UNIVERSITÄT ZU LÜBECK
INSTITUTE OF COMPUTER ENGINEERING

From the Institute of Computer Engineering
of the University of Lübeck

Director: Prof. Dr.-Ing. Mladen Berekovic

The Predicting Swarm: Evolving Collective Behaviors for Robot Swarms by Minimizing Surprise

Dissertation
for Fulfillment of
Requirements
for the Doctoral Degree
of the University of Lübeck

from the Department of Computer Sciences and Technical Engineering

Submitted by

Tanja Katharina Kaiser
from Bamberg, Germany

Lübeck, 2022

First referee: Prof. Dr.-Ing. Heiko Hamann

Second referee: Prof. Dr.-Ing. habil. Sanaz Mostaghim

Date of oral examination: August 29, 2022

Approved for printing. Lübeck, February 2, 2023

Abstract

Robot swarms are decentralized collective systems of simple embodied agents that act autonomously and rely on local information only. Such large-scale multi-robot systems can be beneficial over single robots due to higher potential for robustness and scalability. However, the development of swarm robot controllers is challenging because when implementing a desired swarm behavior one has to take into account local interactions between robots and between robots and the environment. An alternative is the automatic design of swarm robot controllers using methods of evolutionary robotics. Since evolutionary algorithms maximize fitness potentially by every possible way, undesired side effects may occur if a goal-directed fitness function was not specified accurately enough. By contrast, task-independent fitness functions avoid the specific formulation of rewards but do not guarantee that desired behaviors emerge. Our minimize surprise approach relies on such a task-independent fitness function to evolve diverse collective behaviors for robot swarms. Surprise, in its simplest form here, is the difference between observed and predicted sensor values. We minimize surprise over generations by equipping each swarm member with an actor-predictor pair of artificial neural networks and putting direct selection pressure on the predictor. The actor is only indirectly rewarded by being paired with the predictor and thus swarm behaviors emerge as a desired by-product.

In the first part of this thesis, we study minimize surprise as a method. In a simple simulated self-assembly scenario, we show the effectiveness of our approach in comparison to random search, the scalability of the evolved behaviors with swarm density, as well as the robustness of evolution against sensor noise and of the emergent behaviors against damage to the self-assembled structure. We also show that the resulting behavioral diversity of our standard minimize surprise approach is competitive to the behavioral diversity generated by task-independent novelty search and MAP-Elites variants. In addition, we demonstrate that self-organization in minimize surprise can be engineered towards desired behaviors by predefining some or all sensor predictions. In a more realistic simulation, we illustrate how modifications of the environment (e.g., dynamically changing obstacle positions), the agents (e.g., enabling battery level sharing), and the fitness function (e.g., adding a reward for homing) can influence the evolution of behaviors.

In the second part of this thesis, we study the evolution of collective behaviors with minimize surprise in different application scenarios. We evolve collective decision-making behaviors for a collective perception task in the realistic BeeGround simulator, and collective construction behaviors on a simple 2D torus grid. Furthermore, we make the step to real-world setups and evolve basic swarm behaviors and object manipulation behaviors in the realistic Webots simulator and on swarms of real Thymio II robots using an online onboard evolutionary approach to minimize surprise.

Overall, we show that our minimize surprise approach allows the effective evolution of diverse, robust, and scalable swarm behaviors for a variety of application scenarios in simple simulations, realistic simulators, and real-world experiments. Moreover, evolution can be pushed towards desired behaviors through the modification of the environment, robot model, and predictor outputs. Potentially allowing open-ended adaptation to non-anticipated situations, minimize surprise can help tackle the challenges of robotics.

Zusammenfassung

Roboterschwärme sind dezentralisierte kollektive Systeme einfacher verkörperter Agenten, die autonom handeln und sich nur auf lokale Informationen stützen. Solche großen Multi-Roboter-Systeme können im Vergleich zu einzelnen Robotern aufgrund ihrer potenziell höheren Robustheit und Skalierbarkeit von Vorteil sein. Die Entwicklung von Steuerungen für Roboterschwärme ist jedoch eine Herausforderung, da man bei der Implementierung eines gewünschten Schwarmverhaltens lokale Wechselwirkungen zwischen den Robotern sowie zwischen Robotern und der Umgebung berücksichtigen muss. Eine Alternative ist die automatische Entwicklung von Schwarmrobotersteuerungen mit Methoden der evolutionären Robotik. Da evolutionäre Algorithmen Fitness potenziell auf jedem möglichen Weg maximieren, können unerwünschte Nebeneffekte auftreten, wenn eine aufgabenspezifische Fitnessfunktion nicht genau genug spezifiziert wurde. Im Gegensatz dazu vermeiden aufgabenunabhängige Fitnessfunktionen die spezifische Formulierung von Belohnungen, aber garantieren nicht, dass sich gewünschte Verhaltensweisen entwickeln. Unser Minimize Surprise (zu Deutsch etwa Überraschungsminimierung) Ansatz basiert auf einer solchen aufgabenunabhängigen Fitnessfunktion, um vielfältige kollektive Verhaltensweisen für Roboterschwärme zu evolvieren. Überraschung, hier in seiner einfachsten Form, ist die Differenz zwischen beobachteten und vorhergesagten Sensorwerten. Wir minimieren Überraschung über Generationen, indem wir jedes Schwarmmitglied mit einem Aktor-Prädiktor-Paar aus künstlichen neuronalen Netzen ausstatten und direkten Selektionsdruck auf den Prädiktor ausüben. Der Aktor wird nur indirekt aufgrund der Kombination mit einem Prädiktor belohnt, sodass Schwarmverhalten als erwünschtes Nebenprodukt entstehen.

Im ersten Teil dieser Arbeit befassen wir uns mit Minimize Surprise als Methode. In einem einfachen simulierten Selbstassemblierungsszenario zeigen wir die Effektivität unseres Ansatzes im Vergleich zu zufälliger Suche, die Skalierbarkeit der entwickelten Verhaltensweisen mit der Schwarmdichte sowie die Robustheit der Evolution gegenüber Sensorrauschen und der entstehenden Verhaltensweisen gegenüber Beschädigungen der selbstassemblierten Struktur. Wir zeigen zudem, dass die resultierende Verhaltensvielfalt unseres Standardansatzes zur Minimierung von Überraschungen mit der Verhaltensvielfalt konkurriert, die durch aufgabenunabhängige Varianten der Novelty Search (zu Deutsch etwa Neuheitensuche) und MAP-Elites Ansätze entstehen. Darüber hinaus zeigen wir, dass Selbstorganisation in unserem Minimize Surprise Ansatz durch die Vordefinition einiger oder aller Sensorvorhersagen in Richtung gewünschter Verhalten gelenkt werden kann. In einer realistischeren Simulation veranschaulichen wir, wie Modifikationen der Umgebung (z. B. dynamische Änderung der Hindernispositionen), der Agenten (z. B. Teilen des Akkustandes) und der Fitnessfunktion (z. B. Hinzufügen einer Belohnung für Homing (zu Deutsch etwa Heimkehr) Verhalten) die Evolution von Verhalten beeinflussen können.

Im zweiten Teil dieser Arbeit untersuchen wir die Evolution kollektiver Verhaltensweisen mit Minimize Surprise in verschiedenen Anwendungsszenarien. Wir evolvieren Verhalten zur kollektiven Entscheidungsfindung für eine Aufgabe im Bereich kollektiver Wahrnehmung im realistischen BeeGround-Simulator und kollektives Bauverhalten in einer einfachen 2D-Torus-Gitterwelt. Darüber hinaus machen wir den Schritt in die reale

Welt und evolvieren grundlegende Schwarmverhalten und Verhalten der Objektmanipulation im realistischen Webots-Simulator und auf Schwärmen echter Thymio II-Roboter. Dazu verwenden wir einen evolutionären Ansatz für Minimize Surprise bei dem Verhalten direkt auf den Robotern während ihrer Einsatzzeit in der echten Arena evolviert werden.

Insgesamt zeigen wir, dass unser Minimize Surprise Ansatz die effektive Entwicklung von vielfältigen, robusten und skalierbaren Schwarmverhalten für eine Vielzahl von Anwendungsszenarien in einfachen Simulationen, realistischen Simulatoren und in Experimenten in der echten Welt ermöglicht. Darüber hinaus zeigen wir, dass die Evolution durch Modifikation der Umgebung, des Robotermodells und der Prädiktorausgaben in Richtung gewünschter Verhalten gelenkt werden kann. Da Minimize Surprise potenziell die permanente Anpassung an unvorhergesehene Situationen ermöglicht, kann es dazu beitragen die Herausforderungen der Robotik zu lösen.

Contents

Introduction 1

Chapter Contents

In this chapter, we discuss our...

- *Sec. 1.1:* motivation,
- *Sec. 1.2:* objectives, and
- *Sec. 1.3:* contributions and the thesis outline.

1.1 Motivation

Nature frequently serves as a source of inspiration for roboticists. For example, soft robotics is inspired by the use of soft materials in animals [1], such as jellyfish (see Fig. 1.1), and behavior-based robotics takes inspiration from psychological, neuroscientific, and ethological concepts of behavior [2]. This work is in the context of swarm robotics (see Sec. 2.1.2), which is inspired by natural swarms and collective systems, such as ant colonies [3, 4] and flocks of birds [5]. Robot swarms are decentralized collective systems consisting of simple embodied agents that act autonomously and rely on local information only [6]. While these large-scale multi-robot systems have the potential for higher robustness and scalability than single-robot systems, their manual design is challenging. The collective behavior of the robot swarm on the macro-level (i.e., the swarm or global level) emerges from interactions between individual robots and between robots and the environment on the micro-level (i.e., the individual or local level) [7]. That means, the desired collective behavior is specified on the macro-level by the user, but the robot controller has to be implemented by the system designer on the micro-level while considering hard to anticipate interactions and feedback processes [8, 9].

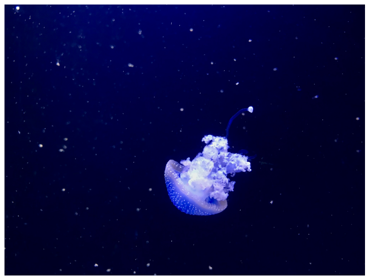


Figure 1.1: Jellyfish (*Phyllorhiza punctata*) at Aquarium Berlin.

The automatic design of swarm robot controllers is a promising alternative [10]. But despite recent improvements in machine learning [11, 12], it is still difficult to automatically generate robot controllers for complex tasks and collective robot systems (see Sec. 2.3.1) [13]. Researchers frequently rely on methods of evolutionary robotics as alternative, optimizing robot controllers using algorithms that are inspired by natural evolution (see Sec. 2.3.2) [14–16]. These methods have also proven to be competitive to standard machine learning methods [17, 18]. In this work, we use methods of evolutionary swarm robotics, which is the application of evolutionary robotics to swarm robotics [8].

In evolutionary swarm robotics, controllers are optimized in an iterative process that is guided by the optimization of a fitness function that also specifies the goal of the search (see Sec. 2.3). The fitness function, that is usually specified by a human designer, measures the performance of a controller candidate when executed in the environment. Higher fitness increases the probability that a controller will be selected and mutated to be evaluated in the next iteration. Consequently, better performing controller candidates survive longer in the evolutionary process.

Most commonly used are task-specific fitness functions that reward the achievement of a desired task or behavior. The designer faces several challenges specifying such goal-directed fitness functions. Evolutionary algorithms maximize fitness potentially by every possible way, which can lead to unwanted behaviors when the fitness function was not specified accurately enough [19]. For example, a system designer may define a fitness function that rewards a robot for every time step it does not hit an obstacle to evolve an obstacle avoidance behavior. In this case, the evolutionary process will most likely result in a behavior that lets the robot stand still, since obstacles cannot be hit when the robot does not move. To avoid such undesired behaviors, fitness functions have to be defined unambiguously, which often requires an extensive iterative refinement process. In addition, there is a tradeoff between rewarding high-level task completion and rewarding specific behavioral features [20]. Since the randomly generated initial controller candidates will most probably not lead to the fulfillment of non-trivial desired tasks, rewarding high-level task completion can cause bootstrapping problems. That is, the fitness of the initial controller candidates is not detectable and thus no selective pressure to guide optimization is generated. By contrast, rewarding specific behavioral features of an assumed solution for a particular task restricts the search to certain solution types potentially causing that more original solutions will not be found.

A different option is to use a task-independent fitness function that puts selection pressure on aspects that are not directly related to a desired task or behavior. While there is no guarantee that desired behaviors will be found, the evolutionary process has the freedom to find original solutions. However, since task-independent approaches frequently lead to the emergence of diverse behaviors within or across evolutionary runs, users can choose from a set of behaviors that may include desired ones. Task-independent fitness functions are studied, for example, in evolutionary robotics to realize engineering tools for the automatic synthesis of robot controllers [21] and in artificial life [22] to enable open-ended evolution and long-term autonomy. A variety of task-independent fitness functions exists, for instance, rewarding behavioral diversity [23], such as in novelty search [24], relying on information-theoretic quantities, such as mutual information [25], or including selective pressure via the environment, such as in robot ecology [26, 27] where agents need to survive long enough to be able to reproduce.

In this work, we investigate a task-independent approach to the evolution of collective behaviors for robot swarms (see Sec. 3.2) in detail. Our minimize surprise approach [28] is inspired by the broader idea of the brain as a prediction machine [29]. Several concepts of artificial intelligence are based on this idea, for example, the so-called Helmholtz machines [30], which are an unsupervised learning approach, and the information-theoretic 'free-energy principle' [31] that states that organisms constantly try to minimize free energy or surprise, that is, in the simplest case, the difference between observed and predicted sensor values (i.e., the prediction error; see Sec. 3.1).

Most studies on the brain as a prediction machine were conducted in single agent settings [32–34] and only recently researchers started to also consider simulated collective systems [35, 36]. In our studies, we evolve collective behaviors in simulation and on real robots by putting selection pressure on minimizing surprise, or, put differently, on maximizing prediction accuracy, in the evolutionary process. For this purpose, each member of a homogeneous swarm is equipped with an actor-predictor pair of artificial neural networks (ANNs; see Sec. 2.3.1). During evolution, direct pressure is only put on the predictor while the actor is only indirectly rewarded by being paired with the predictor. As we minimize surprise over generations, swarm behaviors emerge as a by-product, potentially allowing for a diverse set of solutions [28]. In contrast to single robot settings, a robot's environment is populated by swarm members that are identical to it in our swarm scenarios. Consequently, a robot's sensor input is based both on its own actions and the actions of the other swarm members, which may trigger more active behaviors in a self-referential loop.

First works on minimize surprise [28, 37] showed the potential of the approach to generate diverse collective behaviors across several independent evolutionary runs. However, these first works only evolved basic collective behaviors in simple 1D and 2D torus environments for swarms of simple agents. The resulting behaviors were repetitive, which allows for easy sensor predictions. Furthermore, the analysis focused mainly on the overall success of the approach regarding prediction accuracy and behavioral diversity. Although the results of these studies were promising, they have raised many questions that were left to future work. In this thesis, we address these open questions and even go beyond them by studying minimize surprise in-depth in simple simulations, engineering self-organization towards desired behaviors, and conducting experiments on real robots.

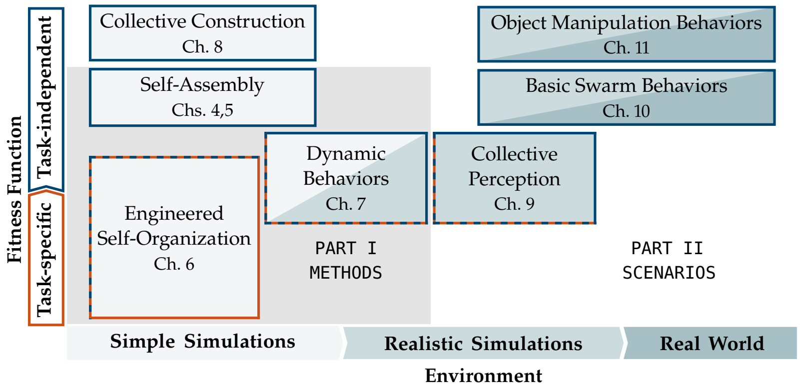


Figure 1.2: Overview of our studies on our minimize surprise approach. The thesis is divided into two parts: Part I focuses on methods and Part II focuses on different scenarios. We do experiments in environments ranging from simple simulations over realistic simulations to the real world, using approaches to minimize surprise that are task-specific, combine task-specific and task-independent rewards, and are fully task-independent. Boxes give the different studies, the color of the outline indicates whether task-specific or task-independent fitness functions or combinations thereof (multicolored lines) were used, and the fill color indicates the used environments.

1.2 Objectives

We address the following six research questions in our studies:

Q1 How robust, scalable, and diverse are collective behaviors evolved with minimize surprise?

Q2 How can we engineer our minimize surprise approach so that desired behaviors emerge?

Q3 How does minimize surprise compare to other approaches?

Q4 How can we evolve dynamic behaviors that adapt their behavior according to varying sensor input using our minimize surprise approach?

Q5 Can we evolve collective behaviors for scenarios with different environmental complexities and agent capabilities with minimize surprise?

Q6 Which adaptations are necessary to apply minimize surprise in real-world settings?

We focus on questions Q1 to Q4 in the first part and on questions Q5 and Q6 in the second part of this thesis.

1.3 Contributions and Thesis Outline

First, we explain the fundamentals of robotics, swarm behaviors, and artificial intelligence to introduce the background for this work in Ch. 2. We then give a detailed introduction to our minimize surprise approach in Ch. 3. As mentioned before, we divide the rest of this thesis into two parts, see Fig. 1.2.

In Part I, we put special focus on methods for our minimize surprise approach and address research questions Q1 to Q4.

Chapter 4 introduces a self-assembly scenario in a simple simulation environment [38–43] (see Fig. 1.3a) that serves as our sample scenario for an in-depth analysis of minimize surprise. We address research question Q1 by studying the emergent behavioral diversity over swarm density and show that minimize surprise outperforms pure random search.

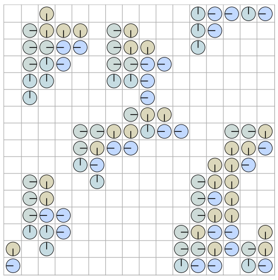
Chapter 5 addresses research questions Q1 and Q3 with an in-depth analysis using our self-assembly scenario [39–43]. We show the robustness of our minimize surprise approach against sensor noise and of the emergent behaviors against damage, the scalability of the behaviors with swarm density, as well as the resulting behavioral diversity in comparison to task-independent novelty search [24] and MAP-Elites variants [44].

Chapter 6 illustrates how self-organization can be engineered towards the evolution of desired behaviors by predefining some or all sensor predictions and shows that this approach is competitive to evolutionary algorithms with task-specific fitness functions using our self-assembly scenario [38–42]. Thus, our minimize surprise approach can be used in variants ranging from being fully task-independent (i.e., no predictions are predefined) to being fully task-specific (i.e., all predictions are predefined). We address research questions Q2 and Q3 in this chapter.

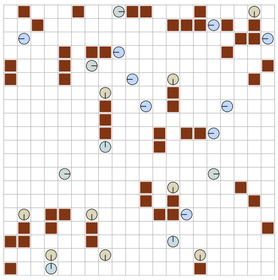
Chapter 7 addresses research question Q4. We study the influence of modifications to the environment, agents, and fitness function on the resulting behaviors aiming to push evolution towards more dynamic behaviors. Here, we also make the first step towards more realistic simulation environments (see Fig. 1.4).

Part II focuses on the evolution of collective behaviors with our minimize surprise approach for various scenarios in simple simulations, realistic simulators and real-world settings, and thus addresses research questions Q5 and Q6.

Chapter 8 investigates the evolution of collective decision-making behaviors with minimize surprise for a collective perception scenario in a realistic simulation environment (see Fig. 1.5a). Here, we show that pure minimize surprise may not be sufficient in all scenarios to evolve behaviors that are useful to the user and that the inclusion of an additional task-specific reward is probably required in some settings.



(a) self-assembly



(b) collective construction

Figure 1.3: Simple simulations

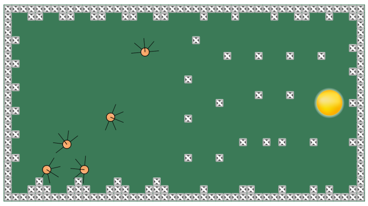
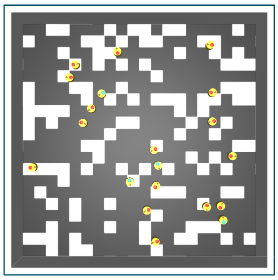


Figure 1.4: Simulation environment for our study on dynamic behaviors.



(a) collective perception in BeeGround [45]

(b) basic swarm behaviors in Webots [46]

Figure 1.5: Realistic simulations

Chapter 9 presents a collective construction scenario in a simple simulation environment [47]. To enable construction, we distribute blocks in the environment that can be pushed around by the agents (see Fig. 1.3b). This is the first scenario in which individual swarm members can not only interact with each other but also with passive objects in their environment.

Chapters 10 and 11 address research question Q6, that is, we finally make the step towards realistic simulations and real-world setups. For this purpose, we present an online onboard evolutionary architecture for minimize surprise. We evolve collective behaviors in two scenarios in realistic simulations (see Fig. 1.5b) and real-world environments (see Fig. 1.6): we aim for basic swarm behaviors [42] in Ch. 10 and for object manipulation behaviors [48] in Ch. 11.

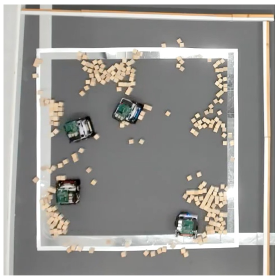


Figure 1.6: Real-world setup for our object manipulation scenario.

Finally, we summarize our results and discuss ideas for future work in Ch. 12.

Fundamentals 2

Chapter Contents

In this chapter, we introduce...

- *Sec. 2.1:* robotics, mobile robotics, and swarm robotics,
- *Sec. 2.2:* swarm behaviors in nature and robotics, and
- *Sec. 2.3:* artificial intelligence including machine learning, evolutionary computation, and intrinsic motivations.

In our research, we aim for the automatic generation of controllers for collective robot systems using an innate motivation as a task-independent reward. This chapter provides an overview over related background to put our research into context.

2.1 Robotics

In 1920, the Czech writer Karel Čapek used the term *robot* for the first time ever to denote a fictional humanoid in his play R.U.R.[1] The term derives from the Czech word *robota* meaning 'forced labor' [49]. Although the word robot is only a century old, the idea of robots and automata goes back to ancient Greek mythology [50]. But it took until the middle of the 20th century until the first real robots were built [51]. Since then, robots have found their way into our daily lives, for example, into manufacturing [52–54], medicine [55–57], space exploration [58–60], agriculture [61–63], and customer service [64–66].

1: Rossum's Universal Robots

Robotics deals with the design, construction, operation, and application of robots. Although there is no single definition of what a robot is, it is generally considered a programmable machine that can automatically perform complex tasks [49, 67, 68]. Robots consist of three main parts: sensors, actuators, and control system [69]. Exteroceptive sensors observe the environment directly (e.g., infrared sensors), while proprioceptive sensors perceive the inner state of the robot related to body position and movement (e.g., rotary encoder), and interoceptive sensors monitor the robot's internal state (e.g., battery charge level). Actuators enable the manipulation of objects by and the locomotion of robots in the environment. Robot control can range from simple reactive behaviors over sophisticated control theory approaches to control based on artificial neural networks. Thus, robotics is a broad research field integrating various scientific and engineering disciplines. We focus on the two most relevant areas for this work, namely mobile robotics and swarm robotics.

"I can't define a robot, but I know one when I see one."
Joseph Engelberger

2.1.1 Mobile Robotics

A common approach to classify robots is based on their operation environment, which is directly connected to their locomotion capabilities, see Fig. 2.1 [67, 70]. *Stationary robots*, such as industrial robotic manipulators, are anchored in the ground and perform specific, repetitive tasks. Since these robots are used in well-defined environments, they can be controlled using open-loop control. This means that the robot's control algorithm uses predefined parameters and does not rely on feedback. Consequently, error correction is impossible. In contrast, *mobile robots*, such as robotic vacuum cleaners, move and execute tasks in uncertain and unstructured environments. These environments are not specifically designed for robots and change over time, for example, due to unpredictable entities, such as humans and animals. Consequently, mobile robots do not know in advance precisely all situations they might encounter. To act autonomously and to be able to react to changes in their uncertain operation environments, mobile robots usually use closed-loop control. This means that robots take actions that minimize the error between desired and actual system states.

