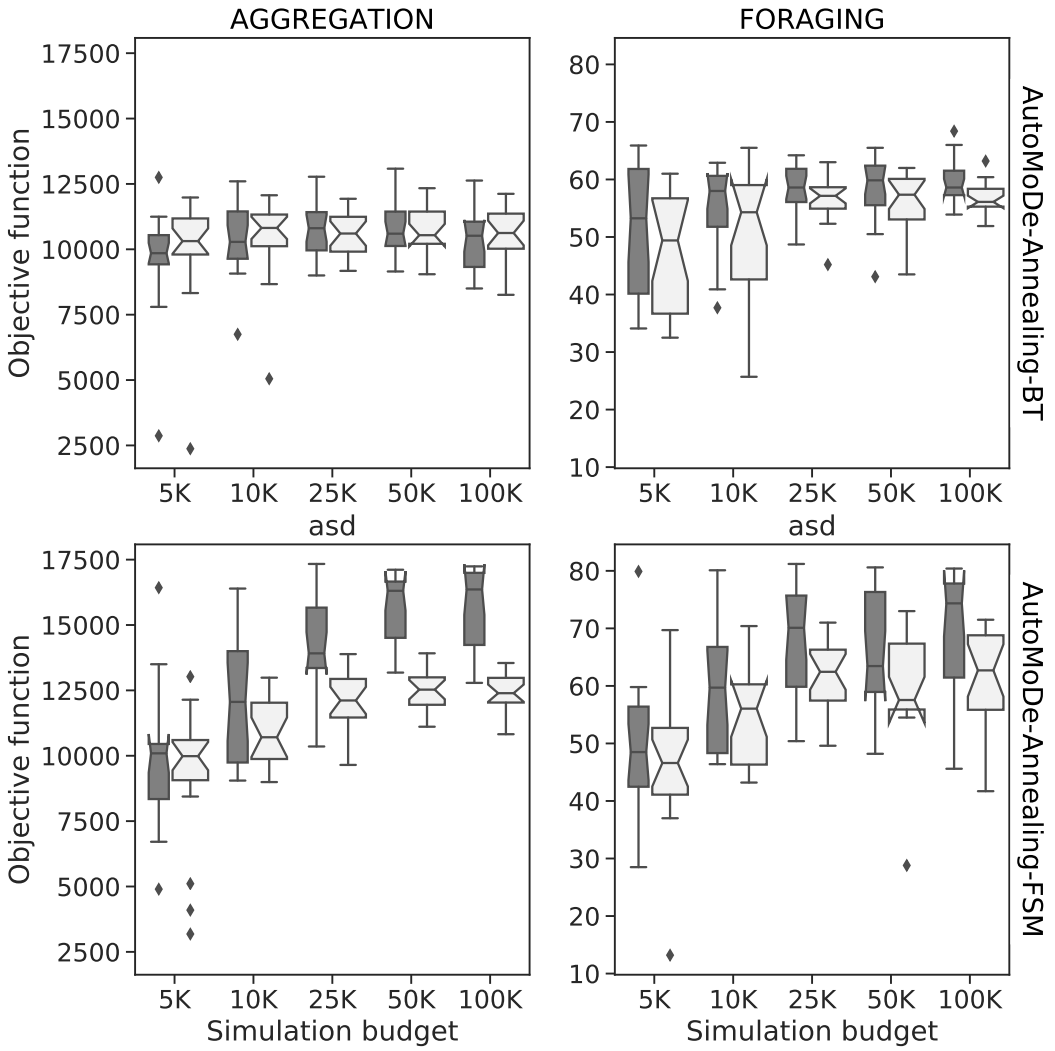


Figure 6.1: **Budget influence**. In this figure, we can observe the performance of Annealing-BT—First row— and Annealing-FSM—Second row— through an increasing simulation budget. The first column corresponds to the aggregation mission and the second to foraging. The assessment in simulation is represented by dark-gray thin boxes, while pseudo-reality is represented by light-gray thicker boxes.

observe that either for simulation and pseudo-reality the modest improvement persists. However, for bigger budgets, 50K and 100K the loss of performance is bigger.

Run time analysis

It is also interesting to observe the evolution of the algorithm in run time. We can observe in Figure 6.2 how the optimization algorithm is evolving in time over five different budgets. The figure shows how for a budget of 5K and 10K there is

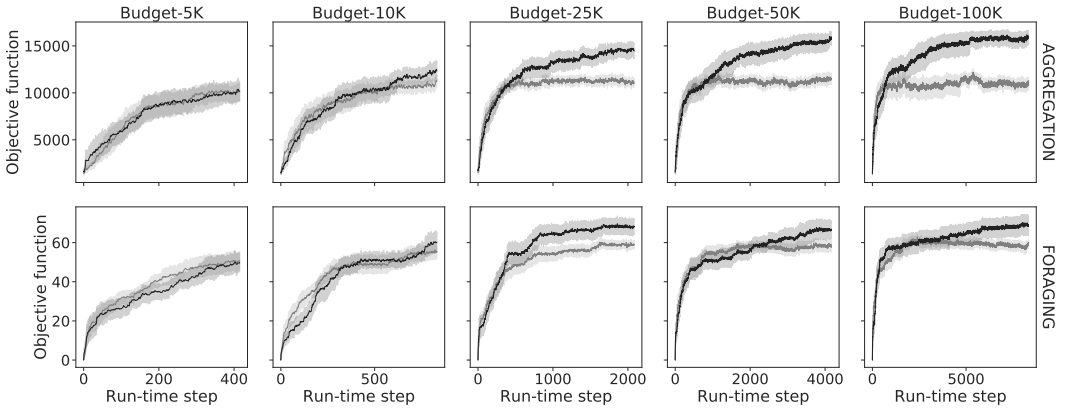


FIGURE 6.2: **Performance during the design process**. This figure shows the evolution of the control software performance in run time. Columns show the results for each budget. Rows represent the missions used for the assessment. The Annealing-BT and Annealing-FSM performance is represented by a light-gray and dark-gray line, accordingly. The performance results are shown within a 95% confidence interval.

no difference in the evolution of the design process in time for Annealing-BT and Annealing-FSM. Nonetheless, for bigger budgets 25K, 50K and 100K Annealing-FSM starts taking advantage of the budget increment over Annealing-BT. Both methods converge rather fast but Annealing-FSM keeps modestly increasing the performance when increasing the simulation budget in both missions.

6.2 Acceptance criterion influence

This is one of the most important parameters in local search algorithms since the algorithm relies on the acceptance criterion as a mechanism of accepting or not a new candidate solution. If the criterion is too tight, the probability of getting trapped in a local optimum is higher. If the criterion accepts any candidate solution then it becomes a random walk and thus the algorithm could never converge. Either for Annealing-BT as for Annealing-FSM, the algorithm uses a custom Metropolis condition acceptance criterion. It is custom because the implemented method evaluates multiple instances of one controller. So, to evaluate the difference in quality, I use by default the mean value of the set of results corresponding to the objective function metric. In this study, I asses the use of Mean, Median and Wilcoxon test as aggregates of the instances to after evaluate the Metropolis condition as an acceptance criterion. The results are shown for 25K, 50K and 100K simulation budgets.

6.2.1 25K

In Figure 6.3, we observe that for a budget of 25K simulations there are no significant differences among the three criteria neither for Annealing-BT nor for Annealing-FSM

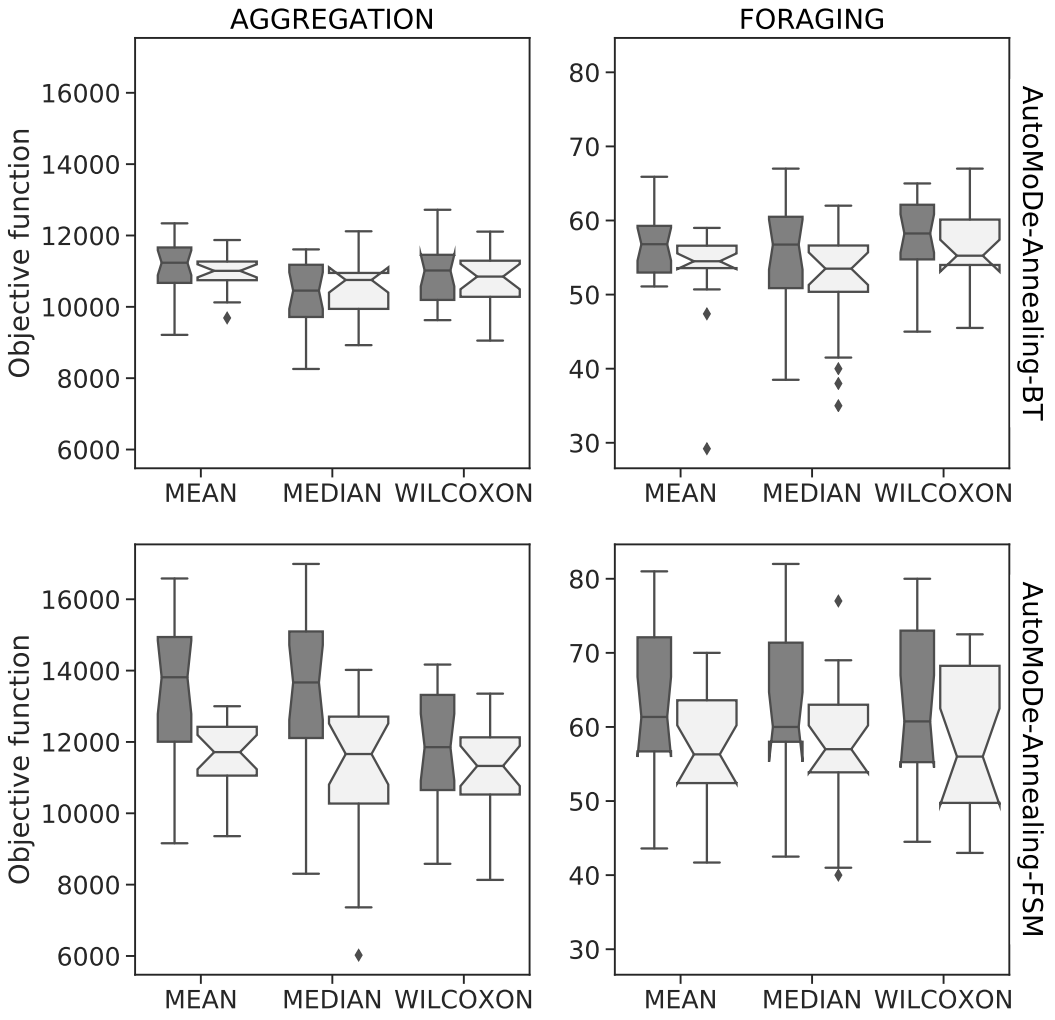


Figure 6.3: **Acceptance criterion 25K**. This figure shows the performance of Annealing-BT and Annealing-FSM over three different acceptance criteria, Mean, Median and Wilcoxon test, in two missions. The thin light-gray boxes represent assessment in simulation and the thicker dark-gray boxes assessment in pseudo-reality.

in both missions. However, we observe that the Wilcoxon test criterion suffers the less with regard to the pseudo-reality gap with exception of Annealing-FSM for foraging. This is expected since the Wilcoxon test is a rank-based test that can better describe differences between sets. However, the Wilcoxon test could need a bigger number of instances to have a better description of the set it is trying to compare. For this case, the sets are about 10 samples when it is empirically recommended to have at least 30 samples to have confidence in the results.

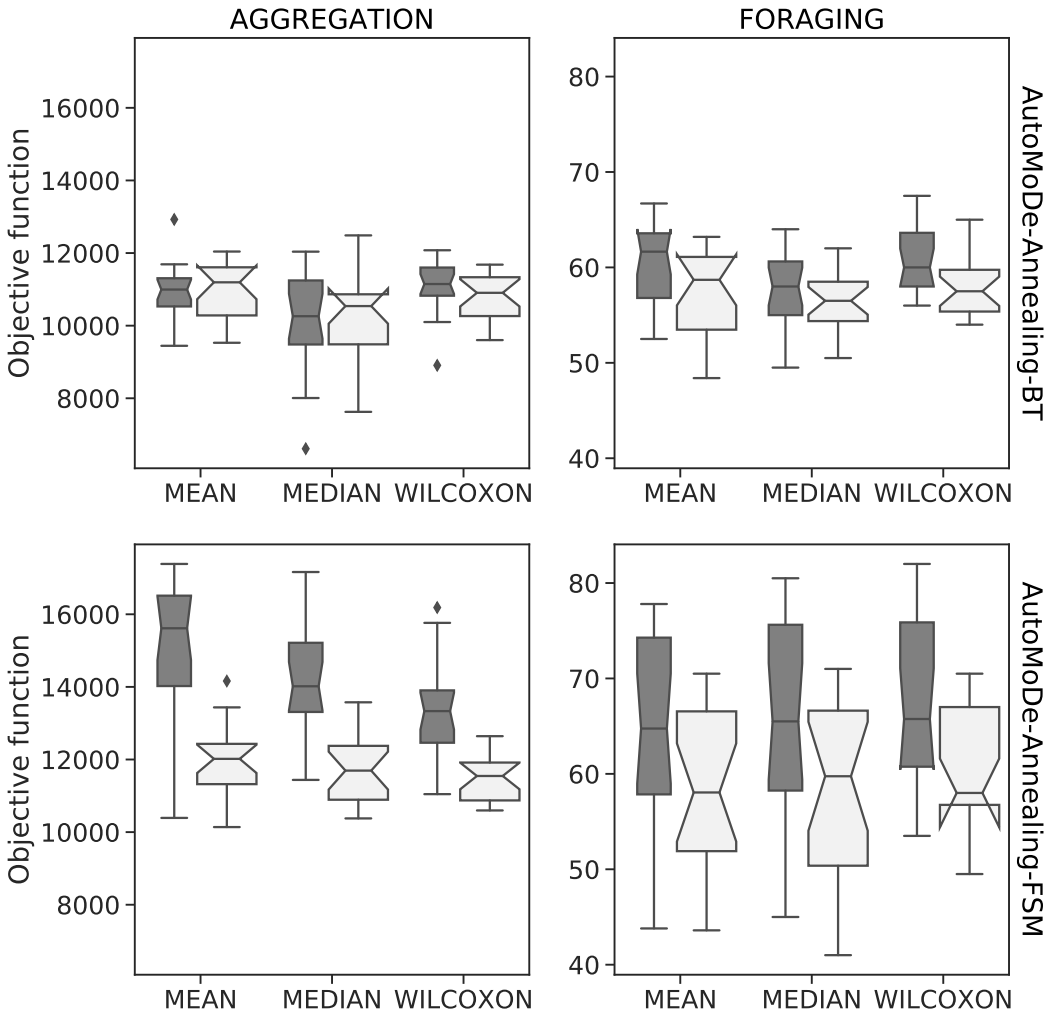


FIGURE 6.4: **Acceptance criterion 50K**. This figure shows the performance of Annealing-BT and Annealing-FSM over three different acceptance criteria, Mean, Median and Wilcoxon test, in two missions. The thin light-gray boxes represent assessment in simulation and the thicker dark-gray boxes assessment in pseudo-reality.

