VRIJE
UNIVERSITEIT
BRUSSEL

Master thesis submitted in partial fulfilment of the requirements for the degree of
Master of Science in Applied Sciences and Engineering: Computer Science

IMPORTANCE SAMPLING WITH INTERMEDIATE REWARDS TO ESTIMATE THE PERFORMANCE OF A ROBOT SWARM

Iulian-Bogdan Vlad

2020-2021

Promotors: Prof. Dr. Mauro Birattari, Dr. Federico Pagnozzi

Science and Bio-Engineering Sciences

DEMIURGE
Automatic Design
of Robot Swarms
An ERC Consolidator Project

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

Abstract

This thesis proposes new methods that use importance sampling with discounted intermediate rewards in order to predict the performance of a robot swarm control software.

In swarm robotics the process of developing control software is an iterative one because of its decentralized nature. An existing approach is to treat the control software design problem as an optimization problem. By using this approach, the control software is automatically created by an optimization algorithm that tries to maximize the score obtained by the swarm. If the performance could be accurately predicted, this could improve the automatic design process and generate better control software.

The technique proposed in this thesis improves on previous methods for estimating the performance of control software using importance sampling. The control software is represented as a finite-state machine and it is obtained by applying a mutation operator to an existing instance of control software. The goal is to predict the impact the mutation has on the performance of the control software. These estimation methods could be used in order to explore more efficiently the neighborhood of solutions by selecting only those ones that are proposed by the estimating module as being the highest performing. During the experiments two types of mutations were used: one where states are removed and one where the properties of the transitions are modified. Multiple experiments were performed on different missions with the goal of understanding how well the methods perform in different environments. The results show a slight improvement in the prediction and a potential for these predictive methods to be used in a local search algorithm.

Contents

List of Figures

Chapter 1

Introduction

In this thesis, new methods for predicting the performance of swarm robotics control software are proposed and a thorough analysis is performed to assess their capabilities.

In the recent years, robots have become an active and important element of our everyday life. They perform a large variety of tasks in many different fields such as constructions, medical sector, agricultural, space exploration, military and many other (Cousineau and Miura, 1998) (Brooks et al., 1990). Robots can solve simple everyday tasks in the house but they also can perform extremely complicated surgeries that can save lives (Beasley, 2012). As the robots become more prevalent, they will have to collaborate in order to accomplish even more complex tasks that can not be solved by a single robot. To successfully cooperate, robots have to communicate among themselves, such that they exchange knowledge about the environment. Moreover, they have to break a complex tasks in smaller ones and perform task planning and division of labour (Labella et al., 2006).

One source of inspiration for designing such a complex system can be found in biological systems. The intelligent behavior of swarms has been studied intensively in the past 50 years. The researchers were trying to understand how a group composed of simple individuals can create complex structures. Ants, wasps and termites are capable of building extremely complex nests. An impressive example of such a structure is the nest of the specie Apicotermes Lamani that can reach 40 cm in height (Schmidt, 1964). The nest has multiple layers and there are passages that connect these layers. Moreover, the nest is constructed in such a way that it facilitates the air flow between the nest and the outside environment. Other examples of complex tasks comprehend: bees working together to select the next best location for their nest (Passino K.M., 2006), ants finding the shortest path between the nest and a food source (Goss et al., 1989) and birds flying in flocks (Reynolds, 1987). There is a contrast between the simplicity of the individuals that compose the swarm and the complex result that emerges from their work. Initially, it was thought that each individual has the knowledge about the final result and that there is a central authority and a hierarchical structure that facilitates the organization and planning. Further research, however, showed that there is no supervision in a swarm and that the individual does not have an overview over the final result. The individuals are behaving according to simple rules that are based only on the local interactions between the members of the swarm.

Swarm intelligence is the discipline that studies these complicated systems. It is a multi-disciplinary field and it can be divided into two categories: scientific and engineering (Dorigo and Birattari, 2007). The goal of the scientific branch is to create an understanding of the mechanisms that govern swarms while the goal of the engineering branch is to leverage this understanding in order to build artificial systems that are capable of solving practical problems.

In 1990, Deneubourg et al. (1990) showed that the ants have a very high probability of picking the shortest path between the nest and a food source. Understanding the probabilistic model that explains how this is possible was an inspiration for building ant colony optimization algorithm. These algorithms are used to solve hard problems such as the Travel Salesman problem. This is an example of a successful interaction between the scientific and engineering sides of the discipline.

Swarm robotics is a subfield of robotics that takes inspiration from biological systems that displays self-organizing and decentralized behavior. Self-organization is represented by the local interactions between the members of the swarm. The local interactions that lead to positive outcomes for the swarm will be encouraged through a positive feedback loop. Swarms have certain properties that make them very appealing in the context of multi-robot systems. Swarms are composed of many individuals that are similar. The individuals are behaving according to some simple rules that are based only on the local interactions between the members of the swarm. The intelligent behavior of the swarm emerges from these local interactions. A swarm robotics system characterized by these properties will have three main advantages (Şahin, 2004).

Firstly, the hardware of each robot composing the swarm is simple. They usually have a small dimension and contain a few motors in order to be able to move and a few sensors to gather knowledge from the environment. This means that there are less components that could break and that the robots will be cheaper to produce making the swarm appealing from an economical point of view (Hilder et al., 2014).

Secondly, because of the decentralized nature of swarms, these systems are fault tolerant. A robot that is malfunctioning can be easily replaced or removed without impacting the performance of the swarm. This is possible because of the lack of central authority and self-organization that results from the local interactions.

Lastly, a swarm robotics system is scalable meaning that the size of the swarm can be modified without having to reprogram the control software of each individual. This is possible because an individual is only interacting and gathering knowledge from its neighbours and the size of the neighborhood does not increase if the size of the swarm increases.

Having these properties in mind, swarm robotics could be deployed in scenarios and in environments where a single robot could not fulfill the task. Because the swarm is fault tolerant, the robot swarm could be used to perform dangerous tasks such as clearing a mine field or search and rescue missions (Cassinis et al., 2000). The swarm can also be used to explore other planets since each individual does not need to have a model of the environment beforehand and it adapts its behavior based on the local interactions. In some missions, such as clearing toxic substances, it is hard to estimate the number of robots needed to accomplish the task and in such cases the scalability property of swarm robotics is extremely useful since the size of the swarm can be adjusted based on the requirements. In an agricultural scenario, the number of swarms deployed could be proportional to the size of the field.

Nevertheless, the swarm robotics field is at its beginning and more work has to be done until these real life scenarios become reality. There are certain standard tasks that are used by researchers to compare the performance of different control softwares. Some examples are: reaching consensus regarding what place to aggregate in, finding food and bringing it to the nest, obstacle avoidance, and forming a certain formation. These tasks are called missions and even though they seem very simple, they do not lack practical importance. Some of the real scenarios presented above can be achieved by splitting the hard problems in smaller tasks such as the ones described above. In (Liekna et al., 2014) the author describes how a complicated practical application can be divided in multiple tasks that can be solved by swarms sequentially or in parallel.

Writing the control software for a swarm of robots is not a straightforward task due to

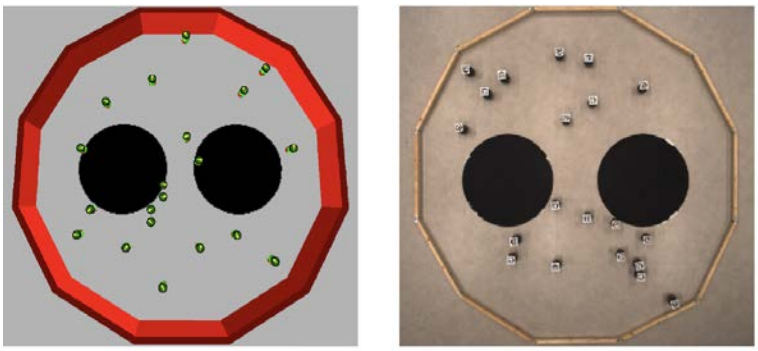

Figure 1.1: Arena for aggregation mission.
On the left side, the arena is represented in the simulator and on the right side it is the physical version of it. (Francesca et al., 2014)

decentralised behavior and self-organization of the robots. The control software is written for one robot and it is deployed on all robots composing the swarm. It is not always clear what each individual of the swarm should do in order for the swarm as a whole to display the emergent behavior. The local interactions from which the behavior of the swarm emerges can not be programmed explicitly. What happens often is that the designer of a swarm will program the control software such that it increases the probability of local interactions that leads to positive outcomes and decreases the probability of negative outcomes. The whole process is an iterative one that requires multiple executions and adjustments until the desired behavior is achieved.

A simplified example will be used to better illustrate this process: a swarm has to reach a consensus regarding the best place to gather in. One possible solution to develop a control software for this mission is to program each robot to stay in a certain place proportional to the quality of the place and to the number of neighbours encountered. By doing this, the local interactions of staying together will be encouraged and staying in a bad place will be discouraged since the robot will spend less time there. In this example, the amount of time the robot spend in a place is a parameter of the algorithm that has to be fine-tuned.

Because of these challenges there is not a clear methodology on how to develop control software for swarm robotics. Brambilla et al. (2013a) categorize the available design methods in behavior-based design methods and automatic design methods.

The behavior based design methods represent the manual way of designing the control software. This is a trial and error process where the software is developed, deployed and then the behavior of the swarm is observed. If the results are satisfactory, the process ends, otherwise the parameters and the implementation are fined tuned and the process is repeated.

The automatic design methods will create the control software without input from the developer. In single robot systems, a very popular automatic design method is Reinforcement Learning (Sutton and Barto, 2018). Using this method, the robot is acting in an environment and it receives rewards based on how well it achieves the task. The goal of the robot is to maximize the rewards and its actions will be chosen accordingly. Nevertheless, this approach does not fit well in the context of swarm systems since the reward is received for the entire group of robots and there is no clear way how to distribute it to each peer in order to facilitate the learning at the individual level. An approach that is better suited for swarms systems and which is used frequently in the literature is evolutionary robotics (Dorigo et al., 2004). This method is inspired from Darwinian principles of natural selection. The control software is represented by

a neural network that maps sensor data to inputs given to motors and actuators. The weights of the network are mutated and selected based on the performance of the swarm. This method gives good results for certain missions, however it has its own drawbacks. Robots are trained using a simulator that reproduces with a certain degree of similarity the real life conditions. Because of the representational power of a neural network, it is possible that it will overfit the simulation and a performance gap will be observed between the simulation and reality (Floreano et al., 2008).

AutoMoDe (Francesca et al., 2015) is an automatic design method that tries to solve the reality gap issue. In AutoMoDe-Vanilla, the control software is represented by a probabilistic finite-state machine which is obtained using an optimization process. In this finite-state machine, a state is represented by a behavior that the robot is performing. This behavior is programmed beforehand in existing modules. The robot can transition from one state to another based on different parameters and observation from the environment. The finite-state machine is the result of an optimization algorithm that explores the search space of all possible finite-state machines and selects the ones that have the highest performance. AutoMoDe contains multiple optimization algorithms that can be used.

A general problem of the automatic design methods is that in order to evaluate the performance of a control software, the simulation has to be executed which means that the whole optimization process can take very long time in order to achieve good results.

In order to speed up the optimization process, a predicting method was proposed (Pagnozzi and Birattari, 2021). Starting from an existing finite-state machine, the method tries to estimate the performance of a new finite-state machine that is obtained by modifying transitions parameters or by removing one state. The method uses off-policy evaluation to estimate the performance of the new finite-state machine based on the data collected from optimization process that generated the original finite-state machine. The collected data contains information such as the state each robot was at any given time during the execution of the simulation, the transition it has performed and the sensor readings.

In the current thesis, new methods of predicting the performance of a control software represented as finite-state machine are proposed and evaluated. The new methods are leveraging more information from the collected data to improve the accuracy of the estimation. In order to test the performance of the prediction, multiple tests were performed with a variety of missions in order to check how general the method is.

The ability to predict the performance of a modified finite-state machine would fit well in an optimization algorithm based on local search. The prediction can be used to rank the neighborhood of finite-state machines by their estimated performance and help the search process make more informed decisions. This would both improve the performance of the resulting control software and it would also reduce the time needed in order to generate it. Part of the work done in this thesis is to explore if these methods of predicting could improve the performance of a local search optimization algorithm.

The results are promising and they show that often the methods are capable of predicting the performance. Moreover, for certain missions, the prediction could improve the result of a local search algorithm.

In the following sections, a brief review of the current state of the art in this research domain will be presented followed by the detailed description of the methods. The methodology for the experiments is presented and the results are showed and discussed. In the last section, the conclusion is drawn and some ideas for future research are proposed.

Chapter 2

State of the art

2.1 Swarm intelligence

The "Swarm Intelligence" term was first mentioned by Beni G (1989), however previous research was performed on this subject long before that especially in the biological field.

To better ground the discussion, a proper definition of the concept is needed. The following definition is proposed by Dorigo et al. (2014):

> "Swarm intelligence is the discipline that deals with natural and artificial systems composed of many individuals that coordinate using decentralized control and self-organization."

This definition touches upon the main properties of a swarm system. Firstly, a swarm is composed of multiple individuals. The individuals are similar and homogeneous. Nevertheless, in some swarm the individuals could have different typologies, meaning each of them being responsible for a different task. The second property of a swarm is decentralized control. In a swarm, no individual has general view over the final objective or current progress and because of that there is no one to coordinate. This property keeps the system flexible and robust since there is no single point of failure. The last property presented in the definition is self-organization. Because of the lack of centralization, the individuals act locally according to some simple rules. The intelligent behavior of the swarm emerges from these local interaction.

The definition presented above makes the distinction between natural and artificial systems. Initially, scientists were trying to understand how simple individuals in nature can cooperate in such a way that together they solve extremely complex tasks.

Reynolds (1987) tried to understand and model the flocking behavior of birds. He realised that the complex patterns that he was observing were the result of just three simple rules that each bird was following: keep a certain distance to the peers, steer towards the average direction of the neighbors, and steer to move to the center of mass of the neighbor peers. By following just these simple rules, the birds are capable of generating incredible formations and patterns in the sky. He built a model that reproduced the patterns formed by the birds using these rules.

Another example of a natural system that was studied is ant colonies. It was observed that ants are following certain paths when they are exploring the territory. In his work, Deneubourg et al. (1990) showed how ants are using pheromone trails in order to mark the paths that should be followed. This type of indirect communication through the environment is called stigmergy and was introduced by Grassé (1950). Initially, the ants are moving randomly and, while doing so, they leave traces of pheromone on the ground. The ants act according to a simple rule:

they choose the path to follow based on the level of pheromone on the ground. Higher level of pheromones means a higher probability of picking that path. This simple rule provides a positive feedback loop where ants are reinforcing the pheromone level of a beneficial paths.

The artificial swarm systems are composed of individuals created by humans that adhere to the same principles observed in the natural systems: local interactions and self-governing.

The behavior studied by Moffett (1988) is an example of natural system that was a source of inspiration for an artificial system. It is presented how groups of ants are working together to carry it a large pray to the nest. This collective behavior was reproduced using a group of robots that cooperate in order to carry a box (C. Ronald Kube, 2000). These types of artificial models reveal important information about the dynamics encountered in nature that are not yet completely understood. Another example of natural behavior modeled using an artificial one is reported by Chris Melhuish (1998) in which the authors reproduced using robots the behavior of ants sorting their brood based on the development stage.

In the taxonomy presented in the paper (Dorigo et al., 2014), the artificial systems presented above belong to the scientific stream of swarm intelligence. The goal of these systems is to understand the mechanisms behind the intelligent behavior of the swarms. Once the mechanisms are understood, the knowledge can be also applied in other domains so as to solve existing practical problems. This is the goal of the engineering streams of swarm intelligence.

One example of a system belonging to the artificial and engineering stream is the Ant Colony Optimization algorithm (Dorigo et al., 2006). This is a meta-heuristic algorithm that can be used to solve hard optimization problems. It was inspired from the foraging behavior of ants. In order to apply this algorithm, the problem needs to be modeled as finding the shortest path through a weighted graph. Virtual ants will be placed randomly in the graph and they will traverse it according to the rule that real ants use. They choose the next path based on a probabilistic model that depends on the level of virtual pheromone found on that path and on an heuristic that is provided. Higher level of pheromone representing a higher probability of the path to be chosen. The pheromone is updated on an edge based on the number of ants that passed through it.