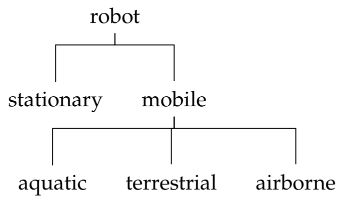


Figure 2.1: Robot classification based on operation environment. Adapted from [67].

Mobile robots can be used in *aquatic*, *terrestrial* or *aerial* environments that require different motion strategies. Aquatic robots include unmanned surface vehicles (i.e., drone ships) that swim on water [71, 72] and underwater robots [51] that swim [73, 74] or walk [75, 76]. Terrestrial robots are either wheeled [77, 78], tracked [79, 80] or legged [81–83] or combine these as hybrids [84, 85]. Airborne robots are autonomous aircraft that are either heavier-than-air, such as drones [86], or lighter-than-air, such as airships [87]. Not all robots can be strictly classified based on their operation environment, since they may fulfill criteria of more than one category. For example, amphibious robots can move on land and in water [88, 89]. In principle, our approach for the automatic generation (see Sec. 2.3) of robot behaviors is not limited to certain operation environments or robot hardware, but we have solely used wheeled indoor robots in our experiments so far. The extension to further platforms and operation environments remains subject of future work.

Although robots that follow a set of pre-programmed motions may appear autonomous, true autonomous robots are capable of making decisions based on their environment [69]. In our work, we aim for fully autonomous robots that are able to perform tasks completely on their own. Nevertheless, many mobile robot systems rely on a human operator: remotely controlled robot systems are fully operated by a human (e.g., robots for repair and recovery in zones of high-level radiation [90]), while semi-autonomous robots can perform subtasks autonomously (e.g., Mars rovers, such as NASA's Perseverance [91]) [67]. The control of fully autonomous robots usually relies on the three primitives *sense*, *plan*, and *act*. The control can then be realized in many ways, but there are essentially three approaches [92]:

Deliberative control. In deliberative control, sense, plan, and act are executed sequentially (see Fig. 2.2). First, robots sense their environment and fuse the sensory data into a world model, that is, a symbolic representation of the environment. The world model is then used by a planner to reason about next actions that will finally be executed by the robot's actuators. While deliberative control allows for finding the best course of action, it is slow and requires an accurate, up-to-date world model. For example, robot navigation can be realized using deliberative control by building a map using *Simultaneous Localization and Mapping* (SLAM) [93], localizing the robot with Markov localization [94], and planning a path using the A* algorithm [95].

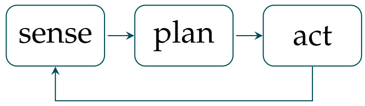


Figure 2.2: Deliberative control: sense, plan, and act are executed sequentially.

Reactive control. Inspired by the reactive behavior of insects [9], reactive control relies on tight sensor-actuator coupling. Simple condition-action rules with no state map sensor inputs directly to actions (see Fig. 2.3). While this allows for real-time responses, reactive control is limited to rather simple tasks. Navigation can be realized by reactive control using an artificial potential field approach [92]. Related to reactive control are behavior-based robotics [2] and Brook's subsumption architecture [96]. Here, several behaviors run concurrently and an action selection mechanism determines which actuator outputs of the active behaviors will be executed. Behavior-based robotics frequently relies on purely reactive behaviors, but it is not limited to those and thus more complex tasks can be accomplished [97].

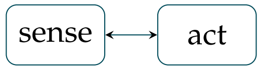


Figure 2.3: Reactive control: sense and act are tightly coupled.

Hybrid control. Hybrid control combines the fast response time of reactive control with the efficiency of deliberative control (see Fig. 2.4). A deliberative layer generates a plan for task accomplishment and a reactive layer executes the behaviors. Consequently, a coordination layer to resolve conflicts between these two layers is required, which is one of the major challenges of hybrid systems [51]. An example for a hybrid approach to navigation is to combine the deliberative A* algorithm to find the shortest path with the reactive potential field approach to avoid obstacles [98].

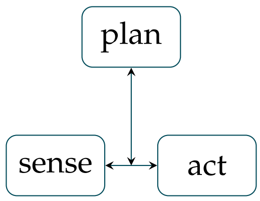


Figure 2.4: Hybrid control: plan, then sense - act.

The realization of all three control approaches can range from fully hand-coded over partially to completely learned using methods of machine learning (see Sec. 2.3). In our work, we use reactive and behavior-based control approaches that are learned using methods of evolutionary computation.

2.1.2 Swarm Robotics

Groups of social animals, and especially of social insects, that consist of relatively simple individuals are able to accomplish collectively tasks of various complexity, that is, they show swarm intelligence [9, 99]. Swarm robotics [6, 100], which was named one of the ten grand challenges of Science Robotics [101], follows this inspiration. It studies how desired collective behaviors emerge from local interactions among many simple, inexpensive, relatively homogeneous[2] robots and between those robots and the environment [102]. The swarms are asynchronous and decentrally controlled [7, 103], usually relying on simple control approaches, such as reactive or behavior-based control [6]. Furthermore, swarm members have only local sensing and communication capabilities [7, 104]. This lack of central or hierarchical coordination and central communication differentiates robot swarms from traditional multi-robot systems [105]. The characteristics of robot swarms lead to three main advantages over monolithic robot systems [7, 106] that can also be found in social animals [107]:

2: Relatively homogeneous swarms consist of robots that are all identical or belong to only few different topologies [99].

Robustness allows the successful execution of the task even when individual swarm members fail.
Scalability means that the collective system is able to solve the task independent from group size.
Flexibility allows the collective system to handle different environments and tasks as well as to adapt to changes.

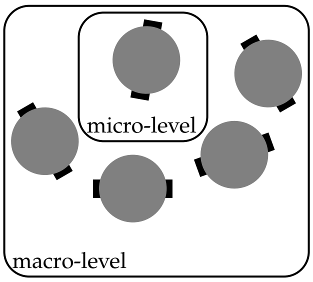


Figure 2.5: Micro-level and macro-level in a robot swarm. Robot controllers are implemented on the micro-level taking into account the individual robot's local interactions with its neighbors and the environment. Collective behaviors are exhibited on the macro-level.

But the **manual design** of such swarm robot systems is challenging. The global, collective behavior of the swarm on the macro-level emerges from the individual behavior on the micro-level, that is, from local interactions between individual robots and between robots and the environment [7]. Thereby, systems rely on self-organization (i.e., the dynamic and adaptive acquisition and maintenance of structure without external control) and emergence (i.e., the exhibition of novel behaviors on the macro-level w.r.t. the individual behaviors) [108]. Thus, robot controllers have to be implemented on the micro-level while considering hard to anticipate interactions and feedback processes to realize the desired collective behavior on the macro-level [8, 9]. Various approaches exist to manually design swarm systems and to derive the **micro-macro link** [109–111]. Alternatively to the manual design, swarm robot controllers can be automatically generated using methods of artificial intelligence. We use such methods in our work here and thus discuss them in more detail in Sec. 2.3.

Swarm robotics research makes use of simulated and real robot swarms. Several robot platforms were built specifically for swarm robot applications including the Kilobot [112] (see Fig. 2.6), the s-bot [113, 114] and the Colias robot [115]. Additionally, simple and inexpensive educational robot platforms

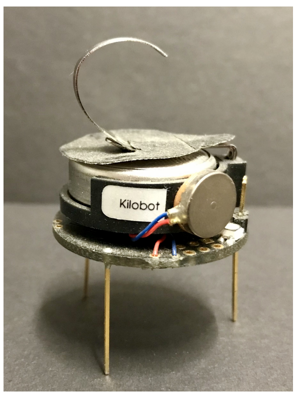


Figure 2.6: Kilobot

are frequently used, for example, the Thymio II [77] and the e-puck [78]. Accordingly, there are simulators that were developed for swarm robotics research, such as BeeGround [45] and ARGoS [116], as well as other suitable simulators that are not specifically intended for swarms, such as Webots [46]. In our work, we use own simple simulations, Webots, and BeeGround as well as real Thymio II robots.

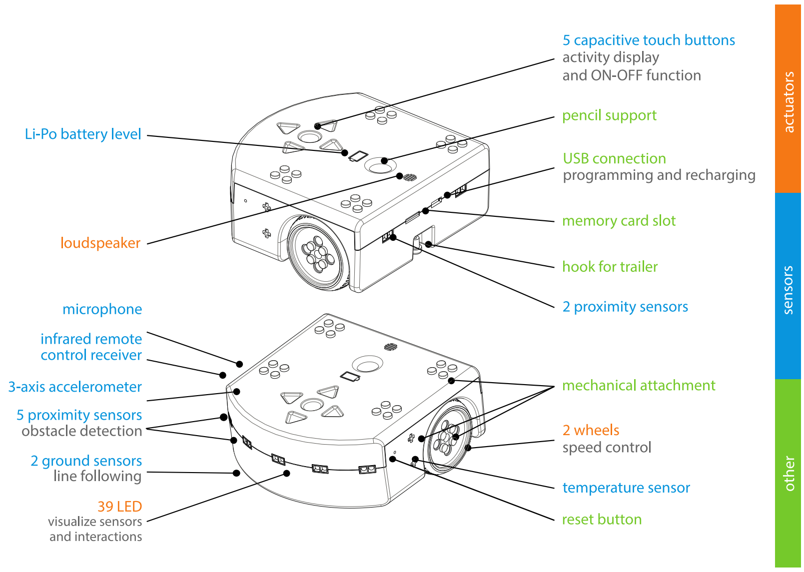


Figure 2.7: Illustration of the Thymio II robot with all its sensors, actuators, and other relevant parts. ©Thymio® (`http://wiki.thymio.org/en:thymiospecifications`) licensed under CC BY-SA 3.0 (`https://creativecommons.org/licenses/by-sa/3.0/`) / modified.

The Thymio II [77] (see Fig. 2.7) is a small, inexpensive mobile robot of 11 cm × 11.2 cm × 5.3 cm. It has a differential drive (i.e., two separately driven wheels), 39 LEDs, and one loud-speaker as actuators. The robot is equipped with several sensors: five capacitive touch buttons, a three-axis accelerometer, a thermometer, a microphone, an infrared (IR) receiver, and nine IR proximity sensors. Seven of the latter sensors are positioned in the front and back of the robot for obstacle detection, and two are underneath the robot that can be used, for example, for line following. The robot makes use of the Aseba framework [117], that is, a modular architecture for event-based control and can be programmed using the Aseba programming language, Blockly, Scratch, and Visual Programming Language (VPL). We use an extended Thymio II robot in our experiments by connecting a Raspberry Pi 3B (RPi) via the robot's USB port that is powered by an external battery (see Fig. 2.8). The command-line utility *asebamedulla* then provides access to the robot's Aseba network through D-Bus and thus the robot can be programmed with third-party languages, such as Python [118]. The external RPi furthermore provides Wi-Fi and allows to add more sensors to the robot.

Figure 2.8: Thymio II with attached Raspberry Pi and external battery.

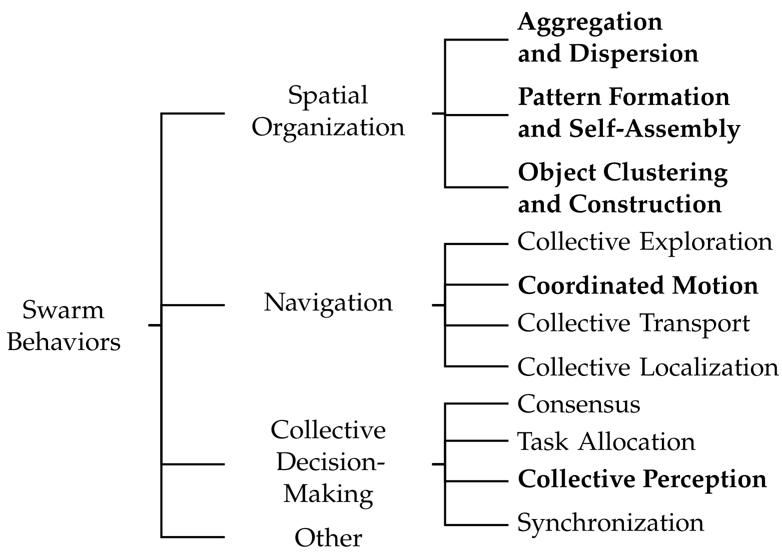


Figure 2.9: Taxonomy of swarm behaviors based on Brambilla et al. [7] and Schranz et al. [119] including a non-exhaustive list of sample behaviors. Behaviors printed in bold are explained in more detail in Sec. 2.2.

2.2 Swarm Behaviors in Nature and Robotics

The main inspiration for collective robot behaviors are collective behaviors of groups of social animals including, among others, ants, bees (see Fig. 2.10), birds, and fish. In this section, we present an overview of swarm behaviors found in nature and exemplary swarm robotics approaches to them.

Figure 2.10: Bees (*Apis mellifera*) on a honeycomb. Printed with permission from ©Richard Kaiser.

Fig. 2.9 visualizes a taxonomy of swarm robotic behaviors based on Brambilla et al. [7] and Schranz et al. [119]. It includes four categories of collective behaviors:

Spatial organization behaviors let robots spatially organize themselves or objects.

Navigation behaviors allow the coordinated movement of the robot swarm.

Collective decision-making behaviors enable robot swarms to make choices.

Other behaviors do not fit in the categories above. Examples are human-swarm interaction and group size regulation.

In this work, we aim for behaviors of the first three categories. Thus, we explain behaviors of those categories and their biological inspiration in more detail in the following.

2.2.1 Spatial Organization Behaviors

Spatial organization behaviors let robots move in the environment to spatially organize themselves or objects [119]. In our work on minimize surprise, we observe several of those behaviors including aggregation and dispersion (see Ch. 10), pattern formation and self-assembly (see Ch. 4), as well as object clustering and construction (see Chs. 9 and 11).

Aggregation

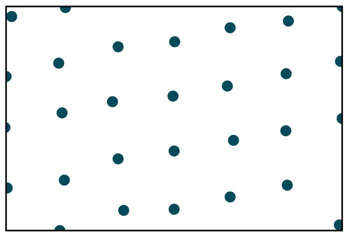
Aggregation is the simplest of the spatial organization behaviors. All swarm members group in a specific region of their environment, that is, they position themselves spatially close to each other and thus enable interactions with each other [7]. This behavior is frequently observed in natural swarms as it allows, among other things, for easier thermoregulation and lower predation risks [120]. Aggregations are formed, for example, in bacteria [121], bees [122] (see Fig. 2.11), cockroaches [123], butterflies [124], mussels [125], and penguins [126].

Figure 2.11: Aggregation of bees (*Apis mellifera*). Printed with permission from ©Richard Kaiser.

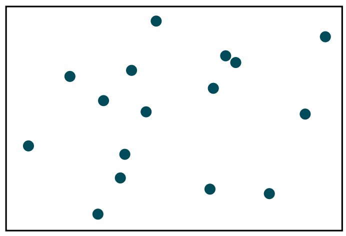
There exist several approaches to aggregation in swarm robotics [127]. A prominent example is the BEECLUST [128, 129] algorithm that is inspired by the grouping of young honeybees at a spot with their preferred temperature on the honeycomb. The behavior can be reproduced with robots using a simple finite state machine. Robots move straightforward until they detect an obstacle or a robot. While they turn away from detected obstacles, they wait for a temperature-dependent time and afterwards turn by 180° when detecting other robots. As robots stop longer in warmer areas, the behavior leads to their aggregation there. Other examples for aggregation in swarm robotics include the replication of the aggregation behavior of cockroaches using probabilistic approaches [130, 131], the combination of four simple behaviors in a probabilistic finite state machine to create a generic aggregation behavior [132], the evolution of aggregation behaviors using artificial neural networks as robot controllers [133, 134], and approaches based on artificial physics [135].

Dispersion

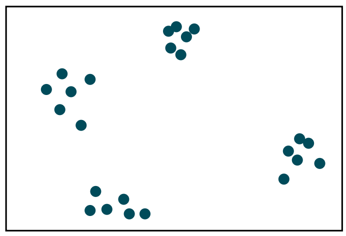
As the opposite behavior to aggregation [6], dispersion lets swarm members distribute over the environment while staying within communication range. While dispersion is used for area coverage and monitoring in swarm robotics, it is only rarely seen in natural swarms [136]. However, dispersion can be found in population distributions in ecology as animals disperse due to territoriality, competition or resource distribution [137]. There are three different dispersion types (see Fig. 2.12): uniform (i.e., individuals are roughly evenly spaced), random (i.e., there is no predictable pattern), and clumped (i.e., individuals group in several areas).



(a) uniform dispersion



(b) random dispersion



(c) clumped dispersion

Figure 2.12: The three types of dispersion of individuals in ecology.

Swarm robotic approaches to dispersion have been studied in simulation and on real robots. Matarić [138] implements dispersion on real robots by making robots move away from their local area of highest robot density. Ugur et al. [139] use wireless signal intensities to estimate distances between robots and let robots move away from or towards other robots depending on a threshold value. Other approaches are based on potential fields [140] or 'virtual pheromones' (i.e., infrared messages allowing to infer the distance to the sender) [141], or evolve artificial neural networks as controllers for dispersion [142, 143].

Pattern Formation

Pattern formation can be found frequently in nature. Examples are animal coat markings, such as the stripes on the coat of zebras (see Fig. 2.13), phyllotaxis, that is, the formation of leaf patterns, the formation of waves [144], and the formation of patterns in crystal growth [145]. Furthermore, spatio-temperal patterns [146] can form, such as the organization of ants or pedestrians into lanes in areas of bidirectional flow [147]. Most of these pattern formation processes can be explained by reaction-diffusion mechanisms and activator-inhibitor schemes [148].

Figure 2.13: Zebra. Image by George Brits released under CC0 / cropped.

In swarm robotics, pattern formation behaviors let robots arrange themselves in desired patterns or shapes [119]. For example, Spears et al. [149] use virtual repulsive and attractive forces to let robot swarms form hexagonal and square lattices. In the approach by Meng et al. [150], robots form user-predefined target patterns, for example, circles or U-curves, using gene regulatory networks. Inspired by the formation of trail networks by foraging ants [151], Nouyan et al. [152] propose a behavior-based architecture based on Arkin's motor schema paradigm [2] and Sperati et al. [153] evolve artificial neural networks as robot controllers for chain formation for swarm robot navigation in unknown environments. Furthermore, Hamann et al. [154] make use of the fact that ants can determine the direction to the nest and food source based on the geometry of trail bifurcations [3] and implement it as a simple reactive swarm robot behavior to allow for the orientation in trail networks.

Self-Assembly

Self-assembly is a process that lets system components organize themselves autonomously into patterns or structures without human intervention [155]. Self-assemblages can be found on different scales ranging from molecules to colonies. Swarm robotics is mainly inspired by those of ants, bees, and wasps, such as ant bridges (see Fig. 2.14) and bee curtains [156]. While Brambilla et al. [7] and Hamann [6] stress that robots have to be physically connected in self-assembly, Schranz et al. [119] include as well virtually (i.e., through communication links) connected robots. Accordingly, the line between pattern formation and self-assembly approaches in the swarm robotics literature is blurred.

Figure 2.14: Ant bridge. ©Igor Chuxlancev (`https://commons.wikimedia.org/wiki/File:AntBridge_Crossing_10.jpg`), licensed under CC BY 4.0 (`https://creativecommons.org/licenses/by/4.0`) / cropped.

One of the most prominent works in that domain is by Rubenstein et al. [157] who implement self-assembly on a swarm of thousand Kilobots using a finite state machine. The robots form user-defined shapes (e.g., a star) by positioning themselves next to each other (i.e., emulated self-assembly). Divband Soorati et al. [158] present a hand-crafted, plant-inspired algorithm for adaptive emulated self-assembly of tree structures also using Kilobots. Similar to these two approaches, we emulate self-assembly in our work as well (see Ch. 4). By contrast, the s-bot is a robot platform that can connect to other s-bots using a gripper and was developed especially for self-assembly [113, 114]. S-bots can self-assemble to accomplish a variety of tasks including forming chains to pass gaps or steps or to go through narrow passages [159], to cross rough terrains with hills [160], and to pull heavy objects [161]. The FireAnt3D robot system by Swissler and Rubenstein [162] provides a new docking mechanism that allows for more flexible 3D self-assembly. The system is used for the self-assembly of four amorphous and environment-adaptive structures, namely towers, chains, cantilevers, and bridges [163]. The next level of self-assembly are reconfigurable, modular robots that self-assemble and self-disassemble to change their morphology [164, 165].

Excursus: Cellular Automata

Related to pattern formation and self-assembly is the research on cellular automata. Cellular automata are mathematical models for dynamical systems that are discrete in time and space (i.e., grid-based) [166] as are our self-assembly experiments with minimize surprise (see Ch. 4). Grid cell states are updated using deterministic update rules that are based on the state of neighboring cells. Those update rules can be manually designed or automatically generated using methods of artificial intelligence.

The most popular cellular automaton is Conway's Game of Life [167, 168]. In this rule-based approach, cells change their state between dead and live based on their Moore neighborhood. Patterns, that can be stable or oscillating, then form depending on the initial configuration (see Fig. 2.15).

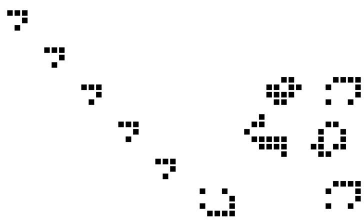


Figure 2.15: The *space rake* in Conway's Game of Life. Image by David Eppstein released to public domain.

Cellular automata are used, among others, in artificial life studies, for example, studying speciation [169] and open-ended evolution [170]. Most relevant for our work here are approaches aiming for the reproduction of target patterns. Elmenreich and Fehérvári [171] evolve time-discrete, recurrent artificial neural networks as update rules for cellular automata to reproduce target images, for example, flags. Hoffmann [172] evolves finite state machines as update rules in a cellular automata agent setup, that is, agents can move over and change grid cells for the formation of path [172], line [173], domino [174], and checkerboard patterns [175]. Öztürkeri and Johnson [176] evolve update rules for developmental cellular models, that is, cellular automata where the update rules influence also neighboring cells. They aim for the self-assembly of target patterns, such as squares, diamonds, and flags. In all those approaches, high resemblance of formed and targeted pattern is rewarded and repetitive patterns are successfully created. This resembles our work on engineering self-organization by predefining predictions in our minimize surprise approach (see Ch. 6).

Object Clustering

In object clustering, robots move objects that are distributed in the environment to aggregate or sort them. In nature, for example, the ant species *Leptothorax unifasciatus* clusters their brood sorted by brood stages [177] and several ant species pile the corpses of dead individuals in 'ant cemetery' clusters [178].

In swarm robotics, there are several approaches to clustering and sorting of objects in simulation and on real robots. Resnick [179] presents a termite-inspired approach to cluster wood chips in simulation. The simulated termites follow two simple rules when finding a wood chip: if they do not carry one yet, they will pick it up; if they carry a wood chip already, they will put it down. Beckers et al. [180] use a behavior-based approach replicating the corpse-gathering behavior of ants. A single cluster is formed by combining moving straightforward, obstacle avoidance and puck dropping behaviors. Scheidler et al. [181] use a probabilistic approach and pick up and drop objects with certain probabilities. A rule-based approach by Holland and Melhuish [182] allows for annular sorting (i.e., objects are sorted in concentric rings) of two different object types using real robots. Wilson et al. [183] extend the approach to work with more object types.

Several other approaches to clustering and sorting rely on artificial neural network controllers that are either evolved [184–186] or trained using supervised learning [187]. We evolve object clustering behaviors with our minimize surprise approach in Ch. 9.

Collective Construction

Animals have proven to be accomplished builders and cooperatively built structures abound in nature. For example, the mounds built by termites of the genus *Macrotermes* can be up to seven meters high, that is, 600 times the size of an individual termite [188]. Further examples include honeycombs [189], ant nests [4], beaver dams [190], and nests of birds [191]. Collective robotic construction became an active research area as the need for safe, inexpensive, and sustainable construction increased [192]. Thereby, not only the coordination of robots but also used material and mechanisms as well as targeted structures have to be taken into account.

A simple, ant-inspired approach is presented by Parker et al. [193]. Ants of the species *Leptothorax tuberointerruptus* build their nests in flat crevices in rock that provide floor and roof. Thus, the ants have to construct solely planar perimeter walls [4]. Exhibiting the so-called 'blind bulldozing' behavior, ants push debris (e.g., stones, sand) outwards from the future nest's center until they detect resistance that exceeds a threshold. Then, they randomly reorientate and continue pushing material outwards, which leads to the self-organized clearance of the nest site and the construction of circular perimeter walls over time. Parker et al. [193] implement this behavior as a three state finite state machine and execute it on a swarm of four real robot bulldozers. Fig. 2.16 shows our reproduction of blind bulldozing with three Thymio II robots that were extended with bulldozer blades [194].