6.2.2 50K

In Figure 6.4 we observe that either in simulation and pseudo-reality, Annealing-BT and Annealing-FSM don't show any significant difference. However, we observe that for Annealing-BT, Wilcoxon and Mean start getting a difference w.r.t the Median criterion. For Annealing-FSM in aggregation, we observe that the loss of performance between simulation and reality is more evident but again Wilcoxon shows more resistance to the pseudo-realty gap.

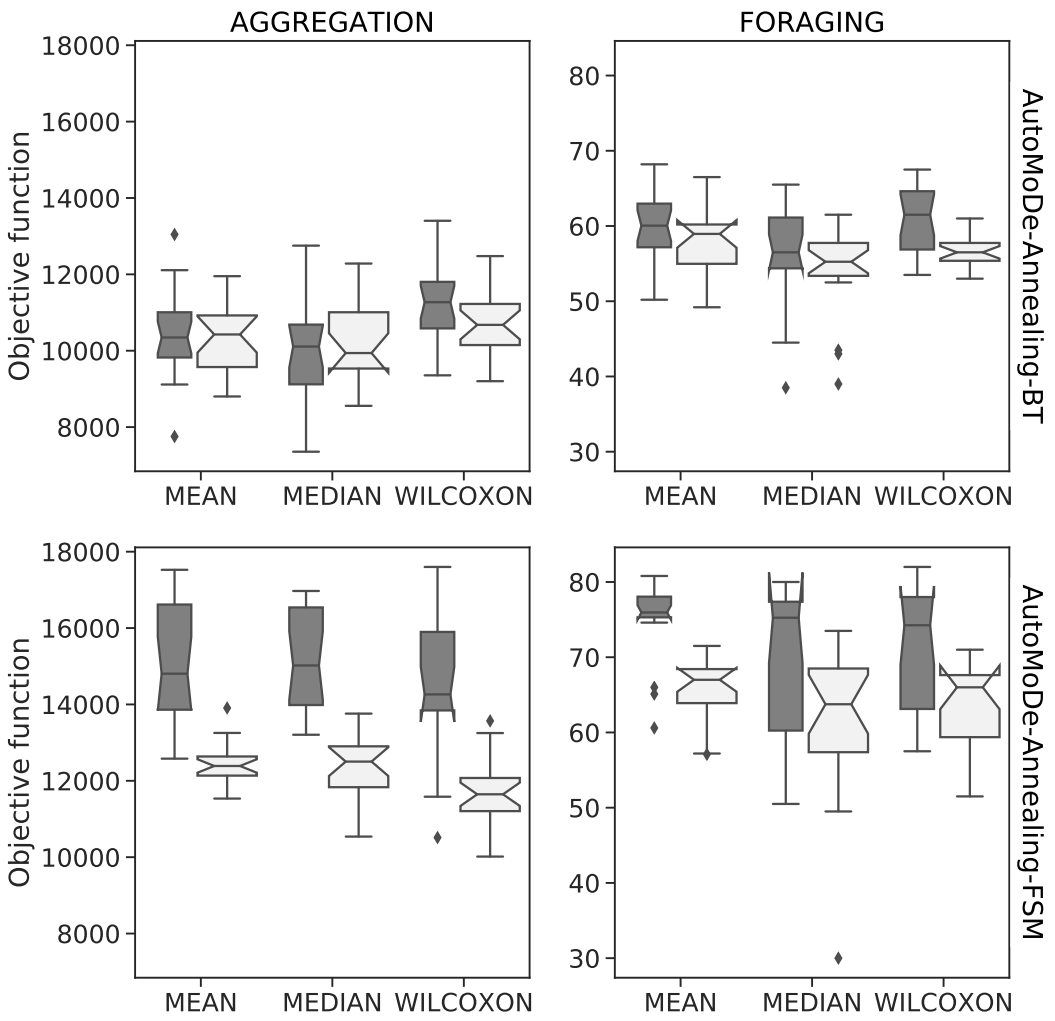


Figure 6.5: **Acceptance criterion 100K**. This figure shows the performance of Annealing-BT and Annealing-FSM over three different acceptance criteria, Mean, Median and Wilcoxon test, in two missions. The thin light-gray boxes represent assessment in simulation and the thicker dark-gray boxes assessment in pseudo-reality.

6.2.3 100K

We observe in Figure 6.5 that for a budget of 100K the difference among criteria is still no clear. Annealing-FSM for aggregation shows in pseudo-reality that the Mean and Median perform significantly better than Wilcoxon. However, Annealing-BT for aggregation Wilcoxon shows a modest improvement in simulation and pseudo-reality. Another observation is that Annelaing-FSM in pseudo-reality keeps increasing when increasing the budget but not as much as it increases in simulation.

6.3 Window size influence

As a reminder, the window size refers to the fact that for evaluating a control software candidate I use 10 instances of one mission. This means that one controller generated by the design method is simulated 10 times by changing the initial conditions of the environment without affecting the objective function of the mission. Decreasing the window size means that I evaluate less the same controller per time step. Increasing the window size means that I evaluate the same controller over more initial conditions. In this study I asses a minor, 5 instances, windows and bigger, 15 instances, windows w.r.t the default which is 10 instances. The results of the study are presented in three subsections: 25K, 50K and 100K. These values represent the simulation budget used to study the variants of the original configuration.

6.3.1 25K

The results of studying the windows size for a budget of 25K simulations is shown in Figure 6.6. We can observe that the only case in which we can find a significant difference is in Annealing-BT. For aggregation Annealing-BT with a window size of 15 is significantly better than a window size of 5. This is interesting because when the window is lower the number of steps the optimization algorithm tries to improve is higher. So, in the case of Annealing-BT for aggregation we observe that more attempts to improve are not helping the algorithm to yield better results. On the other hand, Annealing-FSM doesn't show a clear difference across window sizes, neither simulation nor pseudo-reality. However, we can observe an increase of dispersion when the window size increasing in the foraging mission.

6.3.2 50K

The results for a budget of 50K simulations are shown in Figure 6.7. We can observe that the performance of Annealing-BT for a window size of 15 instances keeps increasing in aggregation and foraging as well. However, the differences between 10 and 15 are not significant. For Annealing-FSM there is no significant difference among the window sizes. One observation is that in aggregation a window size of 15 helps to decrease the dispersion of the results.

6.3.3 100K

The pattern of Annealing-BT with a window size of 15 is repeated for a budget of 100K simulations as we can observe in Figure 6.8. This time the improvement is more clear with respect to a windows size of 10, although it is not significantly different yet. For Annealing-FSM in simulation get better with more budget but the pseudo-reality gap is bigger too. Another observation is that for foraging Annealing-FSM with window a size of 5 get less dispersed than the other sizes, but still, a not significant difference is found.

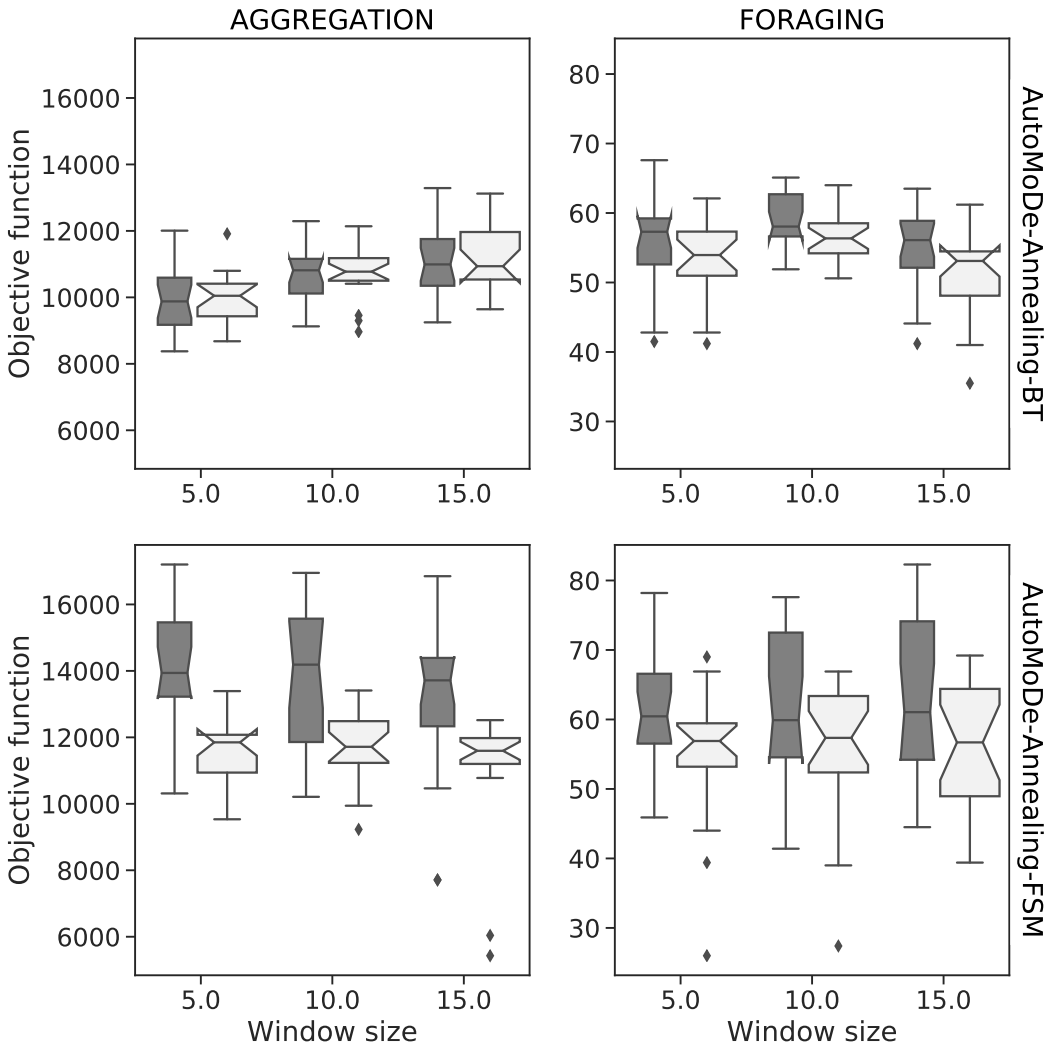


Figure 6.6: **Window size 25K**. This figure shows the performance of Annealing-BT and Annealing-FSM over three different window sizes, 5, 10 and 15. The columns represent the missions: aggregation and foraging. The thin light-gray boxes represent assessment in simulation and the thicker dark-gray boxes assessment in pseudo-reality.

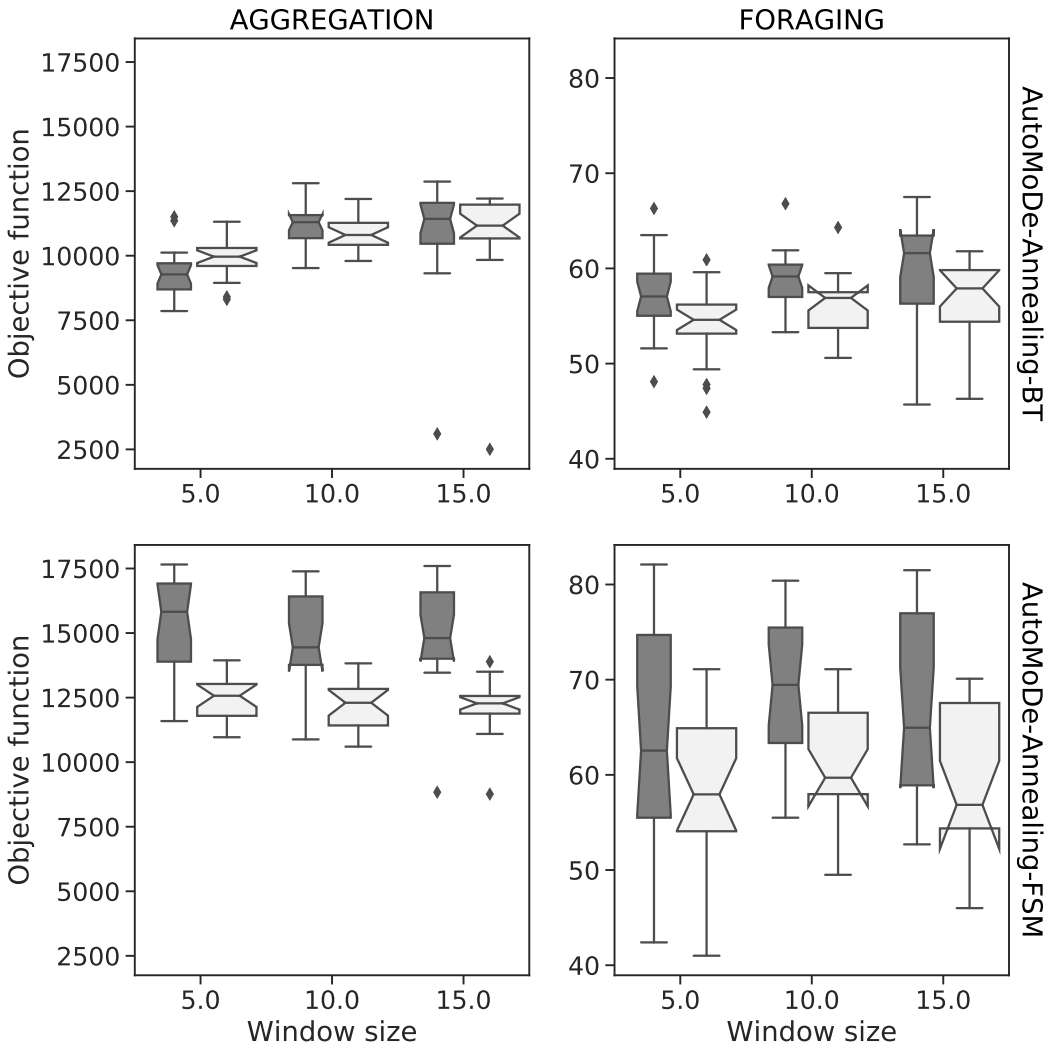


FIGURE 6.7: **Window size 50K**. This figure shows the performance of Annealing-BT and Annealing-FSM over three different window sizes, 5, 10 and 15. The columns represent the missions: aggregation and foraging. The thin light-gray boxes represent assessment in simulation and the thicker dark-gray boxes assessment in pseudo-reality.