Network Management using principles from Swarm Intelligence was proven possible by Di Caro et al. (2005). In the paper they describe AntHocNet, a routing mechanism for mobile ad-hoc network which is inspired from ant colonies. It uses similar mechanisms as Ant Colony Optimization, stygmergy and pheromone, and in doing so it discovers the shortest path in the network.

In Swarm Robotics, there are no solutions deployed in the industry yet. The difficulties of this field will be detailed in the next section. Nevertheless, interesting proof of concepts were proposed: a swarm of robots capable of build a modular construction (Petersen et al., 2011), a group of heterogeneous robots capable of solving a task by grouping together (Dorigo et al., 2013) and a system capable of partitioning tasks in order to solve a task more efficiently (Pini et al., 2009).

2.2 Swarm robotics

Swarm Robotics is the field that studies how to design robots that adhere to the swarm intelligence principles. The robots cooperate using local interaction such that they can perform together tasks that could not be achieved individually. The advantages of this approach are: fault tolerance, scalability and flexibility (Brambilla et al., 2013a).

Nevertheless, designing the control software for a swarm of robots is a difficult task. The control software is designed at the individual level, but the desired behavior of the swarm emerges from the local interactions among the peers. Because of this, designing the control software is

usually an iterative process.

Figure 2.1: An e-puck robot. (Mondada et al., 2009)

Because of its iterative nature, a good testing infrastructure is needed such that good performances are obtained using swarm robotics. While developing the control software there are two possibilities for testing it: running on an actual hardware or running on a simulator. There are multiple options for robot architectures that might be also mission depended. For self-assembling missions, the ATRON robot (Jorgensen et al., 2004) is an almost spherical robot capable of forming different formations by connecting itself to other ATRON robots. Moreover, in the context of self-assembling robots, the S-Bot (Wei et al., 2010) is a cylindrical robot that is capable of clinging to other S-Bots to do missions such as hole avoidance or cooperative transport (Mondada et al., 2003). E-puck (Mondada et al., 2009) is a general purpose robot designed for educational purposes which is frequently used in swarm robotics research. It has IR sensors, a speaker, microphones and accelerometer. For a more thorough analysis of the available hardware options (Patil et al., 2013) does a good presentation of the available hardware options.

Using simulators instead of real robots has its benefits. The control software can be tested in a simulated environment with simulated robots which avoids the risk of damage or malfunctioning. Another advantage is the reduced cost of exploring an idea without having access to physical robots. Moreover, an execution of the control software in a simulation takes less time than deploying the software on real robots. Because of this, the simulations can be used together with an optimization algorithm in order to measure the real performance of the robots and try to maximize it. One popular simulator that it will be also used during the work in this thesis is ARGoS (Pinciroli et al., 2012). It is a simulator that can achieve both flexibility and efficiency. ARGoS uses a physics engine to simulate the interactions between robots and between robots and environment. The simulator can be configurable with various modules that can be plugged in depending on the requirements of the project. These plug-ins can be: sensors, actuators, physics engines or visualizations.

With the aim of clarifying the distinctions between different types of control software design

methods, Brambilla et al. (2013b) specified two categories: behavior-based design methods and automatic design methods.

The behavior-based method implies manual development of the control software that describes the behavior of a single robot. Since the desired behavior emerges from the collective behavior of the swarm, this method is usually a trial and error process. Soysal and Sahin (2005) designed a swarm of robots capable of aggregation using a behavior-based method. In this paper, the control software is modeled as a probabilistic finite-state machine where the states are behaviors that robot can perform: repel, wait, approach, and obstacle avoidance. The transitions between states are based either on the sensor input data or based on probabilities. Task allocation between swarm was also modeled as a probabilistic state machine (Liu et al., 2007).

Automatic design methods are a method of developing the control software where the developer does not intervene. Francesca and Birattari (2016) split this category in two sub-categorises: online methods and offline methods. The difference between the two is when the design process is happening. In the automatic online methods the design process happens after the software has been developed on the robots and in the automatic offline methods the design process happens before the deployment of the control software Birattari et al. (2019). Another categorization that can be made is between semi-autonomous methods, where the designer still tunes the optimization parameters and fully autonomous methods where the designer does not intervene at all (Birattari et al., 2020).

The approach described by Watson et al. (2002) is an example of an automatic online design method that uses an embodied evolution approach. The control software is represented using a neural network and the robots are evolving by communicating some of the weights to the other robots.

One approach that has proven to work in various missions is evolutionary robotics (Dorigo et al., 2004). This method is also in the category of automatic design methods. The control software is represented by a neural network that receives as input the sensor data and the output of the network is redirected to the robot's motors. During the optimization process, the weights of the neurons are not learned since the target function is unknown, but they are being randomly assigned at the beginning of the process. The weights are mutated using the principles of artificial evolution (Trianni et al., 2008). New solutions are generated based on the existing ones by applying mutations and only the ones that achieve the best results will go to the next generation (elitism). This process continues from generation to generation until the budget runs out or when a good enough solution is found. Being a automatic design method, the designer is not directly involved in the development of the control software. The designer does not need a priori knowledge about the particular mission to perform it since the evolutionary algorithm will generate a good control software.

In the work described by Tuci et al. (2008), a group of three robots have the task to reach a light source. Two of the robots have an infrared sensor and an acoustic sensor and the other robot has an ambient light sensor and acoustic sensor. The robots have to cooperate since there is no other way to achieve the task but together. Their control software is represented by neural networks. During the evolution process, the genotypes are modified using elitism, recombination and mutation. The three best performing instances are kept and the others are generated based on their fitness function. The results are promising and they show that a control software can be built using this method. Other work (Quinn et al., 2003) (Duarte et al., 2016) (Baldassarre et al., 2003) also showed good results.

Nevertheless, there are a couple of issues with evolutionary robotics. As it is mentioned by Trianni and Nolfi (2011), even though the designers do not explicitly program the behavior of the robots, in the context of evolutionary robotics, they still have to plan the environment and the characteristics of the robot that will be evolved by the algorithm. There is no clear

methodology on how to perform this and the results obtained depend mostly on the intuitions of the researchers. The paper (Trianni and Nolfi, 2011) emphasize that "there is a strong necessity of formalising an engineering approach for the evolution of robot behaviours, above all within a relatively novel and complex domain like swarm robotics."

Another common and important problem encountered in swarm robotics, especially in evolutionary robotics, is the reality gap problem (Jakobi et al., 1995). During the development process, the control software is tested in simulations that reproduce the reality with various degree of similarity. The problem occurs when the control software developed in simulation has to be deployed on real robots. What happens very frequently is that there is a difference between the performance that is occurring in simulation comparing to the one from reality. This is happening because the control software is optimizing the control software based on the characteristics of the simulation that might be different from the ones presented in real robots. In evolutionary robotics, the neural networks have high representational power and they can overfit these characteristics. When the control software is transferred on real a robot, the neural network is incapable of generalizing to the new characteristics of reality (Hasselmann et al., 2021). Solutions were proposed to address this issue such as adding noise to the readings, sample from real sensor data or continue the training of the control software on real robots (Miglino et al., 1995) (Nolfi et al., 1994) (Miglino et al., 1994).

Another important solution to the reality gap problem is AutoMoDe (Francesca et al., 2014). The authors suggest that the reality gap can be seen as a similar problem to the bias-variance problem from machine learning (Geman et al., 1992). The bias and variance are two properties that are in tension: there is an inverse relation between them. High variance models have a low bias and the other way around. High variance models are capable of fitting the data but there is always the risk of over-fitting which leads to the inability to generalize on unseen data. However, a high bias results in simpler models that are unable to learn the important relations between features. In AutoMoDe, the authors proposed to inject some bias to limit the variance. This would result in diminishing the representational power of the control software which would lead in the end to better results since the noise of the simulation will not be overfit anymore. AutoMoDe is a family of methods belonging to the offline automatic control software design. The robots are controlled by modular control software architectures that are automatically generated by a an optimization algorithm (Birattari et al., 2021). Some examples of architectures used for control software are probabilistic finite-state machines (Francesca et al., 2015) or behavioral trees (Kuckling et al., 2018).

The first implementation of AutoMoDe is called Vanilla and it is intended to be a proof of concept. It is implemented for e-puck robots which are small robots that were designed for educational purposes (Mondada et al., 2009). In AutoMoDe Vanilla, the control software represented as a probabilistic final state machined is obtained using an optimization algorithm called F-race (Birattari et al., 2002). The finite-state machines are represented by state and transitions between them. The states are represented by behaviors that robots can perform at any given moment. The behaviors are:

1. exploration: the robot is moving straight until it reaches an obstacle and then it turns and continues to move.
2. stop: the robots stands still.
3. phototaxis: the robot moves towards a source of light.
4. anti-phototaxis: the robots moves further away from a source of light.
5. attraction: move closer to the central of mass of robots

6. repulsion: move away from the central of mass of robots

For example, when a robot is in state *attraction* it will keep moving towards other robots while if it is in state *stop* it will stand still.

The transitions between states are represented by conditional transitions. They can also be fine-tuned based on some parameters. The conditions that trigger a robot to move from its state to another are environment conditions:

1. Color floor: transition with a certain probability to next state when the floor is of a certain color (black, white, gray)
2. Neighbor-count: transition with a certain probability that is dependent on the number of neighbors that a robot sees
3. Inverted neighbor counting: inverse of "neighbor-count"
4. Fixed-probability : transition to the next state with a fixed probability.

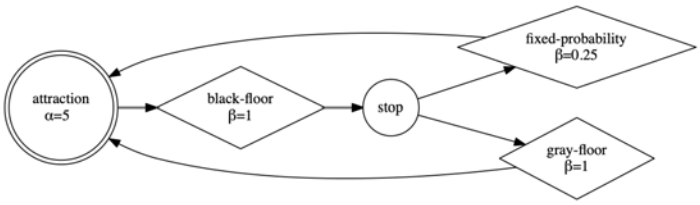

Figure 2.2: Example of an AutoMoDe finite-state machine (Francesca et al., 2014)

An example of a control software represented as a probabilistic finite-state machine can be seen in Figure 2.2. If a robot uses the finite-state machine from the example as control software, it will move closer to other robots until the floor below him is black. When this happens it will transition again to moving towards other robots with a fixed probability or when the floor is gray.

The optimization algorithm tries to find a finite-state machine that maximize the score obtained by the swarm.

It was observed during the experiments that the control software obtained using this automatic design method performs worse than humans programming the control software using exactly the same state modules and transitions as the one presented in AutoMoDe-Vanilla. This is an indication the the bottleneck in performance might be the optimization algorithm itself. In a future improvement called AutoMoDe-Chocolate (Francesca et al., 2015), the optimization algorithm was changed with a more powerful one called Iterated F-Race (López-Ibáñez et al., 2016). A comparison between Vanilla, Chocolate, an evolutionary method and control softwares designed by humans was performed and it showed that AutoMoDe-Chocolate outperforms the evolutionary method and the control software written by human. The experiments showed not only that the control software behaves better when deployed on real robots, but also that AutoMoDe is closing the reality gap better than previous evolutionary methods or even human developed control software.

The AutoMoDe ideas presented in the (Francesca et al., 2014) were further extended into other instantiations. A non-exhaustive list of extensions is presented here:

- AutoMoDe-IcePop (Kuckling et al., 2019) where the optimization algorithm is using simulated annealing.
- In AutoMoDe-Arlequin (Ligot et al., 2020) the behavior modules are trained in a mission agnostic way and then used in the probabilistic finite-state machine.
- AutoMoDe-Gianduja (Hasselmann et al., 2018) where the robots are communicating.
- AutoMoDe-Waffle (Salman et al., 2019) where both the software and the hardware are optimised.
- AutoMoDe-TuttiFrutti (Garzón Ramos and Birattari, 2020) where the robots display and percive colors.

2.3 Off-policy evaluation methods

Reinforcement learning is an artificial intelligence learning technique where a learner tries to figure out what actions should be taken next in a specific configuration of the environment (state) based on the previous experiences. The learner, also called *agent* in the literature, can be a robot (Riedmiller et al., 2009), a car (Kiran et al., 2021), a software component (Stampa et al., 2017) or any other entity capable of learning from experience. While the agent acts in the environment it receives a reward signal that can be both negative or positive. The robot will try to learn what is the best action to take in any given state.

In reinforcement learning, an agent contains a policy and a value function. The policy controls the decision that the robot has to take. Formally, a policy π is a function that receives a state and returns, in deterministic environments, the next action or returns, in non-deterministic ones, a distribution of probabilities over a set of actions. The goal of the agent is to learn the optimal policy which is the policy that maximizes the reward received.

An important concept is the discount factor γ. This value is between interval [0, 1] and it represents how much future rewards should contribute to the value of the current state. The reward obtained after step t can be calculated using the formula below.

$$G_t = R_{t+1} + \gamma \cdot G_{t+1} \tag{2.1}$$

As the discount factor γ is closest to 1, the future rewards are counted entirely and as the γ value is 0 only the current reward is taken into consideration. R_{t+1} represents the reward after step t.

With the goal of evaluating a state, the value function $q(a, s)$ is used. The value function will return, for a specific policy, what is the expected reward to be received if action a is taken in state s.

$$q_\pi(a, s) = E_\pi[G_t | S_t = s, A_t = a] \tag{2.2}$$

$$\pi : A \times S \to [0, 1], \pi(s, a) = Pr[a_t = a, s_t = s] \tag{2.3}$$

There are two methods for estimating the value of a state-action pair: every-visit and first-visit. In the every-visit, all the returns that resulted from all the visits are averaged to get the expectation, while in the first-visit method only the first visits that occur in an episode are used in averaging. Both method converge to the true value with enough episodes (Sutton and Barto, 2018).

In order for an agent to learn all the possible state values, it has to explore. This means that it can not always act according to the policy learned up until that point. The methods where an agent is learning a policy while acting according to the same policy are called on-policy methods. To ensure exploration in this case, mechanisms such as e-greedy needs to be used which make the agent behave randomly with a small probability. By doing this, more parts of the search space will be explored which could lead to better policies. There are also off-policy methods where the agent has two policies: one according to which it is acting in the environment, called behavior policy, and another one that it is learning, called target policy.

The off-policy prediction is the problem where the target and behavior policies are fixed and the performance of the target policy needs be estimated based on the episodes obtained by following the behavior policy. The mechanism that is used to do this is called importance sampling. The ratio between the probability of seeing the sequences of states $\tau(s)$ under the behavior b and target policy π is called importance sampling coefficient $\rho_\tau(s)$ (Rubinstein and Kroese, 2016) (Sutton and Barto, 2018).

$$\rho_\tau(s) = \prod_{k \in \tau(s)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)} \tag{2.4}$$

In equation 2.4, $\pi(A_k|S_k)$ and $b(A_k|S_k)$ represent the probability of taking action A_k in state S_k under policy π, in the case of the nominator and b, in the case of the denominator. If policies π and b are the same, the importance sampling coefficient will be equal to 1. However, for example, if the target policy π gives a probability of almost 0 for all actions in all states, the importance sampling coefficient will be also close to 0.

This coefficient is used to correct the fact that the rewards were obtained from episodes that were drawn from the behavior policy. In order to do this there are two mechanisms available: ordinary importance sampling and weighted importance sampling. To explain these mechanisms, the notation from (Sutton and Barto, 2018) will be used. The time steps are notated in a way that crosses the episodes boundary. If episode 0 has 100 time steps, the last step will be 100 and the first step from episode 1 will be 101. $\tau(s)$ represents the set of all time steps where state s was encountered and $T(t)$ represents the first time step termination after time step t. G_t is the return from time step t up to $T(t)$.

If ordinary importance sampling is used to estimate the state value function under target policy π, the multiplications between the rewards G_t and their associated importance sampling coefficient are averaged (2.5).

$$v_\pi(s) = \frac{\sum_{t \in \tau(s)} \rho_{t:T(t)-1} G_t}{\tau(s)} \tag{2.5}$$

The second estimator is called weighted importance sampling and it is defined as:

$$v_\pi(s) = \frac{\sum_{t \in \tau(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \tau(s)} \rho_{t:T(t)-1}} \tag{2.6}$$

The weighted importance sampling is very similar to the ordinary importance sampling, the only think that differs is the denominator. In the case of the weighted importance sampling the weighted average is used instead of computing the normal average as in the case of the ordinary importance sampling. The ordinary importance sampling is unbiased but the variance can be very high. The weighted importance sampling is biased but the variance tends to be lower (Sutton and Barto, 2018).

2.4 Estimating control software performance

Pagnozzi and Birattari (2021) presented a method that tries to estimate using the estimators described in the previous section the performance of a control software represented as a probabilistic finite-state machine.