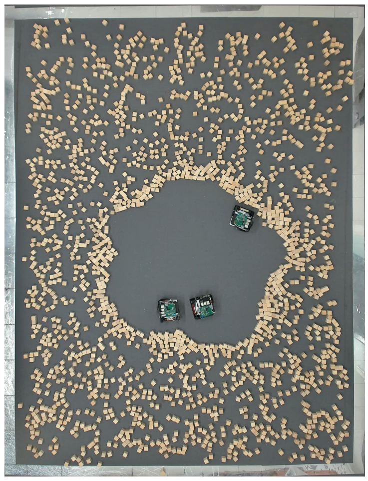


Figure 2.16: Nest constructed by a swarm of three Thymio II robots executing the ant-inspired blind bulldozing algorithm [194].

Werfel et al. [195] present a termite-inspired approach including a dedicated robot platform that can carry and climb bricks. Their system generates low-level rules for the robot swarm to build a user-defined desired structure. The robots rely on local sensing only and act reactively in the construction process.

Other approaches use continuous [196] or amorphous [197] building material, build structures using drones [198], evolve [199] or train controllers for swarm construction with reinforcement learning [200], or encode targeted structures in scalar fields [201, 202] or distance transforms [203]. An extensive overview of the field can be found in the review by Petersen et al. [192]. We present the evolution of construction behaviors with our minimize surprise approach in Ch. 9.

2.2.2 Navigation Behaviors: Coordinated Motion

Many groups of animals exhibit navigation behaviors, such as the coordinated motion[3] of flocks of birds [5], schools of fish [204], swarms of insects [205] or herds of mammals [206]. In 1987, Reynolds [207] presented three simple rules to simulate a bird flock:

Figure 2.17: A flock of birds. Image by Christoffer A. Rasmussen released to public domain.

Cohesion. Agents steer towards the average position of their neighbors to keep the group together.

Separation. Agents steer away from too close neighbors to avoid collisions.

Alignment. Agents steer towards the average heading of their neighbors to match direction of the group.

3: Coordinated motion is also known as flocking [7].

In his model, neighboring robots are determined metrically while real birds probably use a topological approach [208]. Reynolds [207] discusses also that his model is probably closer to a school of fish in murky water or to herds of land animals because his agents have, in contrast to birds, only short-range perception of the environment. Reynolds' approach was implemented on real robots, for example, on wheeled ground robots [209] and on small fixed-wing flying robots [210].

A drawback of Reynolds' approach is that agents require knowledge of their heading and communication to realize the alignment rule. Moeslinger et al. [211, 212] present a minimalist approach to flocking for ground robots with local sensing only. Here, alignment emerges by discretizing the robots' sensor fields into four sectors that lead to either attraction or repulsion. Since flocking is widely studied, there are many other approaches that, for example, rely on potential functions [213, 214] or evolve artificial neural network controllers for coordinated motion [215, 216].

2.2.3 Collective Decision-Making Behaviors: Collective Perception

Collective decision-making is extensively studied in swarm robotics, since it makes the swarm autonomous on the macro-level (i.e., the global or swarm level) by enabling the swarm to get a global view and to make informed collective decisions by combining locally sensed information [6, 119]. Decision-making behaviors include, for example, synchronization, task allocation, and group size regulation [119]. Inspiration for such behaviors can be found in social insects. For example, ants use pheromone trails to collectively find shortest paths to food sources [217, 218] and bees asses the need for comb construction by evaluating search times for empty combs [219]. In this thesis, we evolve decision-making mechanisms for collective perception using our task-independent minimize surprise approach (see Ch. 8).

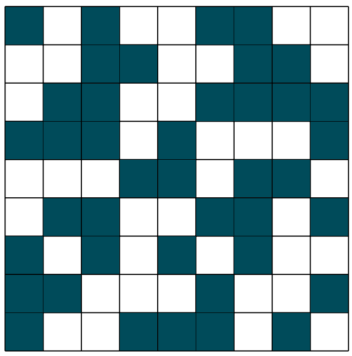


Figure 2.18: Sample arena of the collective perception task as defined by Valentini et al. [220].

Valentini et al. [220] present a collective perception scenario where robots have to collectively decide which of two features is more frequent, that is, if there are more black or white cells on the arena ground (see Fig. 2.18). Consequently, collective perception is a best-of-2 problem [221]. The authors show that the Direct Modulation of Majority-Based Decisions [222] and the Direct Modulation of Voter-Based Decisions [223] strategies perform well in this scenario. Both decision-making strategies are implemented by a state machine that switches between *exploration* and *dissemination*. During exploration, robots sample the environment locally and estimate the quality of their current opinion (i.e., more black or more white). They broadcast their opinion locally in the dissemination state first, collect the opinions of their neighbors afterwards, and finally apply a decision-making mechanism, that is, the majority rule or the voter model, respectively. Concurrently to the decision-making strategy, robots execute random walk and obstacle avoidance. Several new approaches and extensions to the scenario were proposed, including the study of the scenario with multiple features [224, 225], the use as a benchmark task in a blockchain-based approach to handle

Byzantine robots in swarms [226], speeding up the collective decision-making process through isomorphic changes in the environment [227], making decisions in a sparse swarm using a Bayesian algorithm [228], and to realize an Ising-based approach that takes learned preferences of agents into account [229]. Furthermore, Bartashevich and Mostaghim [230] propose nine different visual patterns for benchmarking collective decision-making behaviors in the collective perception scenario, since problem difficulty not only depends on the proportion of features but also on their distribution in the environment. Almansoori et al. [231] evolve collective decision-making behaviors for the collective perception scenario using a task-specific fitness function. The robot swarm constantly executes random walk and obstacle avoidance, as in the approach by Valentini et al. [220], but the decision-making behavior is realized by a neural network that is evolved by rewarding high mean amounts of swarm members with the correct opinion. By contrast, we use the decision-making state machine by Valentini et al. [220] and evolve only the applied decision-making mechanism with our minimize surprise approach in our experiments in Ch. 8. Furthermore, Morlino et al. [232] use a similar scenario to evolve controllers to estimate the density of black spots on gray floor.

Other approaches to collective perception aim, for example, for the recruitment of the adequate number of robots based on the size of the target sites [233] or the classification of objects [234, 235].

2.3 Artificial Intelligence

Turing [236] posed the question 'Can machines think?' already in 1950, but it is the 1956 *Dartmouth Summer Research Project on Artificial Intelligence* that is generally considered as the founding event of artificial intelligence as a research field. Russell and Norvig [95] define artificial intelligence as the study of intelligent agents, that is, agents that perceive and act upon their environment (see Fig. 2.19). According to Pfeifer and Bongard [237], artificial intelligence research has three main goals: (i) understanding intelligence in biological systems, (ii) abstracting general principles of intelligent behavior, and (iii) creating useful artifacts by applying those principles. Initially, researchers focused on the software side to create intelligent artifacts and considered the hardware side rather irrelevant. But due to a paradigm shift in the 1980s, embodiment (i.e., the use of physical agents that act in the real world) is nowadays an essential part of AI research [96, 237]. Overall, artificial intelligence is an interdisciplinary field that draws upon computer science, mathematics, psychology, cognitive science, and many more. It studies problems such

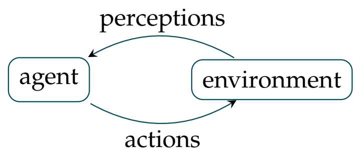


Figure 2.19: Agent-environment interaction.

"I think, therefore I am."
René Descartes

as natural language processing [238], knowledge representation [239], and computer vision [240]. Of particular relevance to our research are the application of machine learning and evolutionary computation to (swarm) robotics, which we will discuss in more detail in the following.

2.3.1 Machine Learning

Machine learning studies algorithms that improve their performance in a task automatically through experience and by learning from data [241]. This allows to develop solutions for a wide variety of tasks that are otherwise difficult or impossible to solve by conventionally developed algorithms [242]. There are three main machine learning categories [243]:

Supervised learning learns rules to map input values to outputs using labeled data, that is, given inputs and corresponding desired outputs. It is used for classification and regression problems.

Unsupervised learning learns to determine structure in unlabeled data including clustering (i.e., finding groups of data), density estimation (i.e., finding the data distribution in the input space), and dimensionality reduction [244].

Reinforcement learning learns a policy that maps agent states to actions by trial and error, that is, by receiving rewards based on an agent's actions in the environment [245].

The lines between supervised and unsupervised learning are blurred and hybrid approaches lie between the two [242]. For example, semi-supervised learning uses few labeled data points and a large number of unlabeled data [246] and self-supervised learning solves an unsupervised learning problem in a supervised way by automatically labeling the input data [247].

Artificial neural networks (ANN) are frequently used models in machine learning. They are inspired by human brains and are essentially weighted directed graphs consisting of layers of artificial neurons (i.e., nodes). Over time, a huge variety of artificial neural network architectures were developed [248]. In our work, we rely on feedforward neural networks where inputs are propagated straight through and recurrent neural networks that take time and sequence into account by using information from the previous pass as additional input. Deep learning, a subfield of machine learning, uses artificial neural networks with many layers [242].

Robot learning uses machine learning methods to accomplish, among other things, perception, control, and navigation [249]. Learning, and in particular reinforcement learning, is also applied to the multi-robot and swarm robotics domain. There are three different settings for multi-agent learning: cooperative, competitive, and mixed cooperative-competitive [250, 251]. In cooperative settings, all agents pursue a collective goal (e.g., avoiding collisions in autonomous driving). By contrast, agents compete with each other to accomplish a goal in competitive settings (e.g., chess). Mixed cooperative-competitive settings combine the two, that is, teams of cooperating agents compete with each other (e.g., team sports, such as basketball). In all three settings, agents can be homogeneous or heterogeneous in their features and behaviors [251, 252]. The training of multi-agent systems can be centralized (i.e., agents learn based on mutual information) or distributed (i.e., agents learn independently of other agents) and execution centralized (i.e., agents share a joint controller) or decentralized (i.e., each agent has its own controller). Compared to single-robot learning, multi-agent learning has to deal with several additional challenges [250–252]. The main problem to address in multi-agent learning is that the environment is non-stationary by default, since other agents are present, and agents have to co-adapt in this constantly changing environment [250]. In addition, the search space grows with the number of agents and thus algorithms for multi-agent learning need to be scalable regarding speed of convergence and solution quality. A high number of agents also causes partial observability, that is, agents have only limited information about the state of their environment, which may lead to suboptimal solutions. Moreover, credit assignment in cooperative settings is complicated, since performance is evaluated at the multi-agent level and the contribution of an individual agent to the overall goal is difficult to assess. Despite these challenges, machine learning, and in particular reinforcement learning [253], has been applied successfully to several multi-agent and swarm scenarios. For example, reinforcement learning and deep reinforcement learning approaches were used for collaborative exploration in deep-space [254], traveling between landmarks [255], and rendezvous (i.e., meeting at the same location) and pursuit evasion (i.e., capturing one or more evaders collectively) [256].

In our minimize surprise approach, we rely on evolutionary computation, that we discuss in the next section in more detail. Nevertheless, we present a machine learning approach to minimize surprise in App. A.

2.3.2 Evolutionary Computation

Evolutionary computation [16] is the umbrella term for algorithms that are inspired by Darwinian evolution [257] and genetics, such as evolutionary programming [258], genetic algorithms [259], evolution strategies [260, 261], and genetic programming [262]. Most evolutionary algorithms are gradient-free, that is, optimization is done in a stochastic trial-and-error style. Both evolutionary computation and reinforcement learning are reward-based, that is, they aim to maximize the fitness or reward of agents in unknown environments, and thus address the same class of problems [252, 263, 264]. Although training agents through supervised learning is theoretically possible in these scenarios, it may be difficult or even impossible to generate labeled training data for environments or tasks where rewards are sparse. Reinforcement learning and evolutionary computation are more suitable in such scenarios as they usually rely on trial and error and do not require knowledge about correct and incorrect decisions [265]. Evolutionary methods have proven to be competitive to reinforcement learning on modern agent-environment benchmarks while offering advantages in code complexity, scaling to large-scale distributed settings, and handling of sparse rewards [17, 266]. Due to these advantages, we use methods of evolutionary computation in our minimize surprise approach.

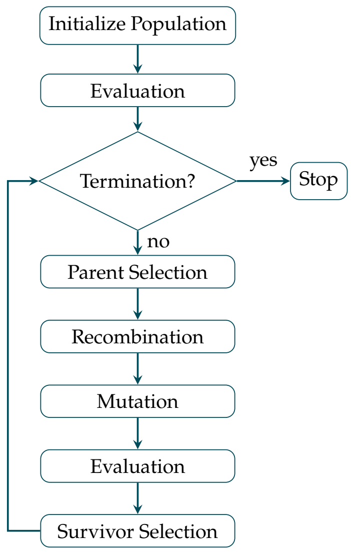


Figure 2.20: Flowchart of the general scheme of an evolutionary algorithm.

Figure 2.20 illustrates the general scheme of an evolutionary algorithm [16]. First, an initial population $\mathcal{P}(0)$ of μ individuals is randomly initialized and evaluated. Each member or individual i of the population is represented by a genome $g_i, i \in [0..\mu-1]$, that is, a candidate solution in the genotype space $\mathcal{C}_G$, on which evolutionary operators for variation (i.e., recombination and mutation) act. Each genome (also chromosome or genotype) encodes a phenotype, that is, a candidate solution in the original problem context, on which evolutionary operators for selection act. For evaluation, individuals are mapped from the genotype space $\mathcal{C}_G$ to the phenotype space $\mathcal{C}_P$ using a mapping function $J : \mathcal{C}_G \to \mathcal{C}_P$. For example, binary strings can be used to encode integers (e.g., 0101 maps to 5) or, as used in our scenario, strings of real values can be used to encode the weights of an artificial neural network. The phenotypes are then evaluated using a fitness function $F : \mathcal{C}_P \to \mathbb{R}$. An iterative process runs until a termination criterion evaluates to true, for example, after a fixed number of generations $g_{\max}$ or when an individual with a minimum fitness $F_{\min}$ is found. In the iterative process, first individuals of the population $\mathcal{P}(g)$ of the current generation g (i.e., iteration) are selected as parents to create offspring. Parent selection methods include, for example, fitness proportionate selection[4] and tournament selection. In fitness proportionate selection, the selection probability of each individual i is $p^i_{\text{FPS}} = F_i / \sum_{j=0}^{\mu-1} F_j$, that is, higher fitness leads to higher selection probability. By contrast, k individuals are picked randomly, compared, and the best one is selected in tournament selection. Then, a set of $\lambda \in \mathbb{N}$ new individuals (i.e., offspring) is created by applying variation operators, that is, recombination and mutation, to the genomes. Recombination combines with a probability p_{rec} two or more parents to create new offspring. For example, one-point crossover recombines two parent individuals by

4: Fitness proportionate selection is also known as roulette wheel selection.

splitting both parents at the same point and creating two children by exchanging the back parts of the genomes (see Fig. 2.21). Mutation alters each gene (i.e., value υ) of a genome with a probability p_{mut}, for example, by flipping the value in a binary encoded genotype or by adding a random number to a value in a genotype consisting of floating point values, that is,

$$\upsilon_{\mathrm{mut}} = \begin{cases} \upsilon + \mathcal{Y}, & \text{if } \mathcal{X} < p_{\mathrm{mut}} \\ \upsilon, & \text{otherwise} \end{cases}, \tag{2.1}$$

with random number $\mathcal{X} \in [0, 1]$ and random number $\mathcal{Y}$ in a predefined interval. Afterwards, the offspring is evaluated. In survivor selection,[5] the next population $\mathcal{P}(g+1)$ of μ individuals is chosen from the μ individuals in the current population (i.e., parents) and the λ offspring based on age or fitness. Age-based survivor selection keeps every individual for a fixed number of generations in the population before replacing it. We follow this strategy in the simple evolutionary algorithm used in Chs. 4-9 and replace all but the best performing parent with offspring after each generation. Consequently, each individual, except the best, exists for only one generation in the evolutionary process. Keeping the current best individual in the population (i.e., elitism) ensures that high performing individuals are not lost. Fitness-based survivor selection replaces the least fit individuals after each generation. For example, in $(\mu + \lambda)$-selection, the μ parents and the λ offspring are ranked according to fitness and the μ best individuals form the next population $\mathcal{P}(g+1)$. We use this approach with $\mu = \lambda = 1$ in Chs. 10 and 11. In the iterative process, fitness will improve over generations. Nevertheless, the stochasticity of the evolutionary process can lead to genetic drift, that is, variety in the population is lost due to random chance.

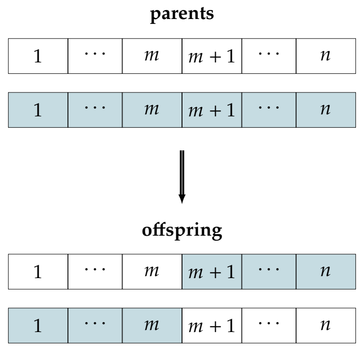


Figure 2.21: One-point crossover. The parent genomes are split after gene m and the back parts of the genomes switched to create offspring.

In the general scheme of an evolutionary algorithm introduced above, one population of individuals is evolved. In coevolution, this scheme is extended to evolve several populations in turns [16]. The fitness of an individual depends on interactions with other individuals in this case. These interactions can be cooperative (e.g., a ground robot and an airborne robot cooperate to collect items in the environment [267]) or competitive (e.g., predator-prey or pursuit-evasion scenarios [268]) as in multi-agent learning (see Sec. 2.3.1).

Neuroevolution evolves artificial neural networks [269]. The simplest case, that we also use in our work, is to evolve individuals that directly encode the weights of a fixed neural network topology. More sophisticated methods evolve both network topology and weights, for example, *NeuroEvolution of Augmenting Topologies* (NEAT) [270] or Analog Genetic Encoding [271].

5: Survivor selection is also called replacement.

Evolution and learning are not mutually exclusive. For example, Nolfi et al. [272, 273] combine learning during the lifetime of an individual with evolution across generations. In their sample setting, an individual simulated agent has to find food in a 2D grid world (evolutionary task) while predicting the sensor values of the next time step, that is, the next position of food (learning task). Although learned weight changes are not inherited (i.e., no Lamarckian inheritance), evolution and learning influence each other, since individuals with high fitness and individuals that reach high fitness because they learn to predict are found by evolution.

Evolutionary computation is model-free by default, but environment or agent models can be evolved or used during evolution. Ha and Schmidhuber [274] train single agents that are equipped with a variational autoencoder (VAE), a world model, and a controller in OpenAI Gym simulations [275]. The authors train VAE, world model, and controller separately, whereby self-supervised learning is used for the VAE and the world model and evolution strategies with a task-specific reward for the controller. Thus, different controllers can be trained using the same world model in their single agent scenario. In collective systems as used in our work, the world model depends not only on the (dynamic) environment but also on the behavior of the surrounding agents and thus controller and world model cannot be trained individually in that case. But Risi and Stanley [18] showed that all three agent components can be trained end-to-end using a genetic algorithm while being competitive to the more complex training regime by Ha and Schmidhuber [274].

Evolutionary (Swarm) Robotics

Evolutionary robotics [13–15] and evolutionary swarm robotics are the application of evolutionary algorithms to robotics and swarm robotics [8], respectively. They are common approaches to automatically generate controllers for monolithic robot systems and for robot swarms [10]. In our work, we use evolutionary swarm robotics, which has proven to be successful in the generation of controllers in simulation and on real robots for a variety of swarm robotics tasks including phototaxis [276], foraging [277, 278], aggregation and coordinated motion [279], cooperative transport [280], and self-assembly [281].

Although in principle various control architectures (e.g., behavior trees [277] or finite state machines [282]) can be evolved, artificial neural networks are typically used in evolutionary robotics. They constrain evolution minimally, that is, ANNs permit the evolution of controllers of unbounded complexity. Additionally, they allow for the use of raw sensor inputs and the generation of low-level commands for actuators [13]. Most of the time, including our approach, the weights of a fixed neural network topology are evolved as robot controllers. But it is also possible to evolve both network weights and topology [283], or even the robot morphology [284–287].

In most cases, the evolutionary search process is guided by a task-specific fitness function that rewards performing a given task or behavior. As evolutionary algorithms maximize fitness potentially by every possible way, unexpected, unwanted behaviors may emerge when the fitness function was not specified accurately enough. Thus, the specification of goal-directed fitness functions is difficult [19]. In addition, there is a tradeoff between rewarding high-level task completion (i.e., aggregate fitness functions) and rewarding specific behavioral features (i.e., behavioral fitness functions) [20]. Since the randomly generated solutions in the initial population will most probably not lead to the fulfillment of non-trivial tasks, rewarding high-level task completion can cause bootstrapping problems. That is, the fitness of the initial population is not detectable and thus no selective pressure to guide optimization is generated. By contrast, rewarding specific behavioral features of an assumed solution for a particular task restricts the search to certain solution types, potentially causing that more original solutions will not be found. In our work, we rely on a task-independent fitness function to drive the search process to allow for the emergence of diverse swarm behaviors. Task-independent fitness functions put selection pressure on aspects that are not directly related to a desired task or behavior (e.g., novelty [24]). While there is no guarantee that desired behaviors will be found, the evolutionary process has the freedom to find original solutions here. However, since task-independent approaches frequently lead to the emergence of diverse behaviors within or across evolutionary runs, users can choose from a set of behaviors that may include desired ones. We will discuss divergent search algorithms, quality-diversity algorithms, and intrinsic motivations as approaches for the generation of diverse behaviors in the next sections in more detail.

Evolutionary robotics allows for offline and online optimization of controllers [288]. Offline evolution separates design and operational phase, that is, controllers are first optimized and transferred to the real robot for task-execution afterwards. Thereby, optimization can be done completely in simulation, which allows minimizing costs by speeding up the search process and robot hardware will not wear out. But the reality gap [289] is a challenge. Since simulation and real world differ, evolution may exploit simulation-specific features and performance on the real robot may be (significantly) worse than in simulation. A potential solution is to conduct some or all evaluations on the real robot, but at the cost of loosing the advantages of optimization in simulation. Online evolution, in contrary, avoids the reality gap by evolving robot controllers on the real robots in the environment during the operational phase (i.e., task execution). This also allows robots to adapt to environmental or task-related changes during evolution. Online evolution can be realized in a centralized or distributed manner [16]. There are two ways to implement centralized online evolution: A master computer runs the evolutionary process for the full swarm (i.e., it collects the fitness from the individual robots, applies selection and variation operators, and distributes new controllers for evaluation to the robots), or each robot runs the evolutionary process encapsulated, that is, independent from other robots. In the first variant, all robots execute the same controller (i.e., homogeneous swarm regarding control) while in the second variant all robots have different controllers (i.e., heterogeneous swarm regarding control). There are also two ways to implement distributed online evolution[6] [237, 291, 292]: Either each robot has one genome and selection and reproduction is based on interaction between robots, or each robot runs an encapsulated evolutionary process but robots also exchange genomes. In distributed online evolution, the swarm is always heterogeneous regarding the executed controllers. In our work, we use a centralized online evolutionary approach that runs onboard the robots (see Ch. 10.1).

6: Distributed online evolution is also termed embodied evolution, and social learning is its generalization to all learning methods [290].

A variety of behaviors were successfully evolved using methods of evolutionary robotics with task-specific rewards both in simulation and on real robots. For example, Christensen and Dorigo [276] evolve phototaxis and hole avoidance in a group of connected robots by rewarding reaching the light source quickly and minimal traction between robots, and penalizing falling into holes. The behaviors are evolved in simulation and transferred to real robots afterwards. Heinerman et al. [293] evolve obstacle avoidance on real Thymio II robots by rewarding high translational speed, low rotational speed, and low proximity sensor values.

By contrast, Turing learning [294, 295] is a novel approach to infer the behavior of natural or artificial systems without requiring predefined metrics. Two populations are concurrently optimized using competitive coevolution: (i) a population of models of the behavior of the system under investigation and (ii) a population of classifiers. The classifiers are rewarded for discriminating model and real system correctly while models aim for tricking the classifier to categorize them as genuine. The approach was applied to swarm robotics to infer aggregation and clustering in swarms of mobile agents.

Divergent Search Algorithms and Quality-Diversity Algorithms

Divergent search algorithms push towards behavioral diversity instead of a task-specific objective in the evolutionary process [296]. They are inspired by the open-endedness in natural evolution, that is, constant morphological and behavioral innovation [13], which is one of the grand open challenges of artificial life [297]. Novelty search [24, 298], for example, aims for behavioral (i.e., phenotypic) diversity by driving the search process using the novelty of individuals with respect to current and past individuals. Thereby, each individual's behavior is characterized by a vector of domain-dependent behavioral features that are supposed to capture relevant aspects of an assumed task. An individual's novelty in behavior space is then calculated as the mean behavioral distance (e.g., the Euclidean distance) to its K nearest neighbors. Out of the vast literature on novelty search, the most relevant work here is that by Gomes et al. [134] who successfully evolve swarm robot controllers for aggregation and resource sharing with novelty search. We compare our minimize surprise approach and novelty search with respect to behavioral diversity in Sec. 5.3. Another example for divergent search algorithms is surprise search [299] that rewards the deviation from expected behaviors.