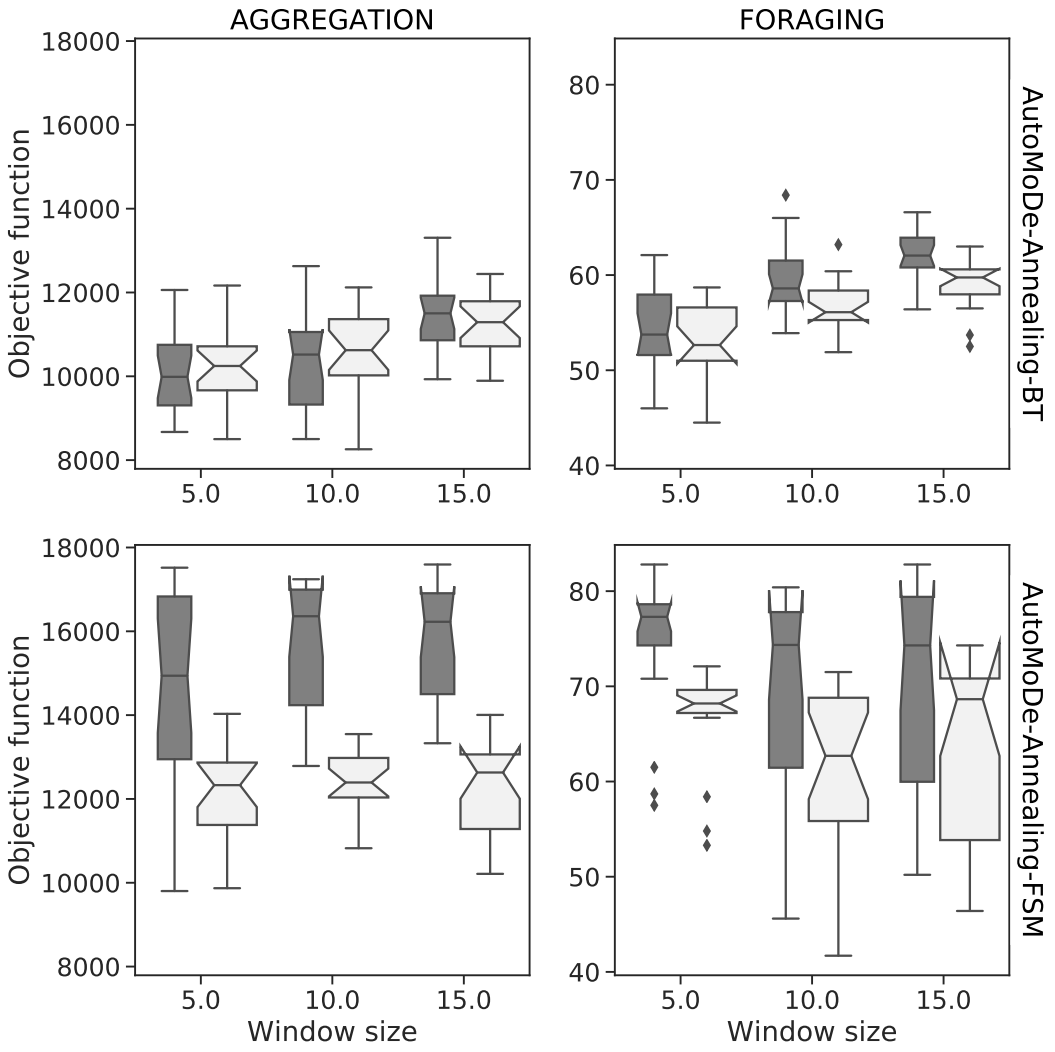


Figure 6.8: **Window size 100K**. This figure shows the performance of Annealing-BT and Annealing-FSM over three different window sizes, 5, 10 and 15. The columns represent the missions: aggregation and foraging. The thin light-gray boxes represent assessment in simulation and the thicker dark-gray boxes assessment in pseudo-reality.

6.4 Restart mechanism influence

Restarting local search algorithms is a technique used to avoid stagnation. That is why I consider the study of the restarting mechanism in my research. Additionally, The default restart mechanism could be affecting the dispersion of solutions, especially for Annealing-FSM as we can observe in the budget analysis (Figure 6.1). I evaluate 4 different restarting mechanisms: Default, NoRestart, RunRestart and Reheat. The study is organized in 3 sections where each one corresponds to the assessment for three different budgets: 25K, 50K and 100K budget of simulations.

6.4.1 25K

The results of the restart mechanism study for a 25K budget of simulations are presented in Figure 6.9. For Annealing-BT the restart mechanism doesn't present a significant difference across mechanisms and missions. The restart mechanism in Annealing-FSM also shows no impact across different mechanisms and missions. It only reports a bigger loss of performance when assessing it in pseudo-reality but no significant difference is found, except for Reheat that is significantly worse in pseudo-reality for the aggregation mission.

6.4.2 50K

The results of this study for a 50K budget of simulations are found in the Figure 6.10. For Annealing-BT as for 25K there is no influence of the restart mechanism across missions. For Annealing-FSM, there are significant differences across mechanisms. For NoRestart and RunRestart the dispersion is very low while for Default and Reheat the dispersion is bigger. These last two mechanisms restart 10 times during the design process. So the result shows that in fact restarting is affecting the dispersion of the solution for the foraging mission.

6.4.3 100K

The results for a 100K budget of simulations are presented in Firugre 6.11. For Annealing-BT the pattern of no influence remains. For Annealing-FSM we observe that for the foraging mission, the three of the alternative restart mechanisms behave similarly and show low dispersion w.r.t the Default mechanism. This gives another hint of what could be affecting the dispersion, which is the temperature value used in the simulated annealing algorithm.

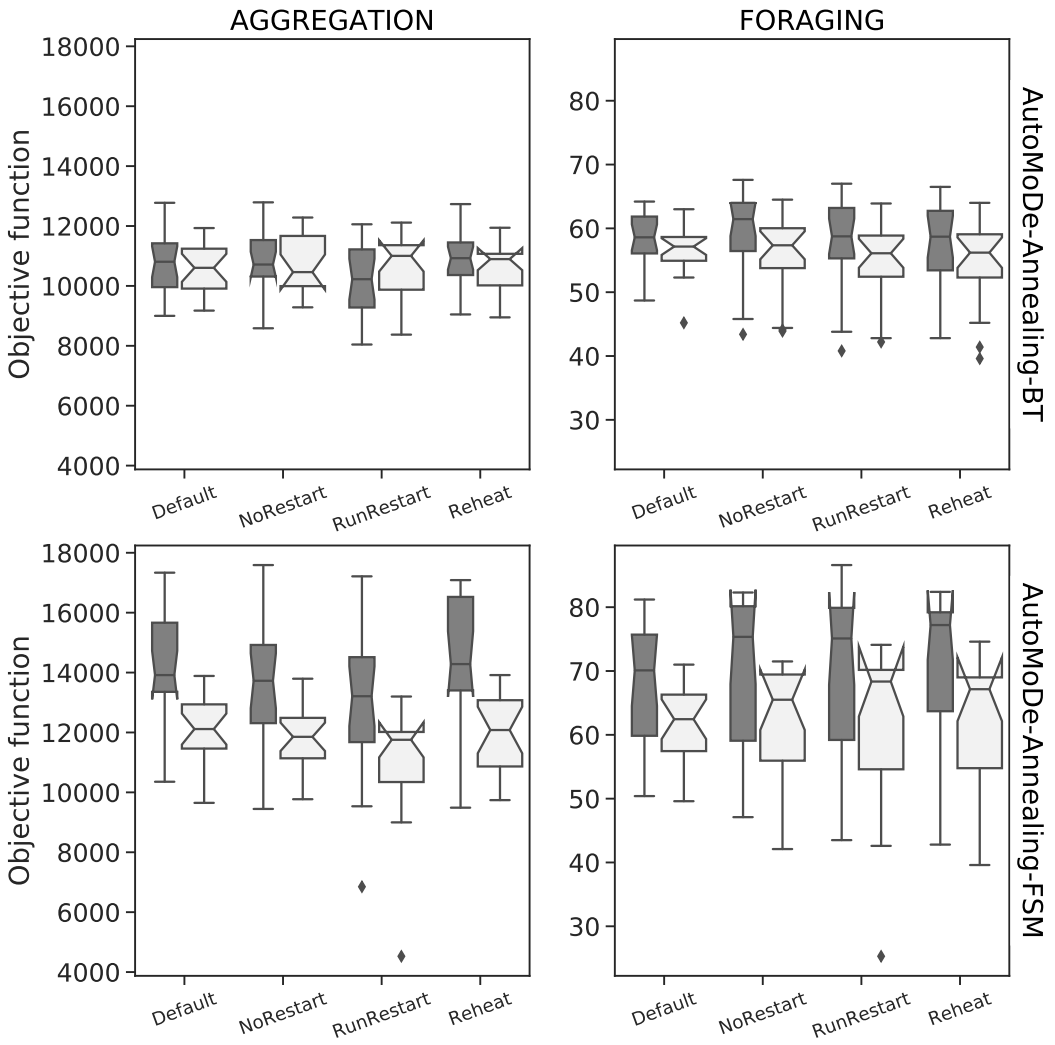


Figure 6.9: **Restart mechanism 25K**. This figure shows the performance of Annealing-BT and Annealing-FSM over four different restart mechanisms: Default, NoRestart, RunRestart and Reheat. The columns represent the missions used for the assessment: foraging and aggregation. The thin light-gray boxes represent assessment in simulation and the thicker dark-gray boxes represent assessment in pseudo-reality.

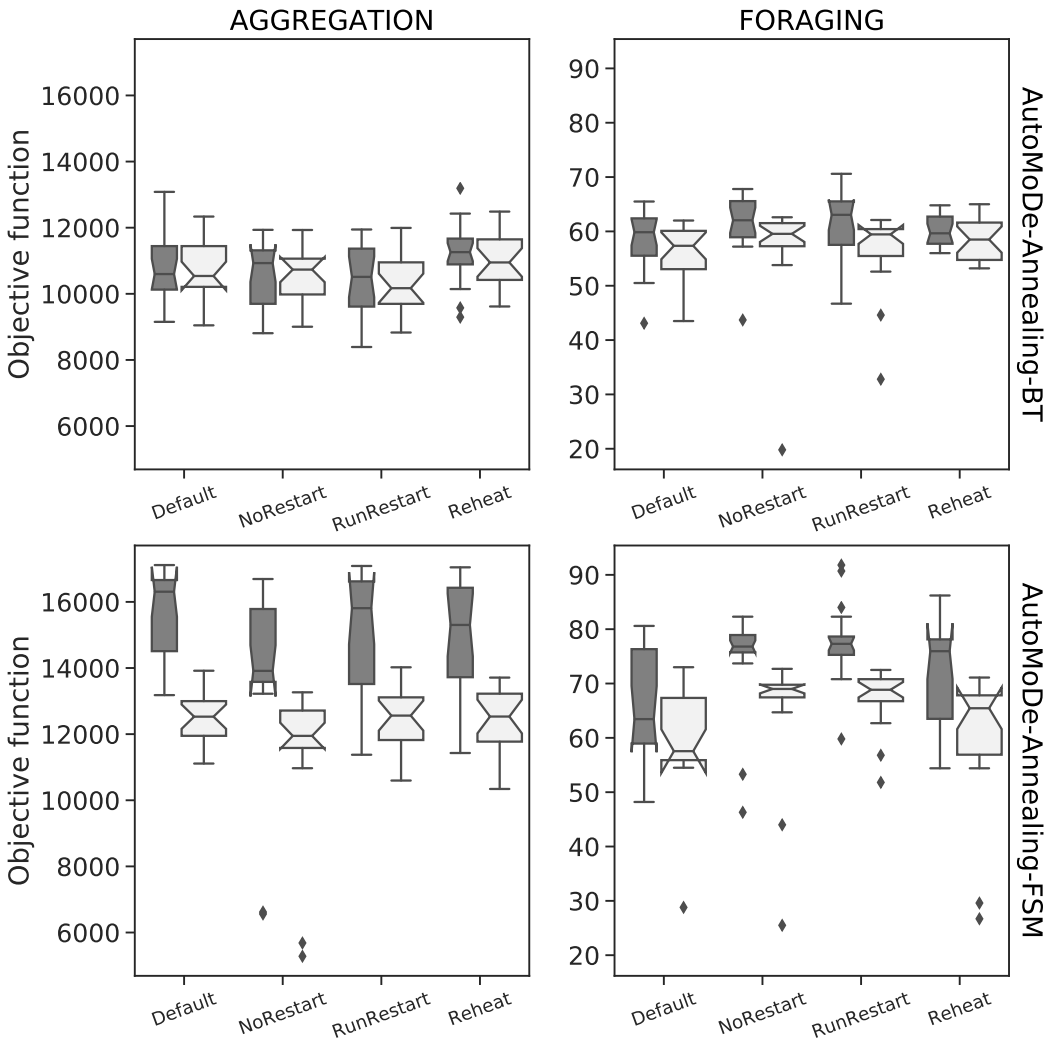


FIGURE 6.10: **Restart mechanism 50K** This figure shows the performance of Annealing-BT and Annealing-FSM over four different restart mechanisms: Default, NoRestart, RunRestart and Reheat. The columns represent the missions used for the assessment: foraging and aggregation. The thin light-gray boxes represent assessment in simulation and the thicker dark-gray boxes represent assessment in pseudo-reality.