Using AutoMoDe-Chocolate, multiple finite-state machines are generated. In effort to know their performance, the optimization algorithm needs to run each control software multiple times. During each run, each robot will save the current state, the active transition and other information needed to be able to calculate the probability of transitioning to another state. For example, in Figure 2.2, a robot might be first in the *attraction* state for 10 steps and then the floor becomes black and it will stop for another 10 steps. In the execution trace of this robot, it will be logged that for step 0 to 10 that the robot was in the *attraction* state and that in step 11 the floor was black which made him transition to state *stop*.

The goal of Pagnozzi and Birattari (2021) is to estimate the performance impact of changing a transition parameter or removing a state. The control software represented as a probabilistic finite-state machine can be seen as a policy. The steps contained in the execution traces file are obtained from a behavior policy which is represented by the original finite-state machine and the new modified finite-state machine is the target policy. The problem of estimating the performance of a new probabilistic finite-state machine obtained by modifying another finite-state machine can be framed as an off-policy evaluation method.

In Swarm Robotics, multiple robots are executing the same control software. In the context of Pagnozzi and Birattari (2021), a simplified approach is taken and each robot's execution trace is considered as a new episode.

Two estimators are used. The first one uses weighted importance sampling and it considers that each state contributed equally to the performance that the swarm obtained at the end of the episode.

$$F_e = \frac{\sum_S \frac{v_t(s)}{v_b(s)}}{|S|} \cdot G_t \tag{2.7}$$

In order to compute the performance of the swarm F_e under the target policy, the average of ratios between the state values of the target policy and behavior policy is multiplied by the swarm reward G_t.

The second estimator is called proportional weighted importance sampling and it is weighting the reward by the time the robot spent in each state.

$$F_e = \sum_S V_t(j) \cdot GP_T \tag{2.8}$$

$$GP_T = F/n \cdot k/steps \tag{2.9}$$

To evaluate the performance of this estimation method, the authors performed a comparison between a naive estimator that always considers that the finite-state machine performance will not change and the estimators mentioned above. The results shows that the weighted importance sampling and the proportional weighted importance sampling estimators perform significantly better than the naive estimator.

Chapter 3

Method

The goal of this thesis is to improve the methods proposed by Pagnozzi and Birattari (2021) and to asses their capabilities under various circumstances. The methods that will be presented in the following chapters are built on top of the ones created by Pagnozzi and Birattari (2021). To evaluate how well the methods can predict the performance impact, a thorough analysis is performed. Moreover, a set of experiments is performed to evaluate how well these methods would fit in a local search environment.

Two different types of methods are presented. The first one starts from an existing control software represented as a finite-state machine created using AutoMoDe-Chocolate and it predicts the performance of a newly created finite-state machine that is obtained by modifying the transition parameters of the existing finite-state machine. The second type also starts from an existing finite-state machine, but the newly created one is obtained by removing one state from the original one. The first method type will be called *parameter variation* and the second one *state pruning*.

The prediction is performed by estimating how likely it is to observe the same sequence of states and state transitions by using the newly obtained finite-state machines. To do this, an off-policy prediction mechanism is used.

3.1 Mathematical foundation

To apply the off-policy prediction, the problem needs to be framed in the same mathematical terms as the ones that describes off-policy methods. The original finite-state machine that generated the execution traces is called the behavior control software and the modified finite-state machine obtained by mutating the original one is called target control software.

In the context of AutoMoDe-Chocolate, the control software is represented by a probabilistic finite-state machine where the states are behaviors that the robots perform in the environment and transitions are changes in the robot's behavior. For example, a state is "moving forward" while a transition can change the behavior of a robot from "moving forward" to "move toward a source of light". A policy function π receives a state s as input and returns the probability of seeing an action a. The function is written as $\pi(a|s)$. With AutoMoDe-Chocolate, it can be considered that the finite-state machine is similar to a policy by assuming the state s is the behavior and the action a is the transition between behaviors.

When the control software is deployed on the robots and tested, the robots will act according to the policy π and they will generate a sequence of states and transitions. Moreover, the performance of the swarm can be evaluated at each state and a score can be assigned. For

example, in a foraging mission where the robots have to find and retrieve food back to their nest, the performance score can be computed as the number of food items retrieved. This performance score can be used as the intermediate reward R_t.

A swarm composed of n robots that behave according to a policy π will generate n sequence of states s. A reward R_{i+1} will be obtained after each state s_i. One characteristic of the swarm is that the robots work together to achieve a task, however a simplified approach will be made and each sequence generated by one of the n robots can be considered an independent episode e, each episode being a sequence of states, actions, rewards and world information received from sensors.

The methods presented by Pagnozzi and Birattari (2021) did not use the intermediate steps or rewards received by the swarm, the final performance score being considered the reward at the end of a whole episode. In this paper, the intermediate steps and rewards are used in the estimator in order to try to use as much information as possible to offer a more reliable prediction.

The reward obtained in a specific state can be computed using the following formula:

$$G_t = R_{t+1} + \gamma \cdot G_{t+1} \tag{3.1}$$

The discount factor γ controls how much future rewards contribute to the value of the current state at moment t. In an on-policy method, multiple returns obtained in the state s can be averaged in order to obtain the expected value E of the reward for a given state S_t using $E[G_t|S_t = s] = v_b(s)$. If the goal is to estimate the state value under a different control software (the target control software), the off-policy methods should be used. The importance sampling coefficient is used to correct for the fact that the rewards were obtained following a different policy. The ordinary importance sampling and weighted importance sampling estimators can be used for this task.

$$v_{o_\pi}(s) = \frac{\sum_{e \in E} \rho_{\tau(s)}(e) G_t(s;e)}{|E|} \tag{3.2}$$

$$v_{w_\pi}(s) = \frac{\sum_{e \in E} \rho_{\tau(s)}(e) G_t(s;e)}{\sum_{e \in E} \rho_{\tau(s)}(e)} \tag{3.3}$$

As mentioned in Section 2.3, the ordinary importance sampling estimator offers an unbiased estimator but the variance obtained is very large while the weighted importance sampling estimator offers a biased estimator but the variance is lower. From the experiments performed, the variance of the ordinary importance sampling estimator makes it impossible to be used in practice. Because of this, in the experiments the focus is on the weighted importance sampling estimator.

To get the estimated value for each state, the every-visit method is used. As mentioned by Sutton and Barto (2018), the every-visit and first-visit are both converging and the every-visit is preferred since the algorithm does not have to keep track of the visited states.

Having obtained the estimated value for each state, the next thing that needs to be done is to evaluate the performance of the swarm as a whole. The same estimation user by Pagnozzi and Birattari (2021) will be used, where the ratio between the state values under policy π and policy b is multiplied by the final reward F.

$$F_e = \frac{\sum_S \frac{V_t(s)}{V_b(s)}}{|S|} \cdot F \tag{3.4}$$

3.2 Off-policy prediction with intermediate rewards

The optimization process in AutoMoDe-Chocolate will execute various control software configurations many times in order to see which one is better performing. In order to apply the off-policy prediction, when a control software is executed it will also save the execution traces in a separate file. This execution trace is a file composed of detailed information about each execution of the control software. This information contains for each robot at each step what state it was in, how many neighbors it was observing, what color the ground had, what transition is performed and the reward obtained at each step. In other words, every information that might affect the decision of a robot is recorder in this file.

The probability for the transitions in AutoMoDe-Chocolate can be computed as in the formulas below. During the experiments five types of transitions were used and each transition type is computed in a different way. In the following equations, the p and w represent transition parameters, p being a probability and w a parameter that is particular to a transition.

1. White ground floor

$$transition\_prob(p) = \begin{cases} p, if\ the ground\ floor\ is\ white \\ 0, if\ the\ ground\ floor\ is\ not\ white \end{cases}$$

2. Black ground floor

$$transition\_prob(p) = \begin{cases} p, if\ the ground\ floor\ is\ black \\ 0, if\ the\ ground\ floor\ is\ not\ black \end{cases}$$

3. Gray ground floor

$$transition\_prob(p) = \begin{cases} p, if\ the ground\ floor\ is\ gray \\ 0, if\ the\ ground\ floor\ is\ not\ gray \end{cases}$$

4. Number of neighbors

$$transition\_prob(w, p, num\_neigh) = \frac{1}{1 + e^{w*(p-num\_neigh)}}$$

5. Inverse number of neighbors

$$transition\_prob(w, p, num\_neigh) = 1 - \frac{1}{1 + e^{w*(p-num\_neigh)}}$$

Using the execution trace files, the prediction algorithm can compute all the transition probabilities needed to estimate the performance of the swarm. The algorithm will receive as input the configuration of the newly obtained finite-state machine and the configuration of the original finite-state machine together with the execution traces file. The execution traces file will be read line by line and, for each execution, the state, transitions and intermediate rewards are saved in data structures such that they can be easily retrieved during the run of the algorithm.

The algorithm will perform the estimation for each execution trace of the control software. For each robot, the estimation is performed by reversely iterating over the steps that the robot encountered. A step is defined by a configuration of state, action, number of neighbors seen and ground color. The probability of taking the action that makes the transition from s_{t-1} to s_t

under the target policy and the probability of doing the same thing under the behavior policy are computed. The two probabilities are divided and the importance sampling coefficient from step $t-1$ to t is obtained. The reward G_t is corrected for the fact that the steps are obtained under another policy by multiplying the importance sampling coefficient with the reward. If the transition probability from state s_{t-1} to state s_t under the target policy would be 0, the importance sampling coefficient will be 0 which will make the expected reward for state s_{t-1} to also be 0. This happens because it is impossible to reach state s_t following the hypothetical target policy that has a null transition probability from s_{t-1} to s_t. On the other hand, if the probabilities are the same, the expected return s_{t-1} under the target policy is the same as under the behavior policy.

The loop continues and the importance sampling coefficient is computed for the sequence s_{t-2}, s_{t-1} and s_t. The probability from s_{t-1} to s_t was already computed previously. The only thing that needs to be done is to multiply the importance sampling coefficient from s_{t-2} to s_{t-1} with it. The importance sampling coefficient computed for this sequence is used to weight the return obtained so far $G_t \cdot \gamma + G_{t-1}$. And the loops continues until the first step is encountered. All the intermediate importance sampling coefficients and rewards are stored in data structures such that they can be retrieved at the end of the algorithm.

At the end of the loop, the intermediate importance sampling coefficients and rewards are used to obtain the value function for each state. The formula 3.3 is used to compute the value function using the weighted version of importance sampling, while the formula 3.2 is used to calculate the ordinary version of importance sampling. The weighted sampling version will be used further on since the variance of the ordinary importance sampling is too high to be used in practice. The state value is used to compute the D_{WIS} estimation using formula 3.4.

During the experiments, for one mission multiple episodes are generated by the optimization algorithm. The D_{WIS} values obtained over all the episodes are averaged in order to obtain one D_{WIS} value that characterises the whole mission.

3.3 Parameter variation method

Modifying the transition parameters of a finite-state machine generated by AutoMoDe can produces important changes in the behavior of the robots. The transition parameters are usually a probability of transitioning to another state. The probability can be specified in two ways. The first one is by a fixed number and the second one is by using a function that depends on the number of neighbors or on environment observations such as ground color. Setting the value of a transition parameter too low might result in a transition not being executed anymore and setting the value too high could result in a transition being always executed. These changes can have a big impact in the performance of the control software. The goal is to quickly predict this impact without running any simulations. This can achieved using the algorithm described in the previous section.

Algorithm 1: Parameter variation algorithm

```
reward = 0;
discount_factor = 0.99;
foreach episode ∈ episodes do
    foreach step ∈ reverse(episode.steps) do
        current_state = step.state ;
        number_of_neighbors = step.number_of_neighbors ;
        ground_color = step.ground_color ;
        r = step.intermediate_reward ;
        target_probability = transition_probability(target_policy, current_state,
         previous_step, number_of_neighbors, ground_color) ;
        behavior_probability = transition_probability(behavior_policy, current_state,
         previous_step, number_of_neighbors, ground_color) ;
        importance_sampling_coeff *= target_probability / behavior_probability ;
        reward = reward * discount_factor + r ;
        if r > 0 then
            sum{current_state} += importance_sampling_coeff * reward ;
            sum_coeff{current_state} += importance_sampling_coeff ;
            count{current_state} += 1 ;
        end
    end
    foreach state ∈ state do
        weighted_is_estimation{state} = sum{current_state} / sum_coeff{current_state} ;
        ordinary_is_estimation{state} = sum{current_state} / count{current_state} ;
    end
end
```

3.4 State pruning method

During the AutoMoDe Chocolate optimization process, multiple configurations of state machines are generated randomly and only the ones that perform the best are kept in the race. It can happen that the best resulting finite-state machines contain states that are never or rarely reached by robots during the execution of the control software. These states can also cause perturbation and better results are often obtained if they are removed.

In this section, a method is proposed that tries to evaluate the performance of a new control software generated by removing a state from a control software generated by AutoMoDe-Chocolate.

Computing the estimation when a state is removed is different from estimating the performance where only transition parameters are changed. To illustrate why this is the case, an example is proposed where a robot passes only once through a state s at step t that is later removed in the new control software. When the importance sampling coefficient is computed its value will be zero since the probability of transition between state s_{t-1} at step $t-1$ to state s_t at step t is zero since the state state s_t does not exist anymore. The estimator's result would always be that the new finite-state machine has a performance score of 0.

Pagnozzi and Birattari (2021) tried to solve this problem by skipping state s_t and compute the probability of transition from state s_{t-1} to s_{t+1}. If a direct transition from s_{t-1} to s_{t+1} does not exist, indirect transitions are also considered. An indirect transition is a transition that

from s_{t-1} reaches the state s_{t+1} by following an intermediate state s_x. Nevertheless, the method described by Pagnozzi and Birattari (2021) does not use intermediate rewards. The importance sampling coefficient is computed over the whole sequence and multiplied with the final reward.

A similar approach was tried when intermediate rewards are taken into consideration. However, the experiments showed that the variance is too large resulting in unusable results. This is caused by the fact that the differences between the two finite-states machines, the original and the modified ones, are important enough to make the value of the importance sampling coefficient irrelevant.

In order to overcome this issue, another approach is proposed which is called Intermediate Reward Estimation (IRE). The sequence of steps s_0, s_1, ..., s_n that a robot encounters while executing the original finite-state machines results in the associated intermediate rewards r_0, r_1, ..., r_n. This information is retrieved by the algorithm from the execution trace. The sequence is split in multiple sub-sequences called $runs$. A run is a sub-sequence of states s_x, s_{x+1}, ..., s_{x+m} where m is the length of the sequence, $r_{x+m} > 0$ and r_x, r_{x+1}, ..., r_{x+m-1} are all 0. In other words, a run is a sub-sequence of steps with no intermediate rewards that ends when an intermediate reward is received. The reason behind this split is the simplifying assumption that the states s_x, s_{x+1}, ..., s_{x+m} are the only ones contributing to the intermediate reward r_{x+m-1} received.

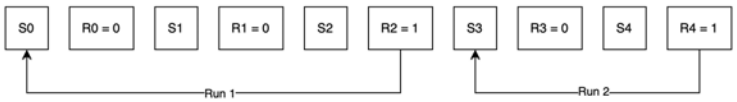

Figure 3.1: Dividing the state sequence in runs during the state removal estimator algorithm.

For each run, the algorithm will check if any state encountered in that run was removed in the altered finite-state machine. If this happens, the intermediate reward will not be distributed to the states contained in that run since that sequence of states is impossible to encounter under the new policy. If all states belong also to the new finite-state machine, the discounted intermediate reward is attributed to each state. The importance sampling coefficient is 1 since the transition parameters are the same. At the end of the algorithm the state value function and performance estimation value are computed in a similar way as the parameter variation method described in Section 3.2.

In Figure 3.1, it can be seen how a sequence of five states and their associated rewards are split into two runs. In this example, the robot has passed through state S0, S1 and S2 and it has received an intermediate reward $R2 = 1$ in state S2. The first three states are part of the run 1. Next the robot passes through states S3 and S4 and it receives a reward $R4 = 1$ in state S4. The run 2 is composed of states S3 and S4. In this example, if a new finite-state machine would be generated where state S1 would be removed, then the reward R2 would not be distributed to the states S0 and S2.