Purely divergent search generates behavioral diversity but does not guarantee solution quality. Quality-diversity (QD) algorithms [296] are an extension to pure divergent search algorithms and guide search towards collections of behaviors that are both maximally diverse and as high performing as possible. For example, the *multidimensional archive of phenotypic elites* (MAP-Elites) [44] approach searches for the highest performing solution with respect to a user-defined performance measure, that is usually related to a desired task, for each cell in a discretized n-dimensional behavior space that covers variations of interest to the user. Cazenille et al. [300], for example, use MAP-Elites to evolve self-assembly behaviors in a swarm of robots. In the standard MAP-Elites approach, the behavior space is discretized per dimension. Thus, it does not scale well to high-dimensional behavior spaces. This problem is addressed by CVT-MAP-Elites [301], which partitions the behavioral space into a desired number of regions using centroidal Voronoi tessellation (CVT). Recent extensions of MAP-Elites combine the standard approach with CMA-ES [302] or CVT-MAP-Elites with differential evolution [303] to find more diverse behaviors with better quality. Moreover, an extension of MAP-Elites allows to solves multiple tasks of the same family simultaneously using task-dependent fitness functions and a distance measure between those tasks [304]. MAP-Elites is also combined with task-independent measures. For example, Gravina et al. [305] use a task-dependent performance measure to fill the behavior-performance map but task-independent measures, such as novelty or surprise, for selection. In Sec. 5.3, we combine standard MAP-Elites with our minimize surprise approach. Other examples for quality-diversity algorithms are novelty search with local competition [306], surprise-search with local competition [307], and sparse reward exploration via novelty and emitters (SERENE) [308].

Conceptually related to approaches that are inspired by open-ended evolution are intrinsic motivations in machine learning, developmental robotics [309], and evolutionary robotics and artificial life [298, 310]. We discuss those in more detail in the next section.

2.3.3 Intrinsic Motivations

In contrast to task-specific rewards, task-independent fitness functions do not reward a specific goal or behavior. Approaches of intrinsically motivated learning combine machine learning approaches with computational approaches of intrinsic motivation, that are task-independent, to aim for open-ended learning [311, 312]. Inspiration is drawn from the psychological concept of intrinsic motivation, that is, activities are done for the joy or challenge of performing them and not for a reward or due to external pressures [313]. A clear definition of the underlying psychological concepts of intrinsic motivation is still missing and consequently, there is also no unique definition for computational approaches to it [310]. But while extrinsic motivations are generally considered to lead to the acquisition of material resources or the accomplishment of the user's goals in the case of robots, intrinsic motivations lead to the acquisition of knowledge and skills [314]. Measures for intrinsic motivation should thus be

- task-independent,
- free of semantics (i.e., meaning of sensor values), and
- applicable to any agent embodiment including its sensory-motoric configuration.

They can be calculated from the agent's perspective and either relate to its knowledge or competence [310, 311, 315]. While the aim of intrinsic motivations is to push agents towards exploration in general, Oudeyer and Kaplan [310] argue that simple variants of computational intrinsic motivations that do not do so should still be conceptualized as such. Frequently, intrinsic motivations are used in single-agent reinforcement learning scenarios as a reward that is not directly related to the agent's task [311].

For example, Schmidhuber [316] proposes curious model-building controllers that are trained using reinforcement learning. Thereby, agents have a controller and an adaptive world model that predicts future perceptions based on its current perceptions and planned actions. Additionally to an extrinsic reward for the given task, the prediction error is used as an intrinsic reward that positively reinforces situations in which the agents fail to predict their environment. Thus, it drives the agent towards improving its own world model by exploring unknown situations and thus resembles curiosity.

Homeokinesis [317] is an intrinsically motivated approach for self-exploration and the emergence of complex motion patterns in simulated and real robots. Each agent has a controller and a forward model that predicts the next sensor values. Both are implemented by neural networks that are trained by gradient descent. While the forward model is trained using the prediction error, the learning signal for the controller is the so-called time-loop error. It is the difference between real and reconstructed sensor values of time step t. The reconstructed sensor values are determined by projecting the real sensor values of time step $t+1$ backward through the forward model and the controller. Minimizing the time-loop error destabilizes the system and is thus a driver for activity. This contrasts to *Homeostasis* where controller and forward model are trained using the prediction error and thus the system stabilizes.

Further examples for intrinsic motivations are empowerment [315, 318], learning progress [319, 320], novelty [321, 322], predictive information [323, 324], expected free energy [325], and surprise that can be based on the prediction error [326, 327] or on how unexpected states are [328]. Surprise-based intrinsic motivations usually encourage open-ended learning by maximizing surprise, but Berseth et al. [328] show that also the minimization of surprise (i.e., rewarding familiar states here) leads to increasingly complex behaviors in highly dynamic environments.

Intrinsic motivations in swarm and multi-agent settings are only rarely studied despite its potential to allow for the emergence of new functionality (e.g., communication) and for the development of new models of motivation (e.g., sharing motivations with other agents) [329]. Similarly, only few works use intrinsic motivations, or the information-theoretic measures they are based on, as drivers for evolutionary search. But, for example, Capdepuy et al. [330] generate coordinated collective behaviors by maximizing empowerment and Klyubin et al. [331] evolve a single agent's sensor layout and actuators to maximize empowerment. Prokopenko et al. [332] evolve spatiotemporal coordination in a modular robotic system using an entropy-based fitness measure and Friedman et al. [35] generate ant colony foraging behaviors by minimizing free energy in simulation. Sperati et al. [25] evolve coordinated group behaviors in simulation by maximizing the mean mutual information and transfer the best evolved individuals to a group of real robots. The application of intrinsic motivations in multi-agent and swarm settings frequently relies on global information (e.g., agent positions [330]) or on information about other agents (e.g., to calculate mutual information [25]), or are mostly applied in discrete state-action spaces [330, 333] because they are computationally expensive when applied in continuous settings. Our minimize surprise approach uses a fitness function that is based on a mathematical simple formulation of surprise as the prediction error (i.e., difference between actual and predicted sensor values) to evolve swarm robotic behaviors. Since fitness can be calculated for each swarm member based on local data, minimize surprise is suitable for centralized and distributed evolutionary setups (see Sec. 2.3.2). The simplicity of our fitness function makes it computationally inexpensive in discrete and continuous settings, and extends the idea of simplicity in swarm robotics from swarm members and control (see Sec. 2.1.2) to the process of the automatic generation of collective behaviors.

The Minimize Surprise Approach 3

Chapter Contents

In this chapter, we introduce...

- *Sec. 3.1:* the inspiration for and
- *Sec. 3.2:* the fundamentals of our minimize surprise approach, and
- *Sec. 3.3:* our previous work.

Here, we introduce our minimize surprise approach that is used in the following chapters as an innate motivation to evolve collective behaviors of varying complexity.

3.1 Inspiration

Biology, neuroscience and other scientific disciplines often serve as an inspiration for artificial intelligence. Our minimize surprise approach is loosely inspired by the information-theoretic 'free-energy principle' that was proposed by neuroscientist Karl J. Friston as a potential general theory for brain and behavior [327, 334, 335]. It is a mathematical formulation of the idea of the 'Bayesian brain', which states that living organisms try to infer the causes of their sensations based on a model of their world [336]. This idea is already discussed since ancient times, for example, by Plato in his *Allegory of the Cave* [337]. Helmholtz [338] formalized it as a theory of 'unconscious inference', that is, perceptions are formed by a probabilistic inference process of the causes of sensations.

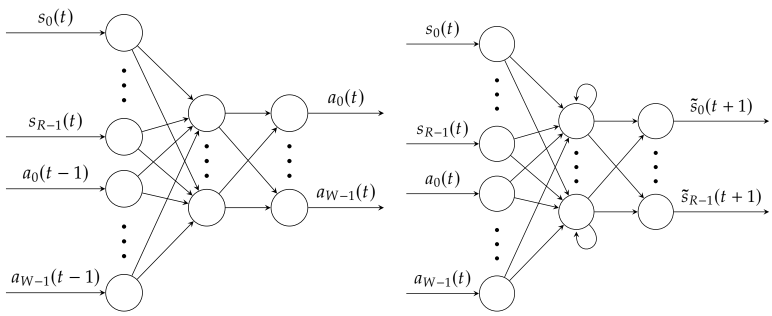
The idea formulated by Helmholtz [338] influenced computer science, psychology, and neuroscience. Dayan et al. [30] propose the so-called Helmholtz machines, that is, an approach of unsupervised machine learning to train a generative model of a data set. Furthermore, Friston formulated his free-energy principle based on Helmholtz's ideas about perception [334, 335]. Free energy is defined, in the simplest case, as the difference between observed and predicted sensor values (i.e., prediction error) and is an upper bound to surprise [339]. Organisms constantly try to minimize free energy, and thus surprise, either by optimizing the internal world model or by adjusting their actions so that they lead to sensor values matching the predictions [31]. Both variants bring advantages: improving the predictor makes an organism more knowledgeable about the world, while the complexity of the predictor can be reduced by choosing actions to actively seek for specific sensory stimulation [29]. This minimization of surprise, or maximization of expectation, brings an evolutionary advantage as organisms will maintain homeostasis by staying in predictable and safe environments by only selectively navigating and sampling their environment. Thus, they survive longer [335]. But the 'Dark-Room Problem' is frequently used as a critique of approaches that rely on surprise minimization [340]. This problem states that while the easiest way to minimize surprise is to search for and stay in a dark, unchanging room, organisms do not behave like this. Friston et al. [340] argue that organisms will be surprised by a dark room though if their world model does not expect it based on its prior beliefs and thus surprise is minimized for an agent's own econiche. Schwartenbeck et al. [341] further point out that minimizing surprise can even lead to exploration and novelty, since an agent can improve its world model by seeking out new states, which in turn can help to minimize surprise in future.

Friston's mathematically complex free-energy principle was applied to single robot systems [32–34] and recently also to simulated collective systems [35, 36]. Here, we use a simpler approach to the minimization of surprise that is only loosely inspired by Friston's work. We directly minimize the prediction error, that is, the absolute difference between predicted and real sensor values, to evolve swarm robot behaviors in simulation and on real robots. Thereby, we focus both on interactions between agents and environment as well as on agent-agent interactions. The latter introduces a self-referential loop, because the sensor inputs of agents are not only based on their own actions and a potentially dynamic environment but also on the actions of other identical, close-by agents. Thus, agents have to predict also the behavior of their neighbors. Besides that, our approach follows the idea of 'swarm cognition' by drawing a connection between a neuroscientific concept and swarm intelligence [342].

3.2 Approach

Following this inspiration by the free-energy principle, our 'minimize surprise' approach uses a task-independent fitness function that is based on the prediction error to drive evolution [28]. For this purpose, we equip each member of a swarm (i.e., robot or agent) with an actor-predictor pair of artificial neural networks (ANN) that are evolved together.

The actor ANN serves as a controller. It is implemented as a feedforward network as visualized in Fig. 3.1a. Those networks have already proven to be suitable as controllers for a variety of swarm robotics tasks, such as foraging [278] or aggregation and coordinated motion [279]. The actor receives the sensor values $s_0(t), \ldots, s_{R-1}(t)$ of the current time step t and all or a subset of the last chosen actions (i.e., outputs $a_0(t-1), \ldots, a_{W-1}(t-1)$ of the action network of the previous time step $t-1$) of the respective swarm member as input. Thereby, we introduce limited memory by



(a) actor **(b)** predictor

Figure 3.1: Actor-predictor ANN pair of each swarm member in minimize surprise. The actor (a) outputs W action values $a_0(t), \dots, a_{W-1}(t)$ and the predictor (b) outputs R sensor value predictions $\tilde{s}_0(t+1), \dots, \tilde{s}_{R-1}(t+1)$ for time step $t+1$. Inputs are R sensor values $s_0(t), \dots, s_{R-1}(t)$ at time step t and the action values $a_0(t-1), \dots, a_{W-1}(t-1)$ of time step $t-1$ or $a_0(t), \dots, a_{W-1}(t)$ of time step t [343].

making the network recurrent although it is implemented as a feedforward network. The actor outputs W action values that encode the swarm member's next action. A variety of encodings are possible, for example, discrete actions, such as straight motion and turning, or continuous motor speeds.

The predictor ANN forecasts the sensor values[1] of the next time step. It resembles world models as used in intelligent agents [95], but here we use it to drive the emergence of collective behaviors rather than to make intelligent decisions. We implement the predictor ANN as a recurrent network as visualized in Fig. 3.1b. Recurrent neural networks are especially suitable for sequential data, such as the time-series of sensor values here, since they have feedback connections. That is, nodes can receive outputs of the previous time step of some hidden nodes additionally to the current data as input and thus the network output is influenced both by the previous and the current time steps [344]. Recurrent networks have thus potential for memory. The predictor receives the swarm member's R sensor values $s_0(t), \dots, s_{R-1}(t)$ of the current time step t and all or a subset of the next actions $a_0(t), \dots, a_{W-1}(t)$[2] of the respective swarm member as input. The action values are fed into the predictor as the agent's actions influence its future sensor readings. The predictor can be both seen as a world model, as mentioned above, or as self-referential depending on the used sensors. The prediction of exteroceptive sensors equals predicting the future state of the environment. That way the predictor operates as a model of the external world. The prediction of interoceptive and proprioceptive sensors is predicting the future state of the robot itself and thus can lead to a sense of agency and a sense of self [345]. In the work presented here, we only predict exteroceptive sensors.

1: These sensor values can be raw sensor readings (e.g., used in Chs. 4 and 10) or aggregated sensor data (e.g., used in Ch. 8).

2: That is, the outputs of the action network of the current time step t.

We reward high prediction accuracy, that is, minimal surprise, by defining the fitness function F as

$$F = \frac{1}{TNR} \sum_{t=0}^{T-1} \sum_{n=0}^{N-1} \sum_{r=0}^{R-1} (1 - |\tilde{s}_r^n(t) - s_r^n(t)|)\,, \qquad (3.1)$$

with evaluation length T in time steps, swarm size N, number of sensors[3] per swarm member R, prediction $\tilde{s}_r^n(t)$ for sensor r of swarm member n, and value $s_r^n(t)$ of sensor r of swarm member n at time step t. Fitness is normalized to a theoretical maximum of 1. The fitness function is not only task-independent but also free of semantics and applicable to any agent embodiment. Furthermore, individual agents can calculate it from their own perspective. Thus, it can be classified as a simple variant of computational intrinsic motivation (see Sec. 2.3.3).

3: In most experiments, we predict all R sensor values of a swarm member. However, predicting only a subset of a swarm member's sensors is also possible (e.g., done in Ch. 7). In such cases, we differentiate number of sensors R_{sen} and number of predicted sensors R_{pred}. Fitness is then calculated considering all predicted sensors and normalized by R_{pred}.

Self-supervised learning [274, 346] could be used to train the predictor because the real sensor values $s_r(t)$ of time step t are the targeted predictor outputs of the previous time step, that is, the sensor predictions $\tilde{s}_r(t)$ for time step t. Since our approach is task-independent, we cannot determine a deviation between a current and a targeted swarm behavior. Consequently, we cannot manually or automatically generate labeled data to train the actor using (self-)supervised learning. By contrast, approaches of evolutionary computation rely on stochastic trial and error and thus do not require labeled data for optimization. This allows us to optimize pairs of actor and predictor networks concurrently while relying on a fitness function that measures only the quality of the predictor (Eq. 3.1). The stochastic processes of evolution (i.e., selection and variation) act then on the actor-predictor pairs. In App. A, however, we present a potential approach to combine machine learning methods with minimize surprise showing potentials and drawbacks. In all other presented experiments, we rely on neuroevolution of the synaptic weights while keeping fixed topologies for both networks. Genomes directly encode the weights of the actor-predictor ANN pairs (see Fig. 3.2) [347]. All swarm members of an evaluation share the same genome (i.e., application of the same synaptic weights), that is, we use a homogeneous swarm. Thus, we have two different population concepts:

- a **population of swarm members** that forms the homogeneous swarm in an evaluation of a genome and applies the actor-predictor ANN pair, and
- a **population of genomes** encoding the weights of pairs of networks in the evolutionary process.

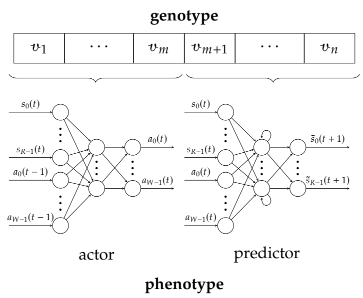


Figure 3.2: Genotype-phenotype mapping. Values v of the genome are used as the synaptic weights of the ANNs.

We use simple evolutionary algorithms here [16] while the study of more sophisticated methods of evolutionary computation will be subject of future work (see Sec. 12.2). The details about the used evolutionary algorithms are explained in more detail in the respective chapters.

By evolving the actor-predictor pairs using our task-independent fitness function (Eq. 3.1), we put selective pressure on the predictor. The actor, by contrary, does not receive direct selective pressure but is still subject to genetic drift. Collective behaviors emerge as a by-product in the evolutionary process as actors and predictors are evolved in pairs. High prediction accuracy (i.e., fitness) is reached when behaviors resulting from the selected actions of the actor lead to sensor values as predicted by the predictor. In turn, higher fitness values result in a higher likelihood to survive in the evolutionary process. It is left to the evolutionary dynamics and generally difficult to analyze whether actors adapt agent behaviors to predictions or predictors adapt their predictions to behaviors in our setup (see Sec. 3.1).

3.3 Previous Work

The first work on minimize surprise was published by Hamann [28] in 2014. The author aims for the evolution of collective behaviors on a simulated 1D torus (see Fig. 3.3) using a simple evolutionary algorithm and the minimize surprise fitness function, that is, rewarding high prediction accuracy (Eq. 3.1). The swarm in these experiments consists of simple agents with four binary sensors that cover their neighborhood. The actor ANN chooses between two actions: staying with the agent's current direction or switching the direction. Thus, the setup allows for the emergence of collective behaviors of low complexity that rely solely on agent-agent interactions. The behaviors can be differentiated based on

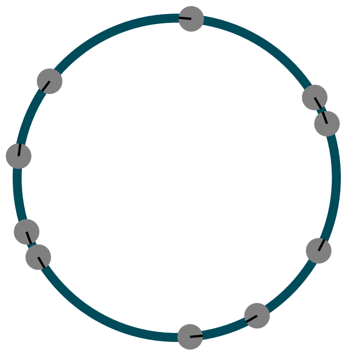


Figure 3.3: Illustration of the 1D torus environment used in the initial study of minimize surprise by Hamann [28].

- **motion**, that is, agents are either moving or stopped, and
- **relative position**, that is, the distance between agents.

moving	**flocking**	**random**
stopped	**aggregation**	**dispersion**
	minimal	maximal

distance

Figure 3.4: Differentiation of behaviors based on motion and relative position (i.e., distance) as used in the initial study of minimize surprise [28].

Fig. 3.4 summarizes the four behaviors that are possible with this differentiation. All four behaviors emerged in these experiments. Furthermore, the author finds that swarm density influences the emergence of behaviors, that is, higher densities lead to more aggregation behaviors (i.e., agents are positioned closer to each other) and low densities to more dispersion behaviors (i.e., agents are spaced further apart). Intermediate densities lead to the emergence of more complex behaviors, that is, flocking here. In this case, sensor inputs are changing over time and thus prediction gets harder.

Zahadat et al. [348, 349] extend the basic minimize surprise approach with an explicit driver for exploration and curiosity. Their so-called *diverse-prediction* method rewards both high prediction accuracy and visiting many different sensory states, that is, combinations of sensor values. As before, experiments are run on a 1D continuous torus with simple agents. The authors show that their approach leads to the emergence of more diverse behaviors independent of swarm density. A drawback of the approach is that it cannot be extended easily to continuous sensors as this would lead to infinitely many sensory states.

Borkowski and Hamann [37] extend the results of the standard minimize surprise approach to a 2D simulated torus environment. Agents are equipped with two or four discrete sensors that cover the neighborhood in front and around the agent, respectively. The actor outputs an action value that switches between straight motion and rotation as well as a linear and an angular speed. The authors find several basic collective behaviors: flocking, two variants of aggregation, and dispersion. In aggregation, agents are either stopped or a rotating cluster of circling agents is formed. Borkowski and Hamann [37] exemplify that evolution can be biased by the agent's sensor model on the basis of the dispersion behavior. The simpler sensor model with two sensors to the front leads to dispersion with circling agents. By contrast, the sensor model with four sensors around the agent leads to less motion, that is, to stopped agents.

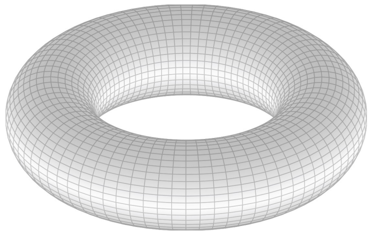


Figure 3.5: Visualization of a 2D torus.

All previous works were conducted in rather simple simulation environments aiming for basic collective behaviors and only provide short studies of the emergent behaviors and diversity. We extend these works considerably by studying more complex behaviors that we analyze more thoroughly with respect to their robustness, scalability, and diversity. Furthermore, we extend the approach to also work in realistic simulations and on real robots.

Methods

Evolution of Self-Assembly Behaviors by Minimizing Surprise

4

Chapter Contents

In this chapter, we study the evolution of self-assembly behaviors with minimize surprise in a simple grid-based environment. We...

- *Sec. 4.1:* introduce the experimental setup, and
- *Sec. 4.2:* the evaluation metrics and methods,
- *Sec. 4.3:* justify the choice of our sensor model,
- *Sec. 4.4:* study the emergent behaviors over grid size,
- *Sec. 4.5:* show the effectiveness of our minimize surprise approach, and
- *Sec. 4.6:* draw a conclusion.

Parts of this chapter are based on [38, 39, 42].

The first works on our minimize surprise approach (see Sec. 3.3) focused on the evolution of basic collective behaviors of rather low complexity in simple simulation environments [28, 37]. In this chapter, we aim for the emergence of more complex self-assembly behaviors (see Sec. 2.2.1) in simulated 2D torus grid worlds [38, 39, 42]. Thereby we not only evolve more complex behaviors than before, but also have a scenario that allows for an in-depth study of the resulting behaviors as computational resources are kept within reasonable limits. In this chapter, we focus on the emergent behaviors over grid sizes to study the resulting behavioral diversity (research question Q1, Sec. 1.2) and on the effectiveness of our approach. In Ch. 5, we then study the robustness, scalability, and diversity of the emergent behaviors in comparison to state-of-the-art approaches in detail to address the remaining aspects of research question Q1.

4.1 Experimental Setup

Aiming for the emergence of self-assembly behaviors increases the potential complexity compared to our previous scenarios considerably. We govern complexity in two ways: (i) by following the approach of emulated self-assembly, and (ii) by restricting ourselves to a simple 2D torus grid world. Emulated self-assembly requires individual agents only to position themselves next to each other without connecting physically or virtually and is frequently used in self-assembly studies [157, 158]. Consequently, we do not have to consider connection mechanisms in our setup. Furthermore, restricting ourselves to a simple 2D grid world with periodic boundary conditions[1] (i.e., a torus) leads to

1: For all coordinates in 2D environments with periodic boundary conditions ($x \bmod L_x$, $y \bmod L_y$) holds with side lengths L_x and L_y in x- and y-direction, respectively.

simplified sensing and enforces equidistant positioning of agents.

On our 2D torus grid lives a simulated swarm of $N = 100$ simple agents. Each grid cell can only be occupied by one agent at a time. Each agent n has a position $P_n(t) = (x_n(t), y_n(t))$ on a grid cell and a discrete heading $H_n(t) = (h_n^x(t), h_n^y(t))$, that is either North $H = (0, 1)$, East $H = (1, 0)$, South $H = (0, -1)$, or West $H = (-1, 0)$, at time step t. In each time step, agents execute one of two possible actions: moving one grid cell forward, that is,

$$\begin{aligned} P_n(t+1) = ((x_n(t) + h_n^x(t)) \bmod L_x, \\ (y_n(t) + h_n^y(t)) \bmod L_y)\,, \\ H_n(t+1) = H_n(t)\,, \end{aligned} \tag{4.1}$$

or rotating ± 90° on the spot, that is,

$$\begin{aligned} P_n(t+1) &= P_n(t)\,, \\ H_n(t+1) &= (-h_n^y(t), h_n^x(t)) \text{ or} \\ H_n(t+1) &= (h_n^y(t), -h_n^x(t))\,. \end{aligned} \tag{4.2}$$

A move forward is only possible if the grid cell in front is not yet occupied by another agent. If an agent attempts to move on a grid cell that is already occupied by another agent, the move forward is prevented and the agent stays on its current grid cell. This is similar to a hardware protection layer in real robots that prevents collisions with other robots or obstacles. We equip each agent with $R = 14$ binary sensors covering its neighboring grid cells as visualized in Fig. 4.1. A sensor value of '0' indicates that the respective grid cell is empty while a '1' means that it is occupied by another agent. The choice of the sensor model is studied in more detail in Sec. 4.3. This overall simple experimental setup allows for a structured and detailed study of the emergent behaviors.