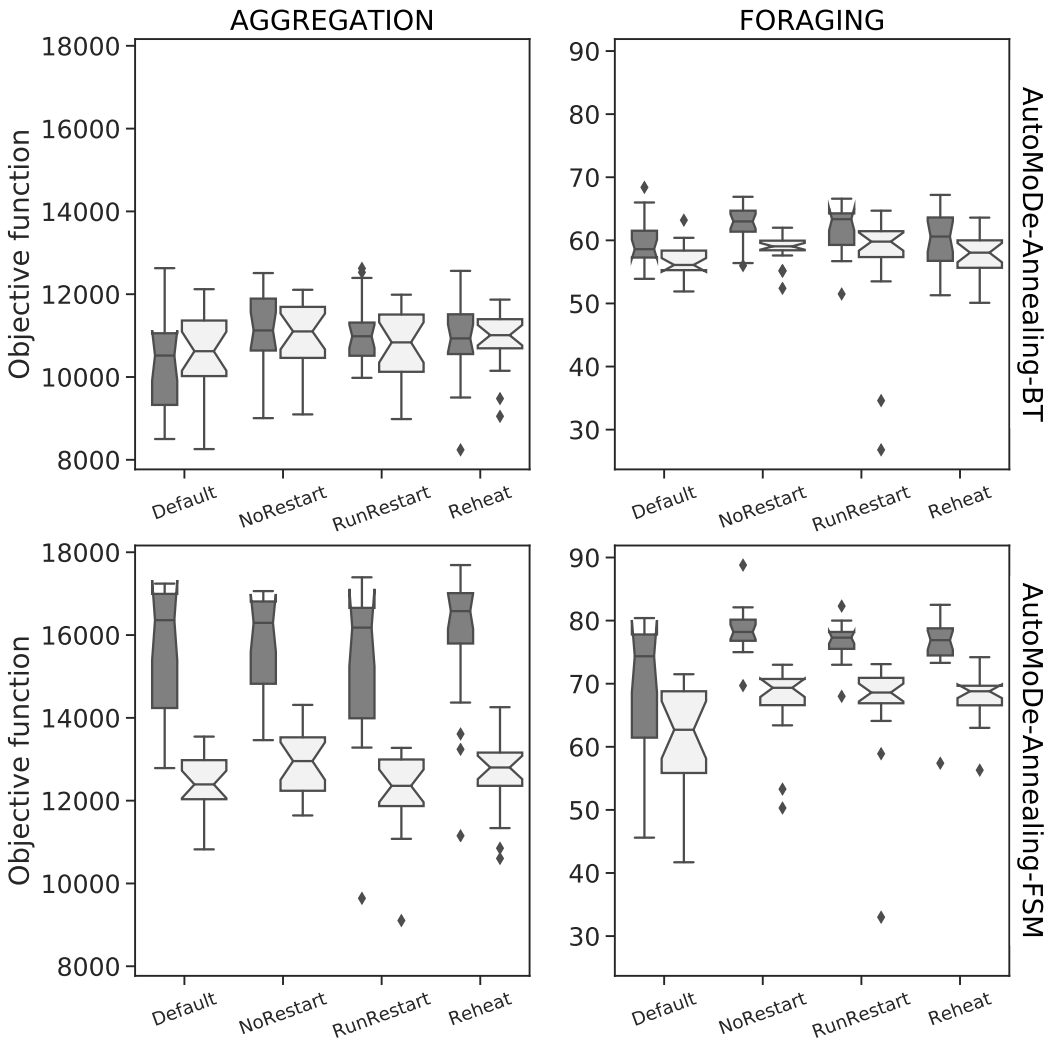


Figure 6.11: **Restart mechanism 100K**. This figure shows the performance of Annealing-BT and Annealing-FSM over four different restart mechanisms: Default, NoRestart, RunRestart and Reheat. The columns represent the missions used for the assessment: foraging and aggregation. The thin light-gray boxes represent assessment in simulation and the thicker dark-gray boxes represent assessment in pseudo-reality.

6.5 Comparison

In this experiment, I compare six different instances of AutoModDe. Three of the instances are based on behavior trees: Maple, Iterated-BT, and Annealing-BT. The other three instances are based on probabilistic finite state machines: Chocolate, Iterated-FSM and Annealing-FSM. These methods were evaluated for three budgets of 25K, 50K and 100K simulations. For this comparison, I use the default implementation of the simulated annealing algorithm. These instances of AutoMoDe differ either for the optimization algorithm used—I-Race, iterative improvement or simulated annealing—and the control software architecture—BT or FSM—. The results are presented in two subsections Aggregation 6.5.1 and Foraging 6.5.2. Each of these sub-sections presents the results of comparing the AutoMoDe instances over the aforementioned budgets in its respective mission, aggregation and foraging accordingly.

6.5.1 Aggregation

In Figure 6.5.1 we can observe results for the aggregation mission assessment. For the BT-based methods, we can observe no significant difference among Maple, Iterated and Annealing is found. They perform rather similar. On the other hand in the FSM-based methods, iterative Iterated and Annealing shows, for all budgets, a significant difference w.r.t Chocolate.

6.5.2 Foraging

In Figure 6.5.2 we can observe results for the foraging mission assessment. For the BT-based methods, we can observe no significant difference among Maple, Iterated and Annealing. They perform rather similar. Another observation is that the three methods decreases the dispersion of solutions when increasing the simulation budget. On the other hand in the FSM-based methods, iterative Iterated and Annealing shows, for all budgets, a significant difference w.r.t Chocolate. But this time in the Chocolate is closer to Iterated and Annealing methods. Another observation is that Chocolate present 2 of 3 cases, a significant difference between simulation and pseudo-reality. Iterated also presents a significant difference between simulation and reality over the three budgets assessed. Only simulated annealing, in the three of the budgets, doesn't show a significant difference between simulation and pseudo-reality.

Figure 6.12: **Aggregation comparison.** The light-gray boxes represent assessment in simulation and the thicker dark-gray boxes assessment in pseudo-reality.

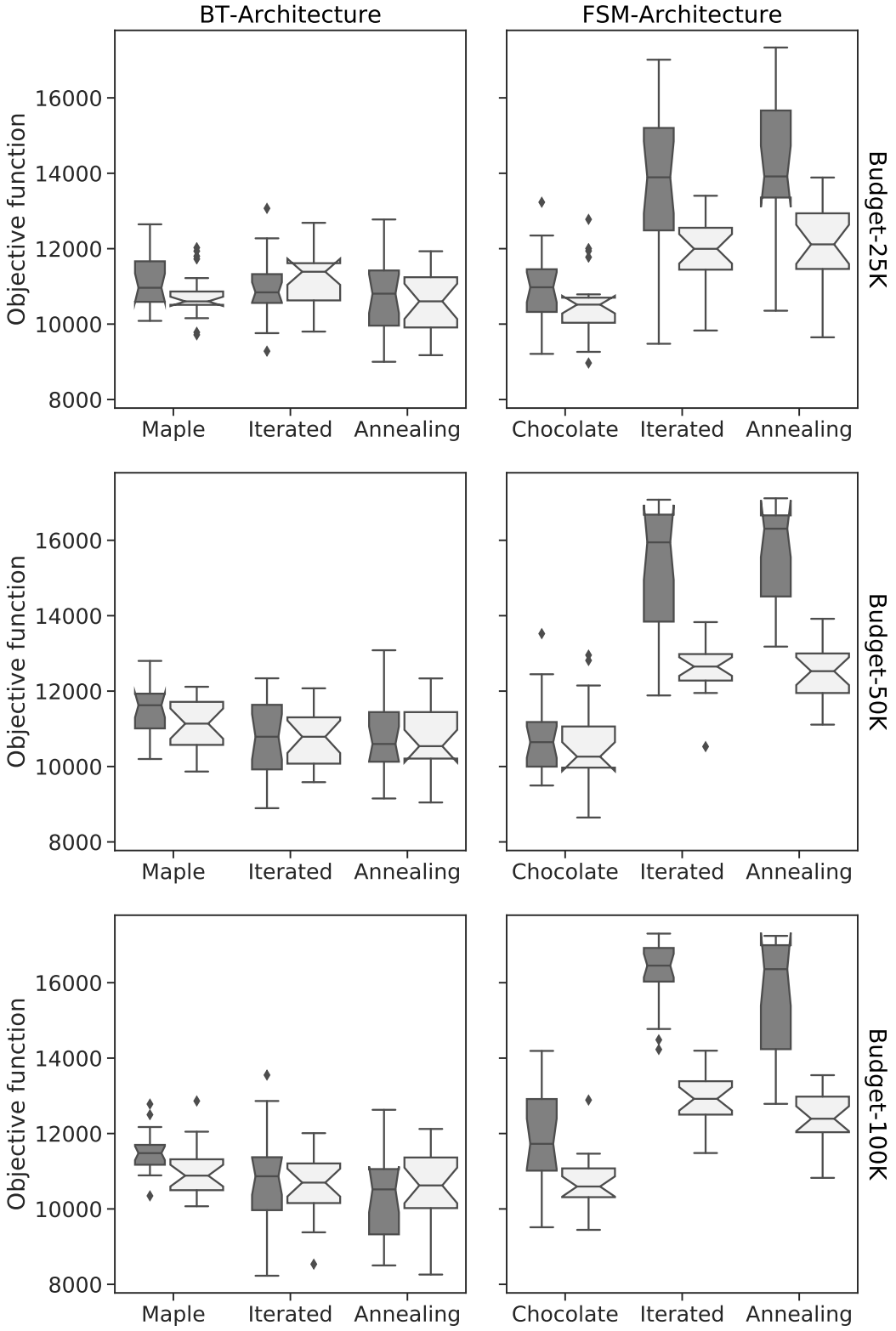
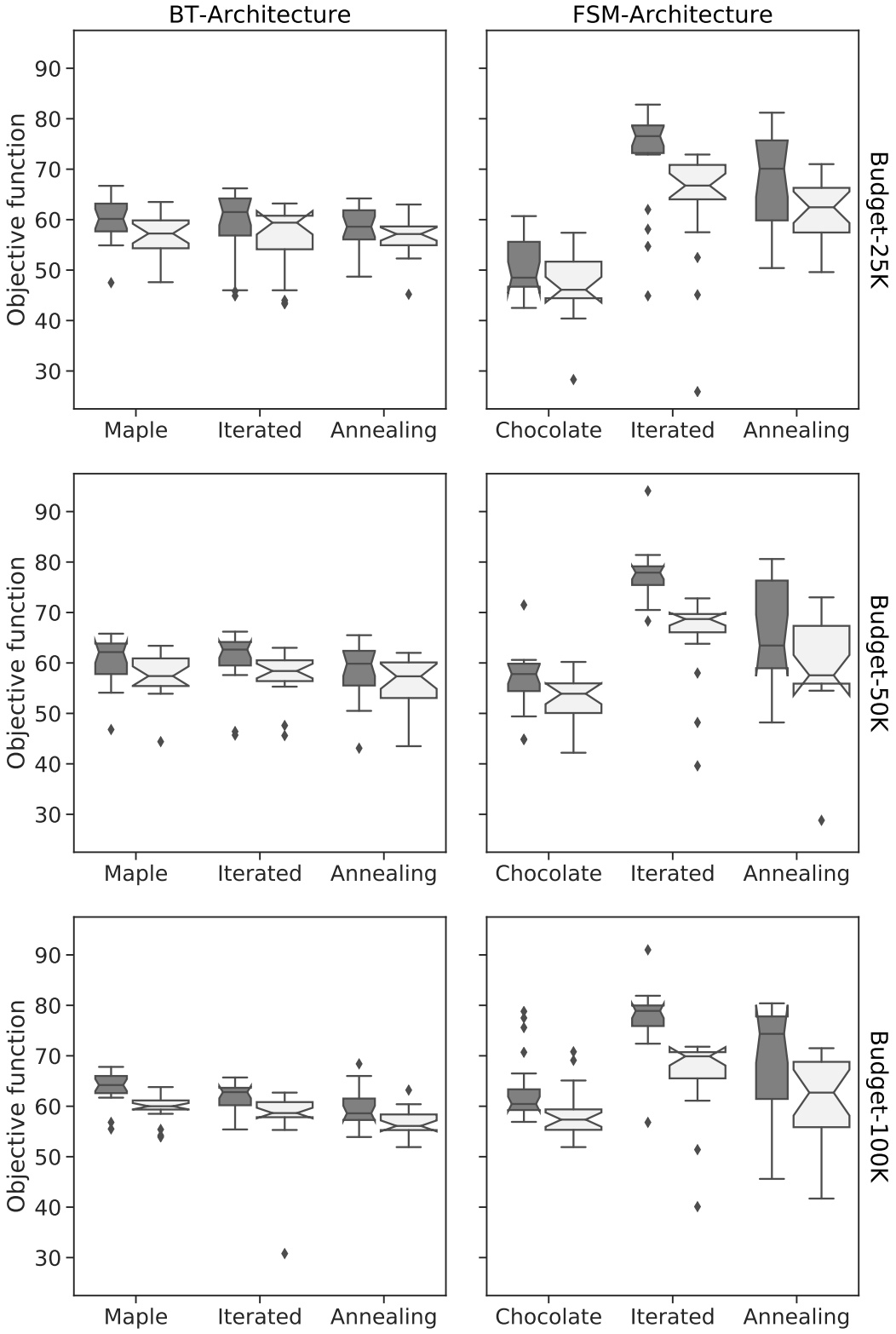


FIGURE 6.13: **Foraging comparison.** The light-gray boxes represent assessment in simulation and the thicker dark-gray boxes assessment in pseudo-reality.



Chapter 7

Conclusions and Future work

In this thesis, I presented AutoMoDe-Annealing: a new instance of AutoMoDe that uses simulated annealing—a method based on local search—as optimization algorithm. AutoMoDe-Annealing operates along two architectures for the control software of robot swarms. I conceived AutoMoDe-Annealing-FSM to operate with probabilistic finite state machines, and similarly, I conceived AutoMoDe-Annealing-BT to operate with behavior trees. I assessed the performance of AutoMoDe-Annealing in two standard swarm missions: aggregation and foraging. In both cases, I studied the influence of the budget of simulations available in the design process. More precisely, I conduced experiments whit AutoMoDe-Annealing and design budgets of 5K, 10K, 25K, and 100K simulations.

The first conclusion I wish to draw lies on the difference of the design by optimization of probabilistic finite state machines and behavior trees. As a matter of fact, the results obtained during the development of this thesis showed that AutoMoDe-Annealing performs differently for these two control architectures. Regarding behavior trees, results showed that higher design budgets do not increase significantly the performance of the swarm. In the best case, higher design budgets were translated in a reduction of the variance in the performance of the solutions—as an example one could compare the results for foraging with budgets of 5K and 100K. On the other hand, higher budgets were translated into a modest but noticeable increase in performance for control software in the form of finite state machines. Differences on the performance of AutoMoDe-Annealing-FSM and AutoMoDe-Annealing-BT can be justified by the difference in the search space. FSM has a bigger search space than BT **??**. Still, AutoMoDe-Annealing was capable of producing appropriate control software in both cases—an indicator of the generalization properties of the method. When only the design budget was considered, AutoMoDe-Annealing converged to stable solutions with budgets of 25K simulations. After 25K simulations, no noticeable increase in performance was visualized neither in simulation nor in pseudo-reality.