Algorithm 2: State removal algorithm

```
rewardstates = 0 ;
foreach episode ∈ episodes do
    foreach step ∈ episode.steps do
        current_state = step.state ;
        r = step.intermediate_reward ;
        if r > 0 then
            foreach state ∈ current_run do
                reward{state} += r ;
                r *= discount_factor ;
            end
        else
            append current_state to the current_run ;
        end
    end
    foreach state ∈ states do
        state_value{state} = average{reward{state}} ;
    end
end
```

Chapter 4

Experiments

The experiments performed to validate the methods described above were executed using ARGoS3 (Pinciroli et al., 2012) and AutoMoDe-Chocolate. Two different types of experiments were performed.

The first type of experiment is similar to the experiments performed by Pagnozzi and Birattari (2021). This was done to make a comparison between the methods used in that paper and the ones proposed in this thesis. Furthermore, more missions are used to present a better picture regarding the generalization power of the method. Moreover, to better understand the predictive power of the methods, a correlation analysis is performed.

The goal of the second type is to understand what impact these methods would have in the context of a local search algorithm used to generate control software. If the methods are able to predict which finite-state machines from the neighborhood produces better performance, then they can be used to guide the search of the neighborhood more efficiently.

4.1 Prediction Analysis

The methods proposed by (Pagnozzi and Birattari, 2021) were compared against a naive estimator that always assumed the performance of the new finite-state machine to be the same as the old finite-state machine. The weighted importance sampling (WIS) and the proportional weighted importance sampling (P_{WIS}) estimators were compared against the naive estimator on multiple control softwares designed for foraging mission.

In this work, a similar experiment is designed where the new estimators D_{WIS} (for parameter variation) and IRE (for state removal) are compared with WIS, P_{WIS} and the naive estimator with the goal of understanding if they produce an improvement in prediction compared to these methods that currently exist in the literature. The experiment is performed on three missions: foraging, aggregation and shelter with constrained access.

For each mission, AutoMoDe-Chocolate was used to produce a set of finite-state machines together with the execution traces generated during the optimization process. For the performance estimation, all the experiments were performed using the ARGoS3 simulator without visual interface. The swarm was composed of 20 robots and the execution lasts 2500 simulation steps.

The iterated racing algorithm (irace) was executed 10 times for each mission with a budget of 10000 evaluations. The finite-state machines that were selected by irace as being the best solutions were kept and the others were removed. The set of finite-state machines was further filtered by removing the ones that have less than 3 states active according to the execution traces.

The remaining ones were sorted based on the number of active states and the first 20 with the highest number of active states were kept.

These 20 finite-state machines are considered the original ones and new finite-state machines will be produced by applying either parameter variation or state state pruning method.

To produce the set of modified finite-state machines for parameter variation analysis, for each finite-state machine, four variations are produced by modifying the transition parameters of the two most active transitions. This results in 80 finite-state machines for which the performance prediction will be executed. Pagnozzi and Birattari (2021) kept only the finite-state machines that showed a significant difference in performance. In this experiment, however, the whole set of finite-state machine is used. The experiment is done like this in order to show capability of the method to predict changes in performance but also the lack of change. This shows a more general view over the performance of the estimators since in a local search algorithm the estimator will have to differentiate between neighbours with worse performance, neighbours that have the same performance and neighbours with improved performance.

For state pruning, the final set of finite-state machines on which the prediction will be executed is generated by removing one state at a time from each of the original finite-state machines.

The finite-state machines generated by both applying the parameter variation and state pruning are executed in the ARGoS3 simulator 30 times. The measured performance of each control software is computed by averaging the performance obtained during the 30 executions. This value is used to measure the accuracy of the prediction. The accuracy of each prediction method is computed using normalized squared error (NSE):

$$normalized\ SE = \begin{cases} \frac{(\pi_i(E) - P_i(E))^2}{\pi_i(E)}, & \pi_i(E) > 0 \\ (P_i(E))^2, & \pi_i(E) <= 0 \end{cases}$$

In this equation, $\pi_i(E)$ is the measured performance of the finite-state machine i over the set of executions E and $P_i(E)$ is the prediction of the performance of finite-state machine i. In the case where the actual performance of the new finite-state machine is 0, which can happen often in the state pruning variation, the error is the prediction value squared. The normalization is performed in order to be able to compare different finite-state machines with different performance scores.

4.2 Local Search

In the context of a local search, a method capable of predicting the performance of a finite-state machine can be used to explore in an informed way the neighborhood. A neighborhood of a solution S is defined as the set of all solution that can be generated by applying an operator on S. A local search algorithm starts from an initial finite-state machine and picks the neighbor that is the most promising as the next solution. The problem is that in order to know which is the best neighbor, all or a subset of the neighborhood needs to be evaluate by running the simulator. This can have a large impact on the execution time of the algorithm. The prediction can help the algorithm make an informed decision which should lead to a more efficient search.

For each mission used in this experiment, 10 finite-state machine are generated using the AutoMoDe-Chocolate optimization algorithm. Then for each of them 30 neighbors are generated. The algorithm for creating the neighborhood is similar to the one used to generate the finite-state machines for the parameter variation experiment: taking the two most active transition and randomly mutating the transition parameter. State pruning was not included in the analysis since the maximum number of finite-state machines that can be created by applying state pruning is very small. For example, if a finite-state machine has 4 states, the maximum number of new finite-state machines that can be created is 4, one for each state that can be removed.

Moreover, as the results will reveal, predicting the performance of a finite-state machine obtained by applying a state removal operator is easier than to predict parameter variation. Keeping this into consideration, the experiment is designed to be as hard as possible for the method.

The control software represented by each finite-state machine in the neighborhood is executed using the ARGoS3 simulator 30 times in order to get its measured performance which is computed as the average of the results obtained during simulation. After that, the estimation methods are executed on the neighborhoods.

In each neighborhood, for each finite-state machine two lists are kept: one with the estimation sorted in descending order and the other one with the measured performance in descending order. The estimator $p_k(N)$ will select the top k finite-state machines from the neighborhood set N according to the estimated value and it will return the one with the highest measured performance. Ideally, if the prediction methods would work perfectly, $p_1(N)$ would return the best performing finite-state machine from the neighborhood. For each estimator, the capacity of finding an improving solution $p_k(N)$ is computed as in the formula presented below.

$$p_k(N) = \frac{p_{original} - p_{modified}}{p_{original}}$$

The values $p_{original}$ and $p_{modified}$ represent the measured performance of the original finite-state machine and the measured performance of the modified finite-state machine. This metric represents the relative deviation from the original performance.

For example, if the best finite-state machine found in the neighborhood by the estimator has a measured performance of 10 and the performance of the original finite-state machine was 5, the value of $p_k(N)$ is -0.2. On the other hand if the solution found by the estimator has a measured performance of 10 while the original performance was 15, the value of $p_k(N)$ is 0.5. In the first case, the estimator was able to find a control software that improves the performance of the initial solution. However, in the second example, the estimator did not find a control software that improves it.

An estimator not finding an improving solution in a neighborhood can happen for two reason. The first one is that the estimator is not performing well and it underestimates improving solutions in the neighborhood. Nevertheless, the second situation is that there are not improving solutions in the neighborhood.

To properly evaluate the method, the results obtained are compared against two estimators used as reference: one that randomly sorts the solution in the neighborhood and another one that acts like a perfect estimator that always chooses the best solution. These two estimators should be seen as the lower and upper bound of the performance: the proposed methods should not perform worse than random and they can't perform better than the perfect estimator.

4.3 Results

4.3.1 Prediction analysis

The results of the prediction analysis are presented here for each of the three missions.

Aggregation

The result for the aggregation mission with parameter variation are presented in Figure 4.1. The configuration of the aggregation mission is the same as the one described by Francesca et al. (2014). The robots have to aggregate in one of the two black circular areas. The color of the floor is gray. The two black areas have a radius of 0.35m and are centered in (0.6, 0), (-0.6,

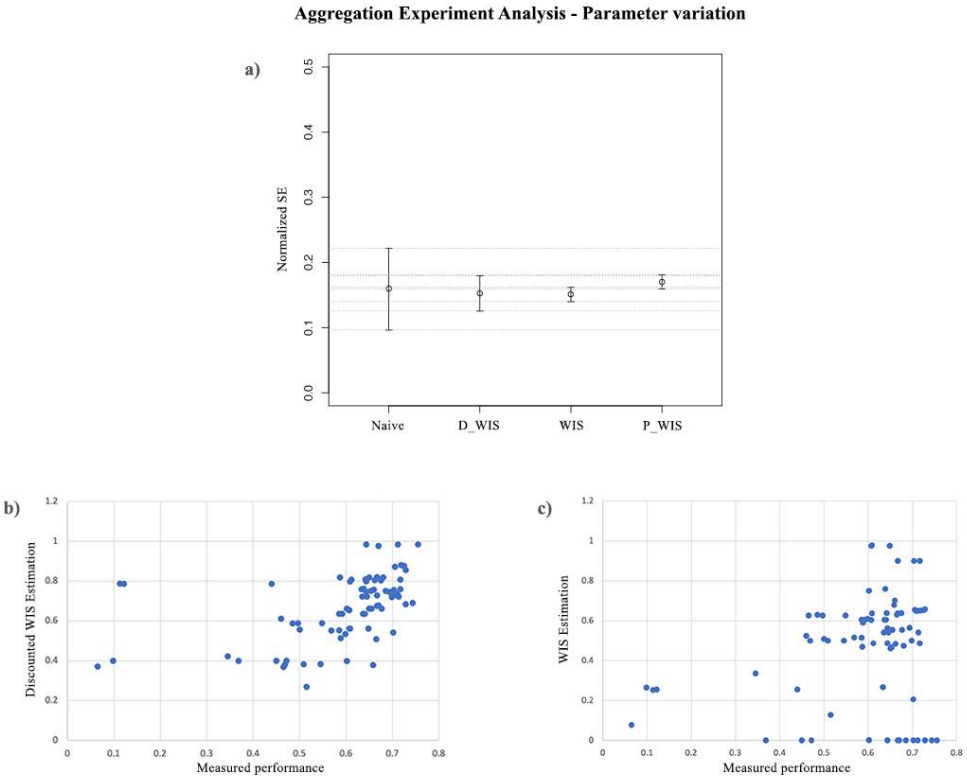

Figure 4.1: The results of the experiments regarding the estimation of the performance of finite-state machines obtained by modifying state transition parameters for the aggregation mission. Figure a) represents the normalized squared error between the measured performance and the performance estimated by different estimators. Figure b) represents the correlation relation between the discounted WIS estimator and the measured performance. Figure c) represents the correlation between the WIS estimator and the measured performance.

0). The objective function is computed at each step as the percentage of the robots that have successfully arrived in one of the two black areas. The value of the objective function is computed for the black area with the most robots.

The normalized squared error is represented on the Y axis, while on the X axis from left to right are: the naive estimator followed by the method proposed in this thesis, the discounted weighted importance sampling D_{WIS}, and then the last two estimators are the methods proposed by Pagnozzi and Birattari (2021): the weighted importance sampling WIS and the proportional weighted importance sampling P_{WIS}.

When the normalized square error is closer to 0 it means the estimator is performing well and as it goes further away from 0 the estimation is worse. From Figure 4.1 a), it can be seen that the estimators are not performing better than the naive estimator.

To understand better the relation between the estimation and the measured performance,

a correlation analysis was performed using a Person's correlation test. A visualization of the correlations presented in Figure 4.1 b) and c). The best two performing methods were analyzed in this section while the analysis for the remaining method is present in the Appendix A. The results show that the estimations of the D_{WIS} method is correlated with the measured performance (r(80) = 2.08, p = .04) and the estimations of the WIS method show a negative correlation with the measured performance (r(80) = -2.19, p = 0.31). This happens because of the errors where the WIS estimator gives an estimation of 0.

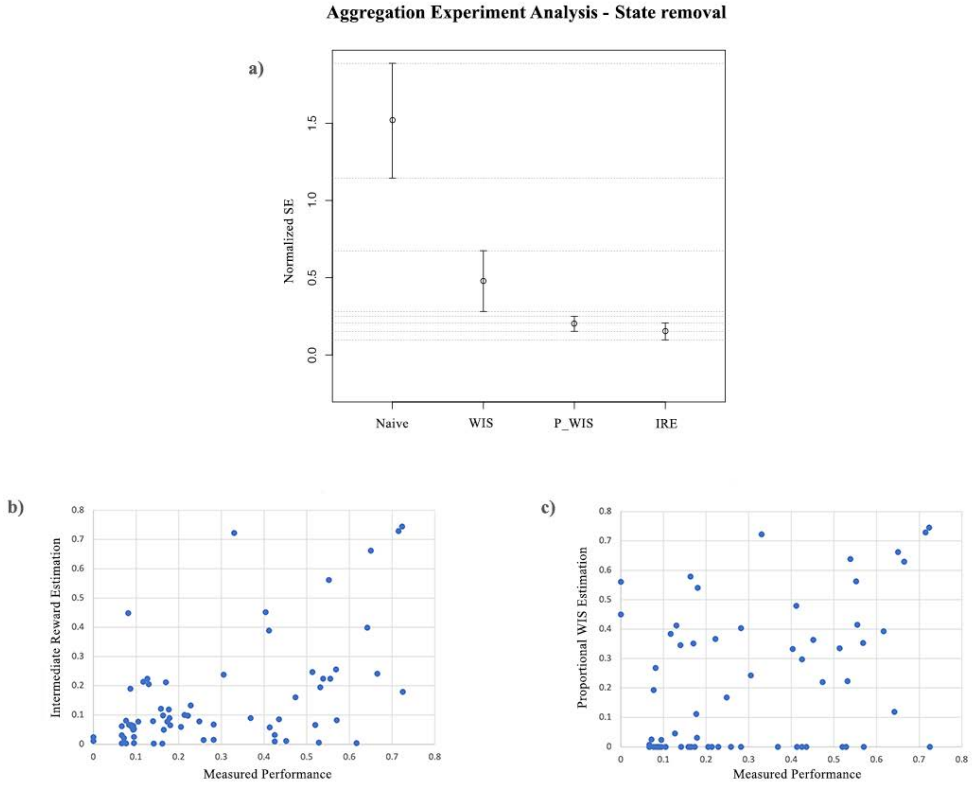

Figure 4.2: The results of the experiments regarding the estimation of the performance of finite-state machines obtained by removing states for the aggregation mission.
Figure a) represents the normalized squared error between the measured performance and the performance estimated by different estimators. Figure b) represents the correlation relation between the estimations of the intermediate reward estimator and the measured performance. Figure c) represents the correlation between the P_{WIS} estimator and the measured performance.

Regarding the state pruning experiments, all methods are performing significantly better than the naive estimator (Figure 4.2). The new estimator proposed in this thesis IRE obtained slightly better performance that the P_{WIS} estimator and significantly better performance than the WIS estimator.

In Figure 4.2 b) and c), the scatter plots illustrate how well the estimators correlate with the measured performance. The Pearson's test shows that both methods shows a high probability of being correlated ($r(68) = 4.86$, $p = 7.58 \cdot 10^{-6}$ for the IRE method and $r(68) = 3.66$, $p = 0.0005$) for the P_{WIS} method). An important aspect to notice is that both methods tend to overestimate the performance of certain finite-state machines. The best performing finite-state machine according to the estimators is actually one of the worst performing according to the measured performance. The IRE method is performing slightly better than P_{WIS} since it also estimates well some of the best performing finite-state machines.