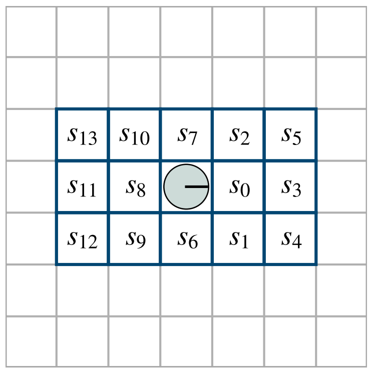
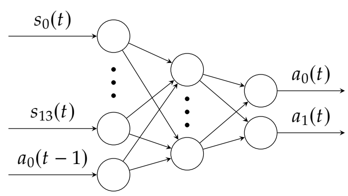
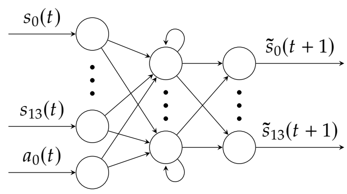


Figure 4.1: Sensor model in the self-assembly scenario with labels for each sensor. Each agent has 14 binary sensors covering its neighborhood. The blue circle represents the agent and the black line indicates its heading.

Following our minimize surprise approach (see Ch. 3), we equip each agent with an actor-predictor ANN pair as visualized in Fig. 4.2. Actor and predictor are both three-layer ANNs with an input, one hidden and an output layer. We use the hyperbolic tangent `tanh` as the transfer function and map the network outputs to our discrete action values and sensor value predictions. The actor network (see Fig. 4.2a) has 15 input neurons, eight hidden neurons and two output neurons. It determines the agent's next action by outputting two action values: $a_0(t)$ decides whether to move or turn and $a_1(t)$ determines the turning direction (i.e., ±90°). The actor receives the agent's 14 current sensor values $s_0(t), \ldots, s_{13}(t)$ and its last action $a_0(t-1)$ as inputs. We use only a_0 as input to the ANNs as in previous works [37]. While the actor always outputs a turning direction $a_1(t)$, it is solely informative when $a_0(t)$ selects to turn and may even be deceptive when $a_0(t)$ selects to move. A study of the influence on the emergent behaviors when $a_1(t)$ is used as an additional ANN input remains subject of future work. The predictor network (see Fig. 4.2b) has 15 input neurons, 14 hidden recurrent neurons and 14 output neurons. It outputs predictions $\tilde{s}_0(t+1), \ldots, \tilde{s}_{13}(t+1)$ for the $R = 14$ sensor values of the next time step $t+1$. That is, agents predict whether they may see other agents. The predictor receives the agent's 14 current sensor values $s_0(t), \ldots, s_{13}(t)$ and its next action $a_0(t)$ as inputs.



(a) actor



(b) predictor

Figure 4.2: Actor-predictor ANN pair in the self-assembly scenario. $a_0(t-1)$ is the agent's last action value and $a_0(t)$ is its next action determining whether to move or turn. $a_1(t)$ determines the turning direction. $s_0(t), \ldots, s_{13}(t)$ are the agent's 14 sensor values at time step t, $\tilde{s}_0(t+1), \ldots, \tilde{s}_{13}(t+1)$ are its sensor predictions for time step $t+1$.

We evolve the actor-predictor ANN pairs using a simple evolutionary algorithm (see Sec. 2.3.2) and reward high prediction accuracy as defined by our minimize surprise fitness function (Eq. 3.1). We run the evolutionary algorithm for $g_{\max} = 100$ generations (i.e., termination criterion) and evaluate each genome in ten independent simulation runs for $T = 500$ time steps each. The fitness of a genome is the minimum fitness observed in ten independent evaluations. Thus, ANN pairs that perform poorly in some runs are eliminated and pressure put on the evolutionary algorithm to find generally well performing solutions. Genomes encode the synaptic weights of both neural networks (see Sec. 3.2) and we randomly generate the initial population $\mathcal{P}(0)$ by drawing the weights from a uniform distribution in $[-0.5, 0.5]$. Our swarm is homogeneous both related to agent model and controller, that is, each swarm member has an instance of the same genome in a given evaluation. We place agents with a uniformly random heading uniformly random on the grid at the beginning of each evaluation. For the evolutionary algorithm, we use a population size μ of 50, proportionate parent selection, age-based survivor selection, and elitism of one. We generate $\lambda = \mu - 1$ offspring for the population of the next generation. We apply mutation only, that is, we do not use recombination. Each value v of a genome is mutated with a probability p_{mut} of 0.1 by adding a uniformly random number from $[-0.4, 0.4]$. For each setting, we do 50 independent evolutionary runs if not indicated otherwise. We post-evaluate the best evolved individuals (i.e., the ANN pairs with the highest fitness in the last generation of the evolutionary runs) for our in-depth study of the emergent behaviors. Tab. 4.1 summarizes all parameters of our experimental setup.

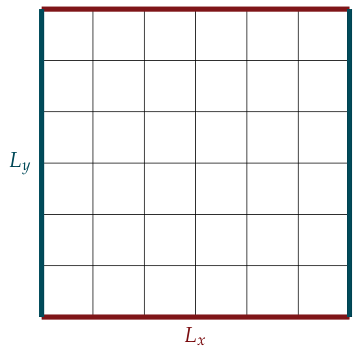


Figure 4.3: Fundamental polygon of the 2D torus grid environment with side lengths L_x in x-direction and L_y in y-direction. A torus (see Fig. 3.5) is formed by connecting the opposite sides of the polygon (marked in the same color).

In our self-assembly experiments, we study the effect of swarm density on the emergent swarm behaviors. Swarm density $D_N \in [0, 1]$ is the fraction of the arena occupied by agents and defined as

$$D_N = \frac{N}{L_x \times L_y}, \tag{4.3}$$

parameter	value
grid side length L	$[11\,..\,30]$
swarm size N	100
# of sensors and predictor outputs R	14
sensor values s_r	$\{0, 1\}$
action value a_0	{straight, turn}
action value a_1	$\pm 90°$
population size μ	50
number of generations $g_{\max}$	100
evaluation length T (time steps)	500
# of simulation runs per fitness evaluation	10
elitism	1
mutation rate p_{mut}	0.1

Table 4.1: Parameters for the self-assembly scenario.

where N is the swarm size and $L_x \times L_y$ is the grid size.[2] We vary swarm density using different sizes $L_x \times L_y$ of the fundamental polygon (see Fig. 4.3) of our 2D torus grid environment while keeping a fixed swarm size N of 100. In the majority of experiments, we use square grid worlds (i.e., $L_x = L_y = L$) with side lengths $L \in [11\,..\,30]$. This leads to swarm densities between 0.11 ($\frac{100}{30 \times 30}$) and 0.83 ($\frac{100}{11 \times 11}$), see Fig. 4.4. A grid size of 10×10 has a swarm density of 1.0, that is, each grid cell is occupied by one agent, and obviously only allows for aggregation. In that case, the actor's outputs are negligible, since agents will be constantly on their initial grid cells and evolution would lead to a predictor constantly outputting '1' for all sensor predictions. We mainly restrict us to square tori, since we do not expect much influence of the torus geometry on the resulting behaviors. But in Sec. 6.2.2, we study the influence of torus geometry on one resulting behavior.

2: L_x and L_y specify the number of grid cells in this scenario.

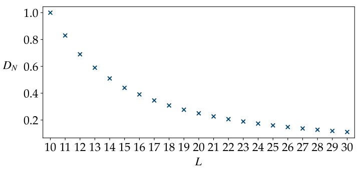


Figure 4.4: Swarm densities D_N for all studied square grid sizes $L \times L$ in the self-assembly scenario.

4.2 Evaluation Metrics and Methods

In this section, we introduce metrics and methods for the quantitative evaluation of the emergent behaviors.

4.2.1 Metrics

Additionally to fitness (i.e., prediction accuracy, Eq. 3.1), we define three other evaluation metrics: (i) temperature Θ of the system, (ii) agent movement M_N, and (iii) intended agent movement I_N.

We define the temperature Θ of the system as the forward motion of the agents in time step t, that is, the mean covered distance of the agents in that time step. The agents' rotations and prevented forward movements are not considered. Our temperature measurement Θ is inspired by thermodynamics where temperature is proportional to the average kinetic energy of the molecules' center-of-mass motions [350]. The temperature decreases with increasing number of agents that stay on their grid cells. Consequently, these agents turn or are blocked in their forward movement as their targeted grid cells are already occupied. A hot system is in a disordered state with many moving agents and a cool system relates to a more ordered system, which has assembled into a structure. We define temperature Θ as

$$\Theta(t) = \frac{1}{N} \sum_{n=0}^{N-1} d_M(P_n(t), P_n(t+1)) \,, \tag{4.4}$$

where N is the swarm size and $d_M(\cdot,\cdot)$ is the Manhattan distance between positions $P_n(t)$ and $P_n(t+1)$ of agent n in time steps t and $t+1$, respectively. We normalize temperature Θ by the number of agents N, which results in $\Theta \in [0,1]$, since each agent can move maximally one grid cell forward per time step. The Manhattan distance $d_M(P_i, P_j)$ between two positions $P_i = (x_i, y_i)$ and $P_j = (x_j, y_j)$ on a torus with a fundamental polygon of size $L_x \times L_y$ (see Fig. 4.3) is given by

$$d_M(P_i, P_j) = d_x(x_i, x_j) + d_y(y_i, y_j) \tag{4.5}$$

with

$$d_x(x_i, x_j) = \min(|x_i - x_j|, L_x - |x_i - x_j|) \tag{4.6}$$

and

$$d_y(y_i, y_j) = \min(|y_i - y_j|, L_y - |y_i - y_j|) \,. \tag{4.7}$$

We measure the motion of agents using the Manhattan distance, since agents cannot move diagonally in our experimental setup.

As a second metric, we introduce agent movement M_N. It is the mean covered distance of agents as an integral of displacement at each time step over a time period of $\tau = \frac{L_x \times L_y}{2}$ time steps as in previous work by Hamann [28]. Time period τ allows agents to cover the same relative distance, that is, 50 % of the grid cells. Tab. 4.2 gives the number τ of time steps per studied square grid size $L \in [11 \,..\, 30]$. Agent movement equals the mean temperature over time period τ. We define agent movement M_N as

Table 4.2: Time period of $\tau = \frac{L_x \times L_y}{2}$ time steps per grid size $L \times L$.

L	τ	L	τ
11	60	21	220
12	72	22	242
13	84	23	264
14	98	24	288
15	112	25	312
16	128	26	338
17	144	27	364
18	162	28	392
19	180	29	420
20	200	30	450

$$
\begin{aligned}
M_N &= \frac{1}{\tau} \sum_{t=T-\tau}^{T-1} \Theta(t) \\
&= \frac{1}{\tau N} \sum_{t=T-\tau}^{T-1} \sum_{n=0}^{N-1} d_M(P_n(t), P_n(t+1)) \,,
\end{aligned}
\tag{4.8}
$$

where $\Theta(t)$ is the temperature at time step t (Eq. 4.4) and $d_M(\cdot,\cdot)$ is the Manhattan distance (Eq. 4.5). We normalize agent movement M_N by swarm size N and time period τ (i.e., $M_N \in [0, 1]$).

Furthermore, we define intended agent movement I_N, that is, how often the actor outputs to move one grid cell forward. As agents are prevented to move forward when a targeted grid cell is occupied, the intended agent movement I_N and the real agent movement M_N can deviate. Consequently, a large deviation between intended and real agent movement serves as an indicator that agents stand still by exploiting the prevention of forward movements. We define intended agent movement I_N as

$$
I_N = \frac{1}{\tau N} \sum_{t=T-\tau}^{T-1} \sum_{n=0}^{N-1} a_0^n(t) \,, \tag{4.9}
$$

where N is the swarm size, τ is a time period of $\frac{L_x \times L_y}{2}$ time steps, and $a_0^n(t)$ is the action value of agent n at time step t, which determines whether to move straight or to turn. The action value is 1 for moving one grid cell forward and 0 for rotation.

4.2.2 Classification of Emergent Structures

We classify structures formed by the agents in our self-assembly scenario based on metrics for nine different patterns. These metrics are defined based on our experience that we gathered during an initial qualitative analysis of the results. This is an empirical approach to classify the emergent self-assembly behaviors (i.e., phenotypes) quantitatively. While we cannot guarantee that our set of patterns is complete, we are still confident that all distinguishable, relevant patterns are covered by our metrics. We differentiate between

- lines (`LN`) and pairs (`PR`),
- four grouping patterns: aggregation (`AG`), clustering (`CL`), loose grouping (`LG`), and swirls (`SW`), and
- three dispersion patterns: random dispersion (`RD`), squares (`SQ`), and triangular lattices (`TL`).

All nine patterns are either rotation symmetric (e.g., squares or triangular lattices) or exploit that the forward movement of agents is prevented when the grid cell in front is already occupied in order to form repetitive patterns (e.g., lines, pairs, swirls). In the case of rotation symmetric patterns, agents stay on their current grid cell by turning constantly. Thus, a 'boring', structured environment is created by the swarm that allows for simple sensor predictions and high prediction accuracy, since all agents in the pattern have the same or similar sensor readings.

Structures formed by the agents in the last time step (i.e., $t = T$)[3] of the post-evaluation runs of the best evolved individuals are automatically classified based on their highest resemblance to one of the nine patterns using Python scripts [39].[4] The highest resemblance to a pattern is determined by measuring the solution quality q_Z for each of the nine possible patterns. We define solution quality q_Z of pattern $Z \in \{\mathrm{LN}, \mathrm{PR}, \mathrm{AG}, \mathrm{CL}, \mathrm{LG}, \mathrm{SW}, \mathrm{RD}, \mathrm{SQ}, \mathrm{TL}\}$ as

$$q_Z = \frac{1}{N} \sum_{n=0}^{N-1} c_Z^n \,, \tag{4.10}$$

with swarm size N and criterion c_Z^n of pattern Z that evaluates to one ($c_Z^n = 1$) if agent n fulfills the pattern criterion and to zero ($c_Z^n = 0$) otherwise.

The self-assembled structure is classified according to the highest solution quality Q, that is, we define the solution quality Q of the best evolved individual as

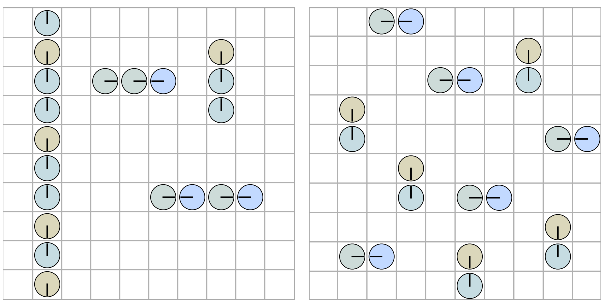
$$Q = \max(\{q_{\mathrm{LN}}, \ldots, q_{\mathrm{TL}}\}) \,. \tag{4.11}$$

Thus, solution quality measures the quantity of agents that are assembled into the dominant pattern in the last time step of the post-evaluation run.[5]

3: For conciseness, we denote by P_n the position $P_n(T)$ and by H_n the heading $H_n(T)$ of agent n in set of all agents $\mathcal{S}_N$ in the last time step T of the post-evaluation run, that is, we omit the time step in our notation in the following. In addition, $P_j = P_i + H_i$ always denotes $P_j = ((x_i + h_i^x) \bmod L_x, (y_i + h_i^y) \bmod L_y)$.

4: https://gitlab.iti.uni-luebeck.de/minimize-surprise/self-assembly

5: In case that we measure solution quality at a different time step, we specify the time step explicitly. For example, $Q(0)$ gives the solution quality at the beginning of a run.



(a) lines **(b)** pairs

Figure 4.5: Example illustrations of lines (LN) and pairs (PR). Agents are represented by circles, their color and the lines indicate their headings.

Lines (LN; Fig. 4.5a) and **pairs** (PR; Fig. 4.5b) are formed by agents that are horizontally or vertically placed next to each other. The criteria for pairs and lines differ only in the length of the structure: pairs consist of exactly two agents, while lines are formed by at least three agents. An individual pair or line is formed out of a set $\mathcal{S}^k_{\mathrm{PL}}$ of adjacent agents n in set of all agents $\mathcal{S}_N$ where

$$\forall n \in \mathcal{S}^k_{\mathrm{PL}} \exists m \in \mathcal{S}^k_{\mathrm{PL}} : P_m = P_n + H_n \,, \tag{4.12}$$

$$\begin{aligned}\forall n, m \in \mathcal{S}^k_{\mathrm{PL}} \exists \epsilon < |\mathcal{S}^k_{\mathrm{PL}}| : P_m = ((x_n \pm \epsilon h^x_n) \bmod L_x,\\ (y_n \pm \epsilon h^y_n) \bmod L_y)\,,\\ \wedge\, d_M(P_n, P_m) = \epsilon\,,\end{aligned} \tag{4.13}$$

and

$$\forall n, m \in \mathcal{S}^k_{\mathrm{PL}} : H_m = \pm H_n \,, \tag{4.14}$$

with positions $P_n = (x_n, y_n)$, $P_m = (x_m, y_m)$ and headings $H_n = (h^x_n, h^y_n)$, $H_m = (h^x_m, h^y_m)$ of agents n and m, respectively, and Manhattan distance $d_M(\cdot,\cdot)$ (Eq. 4.5), hold. This means that each agent has a neighbor on the next grid cell in its heading direction (Eq. 4.12), which allows agents to form stable structures by exploiting that all intended forward movement is blocked (see Sec. 4.1). Furthermore, all agents are positioned on horizontally or vertically adjacent grid cells with no empty grid cells in between (Eq. 4.13) and all agents have parallel headings (Eq. 4.14), that is, $H \in \{(1,0),(-1,0)\}$ for horizontal and $H \in \{(0,1),(0,-1)\}$ for vertical lines and pairs. Lines and pairs can have up to half of their length of neighboring agents[6] on each side next to them (i.e., $|\mathcal{S}^k_{\mathrm{NB},1}| \leq \frac{|\mathcal{S}^k_{\mathrm{PL}}|}{2}$, $|\mathcal{S}^k_{\mathrm{NB},2}| \leq \frac{|\mathcal{S}^k_{\mathrm{PL}}|}{2}$ with sets of neighboring agents per side of the structure $\mathcal{S}^k_{\mathrm{NB},1}$ and $\mathcal{S}^k_{\mathrm{NB},2}$). No two adjacent grid cells parallel to the structure are allowed to be occupied by neighboring agents. Therefore, all neighboring agents must have a Manhattan distance (Eq. 4.5) of at least two from each other. A swarm of agents can self-assemble into several pairs or lines $\mathcal{S}^k_{\mathrm{PL}}$. The set $\mathcal{S}_{\mathrm{PR}}$ of all assembled pairs is given by

$$\begin{aligned}\mathcal{S}_{\mathrm{PR}} = \{\mathcal{S}^k_{\mathrm{PL}} : |\mathcal{S}^k_{\mathrm{PL}}| = 2, |\mathcal{S}^k_{\mathrm{NB},1}| \leq \frac{|\mathcal{S}^k_{\mathrm{PL}}|}{2}, |\mathcal{S}^k_{\mathrm{NB},2}| \leq \frac{|\mathcal{S}^k_{\mathrm{PL}}|}{2},\\ \forall i, j \in (\mathcal{S}^k_{\mathrm{NB},1} \cup \mathcal{S}^k_{\mathrm{NB},2}), i \neq j : d_M(P_i, P_j) \geq 2,\\ 0 \leq k < K\}\,,\end{aligned} \tag{4.15}$$

6: All neighboring agents m above a horizontal line or pair structure (i.e., $H \in \{(1,0),(-1,0)\}$) or, respectively, to the right of a vertical line or pair structure (i.e., $H \in \{(0,1),(0,-1)\}$) are given by set $\mathcal{S}^k_{\mathrm{NB},1} = \{m : (P_m = (x_n + |h^y_n|) \bmod L_x, (y_n + |h^x_n|) \bmod L_y), m \in \mathcal{S}_N, n \in \mathcal{S}^k_{\mathrm{PL}}\}$. Similarly, all neighboring agents m below a horizontal line or pair structure or, respectively, to the left of a vertical line or pair structure are given by set $\mathcal{S}^k_{\mathrm{NB},2} = \{m : P_m = ((x_n - |h^y_n|) \bmod L_x, (y_n - |h^x_n|) \bmod L_y), m \in \mathcal{S}_N, n \in \mathcal{S}^k_{\mathrm{PL}}\}$.

where K is the number of sets of adjacent agents $\mathcal{S}^k_{\mathrm{PL}}$. Hence, we define the criterion c^n_{PR} for agent n being classified as pairs (PR) as

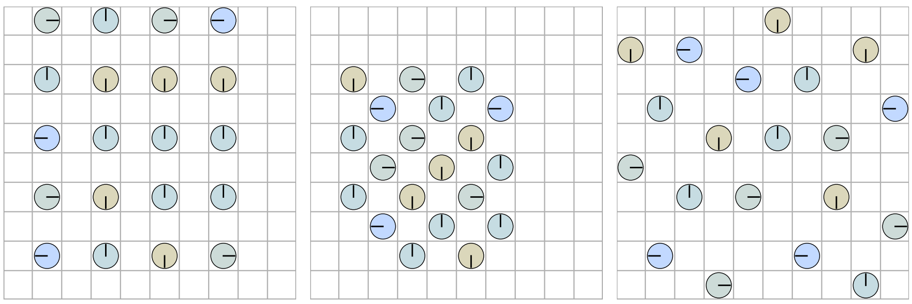
$$c^n_{\mathrm{PR}} = \begin{cases} 1, & \text{if } \exists \mathcal{S}_{\mathrm{PL}} \in \mathcal{S}_{\mathrm{PR}} : n \in \mathcal{S}_{\mathrm{PL}} \\ 0, & \text{otherwise} \end{cases} . \tag{4.16}$$

Accordingly, the set $\mathcal{S}_{\mathrm{LN}}$ of all assembled lines is given by

$$\begin{aligned} \mathcal{S}_{\mathrm{LN}} = \{\mathcal{S}^k_{\mathrm{PL}} : |\mathcal{S}^k_{\mathrm{PL}}| > 2, |\mathcal{S}^k_{\mathrm{NB},1}| \leq \frac{|\mathcal{S}^k_{\mathrm{PL}}|}{2}, |\mathcal{S}^k_{\mathrm{NB},2}| \leq \frac{|\mathcal{S}^k_{\mathrm{PL}}|}{2}, \\ \forall i, j \in (\mathcal{S}^k_{\mathrm{NB},1} \cup \mathcal{S}^k_{\mathrm{NB},2}), i \neq j : d_M(P_i, P_j) \geq 2, \\ 0 \leq k < K\} . \end{aligned} \tag{4.17}$$

As already stated above, lines differ only by the length of the structure from pairs (i.e., $|\mathcal{S}^k_{\mathrm{PL}}| = 2$ for pairs and $|\mathcal{S}^k_{\mathrm{PL}}| > 2$ for lines). We define the criterion c^n_{LN} for agent n being classified as lines (LN) as

$$c^n_{\mathrm{LN}} = \begin{cases} 1, & \text{if } \exists \mathcal{S}_{\mathrm{PL}} \in \mathcal{S}_{\mathrm{LN}} : n \in \mathcal{S}_{\mathrm{PL}} \\ 0, & \text{otherwise} \end{cases} . \tag{4.18}$$



(a) squares **(b)** triangular lattice **(c)** random dispersion

Figure 4.6: Example illustrations of the three dispersion patterns: squares (SQ), triangular lattices (TL), and random dispersion (RD). Agents are represented by circles, their color and the lines indicate their headings.

We define three dispersion patterns: squares (Fig. 4.6a), triangular lattices (Fig. 4.6b), and random dispersion (Fig. 4.6c). These patterns are rotation symmetric, rendering the agent headings irrelevant here.