The second conclusion I wish to draw is that indeed the design process is influenced by the selection and parametrization of the components of my simulated annealing implementation. Yet, further research is necessary to determine and concretize the nature of that influence. I studied in depth the components of simulated annealing

to better understand its potential to become an alternative optimization method for AutoMoDe. In particular, I focused my study on the acceptance criterion, exploration and restart mechanism. I studied the influence of these components over design budgets of 25K, 50K and 100K simulations. Results showed that difference on the components parametrization of the algorithm becomes relevant only in higher design budgets. Experiments with a budget of 25K simulations did not show evident differences in performance. For 50K the method starts showing some differences in performance—still, I could not find enough statistical evidence of them. Results showed that the major influence of the variation of the parameters appears in experiments with a design budget of 100K simulations. However, there is no clear pattern that could highlight the advantage of one specific parametrization of the components I considered—the results vary across different budgets, architectures and missions.

In regards to the acceptance criterion, only AutoMoDe-Annealing-FSM in aggregation showed an statistically significant improvement over the other methods considered. On the other hand, AutoMoDe-Annealing-BT was more influenced than AutoMoDe-Annealing-FSM by changes in the the window size. In both missions, AutoMoDe-Annealing-BT benefited from an increase of the window size up to fifteen instances. On the contrary, AutoMoDe-Annealing-FSM did not present significant differences neither by increasing nor by decreasing the windows size. The last component of the algorithm that I studied was the restarting mechanism. In this case, there was a notorious advantage of changing the default restarting mechanism w.r.t. the original implementation. In the first place, I implemented an obligatory restarting mechanism after ever 5K simulations. After conducting my study, I was able to determine that its better to promote a more exploitation-based optimization algorithm with non-restarting or a restarting-once mechanism.

The third conclusion I wish to draw is that there is a strong evidence that automatic design methods based on local search optimization—such as simulated annealing—can perform as good as standard state-of-the-art design methods. In my experiments, AutoMoDe-Annealing-BT performed similarly to state-of-the-art methods and AutoMoDe-Annealing-FSM was able to outperform them. Despite of being an empirical proof obtained under certain hypothesis, it is a step further to bring diversity into the optimization methods available to the the off-line automatic modular design of robot swarms—so far restricted to rank-based and evolutionary optimization algorithms. I compared AutoMoDe-Annealing-BT and AutoMoDe-Annealing-FSM against two standard design methods—AutoMoDe-Chocolate and AutoMoDe-Maple—and one under-development instance of AutoMoDe based on iterated improvement—AutoMoDe-Iterated. In overall, results showed that AutoMoDe-Annealing and AutoMoDe-Iterated perform similarly, and in some cases better, than the standard methods. Differences are more evident in a per-mission basis. In aggregation, AutoMoDe-Annealing-FSM performed similarly and AutoMoDe-Iterated performed significantly better than AutoMoDe-Maple, and AutoMoDe-Chocolate. On the other hand, AutoMoDe-Annealing-BT performed similarly to AutoMoDe-Chocolate and AutoMoDe-Maple. In foraging, AutoMoDe-Annealing-FSM performed significantly better than AutoMoDe-Chocolate and slightly better than AutoMoDe-Maple in

lower design budgets. AutoMoDe-Annealing-BT performed similarly to AutoMoDe-Chocolate and AutoMoDe-Maple without being affected by the simulation budget. In all cases, the control software was produced by the default configuration of AutoMoDe-Annealing. It would be expected, that better results could be obtained after studying further the influence of the components in the simulated annealing algorithm. For example, the restarting mechanism could be changed in order to promote an exploitation in the optimization algorithm. That could reduce the variance in the performance of AutoMoDe-Annealing in large budgets.

A central point of the automatic modular design is to provide an alternative to the simulation over-fitting of control software produced by means of neuro-evolution. I used pseudo-reality as an indicator to understand how AutoMoDe-Annealing could be affected by the reality gap. Still, a remaining step on my research is the assessment of the control software, I produced, on real robots. I will devote future work to implement, analyze and present the true capabilities of design methods based on local search to surpass the reality gap in the automatic generation of control software for robot swarms. Alongside, I will extend the scope of my experimental set-up by increase the number of components, possible parametrization, and missions I consider to assess AutoMoDe-Annealing. It is my contention that a further exploration of the algorithm properties would bring clarity to the capabilities of simulated annealing as an alternative to the design of robot swarms by optimization.

Appendices

Appendix A

Low and High Budget Analysis

In this appendix shows the budget analysis from another perspective. It shows the budgets categorized in two groups: low budgets and high budgets. It also contains the tables with results obtained when assessing AutoMoDe-Annealing.

A.0.1 Low budgets

This section shows the results obtained when assessing AutoMoDe-Annealing for low budgets: 5K and 10K simulations.

Figure A.1 shows AutoMoDe-Annealing-BT and AutoMoDe-Annealing-FSM in two different mission. It is interesting to observe that Annealing-BT performs slightly better than Annealing-FSM.

In Table A.1 we can observe a statistical summary of the results for low budgets. In Table A.2 we observe the results over the 20 generated controllers for each control software atrchitecture and missions.

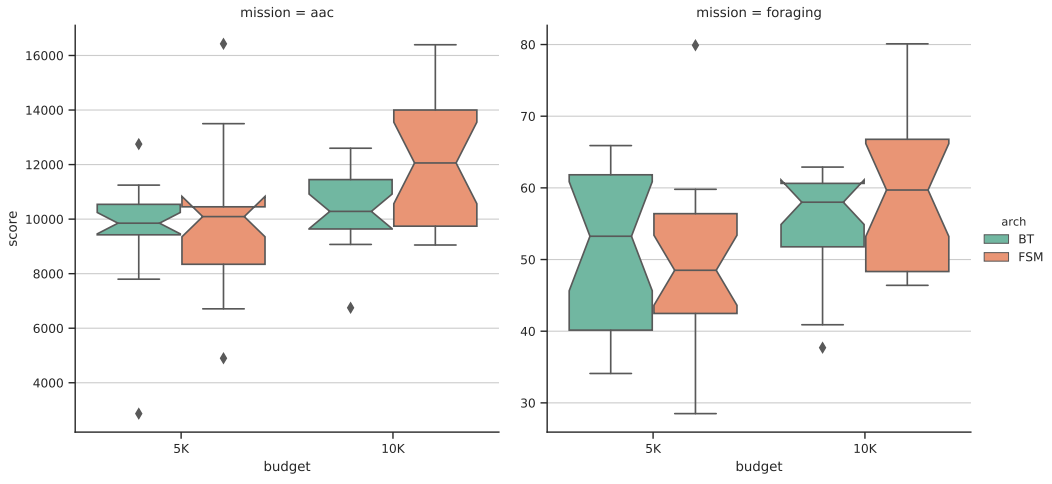


FIGURE A.1: Notch box-plot for low budgets.

	(A) AAC				(B) FOR			
	BT5K	BT10K	FSM5K	FSM10K	BT5K	BT10K	FSM5K	FSM10K
count	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00
mean	9642.00	10420.05	9971.26	11990.52	51.30	55.02	49.64	60.39
std	1914.39	1359.17	2603.80	2356.86	11.45	7.56	11.11	11.47
min	2868.60	6749.10	4898.90	9051.00	34.10	37.70	28.50	46.40
25%	9425.10	9638.75	8344.22	9741.45	40.15	51.78	42.48	48.32
50%	9851.20	10283.55	10092.50	12059.55	53.25	58.00	48.50	59.70
75%	10541.90	11449.30	10451.55	14000.50	61.82	60.62	56.40	66.77
max	12750.40	12599.60	16427.80	16393.60	65.90	62.90	79.90	80.10

Table A.1: Low budget summary table.

	(A) AAC				(B) FOR			
	BT5K	BT10K	FSM5K	FSM10K	BT5K	BT10K	FSM5K	FSM10K
0	7798.0	11796.0	7031.3	9528.9	54.5	60.6	40.0	61.1
1	9526.0	9597.5	10252.6	13965.3	40.0	53.8	39.1	46.4
2	9642.9	9652.5	9603.0	12983.1	48.6	58.7	44.5	61.7
3	10028.1	9806.5	13490.9	10311.8	35.8	37.7	43.8	48.9
4	12750.4	9516.9	10110.0	9221.5	62.6	60.3	56.1	58.1
5	10885.7	10715.1	8540.7	10244.8	49.4	60.3	79.9	46.6
6	10157.6	9071.9	7754.8	12860.8	62.5	42.2	59.8	46.6
7	10744.5	9928.3	4898.9	14605.7	61.3	62.9	55.5	66.6
8	9342.9	10906.2	9925.1	14106.1	34.1	61.4	54.4	56.9
9	10486.4	11775.1	10369.0	10081.0	65.9	60.7	38.7	67.3
10	9541.0	12238.2	10129.4	9051.0	35.6	62.4	44.3	59.6
11	2868.6	10619.0	12066.6	9675.9	60.2	48.7	58.2	80.1
12	9674.3	11918.0	7754.8	15080.7	41.2	56.4	52.5	77.5
13	11247.0	9948.1	10075.0	12830.8	61.6	52.2	42.4	77.3
14	10450.2	10955.3	9881.9	16393.6	57.3	53.3	28.5	46.5
15	8306.1	12599.6	10203.5	9524.0	64.3	50.5	57.6	46.5
16	9452.5	9388.7	6710.6	11288.3	40.2	40.9	57.3	78.3
17	9115.5	11340.7	10699.2	13488.6	35.8	61.4	42.5	58.5
18	10708.4	6749.1	16427.8	9763.3	63.0	58.4	43.4	63.4
19	10113.9	9878.2	13500.1	14805.3	52.0	57.6	54.3	59.8

Table A.2: Performance over 20 different instance of control software generated with AutoMoDe-Annealing.

Run-time analysis

In Figure A.2 we can observe that Annealing-BT and Annealing-FSM behaves similarly. However for 10K simulations FSM seems to get an little advantage at the end of the run. That advantage is not reflected in the assessment, most likely because in the assessment Annealing-FSM lose more quality.

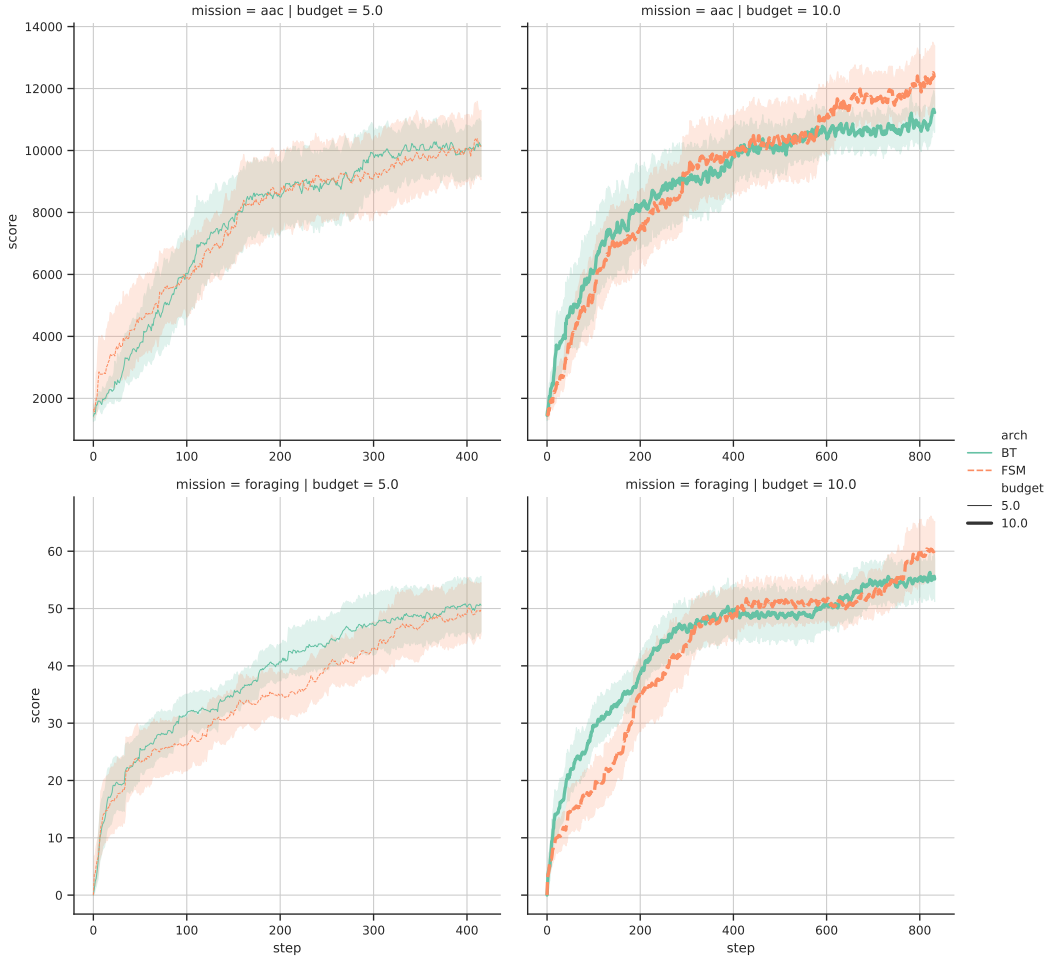


FIGURE A.2: Run time analysis low budgets.

A.0.2 High budgets

This section contains the results of assessing AutoMoDe-Annealing for high budget: 25K, 50K and 100K.

Figure A.3 shows that the difference between Annealing-FSM and Annealing-BT is bigger when increasing the budget for both missions.