Shelter with constrained access

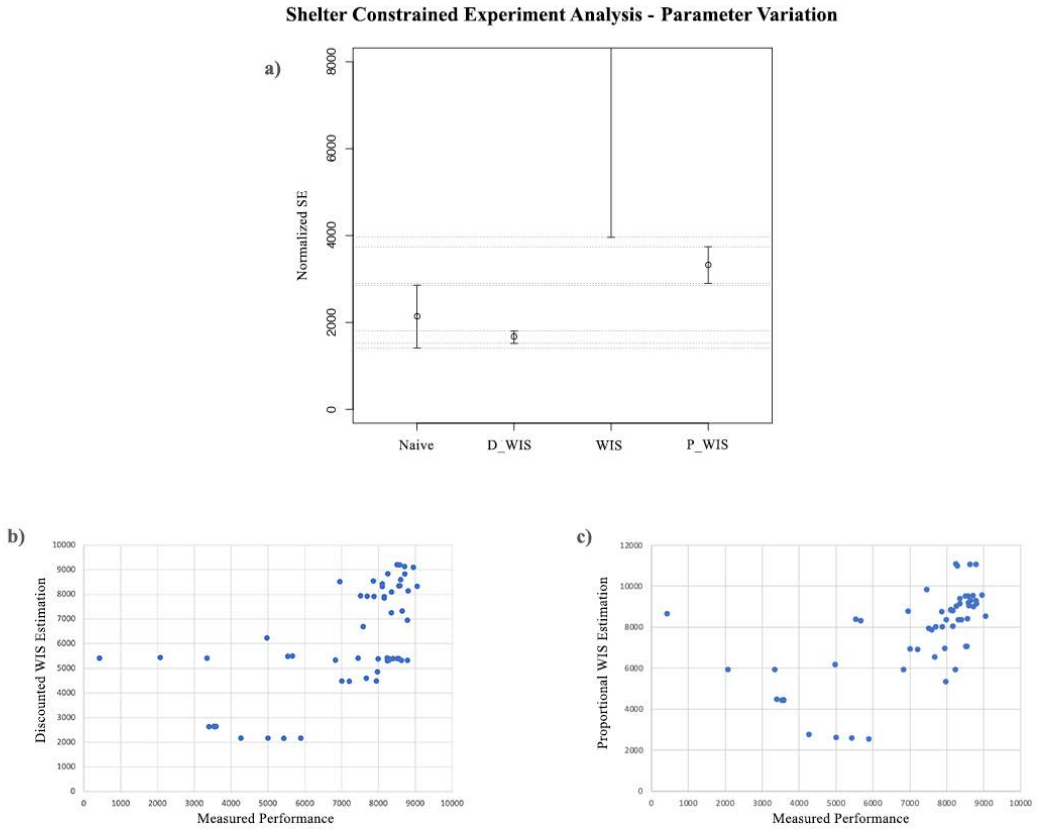

Figure 4.3: The results of the experiments regarding the estimation of the performance of finite-state machines obtained by modifying state transition parameters for the shelter with constrained access mission.
Figure a) represents the normalized squared error between the measured performance and the performance estimated by different estimators. Figure b) represents the correlation relation between the estimations of the discounted WIS estimator and the measured performance. Figure c) represents the correlation between the proportional WIS estimator and the measured performance.

In shelter with constrained access mission, the robots have to aggregate in an area that has the access restricted meaning that the robots can enter only from one side. The objective function is computed at each step and it is represented by the accumulated sum of robots that are in the area until that step. In this mission, the best estimator is the D_{WIS}.

An interesting aspect is that the estimator that was performing well in the previous experiment, WIS, here has a very bad performance caused by a huge variance (Figure 4.3). This difference in estimation performance between missions can be explained by the fact that each

missions requires different dynamics in order to be accomplished successfully and some dynamics' impact might be harder to predict. Another reason could be the way the objective function is computed.

Nevertheless, the performance of the discounted weighted importance sampling with intermediate rewards is consistent with the previous mission performing slightly better than the naive estimator.

The correlation analysis shows that there is a good indication that both of the two best performing methods, D_{WIS} and P_{WIS}, are correlated with the measured performance of the finite-state machines (D_{WIS}: $r(56) = 5.66, p = 5.85 \cdot 10^{-7}$ and P_{WIS}: $r(56) = 4.06, p = 0.15 \cdot 10^{-3}$). In this mission the two methods show less underestimations errors than in the previous mission.

The correlation plot shows that the top ranked finite-state machine from the estimators perspective are also the best ranked finite-state machine according to the actual measurements. This is good news in the context of offering extra information about the neighborhood to the local search algorithm.

The state removal experiments results are presented in Figure 4.4. The estimator that performs better is the IRE estimator. The WIS estimator shows a high variance and the P_{WIS} is performing similar to the native estimator. This result is different from the results obtained in the state removal aggregation experiment. This is an indication that the previous prediction methods do not generalize well to other missions. However, the IRE seems to be consistent in prediction performance using state pruning over multiple missions.

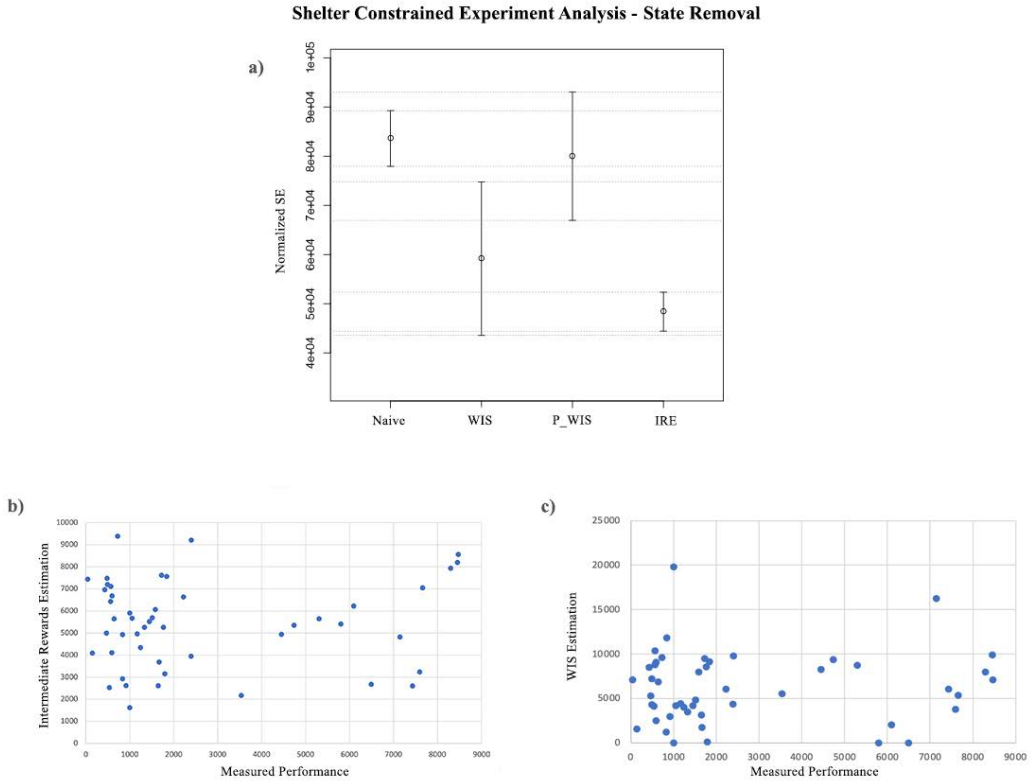

Figure 4.4: The results of the experiments regarding the estimation of the performance of finite-state machines obtained by removing states for the shelter with constrained access mission. Figure a) represents the normalized squared error between the measured performance and the performance estimated by different estimators. Figure b) represents the correlation relation between the estimations of the intermediate reward estimator and the measured performance. Figure c) represents the correlation between the WIS estimator and the measured performance.

Foraging

In the foraging experiment, the two methods proposed in this thesis perform significantly better than the naive, WIS and P_{WIS} estimators. The WIS and P_{WIS} have the same normalized standard error as the naive estimator. In the correlation analysis in Figure 4.5 b) and c), again the D_{WIS} shows better correlation than the WIS estimator. This result is also confirmed by the results of the Pearsons' test. For D_{WIS} the results are $(r(30) = 0.81, p = 3.31 \cdot 10^{-8}$ and for WIS it seems that there is no indication that the estimation is correlated with the measured performance $r(30) = 0.03, p = 0.85$.

The results of the state removal experiment in the foraging missions are showed in Figure 4.6. The IRE estimator is again consistent in prediction performance with the previous experiments performing much better than the naive estimator. In this mission, P_{WIS} performs equally well as the IRE estimator, both of them performing better than the WIS estimator.

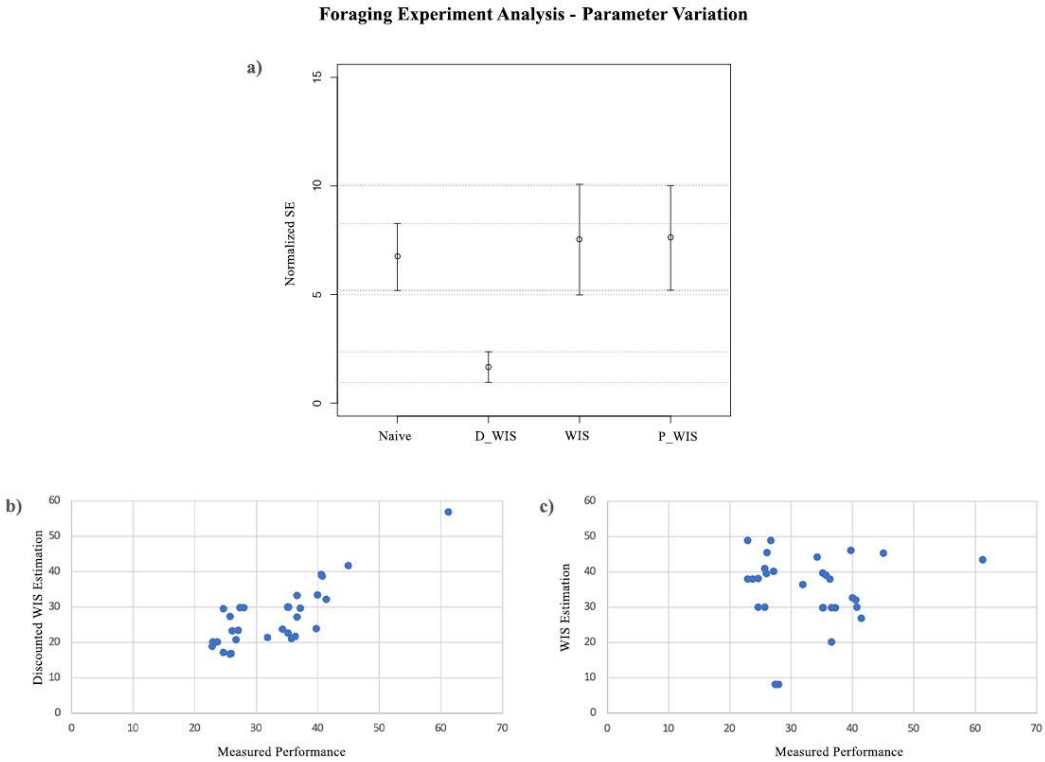

Figure 4.5: The results of the experiments regarding the estimation of the performance of finite-state machines obtained by modifying state transition parameters for the foraging mission. Figure a) represents the normalized squared error between the measured performance and the performance estimated by different estimators. Figure b) represents the correlation relation between the estimations of the discounted WIS estimator and the measured performance. Figure c) represents the correlation between the WIS estimator and the measured performance.

In Figure 4.6 c), the P_{WIS} estimator overestimates the performance of many finite-state machines that have performance below 20. This behavior, however, does not happen when the IRE estimator. The result of the Pearson test (IRE: $r(74) = 0.79, p = 4.05 \cdot 10^{-17}$ and P_{WIS}: $r(74) = 0.72, p = 3.01 \cdot 10^{-13}$) is that there is a significant indication that the results of both methods are correlated with the measured performance.

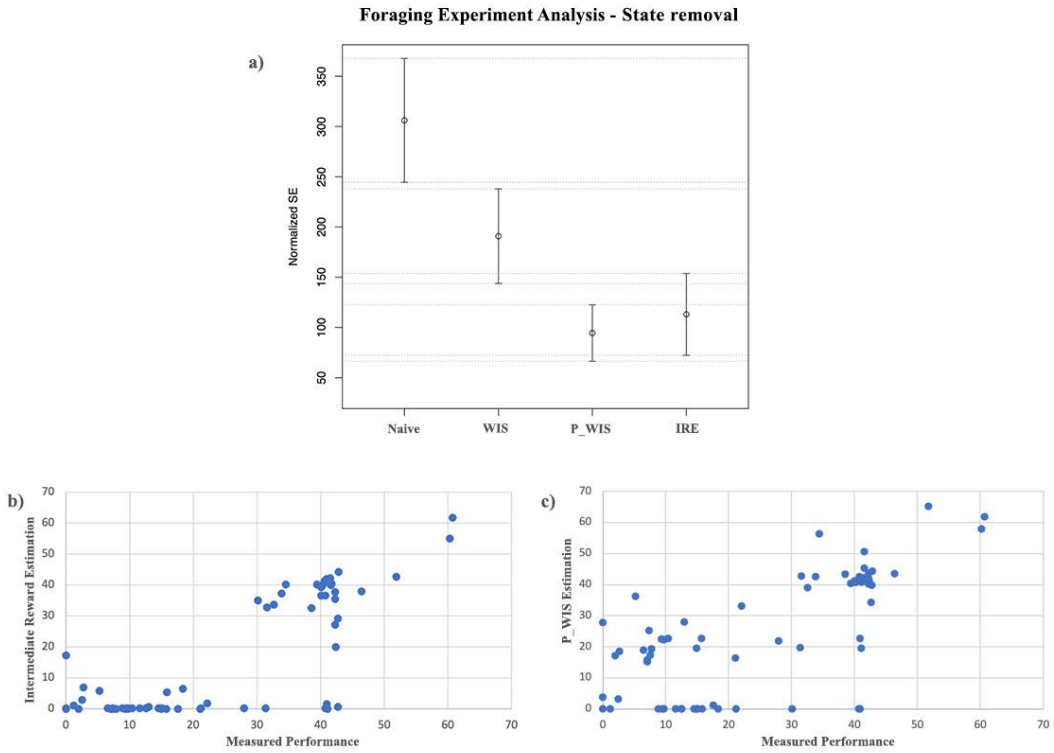

Figure 4.6: The results of the experiments regarding the estimation of the performance of finite-state machines obtained by removing states for the foraging mission.
Figure a) represents the normalized squared error between the measured performance and the performance estimated by different estimators. Figure b) represents the correlation relation between the estimations of the intermediate reward estimator and the measured performance. Figure c) represents the correlation between the P_{WIS} estimator and the measured performance.

Discussion

Having seen how the predictors perform in these three missions, it is important to notice a few important things.

First of all, the state pruning estimators perform better than the transition parameter variation. Almost in all experiments the state pruning estimators perform significantly better than the naive estimator. This is not the case for state transition parameters where the estimators perform almost the same or worst compared to the naive one in aggregation and shelter with constrained missions. This is an indication that estimating the performance of finite-state machines that were modified by changing the state transitions parameters is harder than estimating the performance of finite-state machines obtained by removing a state from an original finite-state machine. Removing a state that is essential for the well functioning of a control software is a

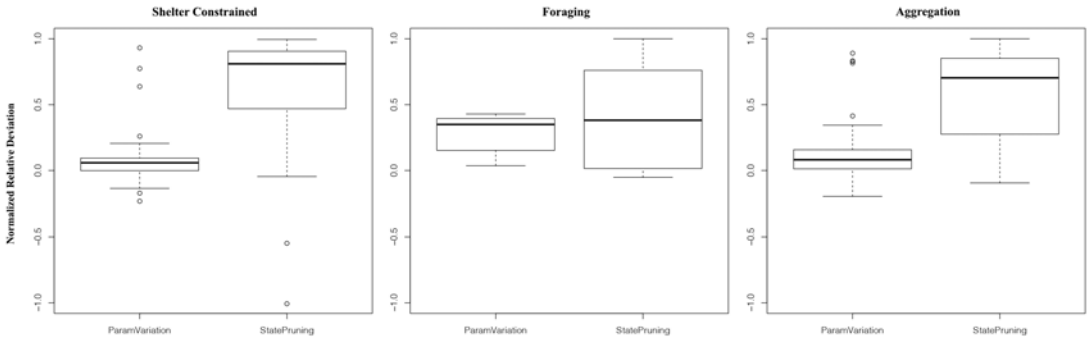

Figure 4.7: The performance variation between the original finite-state machines and the modified finite-state machines in both state variation and state pruning scenario.

perturbation that can make the performance drop to zero. Moreover, removing a state that is rarely used will not have a big impact on the performance. This is a reason why estimating state removal is easier: the change in performance is either very big or very small. In the case of state transition parameter variation, the probability of a transition will change but this will not have such a drastic impact on the new finite-state machine's performance. The predictors will have to estimate more subtle changes which is a harder task to achieve. This can be seen in Figure 4.7. In this plot it is shown what impact the mutation of the original finite-state machine had on the performance. The value on the Y axis, the normalized relative deviation, is computed using the following formula:

$$normalized\_relative\_deviation = \frac{original\_fsm\_performance - mutated\_fsm\_performance}{original\_fsm\_performance}$$

The domain is between minus infinity and 1. The value 0 means that the mutation had no impact on the performance of the finite-state machine, the value 1 means that the new finite-state machine obtained a score of 0 and a negative value means the mutated finite-state machine performs better than the original one. In all missions, the variance of the normalized relative deviation is higher for state pruning than it is for parameter variation. This is an indication that removing a state has a higher impact on the performance of a finite-state machine than just modifying a transition parameter.