Squares (SQ) are formed by agents that are one grid cell apart in each direction, that is, agents are positioned on every other grid cell. An individual square is formed on a 5×5 segment of the torus grid with grid cell $G = (x_G, y_G)$ being the center of this segment, see Fig. 4.7. All agents n in this 5×5 segment are given by

$$\mathcal{S}^G_{5\times5} = \{n : d_x(x_n, x_G) \leq 2, d_y(y_n, y_G) \leq 2, n \in \mathcal{S}_N\}\,, \tag{4.19}$$

with Manhattan distances $d_x(\cdot,\cdot)$ and $d_y(\cdot,\cdot)$ along the x-axis and the y-axis as defined in Eqs. 4.6 and 4.7.

For the squares pattern, the corners of the inner 3×3 grid cells have to be occupied by agents (see Fig. 4.7). The set of agents on the respective grid cells is given by

$$\mathcal{S}^G_{\mathrm{SQ}} = \{n : d_x(x_n, x_G) = 1, d_y(y_n, y_G) = 1, n \in \mathcal{S}_N\}\,. \tag{4.20}$$

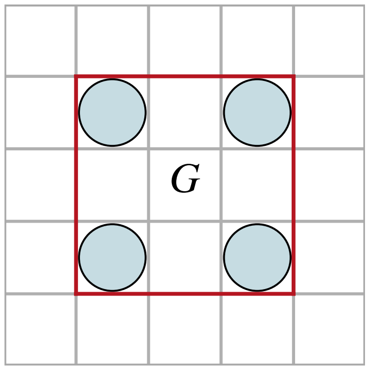


Figure 4.7: Illustration of a 5×5 segment of the grid forming an individual square with grid cell $G = (x_G, y_G)$ being the center of the segment. The inner 3×3 grid cells are indicated by the red line and agents by blue circles. The visualized agents form the set $\mathcal{S}^G_{\mathrm{SQ}}$ (Eq. 4.20).

All other grid cells in the 5×5 segment have to be empty. Hence, we define the criterion c^n_{SQ} for agent n to be classified as squares (SQ) as

$$c^n_{\mathrm{SQ}} = \begin{cases} 1, & \text{if } \exists G : n \in \mathcal{S}^G_{\mathrm{SQ}} \wedge |\mathcal{S}^G_{\mathrm{SQ}}| = 4 \wedge |\mathcal{S}^G_{5\times5}| = 4 \\ 0, & \text{otherwise} \end{cases}\,. \tag{4.21}$$

An agent can be part of more than one formed square as individual squares can overlap.

Triangular lattices (TL) are formed by agents that are positioned in a 2D diagonal square lattice. The structure is rotation symmetric for the center agent i. A triangular lattice segment consists of 21 grid cells on which nine agents are positioned, see Fig. 4.8. Triangular lattice segments can overlap, which means that an agent can be part of several triangular lattice segments.

A triangular lattice is formed by agent i in the center and its neighbors with a Manhattan distance of two. The set $\mathcal{S}^i_{\mathrm{TL}}$ of agents on the respective grid cells is given by

$$\mathcal{S}^i_{\mathrm{TL}} = \{n : d_M(P_i, P_n) = \epsilon, \epsilon \in \{0, 2\}, n, i \in \mathcal{S}_N\}\,, \tag{4.22}$$

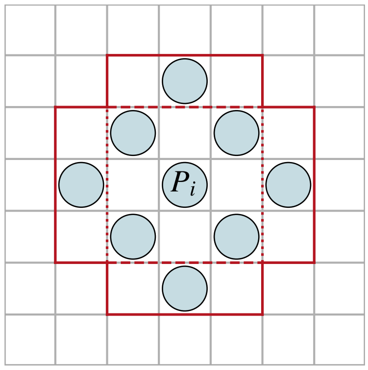


Figure 4.8: Illustration of a triangular lattice segment $\mathcal{S}^i_{\mathrm{TLS}}$ with agent i on position $P_i = (x_i, y_i)$ being the center of the segment. The formed structure is rotation symmetric for agent i, that is, the agent has constant sensor inputs independent from its heading. The segment is indicated by red lines, the sensor view of agent i in horizontal and vertical orientation by dashed and dotted red lines, respectively, and agents by blue circles. The visualized agents form the set $\mathcal{S}^i_{\mathrm{TL}}$ (Eq. 4.22).

with Manhattan distance $d_M(\cdot,\cdot)$ (Eq. 4.5) and position $P_i = (x_i, y_i)$ of agent i in the center of the triangular lattice segment. The other 12 grid cells in the triangular lattice segment have to be empty. The set $\mathcal{S}^i_{\mathrm{TLS}}$ of all agents on the 21 grid cells of a triangular lattice segment is given by

$$\begin{aligned}\mathcal{S}^{i}_{\text{TLS}} = \{n : d_M(P_i, P_n) \leq 2 \vee \\ (d_x(x_i, x_n) = 2, d_y(y_i, y_n) = 1) \vee \\ (d_x(x_i, x_n) = 1, d_y(y_i, y_n) = 2),\ n, i \in \mathcal{S}_N\}\,,\end{aligned} \tag{4.23}$$

with distances $d_x(\cdot,\cdot)$ (Eq. 4.6) and $d_y(\cdot,\cdot)$ (Eq. 4.7) along the x-axis and the y-axis, respectively. Thus, we define criterion c^n_{TL} for agent n to be classified as triangular lattices (TL) as

$$c^n_{\text{TL}} = \begin{cases} 1, & \text{if } \exists i \in \mathcal{S}_N : n \in \mathcal{S}^i_{\text{TL}} \wedge |\mathcal{S}^i_{\text{TL}}| = 9 \wedge |\mathcal{S}^i_{\text{TLS}}| = 9 \\ 0, & \text{otherwise} \end{cases}. \tag{4.24}$$

The **random dispersion** (RD) pattern is formed by agents that have either no neighbors in their von Neumann neighborhood or maximally one neighbor in their Moore neighborhood. Additionally, randomly dispersed agents are not allowed to be part of any other pattern. Based on the pattern criteria, randomly dispersed agents cannot, inherently, be part of one of the four grouping behaviors. Thus, we only explicitly check for triangular lattices, squares, lines, and pairs.

The set $\mathcal{S}^n_{\text{VN}}$ of agents in the von Neumann neighborhood (Fig. 4.9a) of an agent n is given by

$$\mathcal{S}^n_{\text{VN}} = \{m : d_M(P_m, P_n) = 1, m, n \in \mathcal{S}_N\}\,, \tag{4.25}$$

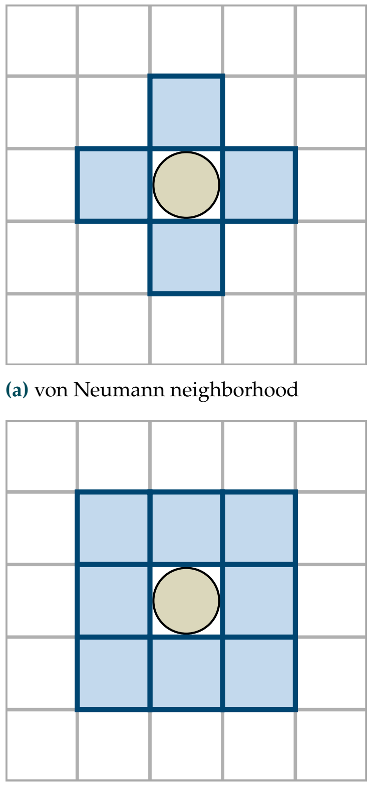
that is, all neighboring agents m with a Manhattan distance $d_M(\cdot,\cdot)$ (Eq. 4.5) of one. The set $\mathcal{S}^n_{\text{M}}$ of agents in the Moore neighborhood (Fig. 4.9b) of an agent n is given by

$$\begin{aligned}\mathcal{S}^n_{\text{M}} = \{m : d_x(x_m, x_n) \leq 1, d_y(y_m, y_n) \leq 1, \\ m, n \in \mathcal{S}_N, m \neq n\}\,,\end{aligned} \tag{4.26}$$

that is, all neighboring agents m with a maximum distance $d_x(\cdot,\cdot)$ (Eq. 4.6) of one in x-direction and a maximum distance $d_y(\cdot,\cdot)$ (Eq. 4.7) of one in y-direction.

We define the criterion c^n_{RD} for agent n to be classified as randomly dispersed (RD) as

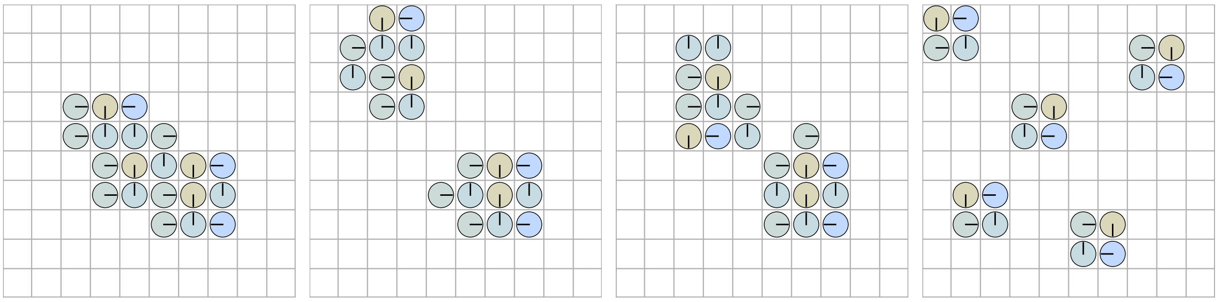
$$c^n_{\text{RD}} = \begin{cases} 1, & \begin{aligned}&\text{if } (|\mathcal{S}^n_{\text{M}}| \leq 1 \vee |\mathcal{S}^n_{\text{VN}}| = 0) \wedge \\ &\quad (c^n_{\text{TL}} = 0 \wedge c^n_{\text{SQ}} = 0 \wedge c^n_{\text{LN}} = 0 \wedge c^n_{\text{PR}} = 0)\end{aligned} \\ 0, & \text{otherwise} \end{cases}. \tag{4.27}$$



(a) von Neumann neighborhood

(b) Moore neighborhood

Figure 4.9: Von Neumann and Moore neighborhoods of an agent. The agent is represented by the beige circle, neighboring grid cells are colored blue.



(a) aggregation **(b)** clustering **(c)** loose grouping **(d)** swirls

Figure 4.10: Example illustrations of the four grouping patterns: aggregation (AG), clustering (CL), loose grouping (LG), and swirls (SW). Agents are represented by circles, their color and the lines indicate their headings.

We differentiate four grouping patterns: aggregation (Fig. 4.10a), clustering (Fig. 4.10b), loose grouping (Fig. 4.10c), and swirls (Fig. 4.10d).

Aggregation (AG), **clustering** (CL), and **loose grouping** (LG) are patterns that are based on agent clusters. Individual clusters are formed by agents with at least three neighbors in their von Neumann neighborhood $\mathcal{S}^n_{\mathrm{VN}}$ (Eq. 4.25) and at least six neighbors in their Moore neighborhood $\mathcal{S}^n_{\mathrm{M}}$ (Eq. 4.26). The agents' neighbors are also part of the cluster, irrespective of whether they fulfill those neighborhood criteria themselves. Hence, we define measure c^m_{CLN} indicating if agent n fulfills these neighborhood conditions as

$$c^n_{\mathrm{CLN}} = \begin{cases} 1, & \text{if } |\mathcal{S}^n_{\mathrm{M}}| \geq 6 \wedge |\mathcal{S}^n_{\mathrm{VN}}| \geq 3 \\ 0, & \text{otherwise} \end{cases} . \tag{4.28}$$

We define the set $\mathcal{S}^k_{\mathrm{GR}}$ of agents grouped into an individual cluster recursively. Agent m fulfills the neighborhood criterion c^n_{CLN} and initializes cluster $\mathcal{S}^k_{\mathrm{GR}}$, that is,

$$m \in \mathcal{S}^k_{\mathrm{GR}} \,, c^m_{\mathrm{CLN}} = 1 \,. \tag{4.29}$$

If an agent n is part of cluster $\mathcal{S}^k_{\mathrm{GR}}$ and fulfills the neighborhood criterion c^n_{CLN}, then all neighbors j in its Moore neighborhood are also part of cluster $\mathcal{S}^k_{\mathrm{GR}}$ independent of whether they fulfill the neighborhood criterion themselves as given by

$$n \in \mathcal{S}^k_{\mathrm{GR}} \wedge c^n_{\mathrm{CLN}} = 1 \wedge j \in \mathcal{S}^n_{\mathrm{M}} \longrightarrow j \in \mathcal{S}^k_{\mathrm{GR}} \,. \tag{4.30}$$

We define $\mathcal{S}_{\mathrm{CL}} = \{\mathcal{S}^k_{\mathrm{GR}} : 0 \leq k < K\}$ as the set of all K clusters $\mathcal{S}^k_{\mathrm{GR}}$ formed by the swarm. We differentiate between aggregation, clustering, and loose grouping based on the number K of formed clusters and whether clusters are connected. Aggregation is the formation of a single cluster, that is, the criterion c^n_{AG} for agent n being classified as aggregation (AG) is

$$c^n_{\mathrm{AG}} = \begin{cases} 1, & \text{if } |\mathcal{S}_{\mathrm{CL}}| = 1 \land \exists \mathcal{S}_{\mathrm{GR}} \in \mathcal{S}_{\mathrm{CL}} : n \in \mathcal{S}_{\mathrm{GR}} \\ 0, & \text{otherwise} \end{cases} . \tag{4.31}$$

Clustering and loose grouping consist of several clusters ($|\mathcal{S}_{\mathrm{CL}}| > 1$) and are differentiated based on the connection of clusters. Two clusters $\mathcal{S}^i_{\mathrm{GR}}$ and $\mathcal{S}^j_{\mathrm{GR}}$ are directly connected if there is at least one agent that is part of both clusters but does not fulfill the neighborhood criterion itself as visualized in Fig. 4.11. The set $\mathcal{E}$ of direct connections between clusters is given by

$$\begin{aligned} \mathcal{E} = \{[i,j] : \mathcal{S}^i_{\mathrm{GR}} \cap \mathcal{S}^j_{\mathrm{GR}} \neq \emptyset, \mathcal{S}^i_{\mathrm{GR}} \in \mathcal{S}_{\mathrm{CL}}, \mathcal{S}^j_{\mathrm{GR}} \in \mathcal{S}_{\mathrm{CL}}, \\ i \neq j, 0 \leq i < K, 0 \leq j < K\} . \end{aligned} \tag{4.32}$$

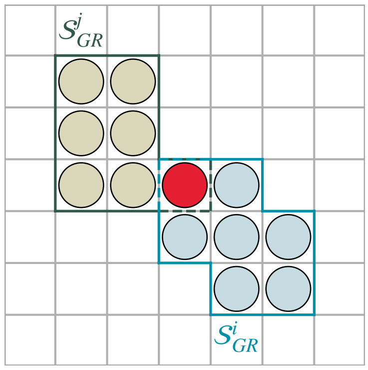


Figure 4.11: Two directly connected clusters. Cluster $\mathcal{S}^j_{\mathrm{GR}}$ is formed by the agents represented by beige circles and cluster $\mathcal{S}^i_{\mathrm{GR}}$ by the agents represented by blue circles. The agent represented by the red circle is part of both clusters. It does not fulfill the neighborhood criterion c^n_{CLN} (Eq. 4.28) itself and thus two individual clusters are formed that are connected via this agent.

Clusters can be either directly connected or connected via other clusters. We can determine whether all clusters are connected (i.e., there is a path from every cluster to every other cluster) by representing the set of clusters and the direct connections between clusters as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The set $\mathcal{V} = \{k : 0 \leq k < K\}$ of vertices represents the K clusters in $\mathcal{S}_{\mathrm{CL}}$ and $\mathcal{E}$ (Eq. 4.32) gives the direct connections between two clusters as edges. We test if the graph is connected using a traversal algorithm, such as Depth First Search, which sets connection criterion c_{connect} to 1 if the graph is connected and to 0 if the graph is not connected.

Loose grouping is the formation of several clusters that are all connected. Thus, we define criterion c^n_{LG} of agent n being classified as loose grouping (LG) as

$$c^n_{\mathrm{LG}} = \begin{cases} 1, & \begin{aligned} &\text{if } |\mathcal{S}_{\mathrm{CL}}| > 1 \land c_{\mathrm{connect}} = 1 \\ &\quad \land \exists \mathcal{S}_{\mathrm{GR}} \in \mathcal{S}_{\mathrm{CL}} : n \in \mathcal{S}_{\mathrm{GR}} \end{aligned} \\ 0, & \text{otherwise} \end{cases} . \tag{4.33}$$

In contrast, clustering is the formation of several clusters that are not connected, that is, there are at least two separate clusters. We define criterion c^n_{CL} of agent n to be classified as clustering (CL) as

$$c^n_{\mathrm{CL}} = \begin{cases} 1, & \begin{aligned} &\text{if } |\mathcal{S}_{\mathrm{CL}}| > 1 \land c_{\mathrm{connect}} = 0 \\ &\quad \land \exists \mathcal{S}_{\mathrm{GR}} \in \mathcal{S}_{\mathrm{CL}} : n \in \mathcal{S}_{\mathrm{GR}} \end{aligned} \\ 0, & \text{otherwise} \end{cases} . \tag{4.34}$$

Swirls (SW) are formed by four agents $\mathcal{S}^i_{\mathrm{SW}}$ that are positioned in a square with no free cells in between. Each agent's heading points to one of the other three agents whereby this neighbor's heading differs by ±90°, see Fig. 4.12. Thus, a stable structure is formed as intended forward movement of the four agents will be prevented. We define the set $\mathcal{S}^i_{\mathrm{SW}}$ of agents assembled into a swirl based on an agent $i \in \mathcal{S}_N$ with position $P_i = (x_i, y_i)$ and heading $H_i = (1, 0)$, that is, $P_i \in \mathcal{S}^i_{\mathrm{SW}}$. Three more agents are required for the formation of one swirl:

$$\begin{aligned} j \in \mathcal{S}_N, P_j = P_i + H_i,\ H_j = (0, \pm 1) \rightarrow j \in \mathcal{S}^i_{\mathrm{SW}}\,, \\ k \in \mathcal{S}_N, P_k = P_j + H_j,\ H_k = -H_i = (-1, 0) \rightarrow k \in \mathcal{S}^i_{\mathrm{SW}}\,, \\ m \in \mathcal{S}_N, P_m = P_k + H_k,\ H_m = -H_j \rightarrow m \in \mathcal{S}^i_{\mathrm{SW}}\,. \end{aligned} \tag{4.35}$$

These four agents are positioned in the center of a 4×4 segment of the grid. We allow maximally one more agent next to the four agents forming the swirl to be positioned in this segment. $\mathcal{S}^i_{4\times 4}$ gives all agents in the 4×4 grid segment as defined by

$$\begin{aligned} \mathcal{S}^i_{4\times 4} = \{n : &-1 \leq \min(x_{\min} - x_n, L - (x_{\min} - x_n)) \leq 2, \\ &-1 \leq \min(y_{\min} - y_n, L - (y_{\min} - y_n)) \leq 2, \\ &n \in \mathcal{S}_N\}\,, \end{aligned} \tag{4.36}$$

with leftmost agent position in the swirl $x_{\min} = x_i$ and lower agent position in the swirl

$$y_{\min} = \begin{cases} y_j, & \text{if } H_j = (0, 1) \\ y_m, & \text{otherwise} \end{cases}\,. \tag{4.37}$$

We define criterion c^n_{SW} of agent n to be classified as swirls (SW) as

$$c^n_{\mathrm{SW}} = \begin{cases} 1, & \text{if } \exists i \in \mathcal{S}_N : n \in \mathcal{S}^i_{\mathrm{SW}} \wedge |\mathcal{S}^i_{\mathrm{SW}}| = 4 \wedge |\mathcal{S}^i_{4\times 4}| \leq 5 \\ 0, & \text{otherwise} \end{cases}\,. \tag{4.38}$$

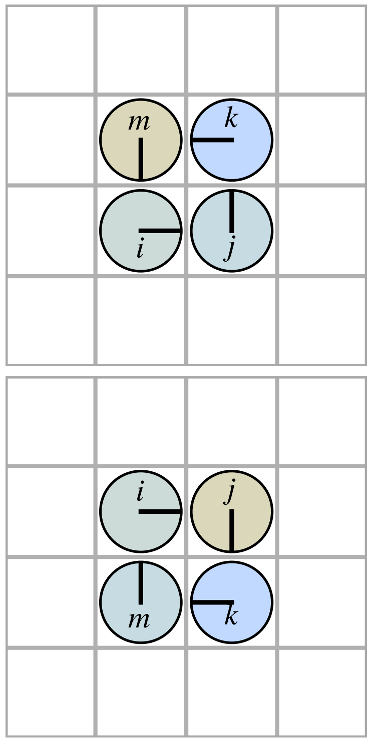


Figure 4.12: Two swirls on a 4×4 grid segment each. Agents are represented by circles, their color and the lines indicate their headings. i, j, k, m give the indices of the agents as used in Eq. 4.35.

4.2.3 Statistical Tests

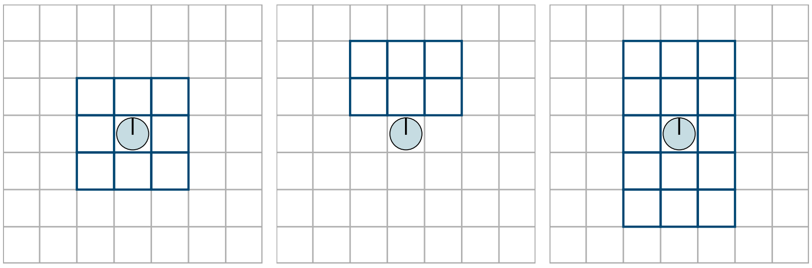
We test for statistically significant differences in fitness (Eq. 3.1), behavior distributions, and solution quality (Eq. 4.11) (i.e., dependent variables) in the different scenarios (i.e., independent variable).[7] Here, different scenarios are different test environments (e.g., varying swarm densities) or different approaches (e.g., novelty search, minimize surprise) that may affect the resulting fitness, solution quality, and behaviors. We use non-parametric tests (i.e., no normality assumption) as recommended for evolutionary robotics [351, 352]. Our data is unpaired, since we randomly initialize all evolutionary runs independently.

7: Dependent variables measure the effect caused by a change of the independent variable.

Fitness and solution quality are continuous values in the range of $[0, 1]$. We want to test for differences in the distributions of groups of this data (e.g., from different scenarios). The Mann-Whitney U test (hereafter abbreviated MW-U) [353] is a non-parametric test for unpaired data with the null hypothesis that two groups of at least ordinally scaled data are equal. The alternative hypothesis is that the two groups of data differ without specifying the direction (i.e., two-sided test). Alternatively, it can be performed as a one-sided test to detect if one group is statistically significantly greater or less than the other. The Kruskal-Wallis test[8] (hereafter abbreviated KW) [354] is the extension of the Mann-Whitney U test for several groups. As it only indicates if there is at least one group statistically significantly different to the others, post-hoc tests are needed to determine which pairs of groups are different. Here, we use the Mann-Whitney U test with Bonferroni correction [355] (hereafter abbreviated BC) that is commonly used [280] to govern the increasing likelihood of type I errors (i.e., the rejection of a true null hypothesis) when comparing multiple times. All of those tests are commonly used in evolutionary swarm robotics [280, 356–359].

8: The Kruskal-Wallis test is the non-parametric equivalent of a one-way ANOVA (analysis of variance).

The emergent behavior distributions (i.e., how often did we observe clustering, dispersion, etc.) can be represented as frequency or percentage distributions. For visualization purposes, we use percentage distributions to allow for easier visual comparison of scenarios. To study if behavior distributions differ between scenarios, we use Fisher's Exact test [360] that in contrast to Pearson's Chi-squared test can also be used when there are expected cell counts lower than five. Fisher's Exact test (hereafter abbreviated FE) is a non-parametric test for frequency distributions to determine if there is an association between two categorical variables. Here, we study if there is a difference in the behavior distributions for different scenarios or, in other words, if there is an association between scenario and resulting behaviors. The test can be used to compare pairwise and to compare several groups. In the latter case, we user Fisher's Exact test with Bonferroni correction as a post-hoc test.