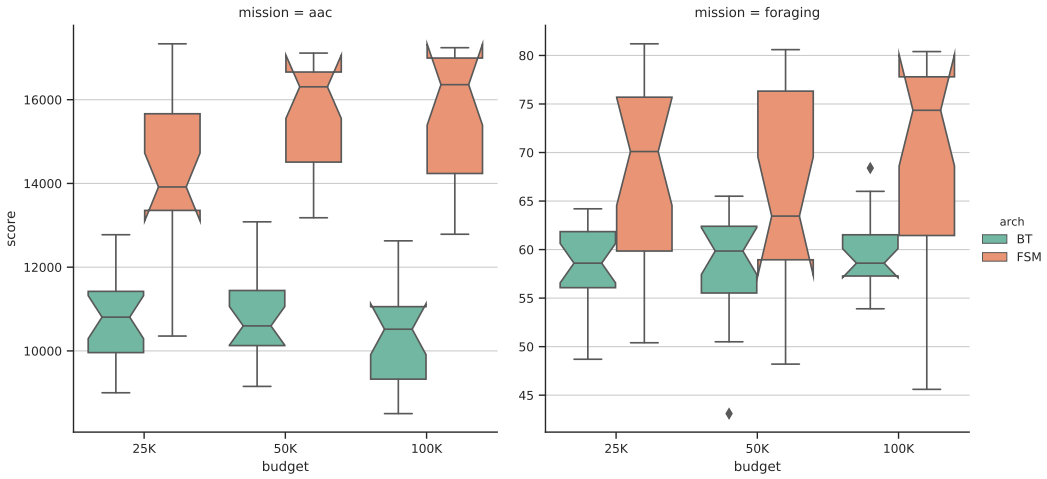


Figure A.3: Notch box-plot for high budgets.

AAC

Table A.3 shows a statistical summary of the results obtained when assessing AutoMoDe-Annealing for the aggregation mission. Table A.4 show the results

	(A) BT			(B) FSM		
	BT25K	BT50K	BT100K	FSM25K	FSM50K	FSM100K
count	20.00	20.00	20.00	20.00	20.00	20.00
mean	10735.44	10822.67	10377.99	14299.07	15561.46	15819.14
std	1095.27	1028.23	1129.67	1868.06	1351.38	1498.11
min	9000.20	9152.20	8502.20	10354.80	13180.20	12786.70
25%	9959.45	10127.20	9325.80	13356.02	14510.03	14237.82
50%	10806.75	10597.65	10518.60	13915.95	16307.85	16359.65
75%	11422.02	11443.10	11056.98	15664.70	16661.03	16994.90
max	12774.90	13083.20	12629.00	17335.20	17113.70	17242.30

Table A.3: High budget summary table for AAC.

of the 20 instances of control software generated with AutoMoDe-Annealing for the aggregation mission.

Foraging

Table A.5 shows a statistical summary of the results obtained when assessing AutoMoDe-Annealing for the foraging mission. Table A.6 show the results of the 20 instances of control software generated with AutoMoDe-Annealing for the foraging mission.

	(A) BT			(B) FSM		
	BT25K	BT50K	BT100K	FSM25K	FSM50K	FSM100K
0	12007.5	10271.8	9494.2	10354.8	16929.9	17242.3
1	10129.6	11676.5	10425.2	13615.0	16277.2	17134.4
2	10287.1	11365.3	12629.0	15966.8	16565.6	16479.0
3	11178.1	10298.3	8804.1	13776.1	17113.7	16240.3
4	11897.9	11973.5	10850.7	13385.3	13399.2	17135.8
5	9606.6	12436.7	12111.5	13745.3	16624.9	16892.7
6	9797.3	9152.2	9294.9	14189.8	16763.9	15812.5
7	11082.9	11010.5	9283.2	13914.3	16694.4	17033.3
8	9082.9	10413.9	9336.1	13268.2	13315.2	13890.3
9	9530.6	12201.7	10719.6	13172.7	16649.9	16963.9
10	10700.7	10781.4	11704.8	14017.0	14097.3	16828.1
11	10013.5	9620.4	10611.8	13917.6	16746.6	15885.9
12	11263.4	10176.1	10425.4	15564.0	14742.4	14061.5
13	10912.8	11040.5	11157.1	17206.6	15402.5	16982.1
14	12774.9	10799.2	9738.0	14821.0	16430.4	14097.5
15	12208.4	13083.2	9270.9	13053.7	15111.7	14284.6
16	10145.7	10393.5	11324.8	16761.4	14597.2	17201.8
17	9000.2	9922.7	11023.6	11132.1	14248.5	13206.0
18	10922.9	9855.5	10852.8	16784.4	13180.2	16224.2
19	12165.9	9980.5	8502.2	17335.2	16338.5	12786.7

Table A.4: Performance over 20 different instance of control software generated with AutoMoDe-Annealing for aggregation in high budgets.

	(A) BT			(B) FSM		
	BT25K	BT50K	BT100K	FSM25K	FSM50K	FSM100K
count	20.00	20.00	20.00	20.00	20.00	20.00
mean	58.67	58.31	59.48	68.11	66.61	69.68
std	4.11	5.81	3.62	9.43	9.81	10.09
min	48.70	43.10	53.90	50.40	48.20	45.60
25%	56.08	55.52	57.28	59.85	58.95	61.45
50%	58.60	59.85	58.60	70.10	63.45	74.35
75%	61.85	62.40	61.52	75.70	76.32	77.80
max	64.20	65.50	68.40	81.20	80.60	80.40

Table A.5: High budget summary table for Foraging.

	(A) BT			(B) FSM		
	BT25K	BT50K	BT100K	FSM25K	FSM50K	FSM100K
0	58.8	51.4	56.4	71.6	59.0	62.2
1	63.6	62.7	62.6	75.2	74.6	74.2
2	55.7	56.5	56.8	50.4	76.3	74.5
3	56.7	60.4	53.9	61.6	67.3	60.7
4	53.4	57.7	62.2	77.6	58.8	77.7
5	48.7	62.3	57.5	57.8	55.8	58.3
6	57.0	52.3	66.0	62.6	76.4	78.1
7	60.5	65.5	58.8	74.9	61.2	45.6
8	64.2	57.2	68.4	81.2	78.3	79.9
9	56.1	43.1	58.4	60.0	48.2	67.5
10	63.9	64.3	57.4	79.8	80.2	76.9
11	56.1	50.5	61.3	59.4	58.7	74.8
12	55.4	65.2	56.9	57.4	60.3	69.7
13	61.3	61.4	64.1	76.6	76.2	80.4
14	63.5	59.3	58.8	74.6	62.6	75.6
15	59.2	57.9	58.8	76.6	59.5	79.7
16	63.6	62.3	57.4	68.6	76.7	61.7
17	56.0	52.6	60.2	66.6	64.3	60.2
18	58.4	60.7	56.2	54.3	57.3	55.4
19	61.2	62.9	57.4	75.4	80.6	80.4

Table A.6: Performance over 20 different instance of control software generated with AutoMoDe-Annealing for foraging in high budgets.

Run-time analysis

In Figure A.4 we can observe that Annealing-BT is outperformed by Annealing-FSM in all budgets. The advantage of Annealing-FSM with respect to Annealing-BT increases when the budget increases. For the aggregation mission the difference is more clear than for foraging.

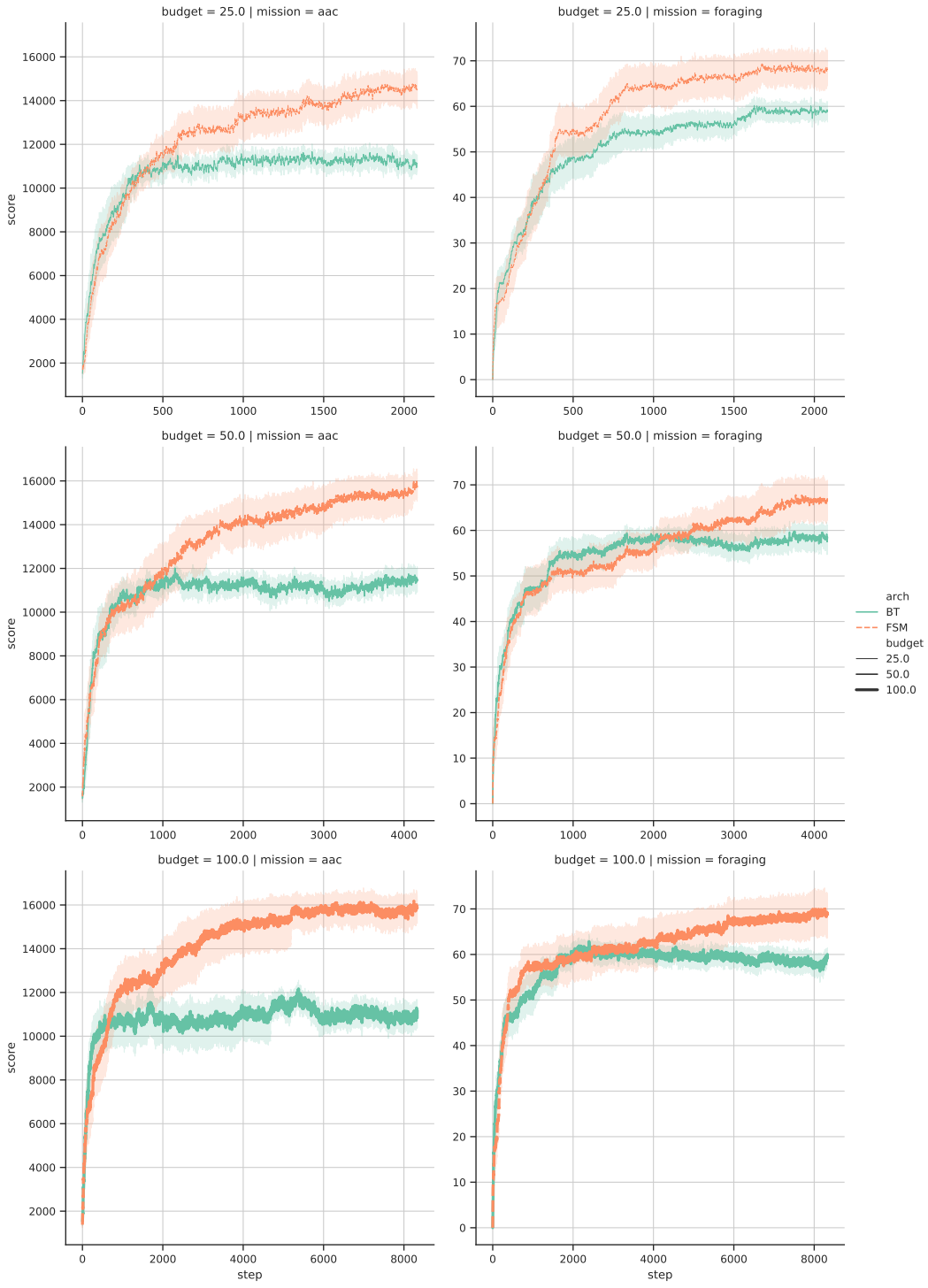


Figure A.4: Run time analysis high budgets.

A.0.3 All budgets

In Figure A.5 we can observe how the budget affects Annealing-BT and Annealing-FSM. Annealing-BT doesn't show a major impact despite of the decrease of dispersion in the foraging mission. On the other hand Annealing-FSM keeps increasing for bigger differences in budget.

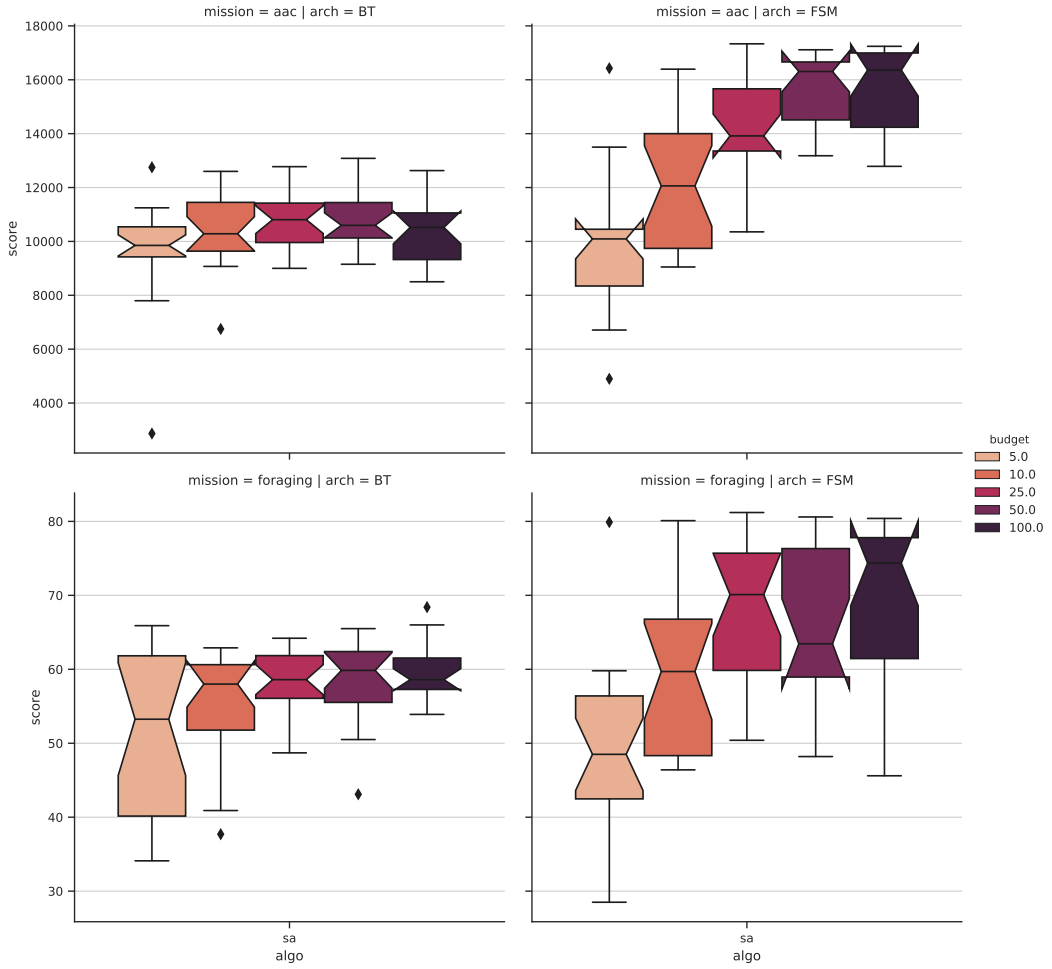


FIGURE A.5: Box plots for all budgets.

Appendix B

Parameters run time analysis

This appendix contains the results in run time for the parameters study presented in Chapter 6. The appendix is divided in parameters. And the run-time analysis are categorized by control software architecture.