Secondly, the D_{WIS} and the IRE estimators are consistent in performance estimation across the missions. In parameter variations, the D_{WIS} is performing better or similar to the naive estimator on all mission. In state removal, IRE is always performing significantly better than the naive estimator. The WIS and P_{WIS} are not consistent in the way they perform across multiple missions. In shelter with constrained mission WIS is performing better and in the others P_{WIS} is performing better. This is problematic if this would be deployed in a local search algorithm since the researcher would have to know which estimator to use for a particular mission.

4.3.2 Local search experiments

The results of the local search experiments for the foraging mission are presented in Figure 4.8. The plot presents the results for different K values between 1 and 4. The perfect estimator always estimates without any mistakes while the random estimator estimates randomly. On the Y axis

it is represented the relative deviation from the original finite-state machine computed using the following formula: $p_k(N) = \frac{p_{original} - p_{modified}}{p_{original}}$.

As the p_k is closer to 0 it means that the modified finite-state machines obtain the same score as the original one. If the p_k value is positive the modified finite-state machines perform worse than the originals and if the p_k value is negative they perform better.

The value of the perfect estimator indicates that the best performing modified finite-state machines perform slightly better than the original ones. When K is 1, only the best performing finite-state machine is selected. The normalized relative deviation of the D_{WIS} and WIS estimators shows that when only the best performing control software according to their estimations is considered, the measured performance decreases compared to the original finite-state machine. As the K values increases, the D_{WIS} and WIS estimators are getting closer to the performance of the perfect estimator. The gap in estimation performance between the D_{WIS} and the WIS estimator is getting larger as the K value increases, the D_{WIS} estimator performs significantly better starting from K=3.

This results suggest that if the D_{WIS} estimator would be used in a local search algorithm, the top three finite-state machines from the neighborhood can be selected based on the performance estimation and that would result on average in a better solution for the foraging mission.

The same experiment was performed for the aggregation mission and the results are presented in Figure 4.9. The performance of the perfect estimator is an indication that the neighborhood generated by modifying the state transitions contains also improving solutions. When K is equal to 1, the methods are not performing better than a random estimator. When K is 2 or larger, the D_{WIS} approaches the performance of the perfect estimator and it is performing significantly better than the random estimator, while the WIS is performing similar to the random estimator. When K reaches 4, D_{WIS}, WIS and the random estimator have similar prediction performance.

The D_{WIS} is performing better than the random estimator when K has the value 2 or 3 and the random estimator is catching up when K is 4. In both missions, the estimators do not achieve the same performance as the perfect estimator which is an indication that there is room for improvements. Nevertheless, from these results it seems that the D_{WIS} method could improve the search of a neighborhood by suggesting three possible finite-state machines from the neighborhood that are more likely to offer better performances.

Lastly, the results of the local search experiment for the shelter with constrained access mission are presented in Figure 4.10. For this particular mission, it seems that the random method behaves similar or better than all the other methods except the perfect estimator.

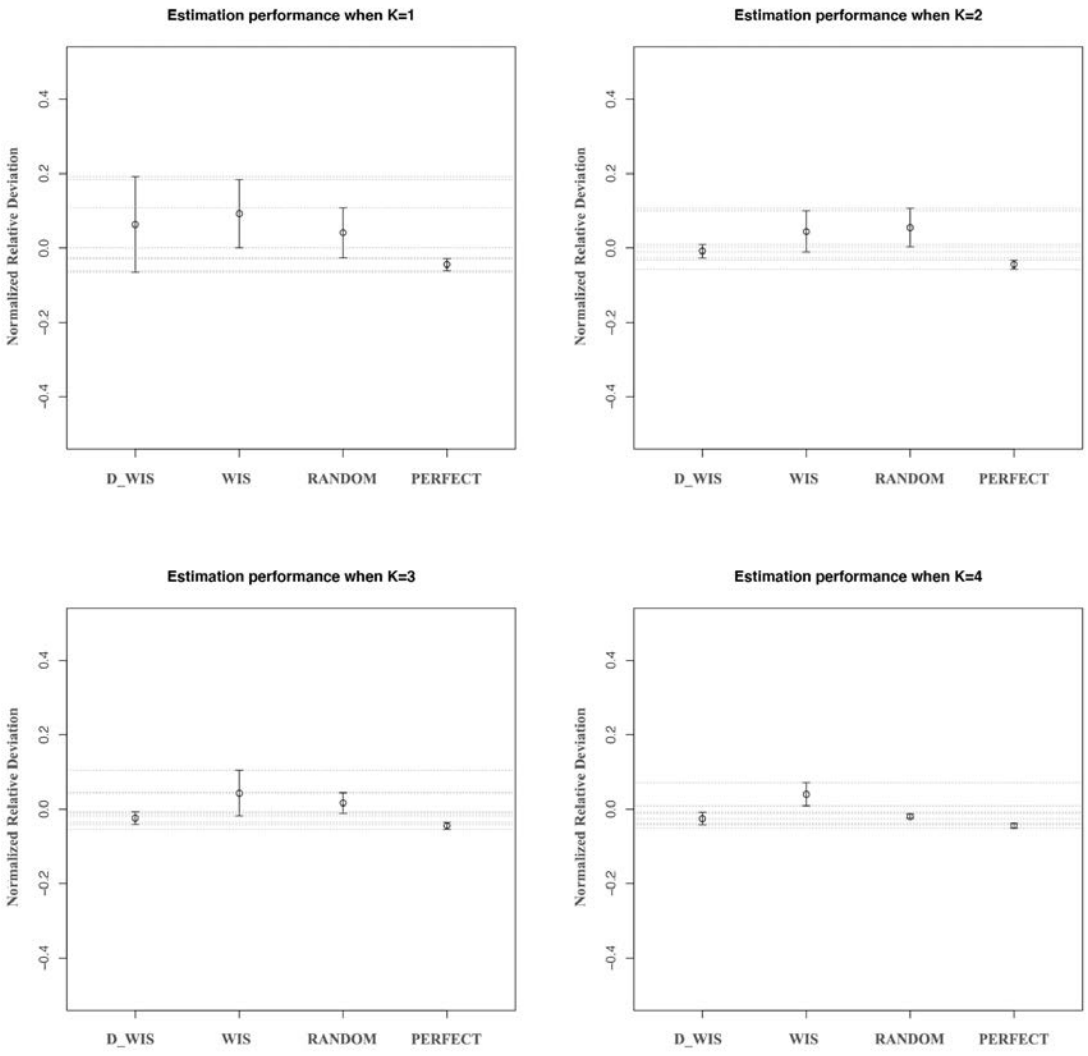

Figure 4.8: The results of the local search experiment for the foraging mission. The 4 plots correspond to different K values.

The K value represents the number of estimations from which the finite-state machine is selected based on the measured performance.

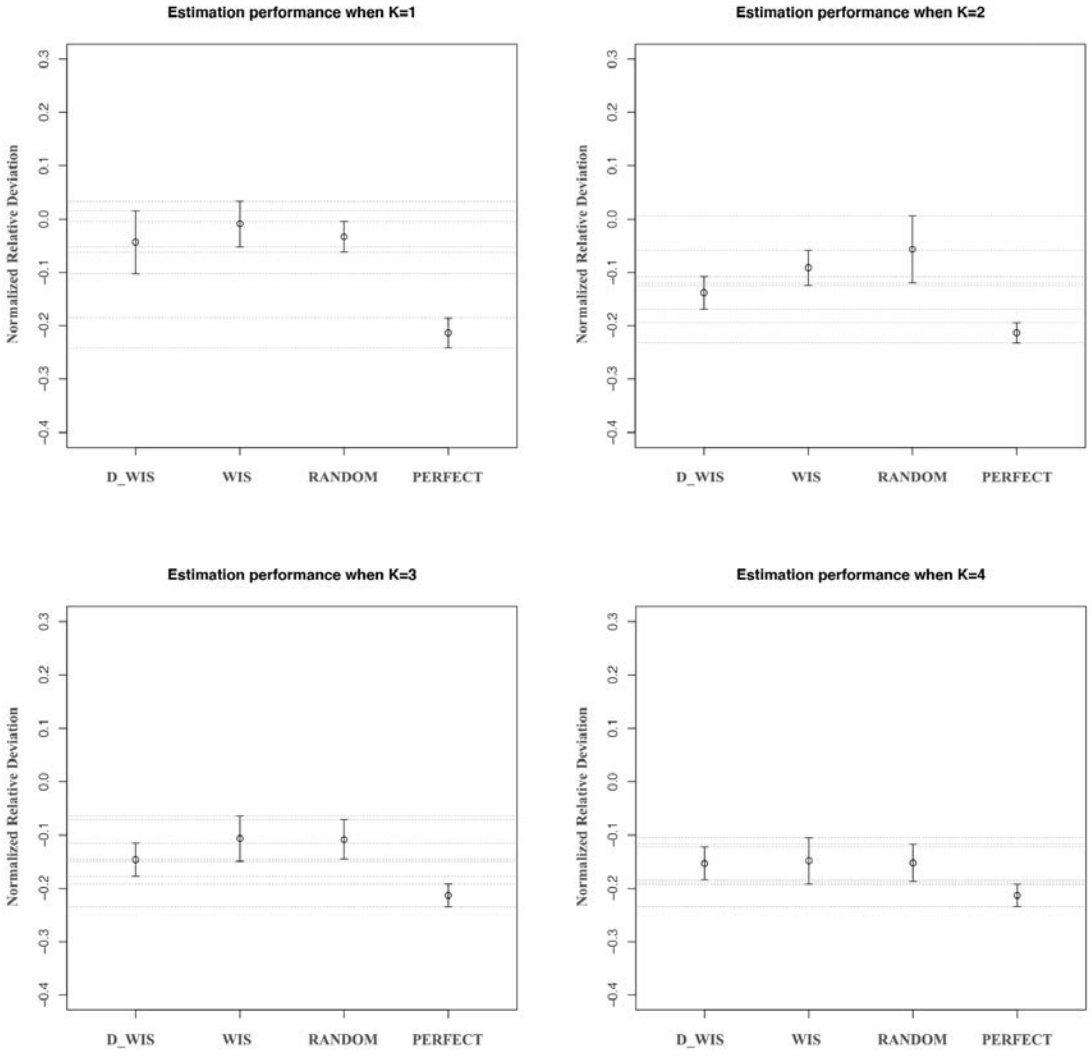

Figure 4.9: The results of the local search experiment for the aggregation mission. The 4 plots correspond to different K values. The K value represents the number of estimations from which the finite-state machine is selected based on the measured performance.

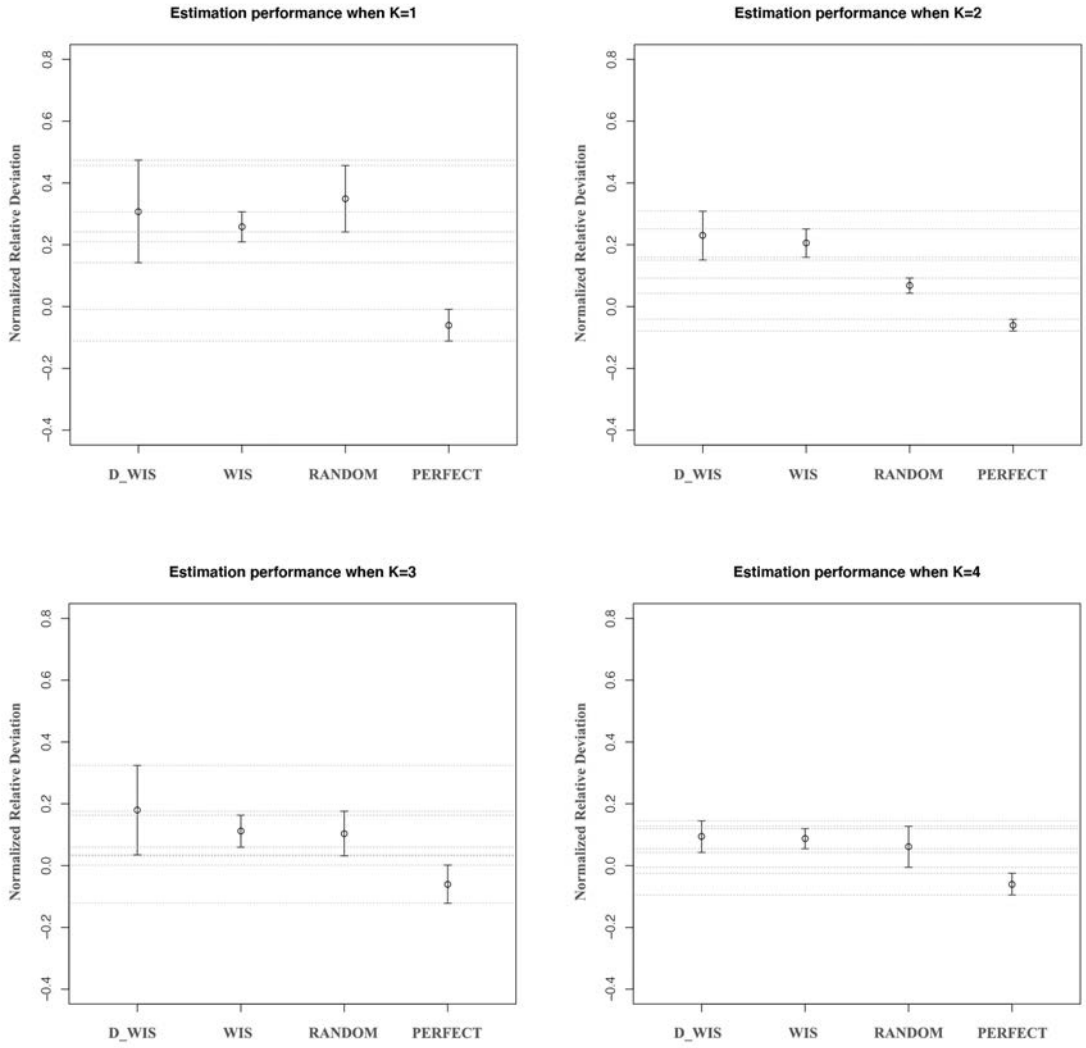

Figure 4.10: The results of the local search experiment for the shelter with constrained access mission.
The 4 plots correspond to different K values. The K value represents the number of estimations from which the finite-state machine is selected based on the measured performance.

Chapter 5

Conclusion

Swarm robotics is a field that offers great prospects for the future. To achieve its full potential, the process of developing control software for a swarm of robots needs to be improved. In the literature, there are many methods to generate the control software: manual, automatic online and automatic offline methods. The automatic methods are preferable otherwise the control software has to be manually developed for every mission that the robots have to achieve. One proposed solution that produced good results is AutoMoDe. AutoMoDe-Chocolate is an automatic offline method that uses an optimization algorithm to generate a finite-state machine which represents the control software. In the reported results, AutoMoDe is capable of crossing the reality gap better than other automatic methods.

The finite-state machine can be produced using various optimization algorithms. One possible choice is to use local search. A local search algorithm would start from an initial control software generated randomly and it would apply mutations on it in order to generate a neighborhood. The best performing neighbor is picked as the new solution and its neighborhood is generated. This process continues until a good enough solution is found. To evaluate the neighborhood of finite-state machines, each one has to be executed in a simulator which can make the algorithm run for a very long time.

A solution proposed by Pagnozzi and Birattari (2021) is to use importance sampling to predict the performance of a finite-state machine that was obtained by modifying some state or transition parameters from an original finite-state machine. In this thesis, new methods of predicting the performance of a finite-state machine are proposed. These methods use more information presented in the execution traces of the original control software such as intermediate states and rewards to improve the prediction. The performance of the new methods and the ones presented by Pagnozzi and Birattari (2021) are compared against each other and to a naive estimator by running multiple experiments with multiple missions. Moreover, experiments in the context of local search are also performed in order to understand how well the methods can predict the performance of the neighborhood of a giving solution.

In the experiments where the methods are compared against a naive estimator, the new methods proposed in this thesis were consistent across multiple missions. They performed similar or better than the naive estimator and there is no performance variance relative to the performance of the naive estimator. However, the methods proposed by Pagnozzi and Birattari (2021) show a larger variance in estimation performance across different missions. For example, in the aggregation mission the WIS estimator was performing well, however, in the shelter with constrained access mission its performance dropped and showed a large variance. A correlation analysis was performed using Person's test and it revealed that in most cases the new proposed estimation

methods were positively correlated with the measured performance of the modified finite-state machines.