(a) Moore neighborhood **(b)** six sensors **(c)** 14 sensors

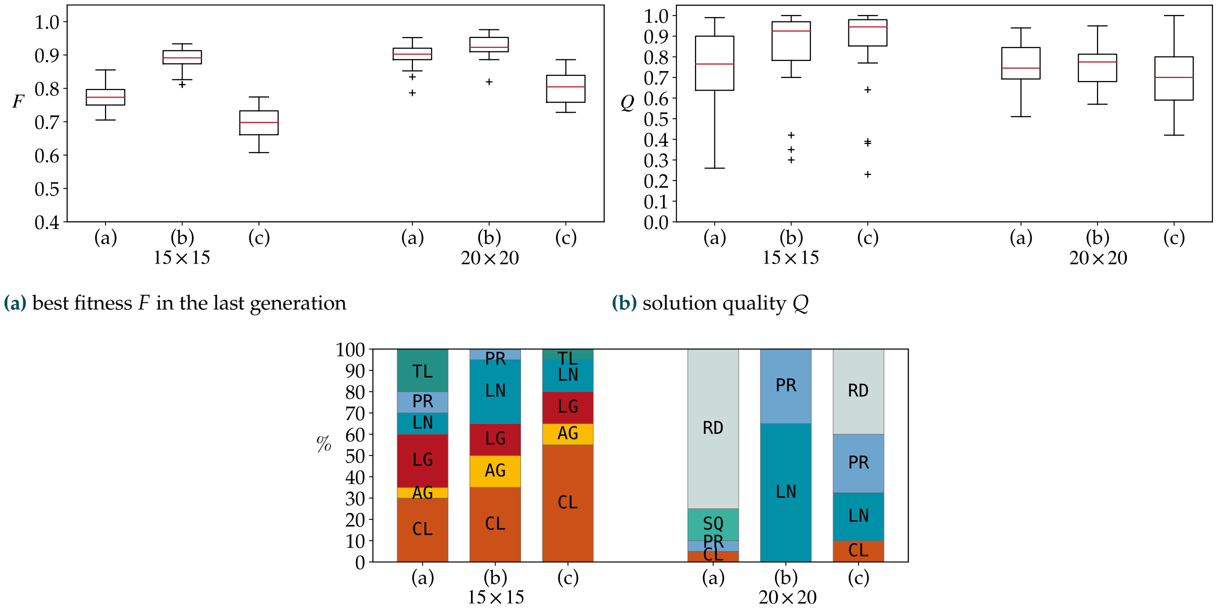
Figure 4.13: The three different sensor models for the self-assembly scenario that we evaluate in preliminary investigations. Blue circles represent agents, black lines indicate their headings.

4.3 Choice of Sensor Model

In this first study, we justify the choice of our sensor model that we have presented in Sec. 4.1 by comparing three different sensor models. All three sensor models use binary sensors that indicate whether the respective grid cell is occupied by an agent. We compare sensor model (a) that covers the agent's Moore neighborhood (Fig. 4.13a), sensor model (b) that covers six grid cells in front of the agent (Fig. 4.13b), and sensor model (c) that covers an agent's 14 surrounding grid cells (Fig. 4.13c). On the 15×15 and 20×20 torus grids (i.e., $L \in \{15, 20\}$), we evaluate each sensor model in 20 independent evolutionary runs and classify the resulting structures based on the metrics presented in Sec. 4.2.2. We compare best fitness F in the last generation of the evolutionary run (Eq. 3.1; Fig. 4.14a), solution quality Q of the formed structures (Eq. 4.11; Fig. 4.14b), and behavior distributions (Fig. 4.14c) of the three sensor models per grid size.

Fitness (Fig. 4.14a) is statistically significantly different for all three sensor models on both grid sizes (KW, $p < 0.05$). Sensor model (b) reaches significantly greater and sensor model (c) significantly less fitness than the respective other two sensor models (MW-U with BC, $p < 0.05$). We expected this ranking (i.e., (b) $>$ (a) $>$ (c)) of the sensor models in fitness, since each additional sensor value increases the overall difficulty of the prediction task. This increase in difficulty is likely to be non-linear, since each additional sensor value reduces the weight of each prediction in the overall fitness, also reducing the impact of each incorrectly predicted sensor value. In total, we find high fitness for all sensor models with a median fitness of at least 0.7 on the 15×15 grid and 0.8 on the 20×20 grid. Furthermore, there are no statistically significant differences in solution quality (Fig. 4.14b) of the best evolved individuals between the three sensor models on both grid sizes (KW, $p < 0.05$). We conclude that all three sensor models are suitable candidates regarding fitness and solution quality.

As we aim for the emergence of diverse collective behaviors using our minimize approach, we also compare the resulting behavior distributions, see Fig. 4.14c. On the 15×15 grid, we do not find statistically significant differences (FE, $p <$



(a) best fitness F in the last generation

(b) solution quality Q

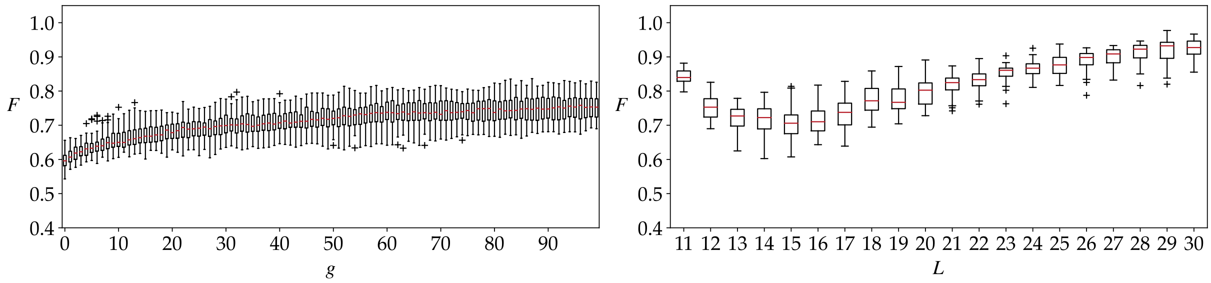
(c) behavior distributions

Figure 4.14: Fitness F (Eq. 3.1), solution quality Q (Eq. 4.11), and behavior distribution of the best evolved individuals of 20 independent evolutionary runs per grid size $L \in \{15, 20\}$ and sensor model in our study of sensor models for the self-assembly scenario. The behavior distributions give the percentage of resulting structures with clustering (CL), aggregation (AG), loose grouping (LG), lines (LN), pairs (PR), triangular lattices (TL), squares (SQ), and random dispersion (RD). Medians are indicated by the red bars in the box plots.

0.05). All three sensor models lead to the emergence of a variety of behaviors here. We find six different behaviors for sensor model (a) and five different behaviors for sensor models (b) and (c). Sensor model (b) did not lead to the emergence of triangular lattices, while we do not find pairs using sensor model (c). The emergent behavior distributions on the 20×20 grid are significantly different between the sensor models (FE, $p < 0.001$), see Fig. 4.14c. Sensor model (b) results solely in the formation of lines and pairs and has the lowest number of different behaviors. Both sensor models (a) and (c) lead to the emergence of four different structures. Sensor model (a) leads to the emergence of several behaviors that result in the formation of clusters, pairs, and squares, but the majority of behaviors leads to random dispersion. In contrast, sensor model (c) results in a more uniform distribution of clustering, lines, pairs, and random dispersion. We use sensor model (c) in all following experiments, since it enables us to evolve a variety of behaviors independent from swarm density.

4.4 Emergent Behaviors

In this section, we investigate the emergent behaviors over grid size when applying our minimize surprise approach to our self-assembly scenario (see Sec. 4.1). As we vary swarm density by changing grid size $L \times L$, it allows us to study the effects of swarm density on the emergence of structures (see



(a) best fitness F over generations g on the 12×12 grid **(b)** best fitness F over grid size L

Figure 4.15: Best fitness F (Eq. 3.1) of 50 independent minimize surprise runs over generations on the 12×12 grid ($L = 12$) and for the last generation per grid size $L \times L$ for the self-assembly scenario. Medians are indicated by the red bars [42, 343].

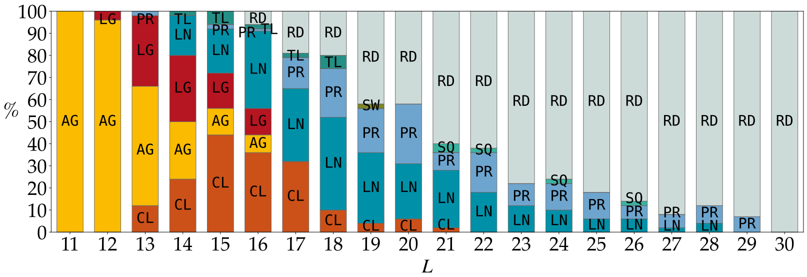


Figure 4.16: Behavior distributions, that is, percentage of resulting structures, in our self-assembly scenario for grid sizes $L \times L$ with $L \in [11 .. 30]$ with clustering (`CL`), aggregation (`AG`), loose grouping (`LG`), lines (`LN`), pairs (`PR`), triangular lattices (`TL`), squares (`SQ`), swirls (`SW`), and random dispersion (`RD`) [41, 42].

Fig. 4.4) [39, 42]. We do 50 independent evolutionary runs for each grid size $L \in [11 .. 30]$ and classify the structures formed by the best individuals in their evaluation's last time step T using the metrics presented in Sec. 4.2.2. We compare fitness F (Eq. 3.1), behavior distributions, solution quality Q (Eq. 4.11), temperature Θ (Eq. 4.4), agent movement M_N (Eq. 4.8), and intended agent movement I_N (Eq. 4.9) of the best evolved individuals. A video of emergent behaviors is online.[9]

9: `https://youtu.be/KWJIgPZd060`

Fitness Fitness F (Eq. 3.1), that is, prediction accuracy, is a measure of success for our minimize surprise approach. Fig. 4.15a visualizes the increase of best fitness of 50 independent evolutionary runs over generations on the 12×12 grid and is representative for all observed fitness curves. Fig. 4.15b shows best fitness in the last generation over grid size $L \in [11 .. 30]$. The median best fitness in the last generation is between 0.71 ($L = 15$) and 0.93 ($L = 29$). That means at least 71 % of the sensor values were predicted correctly by the evolved predictor networks. We find statistically significant differences in the best fitness of the various grid sizes (KW, $p < 0.001$). The prediction task is easiest for dense ($L \in \{11, 12\}$) and sparse ($L \in [18 .. 30]$) swarm densities, that is, we find high fitness values. For intermediate densities ($L \in [13 .. 17]$), we find statistically significantly lower fitness values (MW-U with BC, $p < 0.05$) and thus the prediction task is harder here.

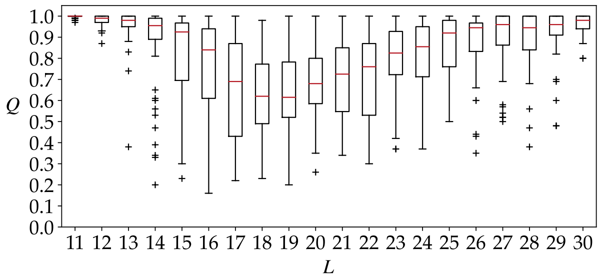


Figure 4.17: Solution quality Q (Eq. 4.11) of the best evolved individuals of 50 evolutionary runs per grid size L in our self-assembly scenario. Medians are indicated by the red bars [42, 343].

Behavior Distributions Fig. 4.16 shows the resulting behavior distributions, that is, the percentage of formed structures, over grid size. The behavior distributions for the different grid sizes are statistically significantly different and thus swarm density affects the emergence of behaviors (FE, $p < 0.01$). We find that this significant difference occurs between two behavior distributions with increasing difference in swarm density (FE with BC, $p < 0.05$), that is, neighboring distributions usually do not vary to a statistically significant degree. Grouping behaviors (i.e., aggregation, loose grouping, and clustering) prevail on small grid sizes as these structures form easily in high swarm densities. On the smallest grid sizes $L \in \{11, 12\}$, movement is barely possible and thus only aggregation and loose grouping can emerge. We note a shift in the distributions towards pairs, lines, and dispersion with increasing grid size. The decrease in swarm density allows for the formation of patterns that require agents to be distributed in space. For example, a swarm density of 0.5 or less is required for the formation of perfect line structures (i.e., the grid has to have twice the swarm size of grid cells). Triangular lattices, squares and swirls emerge rarely over all grid sizes, probably because they require exact positioning of four to nine agents. For instance, we find that 12 % of the runs on the 20×20 grid show a visible tendency towards the formation of triangular lattices or squares but are classified as random dispersion. We find the most diverse behavior distributions with seven and six out of nine possible structures on the 16×16 grid and 15×15 grid, respectively.

Solution Quality We compare the solution quality Q (Eq. 4.11), that is, the percentage of agents forming the dominant structure, over grid size next (see Fig. 4.17). The median solution quality ranges from 0.62 ($L = 19$) to 1.0 ($L = 11$). That means at least 62 agents assemble into the dominant structure. We find statistically significant differences for the different swarm densities (KW, $p < 0.001$). As expected, solution quality is highest on the 11×11 grid as the great majority of agents will be aggregated by default due to the high swarm density. In general, sparse and dense settings lead to higher solution quality than intermediate densities.

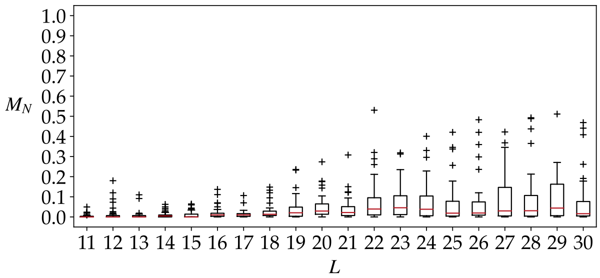
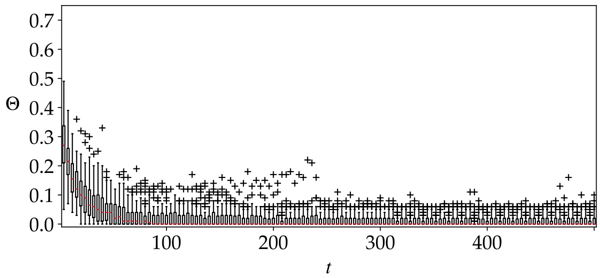


Figure 4.18: Agent movement M_N (Eq. 4.8) of the best individuals of 50 evolutionary runs per grid size $L \times L$ in our self-assembly scenario. Medians are indicated by the red bars.

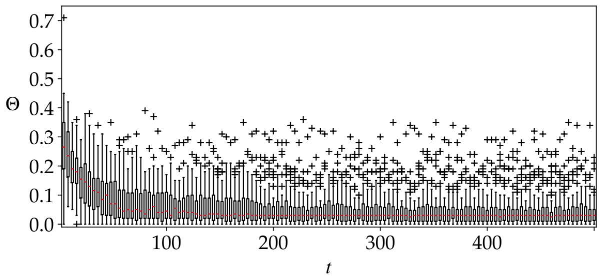
Agent Movement Agent movement M_N (Eq. 4.8) is an indicator whether the swarm self-assembles into a stable structure. We find low agent movement M for all grid sizes with median values ranging from zero ($L \in \{11, 12, 13, 15\}$) to 0.05 ($L = 23$), see Fig. 4.18. Thus, agents stay on their current grid cells either by rotation or by prevented attempts to move forward because their targeted grid cells were already occupied by other agents (see Sec. 4.1). Consequently, we find that stable structures form.[10]

10: This allows us to classify the resulting behaviors based on the agent positions in the last time step T of the run.

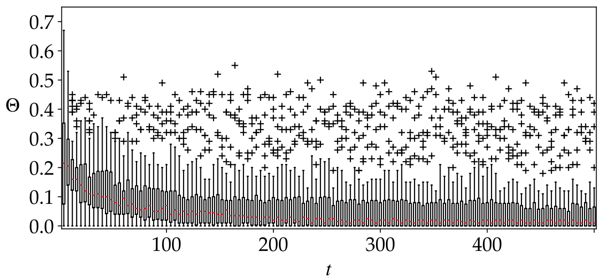
Temperature The temperature Θ (Eq. 4.4) measured over runtime illustrates how the systems cool down over time, that is, how quickly the agents form stable structures, see Fig. 4.19. After around 150 time steps almost all agents are staying stopped at the latest and thus agents quickly assembly into a structure. We measure median temperature values between zero ($L = \{11, 12, 13, 14, 15, 16, 30\}$) and 0.04 ($L = \{22, 23\}$) in the last time step. This means that up to 4 % of the agents moved one grid cell forward in that time step. We find that variability and outliers increase with increasing grid size, see Fig. 4.19. To elaborate, we find low temperature values (i.e., low variability) and outliers with still rather low values for small grids (Fig. 4.19a). With increasing grid size, we find more variability in the temperature values and outliers with higher values (Figs. 4.19b and 4.19c). Most of these outliers, and especially those with high temperature values, are caused by random dispersion behaviors, which can be seen when measuring temperature separately for random dispersion behaviors and all other behaviors. As a representative example for grid sizes $L \geq 16$, we show the measured temperature separately for random dispersion behaviors and all other behaviors for the 20×20 grid, see Fig. 4.20. The temperature converges to low values quickly and there are only few outliers for all behaviors except for random dispersion, see Fig. 4.20a. For the random dispersion behaviors, the temperature decreases slightly in the beginning, but we still find high variability in temperature until the end of the run (see Fig. 4.20b). A reason may be that behaviors leading to random dispersion can reach high solution quality with different approaches. Agents can either stay on a grid cell as soon as they have no (or few) neighbors by rotating on the spot or they can constantly move around and avoid neighbors



(a) 15 × 15 grid

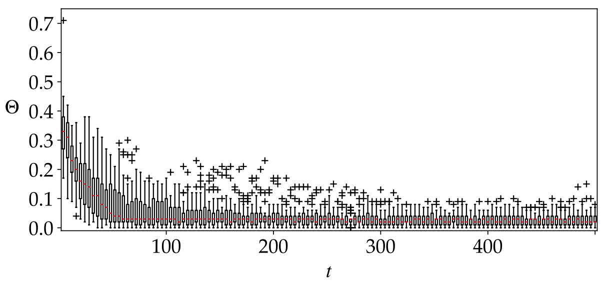


(b) 20 × 20 grid

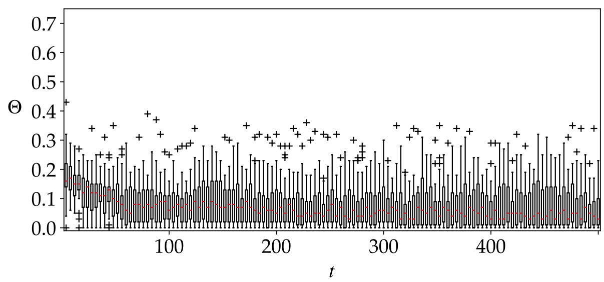


(c) 25 × 25 grid

Figure 4.19: Temperature Θ (Eq. 4.4) measured over runtime of 50 independent runs for grid sizes $L \in \{15, 20, 25\}$ in our self-assembly scenario as representative examples. Medians are indicated by red bars. Only data of every fourth time step is plotted for a clearer visualization.



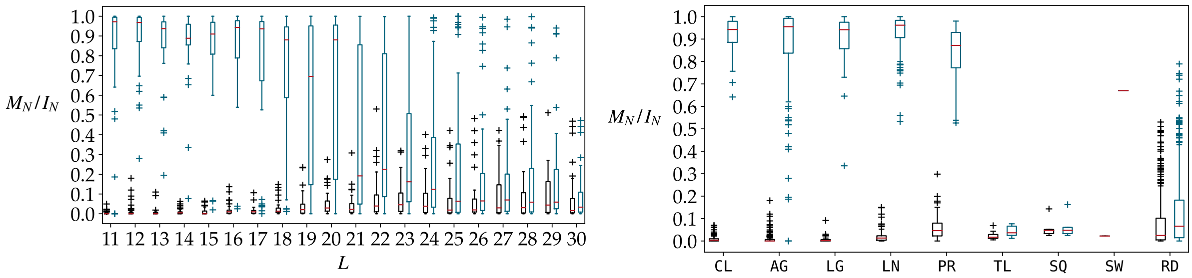
(a) all behaviors except for random dispersion



(b) random dispersion behaviors

Figure 4.20: Temperature Θ (Eq. 4.4) over runtime of 50 independent runs on the 20 × 20 grid in our self-assembly scenario measured separately for random dispersion behaviors and all other behaviors. Medians are indicated by red bars. Only data of every fourth time step is plotted for a clearer visualization.

that are detected by the front outer sensors (i.e., s_3, s_4, s_5; see Fig. 4.1). All other structures require exact positioning of several agents and thus staying on the current grid cell is essential to keep the structure intact. On grid sizes $L < 16$, no random dispersion behaviors emerge and we find less outliers in the temperature curves.



(a) per grid size L

(b) per pattern

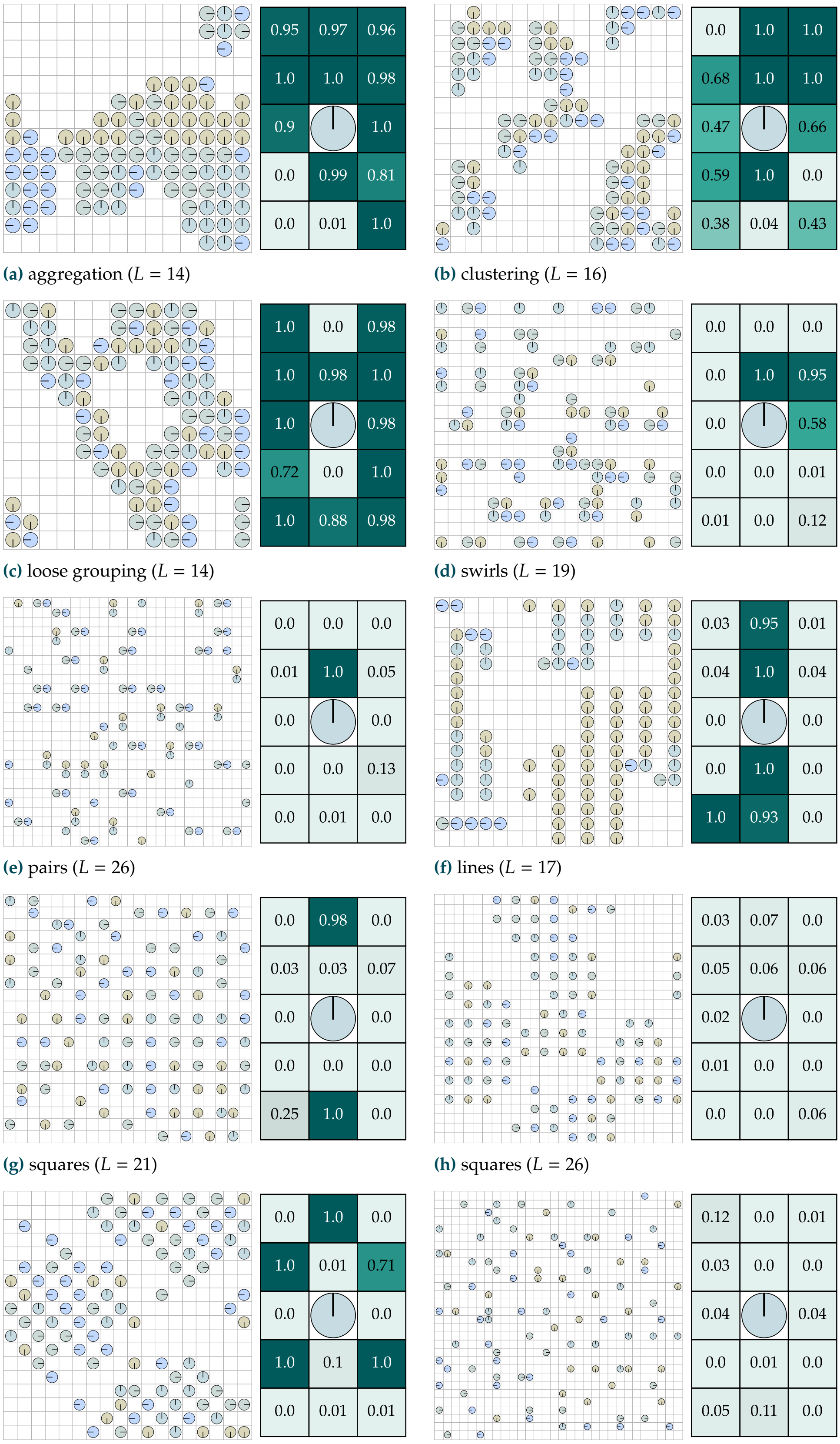
Figure 4.21: Agent movement M_N (Eq. 4.8; black boxes, left) and intended agent movement I_N (Eq. 4.9; blue boxes, right) of the best individuals of 50 evolutionary runs per grid size $L \times L$ and per pattern with clustering (CL), aggregation (AG), loose grouping (LG), lines (LN), pairs (PR), triangular lattices (TL), squares (SQ), swirls (SW), and random dispersion (RD) in our self-assembly scenario. Medians are indicated by the red bars.