B.1 Acceptance criterion

In Figure B.1 and Figure B.2 we can observe the performance of the control software generated by AutoMoDe-Annealing in run time. The figures show that the performance is rather similar. However Wilcoxon test looks less noisy than the other criteria.

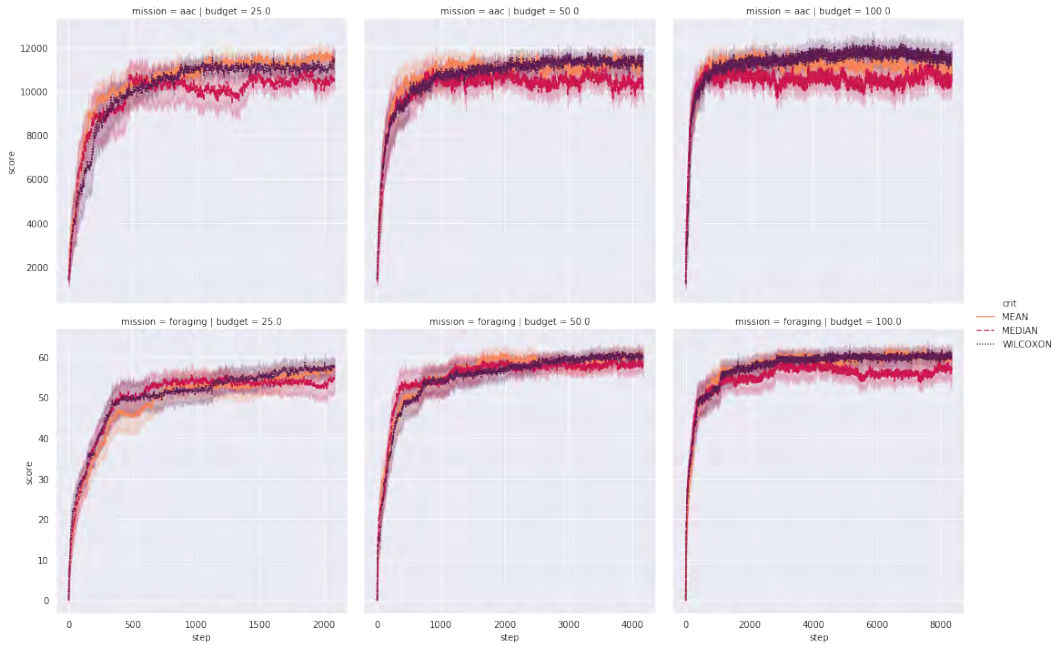


FIGURE B.1: Annealing-BT run time performance, for acceptance criterion.

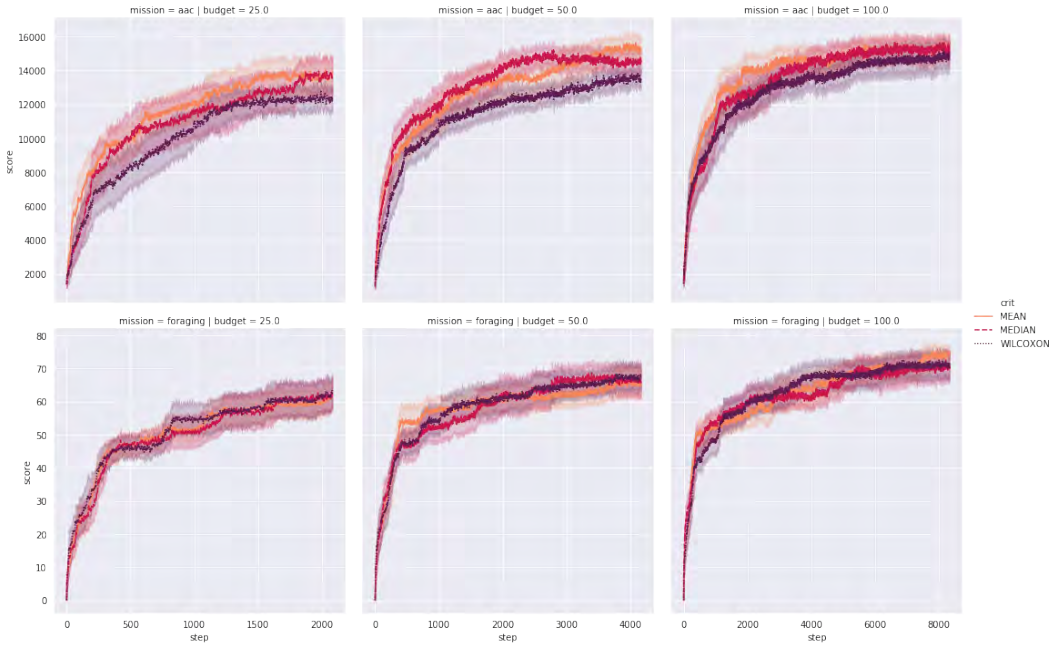


Figure B.2: Annealing-FSM run time performance, for acceptance criterion.

B.2 Window size

The performance of AutoMoDe-Annealing when using different window sizes values is shown in Figure B.3 and Figure B.4. We can observe that the biggest the windows the less attempts of improvement the algorithm will do. However in all the cases a window size of 15 seems to be superior to the others.

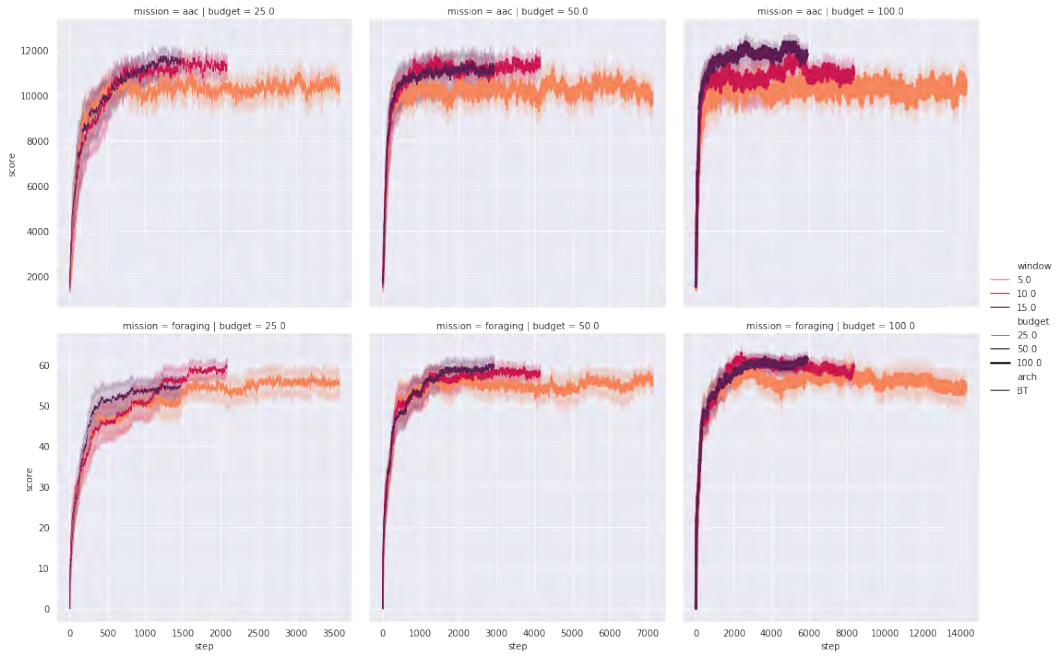


FIGURE B.3: Annealing-BT run time performance, for window size.

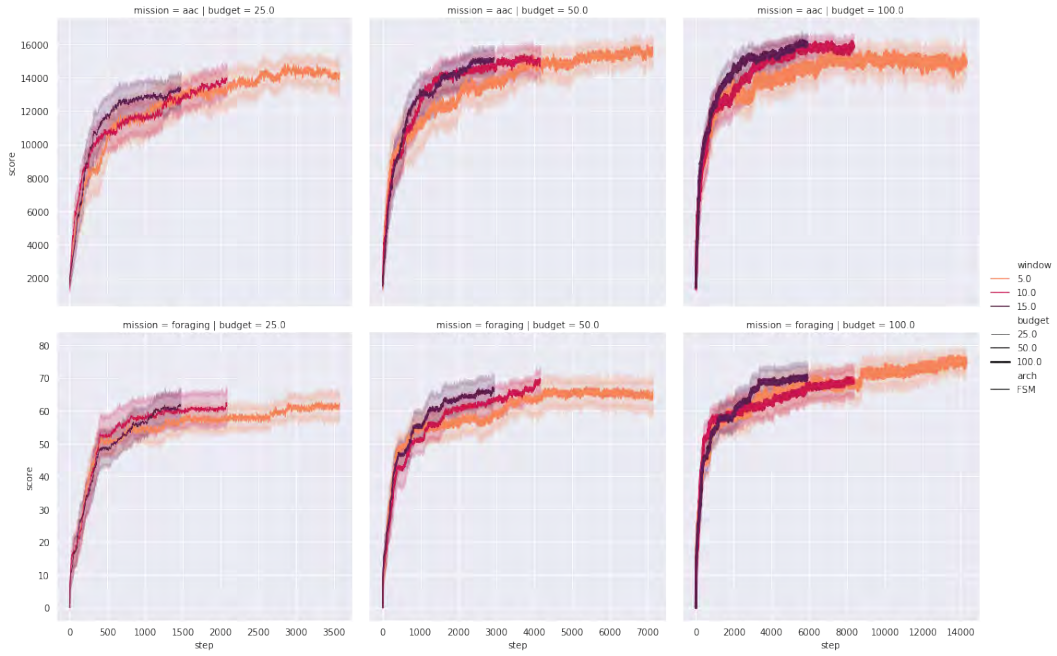


FIGURE B.4: Annealing-FSM run time performance, for window size.

B.3 Restart mechanism

The performance of AutoMoDe-Annealing when using different restart mechanism is shown in Figure B.5 and Figure B.6. We can observe that the for aggregation all the mechanisms perform quite similar. However we can observe a different trend in the forging mission. We observe that Not restarting take more time to converge but at the end it converges. Also it is shown that the algorithm converges in approximately 2000 time steps.

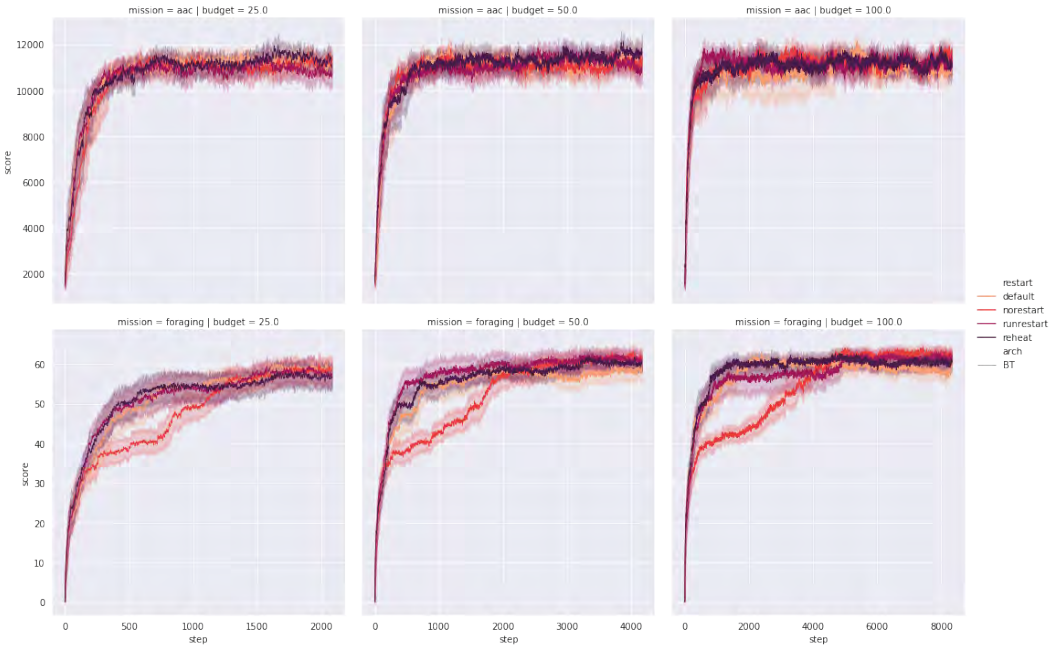


Figure B.5: Annealing-BT run time performance, for restart mechanism.

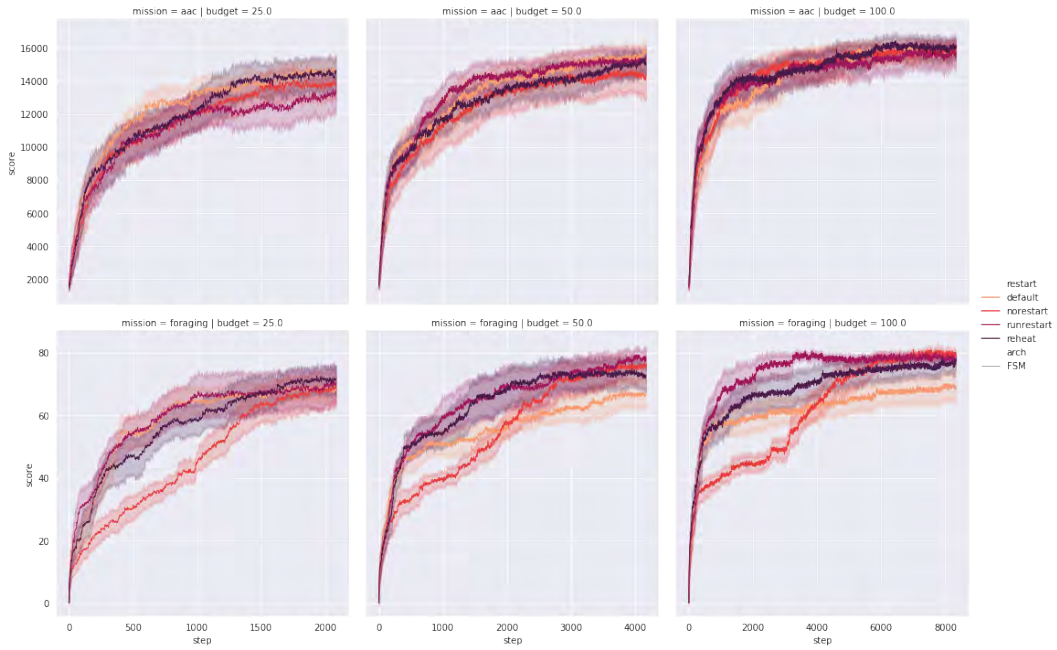


FIGURE B.6: Annealing-FSM run time performance, for restart mechanism.