Another conclusion that can be drawn from the results is that the methods can predict more easily the performance of the finite-state machines obtained by removing states from the original finite-state machine than to predict the change in performance from the ones obtained by modifying state transition parameters. The main reason behind this might be that removing a state causes bigger perturbation in a control software than modifying the parameter of a transition. The methods can offer better estimations when the changes in performance are high and they have a harder time estimating subtle changes.

Three experiments, one for each mission, were executed to understand how well could these methods estimate the performance of neighbor solutions in the context of local search. The results showed that the D_{WIS} method can help achieve a slightly better solution than the random estimator on average by running the simulator only for the top three finite-state machines that obtained the highest performance prediction according to the method. An exception is the shelter with constrained mission where the methods did not provide an improvement in performance compared to the random estimator. The WIS estimator that was presented by Pagnozzi and Birattari (2021) does not seem to achieve better solutions than the random estimator on average. The gap between the performance of the methods presented here and the performance of the perfect estimator shows that there is still room for improvements.

In all the experiments it can be noticed that the performance of the estimator varies from mission to mission. This effect is less seen in the case of the D_{WIS} estimator which seems to generalize better across missions. One of the reasons why the predictor could behave badly is a high variance in the performance of a control software. This means that a control software that is deployed on a swarm of robots could produce a score that varies substantially from one run to another. The score that the predictor is trying to estimate is the average score obtained by a particular finite-state machine over multiple runs and a large variance around this mean could have a negative impact on this prediction.

The results presented here are promising and future work might improve the results. In the methods developed so far a simplifying assumption was taken: the robots are behaving independently and the traces of one robot are considered a single, independent episode. Nevertheless, the behavior of the swarm emerges from the local interactions between the robots. A future improvement that takes into consideration the joint probability of transitioning between states across all the robots might offer a better performance. However, a problem with this approach might be that more simulations have to be run in order to gather enough data to cover all possible combinations of states and state transitions.

One problem with the way importance sampling is used in the D_{WIS} is the fact that its variance can become very large or even infinite (Sutton and Barto, 2018). In each experiment performed in this thesis, the length of an episode is 2500. In order to compute the return for the first state, the intermediate reward obtained in that state will have to be multiplied by the importance sampling coefficient that is the result of 2500 multiplications. This can add a lot to the variance without adding any more value especially if the discount factor is closer to 0. The idea presented by Sutton and Barto (2018) is to use a discounting-aware importance sampling. By using this, the importance sampling coefficients used to weight the returns obtained at the beginning of an episode are computed until the horizon needed to cover the discounted intermediate rewards that contribute to that state. By doing this, lower values of discount factor can be used and the variance of the method might be reduced.

A good prediction mechanism that is capable of predicting the performance of a control software can be used to improve the existing automatic design methods or even create new better ones. Improved methods of designing control software for swarm robotics can help the

field reach its full potential.

Code availability statement

The source code of the proposed technique can be found at `https://github.com/vladiulianbogdan/MasterThesisVUB-Prediction-Using-Importance-Sampling`.

Appendix A

Correlation Analysis

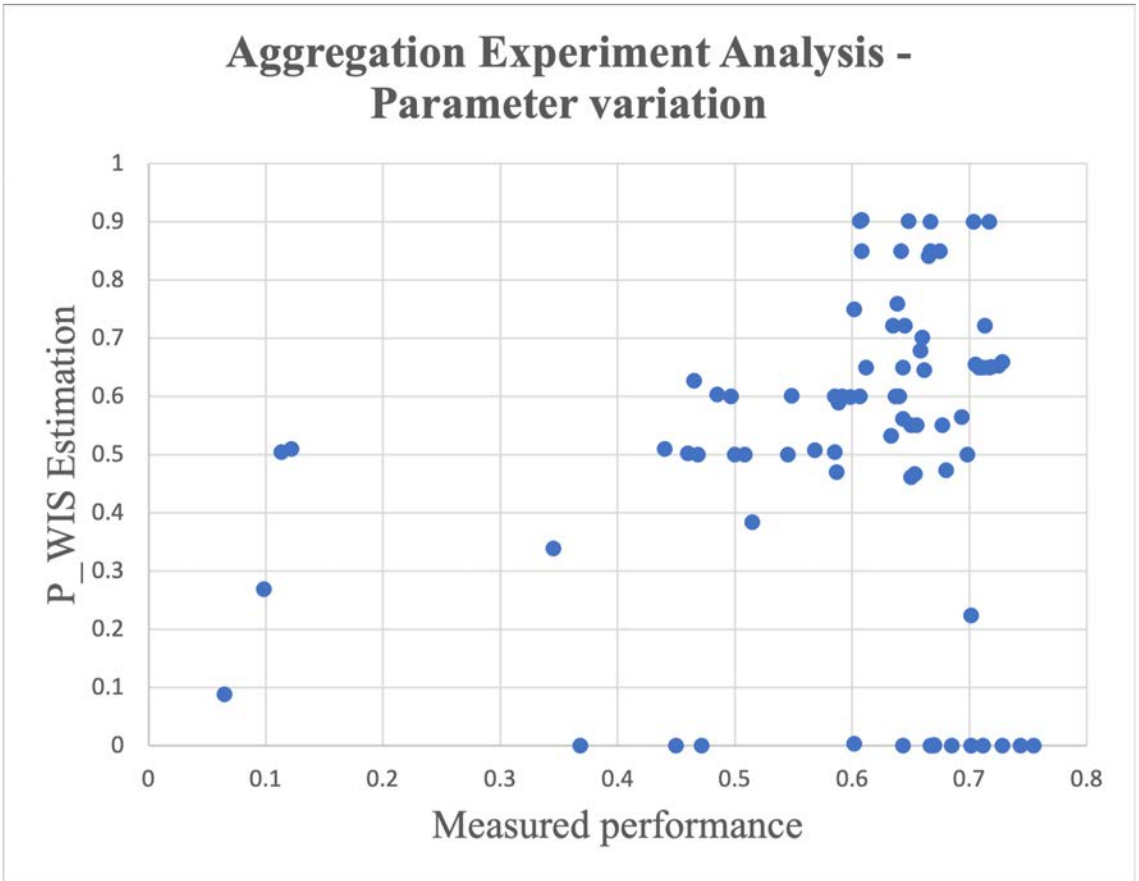

Figure A.1: The correlation between the P_{WIS} estimator and the measured performance in the aggregation parameter variation experiment.

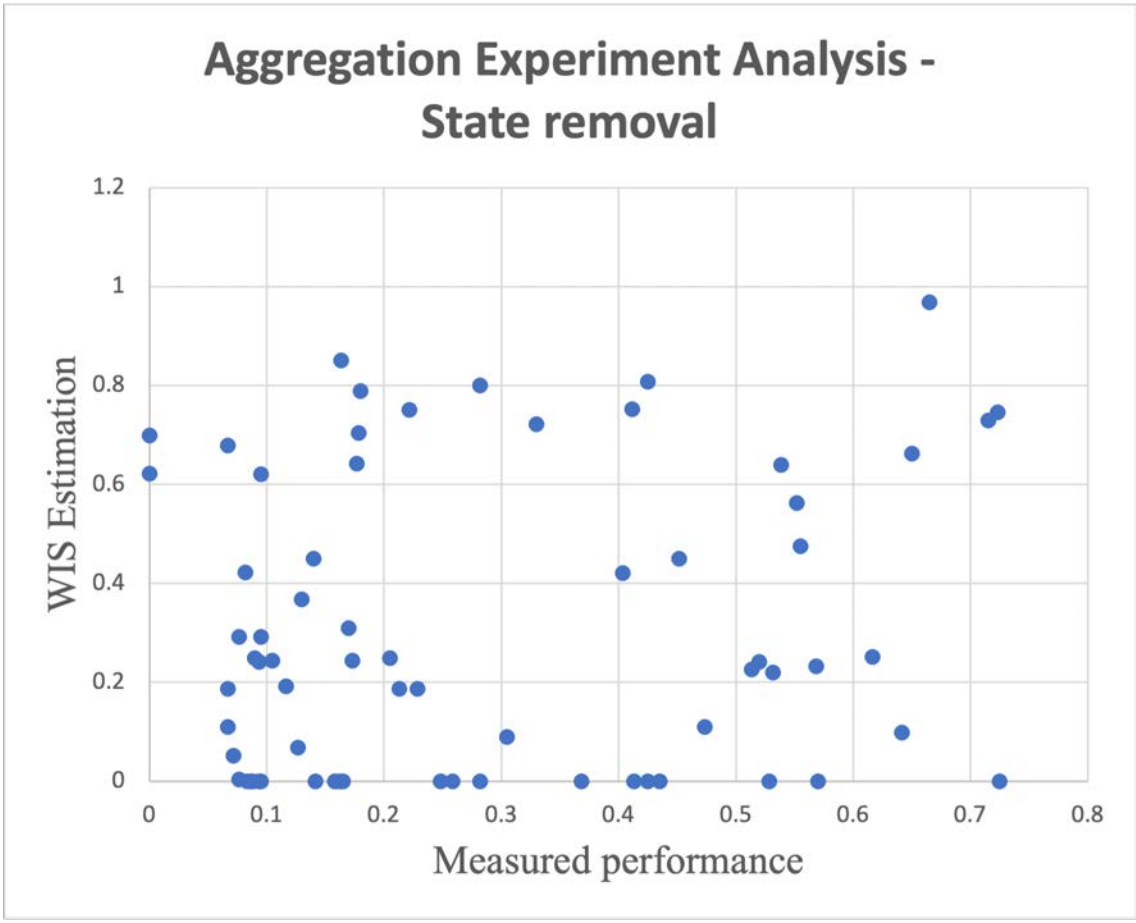

Figure A.2: The correlation between the WIS estimator and the measured performance in the aggregation state removal experiment.

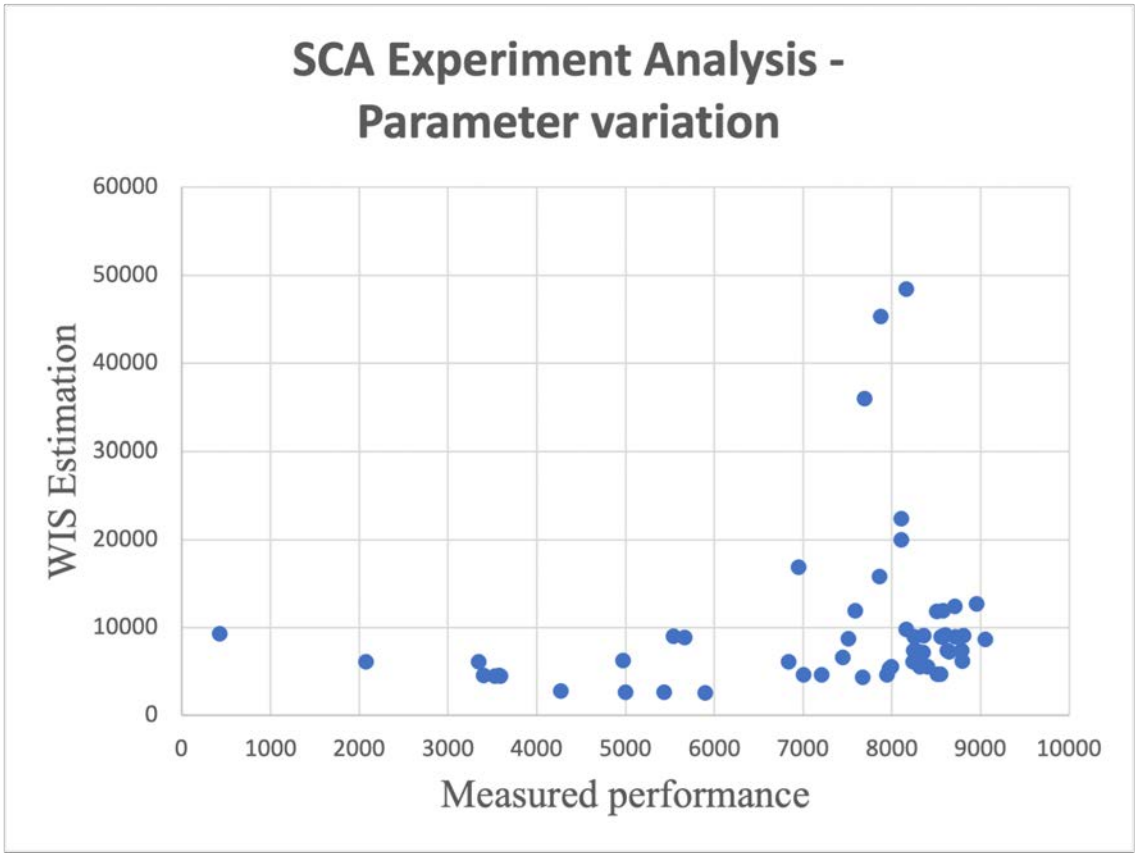

Figure A.3: The correlation between the WIS estimator and the measured performance in the shelter with constrained access parameter variation experiment.

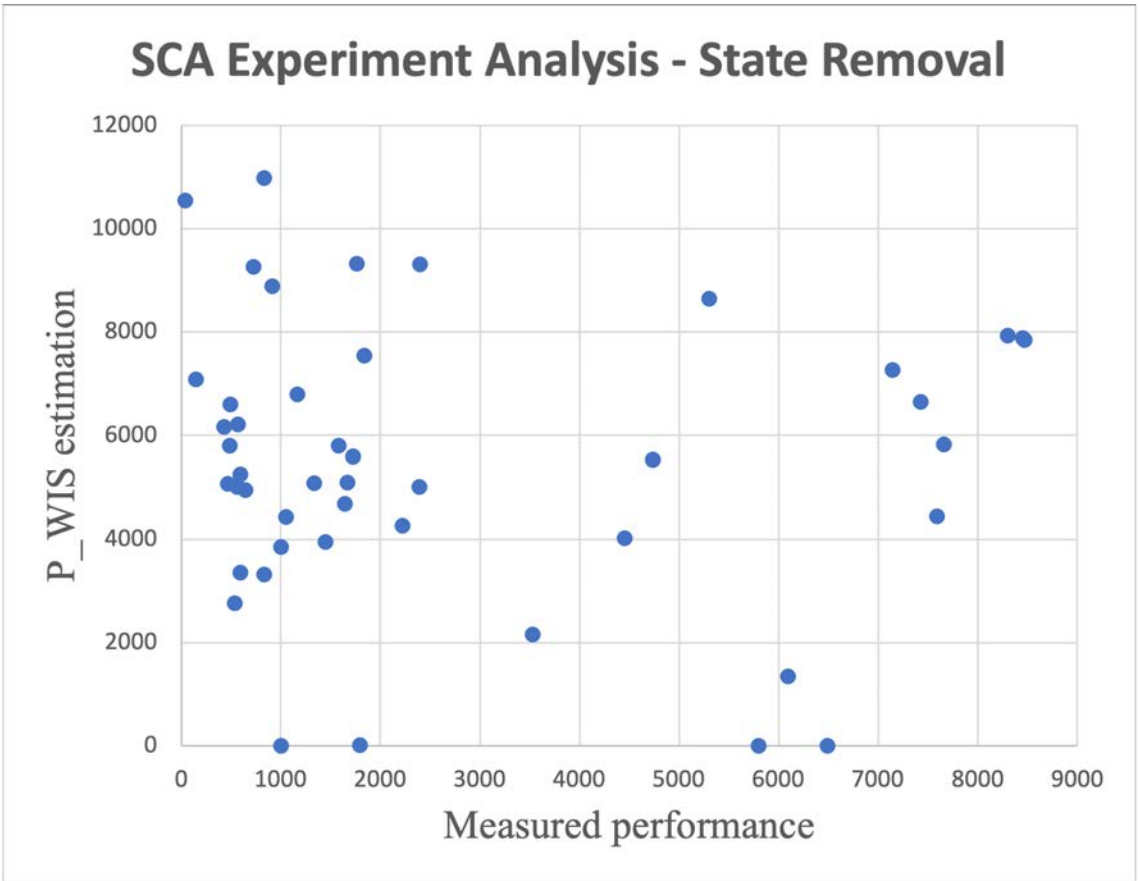

Figure A.4: The correlation between the P_{WIS} estimator and the measured performance in the shelter with constrained access state removal experiment.