Bibliography

[1] E. Aarts, J. Korst, and W. Michiels. *Simulated Annealing*, pages 187–210. Springer US, Boston, MA, 2005.

[2] P. Balaprakash, M. Birattari, and T. Stützle. Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In *Hybrid Metaheuristics, 4th International Workshop, HM 2007*, volume 4771 of *LNCS*, pages 108–122. Springer, Berlin, Germany, 2007.

[3] G. Baldassarre, V. Trianni, M. Bonani, F. Mondada, M. Dorigo, and S. Nolfi. Self-organized coordinated motion in groups of physically connected robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(1):224–239, 2007.

[4] G. Beni and J. Wang. Swarm intelligence in cellular robotic systems, proceed. nato advanced workshop on robots and biological systems, tuscany, italy, june 26-30. *Y.: NATO*, 1989.

[5] M. Birattari. *Tuning Metaheuristics: A Machine Learning Perspective*. Springer, Berlin Heidelberg, Germany, 2009.

[6] M. Birattari, A. Ligot, D. Bozhinoski, M. Brambilla, G. Francesca, L. Garattoni, D. Garzón Ramos, K. Hasselmann, M. Kegeleirs, J. Kuckling, et al. Automatic off-line design of robot swarms: A manifesto. *Frontiers in Robotics and AI*, 6:59, 2019.

[7] C. Blum and X. Li. Swarm intelligence in optimization. In *Swarm intelligence*, pages 43–85. Springer, 2008.

[8] E. Bonabeau, D. d. R. D. F. Marco, M. Dorigo, G. Theraulaz, et al. *Swarm intelligence: from natural to artificial systems*. Number 1. Oxford university press, 1999.

[9] J. Bongard. Evolutionary robotics, vol. 56, 2013.

[10] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013.

[11] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013.

[12] N. Bredeche, J.-M. Montanier, W. Liu, and A. F. Winfield. Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. *Mathematical and Computer Modelling of Dynamical Systems*, 18(1):101–129, 2012.

[13] R. Brooks. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, 2(1):14–23, 1986.

[14] A. Brutschy, L. Garattoni, M. Brambilla, G. Francesca, G. Pini, M. Dorigo, and M. Birattari. The tam: abstracting complex tasks in swarm robotics research. *Swarm Intelligence*, 9(1):1–22, 2015.

[15] V. Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1):41–51, 1985.

[16] J. Chambers, W. Cleveland, B. Kleiner, and P. Tukey. *Graphical Methods For Data Analysis*. Wadsworth, Belmont CA, 1983.

[17] A. L. Christensen and M. Dorigo. Evolving an integrated phototaxis and hole-avoidance behavior for a swarm-bot. In *Proceedings of the 10th International Conference on the Simulation and Synthesis of Living Systems (Alife X)*, pages 248–254. Citeseer, 2006.

[18] R. J. Clark, R. C. Arkin, and A. Ram. Learning momentum: online performance enhancement for reactive systems. In *Proceedings 1992 IEEE International Conference on Robotics and Automation*, pages 111–116. IEEE, 1992.

[19] G. Di Caro. Ant colony optimization and its application to adaptive routing in telecommunication networks. 2004.

[20] G. A. Di Caro, F. Ducatelle, and L. M. Gambardella. Theory and practice of ant-based routing in dynamic telecommunication networks. In *Reflexing interfaces: The complex coevolution of information technology ecosystems*, pages 185–216. IGI Global, 2008.

[21] S. Doncieux and J.-B. Mouret. Beyond black-box optimization: a review of selective pressures for evolutionary robotics. *Evolutionary Intelligence*, 7(2):71–93, 2014.

[22] M. Dorigo and M. Birattari. Swarm intelligence. *Scholarpedia*, 2(9):1462, 2007. revision #138640.

[23] M. Dorigo, M. Birattari, and M. Brambilla. Swarm robotics. *Scholarpedia*, 9(1):1463, 2014.

[24] M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, et al. Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics & Automation Magazine*, 20(4):60–71, 2013.

[25] M. Dorigo, V. Maniezzo, A. Colorni, et al. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, man, and cybernetics, Part B: Cybernetics*, 26(1):29–41, 1996.

[26] M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications, and advances. In *Handbook of metaheuristics*, pages 250–285. Springer, 2003.

[27] M. Dorigo, V. Trianni, E. Şahin, R. Groß, T. H. Labella, G. Baldassarre, S. Nolfi, J.-L. Deneubourg, F. Mondada, D. Floreano, et al. Evolving self-organizing behaviors for a swarm-bot. *Autonomous Robots*, 17(2-3):223–245, 2004.

[28] M. Duarte, S. Oliveira, and A. Christensen. Evolution of hierarchical controllers for multirobot systems. In *Artificial Life Conference Proceedings 14*, pages 657–664. MIT Press, 2014.

[29] M. Duarte, S. Oliveira, and A. Christensen. Hybrid control for large swarms of aquatic drones. In *Artificial Life Conference Proceedings 14*, pages 785–792. MIT Press, 2014.

[30] E. Ferrante, A. E. Turgut, E. Duéñez-Guzmán, M. Dorigo, and T. Wenseleers. Evolution of self-organized task specialization in robot swarms. *PLoS computational biology*, 11(8):e1004273, 2015.

[31] G. Francesca and M. Birattari. Automatic design of robot swarms: achievements and challenges. *Frontiers in Robotics and AI*, 3(29):1–9, 2016.

[32] G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, and M. Birattari. Automode: A novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence*, 8(2):89–112, 2014.

[33] G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, and M. Birattari. AutoMoDe: A novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence*, 8(2):89–112, 2014.

[34] A. Franzin and T. Stützle. Revisiting simulated annealing: A component-based analysis. *Computers & operations research*, 104:191–206, 2019.

[35] R. Fujisawa, S. Dobata, K. Sugawara, and F. Matsuno. Designing pheromone communication in swarm robotics: Group foraging behavior mediated by chemical substance. *Swarm Intelligence*, 8(3):227–246, 2014.

[36] F. Glover and M. Laguna. Tabu search. In *Handbook of combinatorial optimization*, pages 2093–2229. Springer, 1998.

[37] D. E. Goldberg and J. H. Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

[38] J. Gomes, P. Urbano, and A. L. Christensen. Evolution of swarm robotics systems with novelty search. *Swarm Intelligence*, 7(2-3):115–144, 2013.

[39] E. Haasdijk, N. Bredeche, and A. Eiben. Combining environment-driven adaptation and task-driven optimisation in evolutionary robotics. *PloS one*, 9(6):e98466, 2014.

[40] H. Hamann. Towards swarm calculus: Universal properties of swarm performance and collective decisions. In *Swarm Intelligence*, pages 168–179, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[41] K. Hasselmann, A. Ligot, G. Francesca, and M. Birattari. Reference models for AutoMoDe. Technical Report TR/IRIDIA/2018-002, IRIDIA, Université libre de Bruxelles, Belgium, 2018.

[42] K. Hasselmann, F. Robert, and M. Birattari. Automatic design of communication-based behaviors for robot swarms. In *International Conference on Swarm Intelligence*, pages 16–29. Springer, 2018.

[43] S. Hauert, J.-C. Zufferey, and D. Floreano. Evolved swarming without positioning information: an application in aerial communication relay. *Autonomous Robots*, 26(1):21–32, 2009.

[44] J. H. Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[45] H. Hoos and T. Stüzle. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann, San Francisco CA, 2004.

[46] N. Jakobi, P. Husbands, and I. Harvey. Noise and the reality gap: the use of simulation in evolutionary robotics. In F. Morán and et al., editors, *Advances in Artificial Life*, volume 929 of *LNCS*, pages 704–720, London, UK, 1995. Springer.

[47] F. Janabi-Sharifi and D. Vinke. Integration of the artificial potential field approach with simulated annealing for robot path planning. In *Proceedings of 8th IEEE International Symposium on Intelligent Control*, pages 536–541. IEEE, 1993.

[48] D. S. Johnson and M. R. Garey. *Computers and intractability: A guide to the theory of NP-completeness*, volume 1. WH Freeman San Francisco, 1979.

[49] S. Kazadi et al. Swarm engineering. 2000.

[50] J. Kennedy. Particle swarm optimization. *Encyclopedia of machine learning*, pages 760–766, 2010.

[51] S. Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of statistical physics*, 34(5-6):975–986, 1984.

[52] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[53] J. Kuckling, A. Ligot, D. Bozhinoski, and M. Birattari. Behavior trees as a control architecture in the automatic modular design of robot swarms. In M. Dorigo and et al., editors, *Swarm Intelligence, ANTS*, volume 11172 of *LNCS*, pages 30–43. Springer, Cham, Switzerland, 2018.

[54] J. Kuckling, A. Ligot, D. Bozhinoski, and M. Birattari. Search space for AutoMoDe-`Chocolate` and AutoMoDe-`Maple`. Technical Report TR/IRIDIA/2018-012, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2018.

[55] T. H. Labella, M. Dorigo, and J.-L. Deneubourg. Division of labor in a group of robots inspired by ants' foraging behavior. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 1(1):4–25, 2006.

[56] J. B. Lee and R. C. Arkin. Adaptive multi-robot behavior via learning momentum. In C. S. George Lee, editor, *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 2029–2036, Piscataway NJ, 2003. IEEE Press.

[57] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[58] A. Ligot and M. Birattari. On mimicking the effects of the reality gap with simulation-only experiments. In M. Dorigo and et al., editors, *Swarm Intelligence, ANTS*, volume 11172 of *LNCS*, pages 109–122. Springer, Cham, Switzerland, 2018.

[59] A. Ligot, K. Hasselmann, B. Delhaisse, L. Garattoni, G. Francesca, and M. Birattari. AutoMoDe, NEAT, and EvoStick: implementations for the e-puck robot in ARGoS3. Technical Report TR/IRIDIA/2017-002, IRIDIA, Université libre de Bruxelles, Belgium, 2017.

[60] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[61] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[62] H. Martınez-Alfaro and S. Gomez-Garcıa. Mobile robot path planning and tracking using simulated annealing and fuzzy logic control. *Expert Systems with Applications*, 15(3-4):421–429, 1998.

[63] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren. Towards a unified behavior trees framework for robot control. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5420–5427. IEEE, 2014.

[64] M. Matarić and D. Cliff. Challenges in evolving controllers for physical robots. *Robotics and autonomous systems*, 19(1):67–83, 1996.

[65] M. J. Matarić. Learning in behavior-based multi-robot systems: policies, models, and other agents. *Cognitive Systems Research*, 2(1):81–93, 2001.

[66] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[67] H. Miao and Y.-C. Tian. Dynamic robot path planning using an enhanced simulated annealing approach. *Applied Mathematics and Computation*, 222:420–437, 2013.

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

[69] F. Mondada, G. C. Pettinaro, A. Guignard, I. W. Kwee, D. Floreano, J.-L. Deneubourg, S. Nolfi, L. M. Gambardella, and M. Dorigo. Swarm-bot: A new distributed robotic concept. *Autonomous robots*, 17(2-3):193–221, 2004.

[70] A. G. Nikolaev and S. H. Jacobson. *Simulated Annealing*, pages 1–39. Springer US, Boston, MA, 2010.

[71] N. J. Nilsson. Shakey the robot. Technical report, SRI INTERNATIONAL MENLO PARK CA, 1984.

[72] S. Nolfi, D. Floreano, and D. D. Floreano. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press, 2000.

[73] M. G. Park, J. H. Jeon, and M. C. Lee. Obstacle avoidance for mobile robots using artificial potential field approach with simulated annealing. In *ISIE 2001. 2001 IEEE International Symposium on Industrial Electronics Proceedings (Cat. No. 01TH8570)*, volume 3, pages 1530–1535. IEEE, 2001.

[74] L. E. Parker. Task-oriented multi-robot learning in behavior-based systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 1478–1487, Piscataway NJ, 1996. IEEE Press.

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

[76] J. Pugh and A. Martinoli. Distributed scalable multi-robot learning using particle swarm optimization. *Swarm Intelligence*, 3(3):203–222, 2009.

[77] M. Quinn, L. Smith, G. Mayley, and 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):2321–2343, 2003.

[78] E. Şahin. Swarm robotics: From sources of inspiration to domains of application. In E. Şahin and W. M. Spears, editors, *Swarm Robotics, SAB*, volume 3342 of *LNCS*, pages 10–20, Berlin Heidelberg, Germany, 2004. Springer.

[79] E. Şahin. Swarm robotics: From sources of inspiration to domains of application. In *International workshop on swarm robotics*, pages 10–20. Springer, 2004.

[80] A. Seeni, B. Schäfer, and G. Hirzinger. Robot mobility systems for planetary surface exploration–state-of-the-art and future outlook: a literature survey. *Aerospace Technologies Advancements*, page 492, 2010.

[81] F. Silva, M. Duarte, L. Correia, S. M. Oliveira, and A. L. Christensen. Open issues in evolutionary robotics. *Evolutionary computation*, 24(2):205–236, 2016.

[82] K. Sugawara and T. Watanabe. Swarming robots-foraging behavior of simple multirobot system. In *IEEE/RSJ International Conference on intelligent robots and systems*, volume 3, pages 2702–2707. IEEE, 2002.

[83] R. S. Tavares, T. Martins, and M. d. S. G. Tsuzuki. Simulated annealing with adaptive neighborhood: A case study in off-line robot path planning. *Expert Systems with Applications*, 38(4):2951–2965, 2011.

[84] V. Trianni. *Evolutionary swarm robotics: evolving self-organising behaviours in groups of autonomous robots*, volume 108. Springer, 2008.

[85] V. Trianni. Evolutionary robotics: model or design? *Frontiers in Robotics and AI*, 1:13, 2014.

[86] M. Waibel, L. Keller, and D. Floreano. Genetic team composition and level of selection in the evolution of cooperation. *IEEE Transactions on Evolutionary Computation*, 13(3):648–660, 2009.

[87] R. Watson, S. Ficici, and J. Pollack. Embodied evolution: distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1–18, 2002.

[88] A. F. Winfield, C. J. Harper, and J. Nembrini. Towards dependable swarms and a new discipline of swarm engineering. In *International Workshop on Swarm Robotics*, pages 126–142. Springer, 2004.