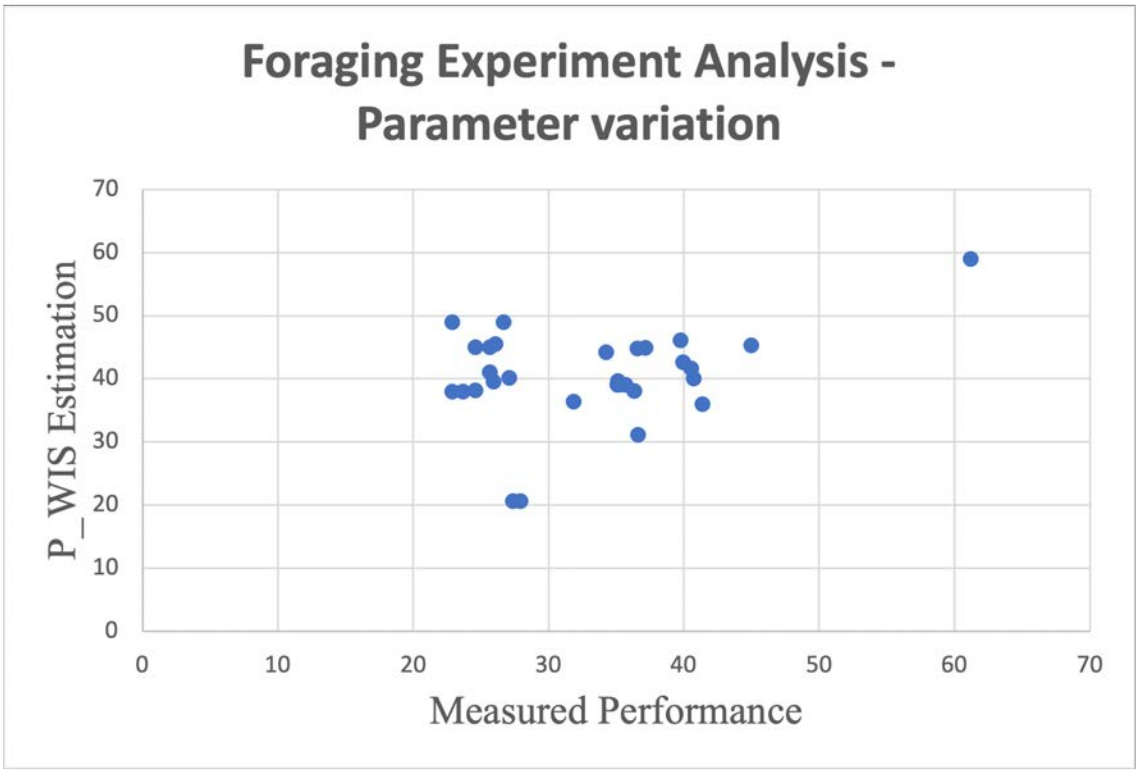

Figure A.5: The correlation between the P_{WIS} estimator and the measured performance in the foraging parameter variation experiment.

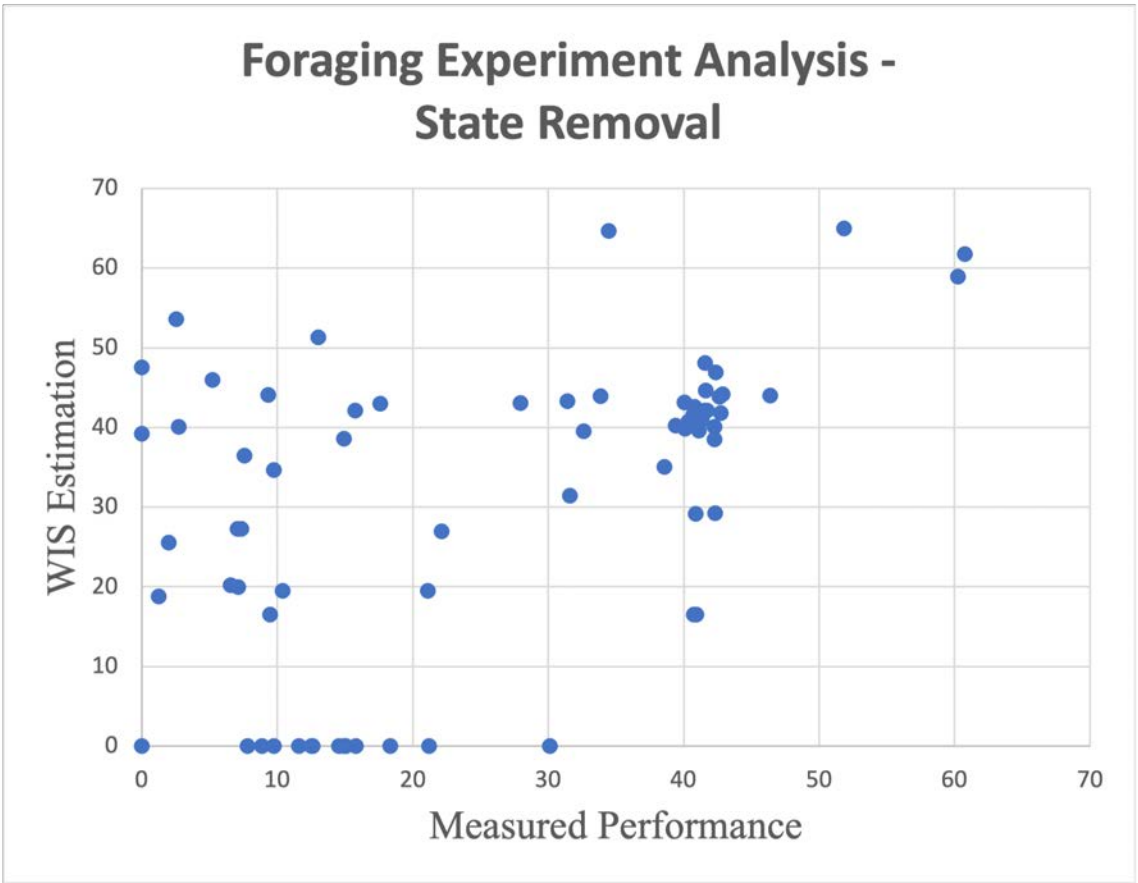

Figure A.6: The correlation between the WIS estimator and the measured performance in the foraging state removal experiment.

Bibliography

Baldassarre, G., Nolfi, S., and Parisi, D. (2003). Evolving mobile robots able to display collective behaviors. *Artificial life*, 9(3):255–267.

Beasley, R. A. (2012). Medical robots: current systems and research directions. *Journal of Robotics*, 2012.

Beni G, W. J. (1989). Swarm intelligence in cellular robotic systems. *Proceed. NATO Advanced Workshop on Robots and Biological Systems*.

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

Birattari, M., Ligot, A., and Francesca, G. (2021). Automode: a modular approach to the automatic off-line design and fine-tuning of control software for robot swarms. In Pillay, N. and (Eds.), R. Q., editors, *Automated Design of Machine Learning and Search Algorithms*, pages 73–90, Cham, Switzerland. Springer Nature.

Birattari, M., Ligot, A., and Hasselmann, K. (2020). Disentangling automatic and semi-automatic approaches to the optimization-based design of control software for robot swarms. *Nature Machine Intelligence*, 2(9):494–499.

Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In Langdon, W. B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M. A., Schultz, A. C., Miller, J. F., Burke, E. K., and Jonoska, N., editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO*, pages 11–18, San Francisco, CA, USA. Morgan Kaufmann Publishers.

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

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

Brooks, R. A., Maes, P., Mataric, M. J., and More, G. (1990). Lunar base construction robots. In *EEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications*, pages 389–392. IEEE.

C. Ronald Kube, E. B. (2000). Cooperative transport by ants and robots. *Robotics and Autonomous Systems 30*, pages 85–110.

Cassinis, R., Bianco, G., Cavagi, A., and Ransenigo, W. (2000). Landmines detection methods using swarms of simple robots. In *Proc. International Conference on Intelligent Autonomous Systems*, volume 6.

Chris Melhuish, Owen Holland, S. H. (1998). Collective sorting and segregation in robots with minimal sensing. *5th Int. Conf. on the Simulation of Adaptive Behaviour*.

Cousineau, L. and Miura, N. (1998). *Construction robots: the search for new building technology in Japan*. ASCE Publications.

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

Deneubourg, J.-L., Aron, S., Goss, S., and Pasteels, J. (1990). The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior*, 3(2):159–168.

Di Caro, G. A., Ducatelle, F., and Gambardella, L. M. (2005). AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks. *European Transactions on Telecommunications*, 16(5):443–455.

Dorigo, M. and Birattari, M. (2007). Swarm intelligence. *Scholarpedia*, 2(9):1462.

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

Dorigo, M., Birattari, M., and Stutzle, T. (2006). Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39.

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

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

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

Floreano, D., Husbands, P., and Nolfi, S. (2008). Evolutionary robotics. In Siciliano, B. and Khatib, O., editors, *Springer Handbook of Robotics*, Springer Handbooks, pages 1423–1451. Springer, Berlin, Heidelberg, Germany. First edition.

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

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

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

Garzón Ramos, D. and Birattari, M. (2020). Automatic design of collective behaviors for robots that can display and perceive colors. *Applied Sciences*, 10(13):4654.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58.

Goss, S., Aron, S., Deneubourg, J.-L., and Pasteels, J. M. (1989). Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76(12):579–581.

Grassé, P. P. (1950). Termites et sols tropicaux. *Journal d'agriculture traditionnelle et de botanique appliquée*, 30(337):549–554.

Hasselmann, K., Ligot, A., Ruddick, J., and Birattari, M. (2021). Empirical assessment and comparison of neuro-evolutionary methods for the automatic off-line design of robot swarms. *Nature communications*, 12(1):1–11.

Hasselmann, K., Robert, F., and Birattari, M. (2018). Automatic design of communication-based behaviors for robot swarms. In Dorigo, M., Birattari, M., Garnier, S., Hamann, H., Montes de Oca, M., Solnon, C., and Stützle, T., editors, *Swarm Intelligence – ANTS*, volume 11172 of *LNCS*, pages 16–29, Cham, Switzerland. Springer.

Hilder, J., Naylor, R., Rizihs, A., Franks, D., and Timmis, J. (2014). The pi swarm: A low-cost platform for swarm robotics research and education. In *Conference Towards Autonomous Robotic Systems*, pages 151–162. Springer.

Jakobi, N., Husbands, P., and Harvey, I. (1995). Noise and the reality gap: the use of simulation in evolutionary robotics. In Morán, F., Moreno, A., Merelo, J. J., and Chacón, P., editors, *Advances in Artificial Life: Third european conference on artificial life*, volume 929 of *Lecture Notes in Artificial Intelligence*, pages 704–720, Berlin, Germany. Springer.

Jorgensen, M. W., Ostergaard, E. H., and Lund, H. H. (2004). Modular atron: Modules for a self-reconfigurable robot. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 2, pages 2068–2073. Ieee.

Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Al Sallab, A. A., Yogamani, S., and Pérez, P. (2021). Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*.

Kuckling, J., Arriaza, K. U., and Birattari, M. (2019). Automode-icepop: Automatic modular design of control software for robot swarms using simulated annealing. In *Artificial Intelligence and Machine Learning*, pages 3–17. Springer.

Kuckling, J., Ligot, A., Bozhinoski, D., and Birattari, M. (2018). Behavior trees as a control architecture in the automatic design of robot swarms: supplementary material. `http://iridia.ulb.ac.be/supp/IridiaSupp2018-004/index.html`.

Labella, T. H., Dorigo, M., and Deneubourg, J.-L. (2006). 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.

Liekna, A., Grundspenkis, J., et al. (2014). Towards practical application of swarm robotics: overview of swarm tasks. *Engineering for rural development*, 13:271–277.

Ligot, A., Hasselmann, K., and Birattari, M. (2020). AutoMoDe-Arlequin: neural networks as behavioral modules for the automatic design of probabilistic finite state machines: supplementary material. http://iridia.ulb.ac.be/supp/IridiaSupp2020-005/index.html.

Liu, W., Winfield, A. F., Sa, J., Chen, J., and Dou, L. (2007). Towards energy optimization: Emergent task allocation in a swarm of foraging robots. *Adaptive behavior*, 15(3):289–305.

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

Miglino, O., Lund, H. H., and Nolfi, S. (1995). Evolving mobile robots in simulated and real environments. *Artificial Life*, 2(4):417–434.

Miglino, O., Nafasi, K., and Taylor, C. E. (1994). Selection for wandering behavior in a small robot. *Artificial Life*, 2(1):101–116.

Moffett, M. W. (1988). Cooperative transport by an asiatic ant. *National Geographic Research 4*, pages 386–394.

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

Mondada, F., Guignard, A., Bonani, M., Bar, D., Lauria, M., and Floreano, D. (2003). SWARM-BOT: from concept to implementation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, volume 2, pages 1626–1631, Piscataway, NJ, USA. IEEE.

Nolfi, S., Floreano, D., Miglino, O., and Mondada, F. (1994). How to evolve autonomous robots: different approaches in evolutionary robotics. In Brooks, R. A. and Maes, P., editors, *Artificial Life IV: Proceedings of the Workshop on the Synthesis and Simulation of Living Systems*, pages 190–197, Cambridge, MA, USA. MIT Press. A Bradford Book.

Pagnozzi, F. and Birattari, M. (2021). Off-policy evaluation of the performance of a robot swarm: Importance sampling to assess potential modifications to the finite-state machine that controls the robots. *Frontiers in Robotics and AI*.

Passino K.M., S. T. (2006). Modeling and analysis of nest-site selection by honeybee swarms: the speed and accuracy trade-off. *Behav Ecol Sociobiol*, page 427–442.

Patil, M., Abukhalil, T., and Sobh, T. (2013). Hardware architecture review of swarm robotics system: Self-reconfigurability, self-reassembly, and self-replication. *International Scholarly Research Notices*, 2013.

Petersen, K. H., Nagpal, R., and Werfel, J. K. (2011). Termes: An autonomous robotic system for three-dimensional collective construction. *Robotics: science and systems VII*.

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

Pini, G., Brutschy, A., Birattari, M., and Dorigo, M. (2009). Task partitioning in swarms of robots: reducing performance losses due to interference at shared resources. In Andrade Cetto, J., Filipe, J., and Ferrier, J.-L., editors, *Informatics in Control Automation and Robotics*, volume 85 of *Lecture Notes in Electrical Engineering*, pages 217–228, Berlin, Heidelberg, Germany. Springer.

Quinn, M., Smith, L., Mayley, G., and Husbands, P. (2003). 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.

Reynolds, C. (1987). Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*.

Riedmiller, M., Gabel, T., Hafner, R., and Lange, S. (2009). Reinforcement learning for robot soccer. *Autonomous Robots*, 27(1):55–73.

Rubinstein, R. Y. and Kroese, D. P. (2016). *Simulation and the Monte Carlo method*, volume 10. John Wiley & Sons.

Salman, M., Ligot, A., and Birattari, M. (2019). Concurrent design of control software and configuration of hardware for robot swarms under economic constraints. *PeerJ Computer Science*, 5:e221.

Schmidt, R. S. (1964). Apticotermes nests. *American Zoologist*, pages 221–225.

Soysal, O. and Sahin, E. (2005). Probabilistic aggregation strategies in swarm robotic systems. In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, pages 325–332. IEEE.

Stampa, G., Arias, M., Sánchez-Charles, D., Muntés-Mulero, V., and Cabellos, A. (2017). A deep-reinforcement learning approach for software-defined networking routing optimization. *arXiv preprint arXiv:1709.07080*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Trianni, V. and Nolfi, S. (2011). Engineering the evolution of self-organizing behaviors in swarm robotics: A case study. *Artificial life*, 17(3):183–202.

Trianni, V., Nolfi, S., and Dorigo, M. (2008). Evolution, self-organization and swarm robotics. In *Swarm Intelligence*, pages 163–191. Springer.

Tuci, E., Ampatzis, C., Vicentini, F., and Dorigo, M. (2008). Evolving homogeneous neurocontrollers for a group of heterogeneous robots: Coordinated motion, cooperation, and acoustic communication. *Artificial Life*, 14(2):157–178.

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

Wei, H., Cai, Y., Li, H., Li, D., and Wang, T. (2010). Sambot: A self-assembly modular robot for swarm robot. In *2010 IEEE International Conference on Robotics and Automation*, pages 66–71. IEEE.