Intended Agent Movement Based on the intended agent movement I_N (Eq. 4.9), we can determine whether rotation or prevented forward movement allows agents to remain on their current grid cells. The mean intended agent movement I_N ranges from 0.03 ($L = 30$) to 0.97 ($L = 11$), see Fig. 4.21a. On the smaller grids, intended agent movement I_N is high, which means that prevented forward movement allows agents to stay on their grid cells. We find a drop to lower intended agent movement I_N between grid sizes $L = 20$ and $L = 21$, that is, agents mainly rotate on large grids.

The study of agent movement M_N and intended agent movement I_N per pattern (Fig. 4.21b) enables us to identify differences in the behavioral characteristics of our nine patterns and shows the correlation between the behavior distributions and (intended) agent movement. Aggregation, clustering, swirls, and loose grouping lead to low agent movement M_N and high intended agent movement I_N, that is, agents self-assemble into stable grouping structures by exploiting that forward movement is prevented when the targeted grid cell is already occupied. These grouping behaviors can also form stable structures by constantly rotating, which seems to happen in rare cases (e.g., see the outliers for aggregation). As expected, lines and pairs lead also to low agent movement M_N and high intended agent movement I_N. Consequently, agents in line and pair structures keep their positions and headings by intending to move straight as the grid cell in front is occupied. Here, the correct positioning of the agents ensures a stable structure and constant sensor values. Triangular lattices and squares lead to low agent movement M_N and low intended agent movement I_N, that is, agents rotate when forming these structures. Both patterns are rotation symmetric and thus agents can keep their sensor values constant by turning. Random dispersion also leads to low agent movement M_N and low intended agent movement I_N, but we find a higher variability. As mentioned before, agents can randomly disperse either by rotating on the spot or by moving and avoiding other agents. The latter can lead to high agent movement M_N. High intended agent movement I_N can occur if some agents on the grid are grouped or assembled into lines and pairs, but the swarm members are mostly randomly dispersed. With the shift in the behavior distributions from grouping behaviors on the smaller grids ($L \in [11\,..\,13]$) over diverse behavior distributions on intermediate grids ($L \in [14\,..\,18]$) to mainly dispersion behaviors on large grids ($L \in [19\,..\,30]$), we also find the shift from high intended agent movement to low intended agent movement.

Predictions To reach both high solution quality and fitness (i.e., prediction accuracy), a high percentage of agents needs to be assembled in a pattern and the predictions have to match this formed structure closely. Therefore, we assess the average sensor predictions during the evolutionary run of the best individuals, which correspond to the agent's anticipated environment, and compare them to the formed structures. Fig. 4.22 visualizes the mean predictions for all nine patterns. As expected, in aggregation (Fig. 4.22a), clustering (Fig. 4.22e), and loose grouping (Fig. 4.22c), agents predict that the majority of their adjacent grid cells are occupied. Predictions of swirls (Fig. 4.22d) contain three neighbors, whereby one neighbor is directly in front of the agent and two more to their right (or left). Agents in pair structures (Fig. 4.22e) only expect to sense a neighbor on the grid cell directly in front of them, while agents in line structures (Fig. 4.22f) predict all four grid cells directly in front and behind them to be occupied. The sensor predictions in triangular lattices follow this intuitive scheme, too, and match almost completely the visually observed agent structure as shown in Fig. 4.22i. In squares, we expect agents to predict neighbors two grid cells directly in front and behind them, as shown in Fig. 4.22h. However, we find that most formed square structures lead to the prediction of no neighbors at all, as for random dispersion, see Figs. 4.22g and 4.22j. This is probably because we have a total of 14 sensors. Small deviations between mean predictions and the sensor values induced by the formed structures have little impact on prediction accuracy. While predicting two empty grid cells directly in front or behind the agent does not match the formed square pattern, prediction accuracy is only reduced by $\frac{2}{14}$th. Thus, high fitness is still reached. The values close to 1.0 in the sensor prediction plots indicate that agents have rather constant sensor predictions over the full runtime, which, in combination with the found high solution quality Q (see Fig. 4.17), results in the observed high fitness (see Fig. 4.15). Overall, as expected, we find that the formed patterns and the mean sensor predictions coincide closely.

In summary, our minimize surprise approach leads to diverse and non-trivial collective behaviors that form stable structures despite the task-independent selective pressure to minimize the prediction error. We find high fitness (i.e., prediction accuracy) and solution quality for all behaviors



(a) aggregation ($L = 14$)
(b) clustering ($L = 16$)
(c) loose grouping ($L = 14$)
(d) swirls ($L = 19$)
(e) pairs ($L = 26$)
(f) lines ($L = 17$)
(g) squares ($L = 21$)
(h) squares ($L = 26$)
(i) triangular lattice ($L = 18$)
(j) random dispersion ($L = 30$)

Figure 4.22: Resulting structures (left) and mean sensor predictions (right; 1: always predict occupied, 0: always predict non-occupied) of the agents for all nine patterns in our self-assembly scenario. The structures in (a)-(f) are static, that is, agents stay on their grid cell by exploiting the prevention of forward movement when a targeted grid cell is occupied. By contrast, the structures in (g)-(j) are dynamic, that is, agents either rotate on the spot because the structures are rotation symmetric (i.e., structures in (g) - (i)) or move and avoid other agents (i.e., structure in (j)). Agents are represented by circles, their color and the lines give their headings.

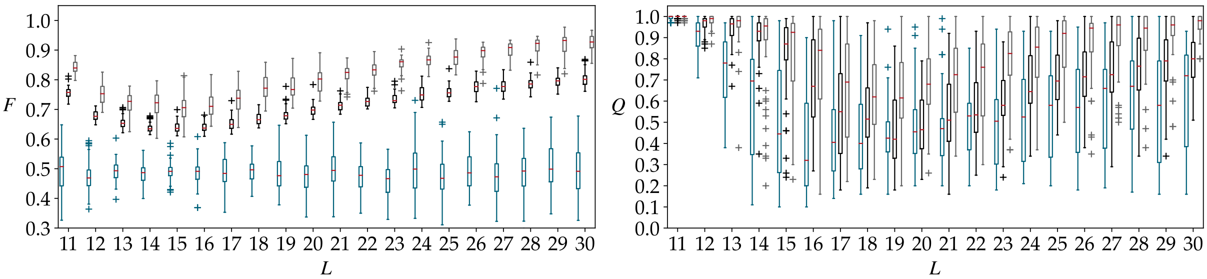
and, accordingly, sensor predictions that closely match the formed patterns. Intermediate swarm densities complicate the prediction task and we find lower fitness and solution quality but higher behavioral diversity on these grid sizes than for high and low swarm densities. In total, swarm density influences the emergence of behaviors.

4.5 Effectiveness of the Approach

In the previous section, we have shown that our minimize surprise approach leads to a variety of behaviors across several independent evolutionary runs. Since minimize surprise is not working towards solutions for a given task but rather implements an exploratory search, we want to test its effectiveness in finding relevant and interesting behaviors at all [42]. For this purpose, we compare the best evolved individuals of the evolutionary runs with minimize surprise to randomly generated actor-predictor pairs. We generate two sets of 50 random ANN pairs per grid side length $L \in [11..30]$: (i) by creating 50 random individuals, and (ii) by creating 50 times a population of 5,000 random individuals and selecting the best individual based on prediction success (Eq. 3.1; hereafter referred to as random individuals with selection). In the latter case, the number of evaluated random individuals equals the number of evaluations in minimize surprise (i.e., 50 independent evolutionary runs with 50 individuals $\times$ 100 generations = 5,000 evaluations each). We compare best fitness F (Eq. 3.1; see Fig. 4.23a), solution quality Q (Eq. 4.11; see Fig. 4.23b), and behavior distributions (see Fig. 4.24) of the best evolved individuals of our minimize surprise approach, the random individuals, and the random individuals with selection per grid size $L \times L$.

First, we compare fitness F (Eq. 3.1) of the best evolved individuals and the randomly generated ANN pairs, see Fig. 4.23a. For all grid sizes, the random individuals have statistically significantly less fitness than the random individuals with selection and the best evolved individuals of our evolutionary runs with minimize surprise. As expected, the random individuals with selection reach statistically significantly less fitness than the best evolved individuals (MW-U with BC, $p < 0.01$). Consequently, we find that our minimize surprise approach successfully improves fitness or, put differently, prediction accuracy, over generations and outperforms pure random search.

Next, we compare solution quality Q (Eq. 4.11), see Fig. 4.23b. We find that the random individuals reach statistically significantly lower solution quality than the random individuals with selection on all grid sizes except for $L \in \{11, 17, 18, 19, 20, 21, 22, 28\}$ and than the best evolved individuals of our minimize surprise approach on all grid sizes except for $L = 11$ (MW-U with BC, $p < 0.05$). Compared



(a) fitness F

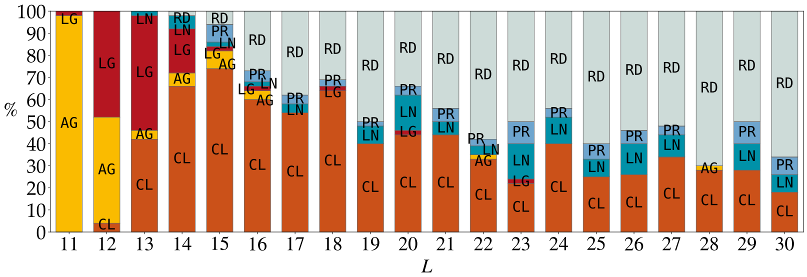
(b) solution quality Q

Figure 4.23: Fitness F (Eq. 3.1) and solution quality Q (Eq. 4.11) of 50 random individuals (blue boxes, left), 50 random individuals with selection (black boxes, middle) and 50 best evolved individuals of our minimize surprise approach (gray boxes, right) in our self-assembly scenario per grid size $L \times L$, $L \in [11..30]$. Medians are indicated by the red bars [42, 343].

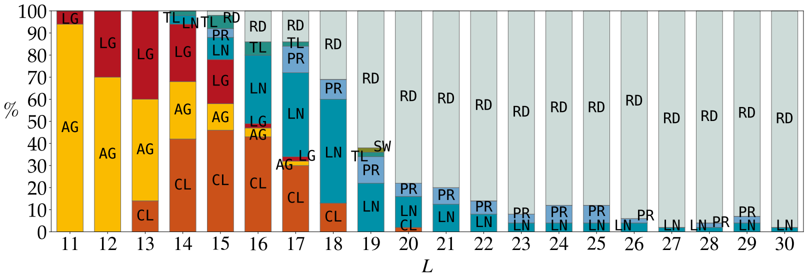
to the random individuals with selection, the best evolved individuals of minimize surprise reach significantly better solution quality on grid sizes of 19×19 or larger, while we do not find significant differences on smaller grid sizes. Thus, the ANN pairs evolved with minimize surprise outperform pure random search on the majority of grid sizes also regarding solution quality, that is, the best evolved individuals lead to a higher percentage of swarm members assembling into the dominant structure.

Furthermore, we compare the behavior distributions, see Fig. 4.24. The behavior distributions of the random individuals and the random individuals with selection differ significantly for all grid sizes except for $L \in \{11, 12, 14\}$ (FE with BC, $p < 0.05$). We find statistically significant differences for the behavior distributions of the best evolved individuals of our minimize surprise approach and the random individuals for all grid sizes except for $L = 11$. In contrast, only two out of the 20 tested grid sizes (i.e., $L \in \{12, 20\}$) lead to significant differences in the behavior distributions of the random individuals with selection and the best evolved individuals.

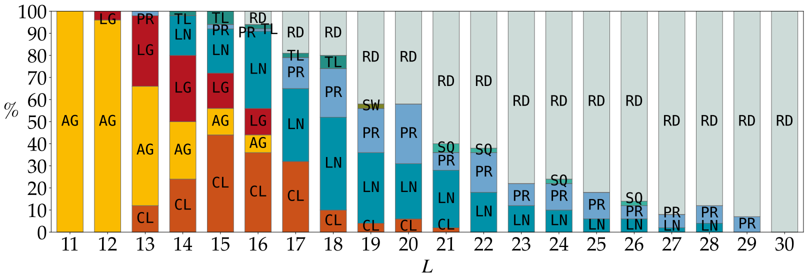
Due to the found differences in the behavior distributions, we want to investigate if some of the behaviors are found more or less likely in the evolutionary runs using our minimize surprise approach than in the case of generating the ANN pairs randomly by either of the two presented approaches. This allows us to study if evolution's selection and improvement over generations influences the resulting behaviors in general. We compare the frequency of each pattern over all grid sizes $L \in [11..30]$ for the best evolved individuals, the random individuals, and the random individuals with selection as visualized in Fig. 4.25 (FE with BC, $p < 0.05$). Clustering is most likely for the random individuals and similar likely for the random individuals with selection and the best evolved individuals of minimize surprise. Aggregation is more likely for the random individuals with selection and the best evolved individuals than for the random individuals. By contrast, loose grouping is equally likely in



(a) random individuals



(b) random individuals with selection



(c) best evolved individuals of minimize surprise (reprint of Fig. 4.16 for easier comparison)

Figure 4.24: Behavior distributions, that is, percentage of structures formed by 50 random individuals, 50 random individuals with selection, and the 50 best evolved individuals of our minimize surprise approach in our self-assembly scenario with clustering (`CL`), aggregation (`AG`), loose grouping (`LG`), swirls (`SW`), lines (`LN`), pairs (`PR`), triangular lattices (`TL`), squares (`SQ`), and random dispersion (`RD`) per grid size $L \times L, L \in [11 .. 30]$ [42, 343].

all three cases. Lines and pairs are most likely for the best evolved individuals of minimize surprise. These behaviors require correct positioning to reach high fitness and solution quality (see Sec. 4.2). Thus, lines and pairs seem to rely on the selection and improvement over generations of the evolutionary process. Triangular lattices, squares, and swirls are generally rarely found. Nevertheless, triangular lattices are more likely for the random individuals with selection and the best evolved individuals of minimize surprise than for the random individuals. Squares are most likely for the best evolved individuals while swirls are equally likely for all three cases. Last but not least, random dispersion is most likely for the random individuals with selection and least likely for the random individuals.

Overall, this indicates that evolution successfully enables the adaptation of actors and predictors. Actor outputs lead to behaviors that are correctly predicted by the predictors and predictors are adapted to behaviors to optimize prediction accuracy. Complex behaviors (e.g., pairs), that are rarely generated in the random runs, are selected and improved over generations to reach high fitness and solution quality. While randomly generating individuals leads to similar or

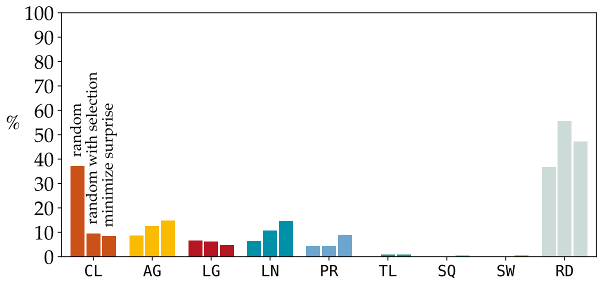


Figure 4.25: Percentage of clustering (CL), aggregation (AG), loose grouping (LG), lines (LN), pairs (PR), triangular lattices (TL), squares (SQ), swirls (SQ), and random dispersion (RD) formed by 50 random individuals (left), 50 random individuals with selection (middle), and the 50 best evolved individuals of our minimize surprise approach (right) in our self-assembly scenario over all grid sizes $L \times L, L \in [11 .. 30]$.

higher behavioral diversity than evolution for low swarm densities, such as on grid size $L = 30$, solution quality is significantly lower for randomly generated ANN pairs, that is, less agents are assembled into the structure. Thus, we conclude that minimize surprise is an effective approach that outperforms random search.

4.6 Discussion and Conclusion

In this chapter, we have shown that we can successfully evolve diverse swarm behaviors using our minimize surprise approach despite the task-independent fitness function rewarding prediction accuracy (Eq. 3.1). To be able to analyze the results in depth, we restricted us to a simple grid-based experimental setup aiming for the emergence of self-assembly behaviors. We studied the influence of the sensor model on the emergence of behaviors and showed that it affects the behavioral diversity. Thus, a careful configuration of the agents and the environment allows to bias evolution. Furthermore, we found that the emergent behaviors depend on swarm density. High swarm densities lead mainly to grouping behaviors (aggregation, clustering, loose grouping) while low densities lead mostly to random dispersion. The prediction task is hardest for intermediate densities, which potentially causes the wide range of behaviors there (lines, pairs, clustering, triangular lattices, etc.). As expected, the agents' predictions match the formed structures closely and can serve as an indicator for the assembled pattern. Last but not least, we have proven that our minimize surprise is effective by comparing it to random search. In the next chapter, we study the robustness against sensor noise and damage, the scalability in swarm density, and the diversity of the best evolved individuals in more detail.

In-Depth Analysis Using the Self-Assembly Scenario 5

Chapter Contents

In this chapter, we present an in-depth study of minimize surprise using our self-assembly scenario. We study...

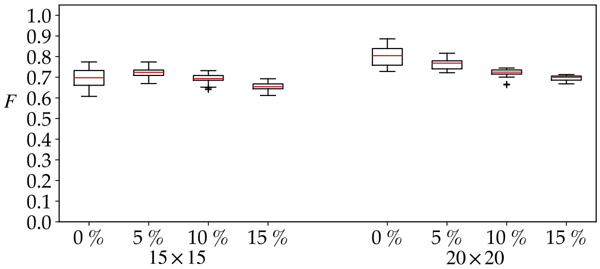
- *Sec. 5.1:* robustness against sensors noise and damage,
- *Sec. 5.2:* scalability with swarm density,
- *Sec. 5.3:* behavioral diversity in comparison to novelty search and MAP-Elites,
- *Sec. 5.4:* the influence of hyperparameters, and
- *Sec. 5.5:* draw a conclusion.

Parts of this chapter are based on [39–43].

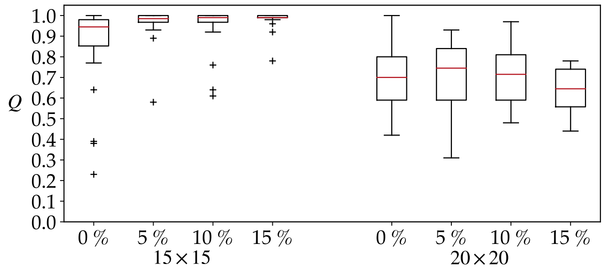
In the previous chapter, we have shown that minimize surprise results in the emergence of diverse behaviors and outperforms pure random search. In this chapter, we analyze the evolution of swarm behaviors with minimize surprise in our self-assembly scenario in more detail. Our in-depth study has four parts: First, we study the robustness of the evolutionary process against sensor noise and the robustness of the emergent behaviors against damage to the self-assembled structures. Second, we investigate the scalability of the evolved self-assembly behaviors with swarm density. Third, we compare our standard minimize surprise approach with novelty search [24] as an example for divergent search algorithms and MAP-Elites [44] as an example for quality-diversity algorithms (research question Q3, Sec. 1.2). For this purpose, we introduce task-independent novelty search and MAP-Elites variants. Last but not least, we study the influence of the hyperparameters on the performance of the evolutionary algorithm. Together, these four studies provide a detailed view on the strengths and weaknesses of our minimize surprise approach and show the robustness, scalability, and diversity of the emergent behaviors (research question Q1, Sec. 1.2).

5.1 Robustness

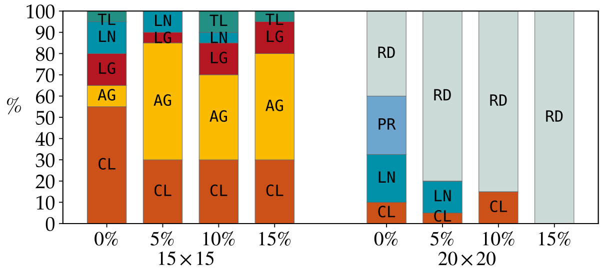
Our initial setup for the self-assembly scenario is fully deterministic resulting in a variety of behaviors, see Ch. 4. But noise may impact the emergent behavioral diversity, or emergent self-assembly behaviors could be vulnerable to external disturbances. Thus, we study the robustness of our minimize surprise approach against sensor noise and the resilience of the emergent behaviors against damage to the initially formed structure here [39, 42].



(a) best fitness F in the last generation



(b) solution quality Q



(c) behavior distributions

Figure 5.1: Fitness F (Eq. 3.1), solution quality Q (Eq. 4.11), and behavior distributions of the best evolved individuals of 20 independent evolutionary runs with minimize surprise per sensor noise level in $\{0\ \%, 5\ \%, 10\ \%, 15\ \%\}$ and grid size $L \times L, L \in \{15, 20\}$ in our self-assembly scenario. Behavior distributions give the percentage of resulting structures with clustering (CL), aggregation (AG), loose grouping (LG), lines (LN), pairs (PR), triangular lattices (TL), and random dispersion (RD). Medians are indicated by the red bars in the box plots.

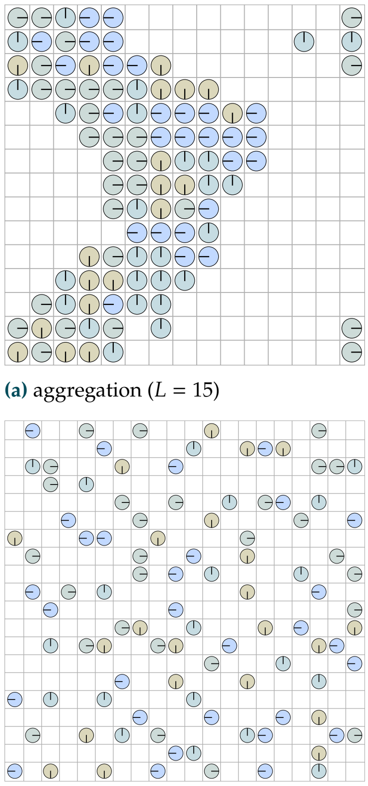
5.1.1 Sensor Noise

As a first step towards more realistic environments as experienced by physical robots, we introduce sensor noise into our otherwise fully deterministic simulation environment (see Sec. 4.1). We flip each binary sensor value with a probability of 5 %, 10 % or 15 %. As examples for denser and sparser swarm density settings, we restrict ourselves to grid sizes 15×15 and 20×20. We do 20 independent evolutionary runs per sensor noise level and arena size, and compare the results with the results of our deterministic setup (i.e., 0 % sensor noise).

Fig. 5.1a shows the median best fitness F (Eq. 3.1) in the last generation per sensor noise level and grid size. We find a median best fitness of at least 0.65 on the 15×15 grid and of at least 0.7 on the 20×20 grid. On both grid sizes, best fitness is statistically significantly different for the four sensor noise levels (KW, $p < 0.01$). On the 15×15 grid, we find significantly lower fitness for 15 % sensor noise (MW-U with BC, $p < 0.05$). On the 20×20 grid, fitness decreases significantly with increasing sensor noise. This is likely because task complexity increases with non-determinism, since noise is inherently unpredictable.

We find median solution qualities Q (Eq. 4.11) of at least 0.84 on the 15×15 grid and of at least 0.63 on the 20×20 grid, see Fig. 5.1b. Only the formed structures on the 15×15 grid lead to statistically significant differences in solution quality in the different sensor noise levels (KW, $p < 0.01$). Solution quality is significantly better for 15 % sensor noise than for 0 % sensor noise (MW-U with BC, $p < 0.01$), which may be because the greatest amount of grouping behaviors emerges in this setup. The greater amount of grouping behaviors probably also causes that solution quality is generally higher for the 15×15 grid than for the 20×20 grid. The behaviors found on the 20×20 grid often require the exact positioning of several agents (e.g., lines and pairs) or have strict criteria regarding the maximally allowed number of neighbors (e.g., random dispersion; see Sec. 4.2), potentially complicating the achievement of high solution quality compared to the rather weak criteria for grouping.

The behavior distributions for the four sensor noise levels are only statistically significantly different on the 20×20 grid (FE, $p < 0.001$), see Fig. 5.1c. Behavioral diversity decreases with increasing sensor noise as more robust structures tend to dominate in non-deterministic environments. Robust structures do not rely on the exact positioning and heading of agents and cannot be destroyed by the behavior of an individual agent. Such robust structures are aggregation (Fig. 5.2a), clustering, loose grouping, and random dispersion (Fig. 5.2b). In aggregation, clustering, and loose grouping, most agents are surrounded by other agents and cannot leave the structure quickly in any event, including false sensor readings. The more agents are in a cluster, the harder it is to dissolve it. Such grouping behaviors are already favored on the 15×15 grid for 0 % sensor noise due to the high swarm density. Consequently, behavior distributions do not change significantly with increasing sensor noise on the 15×15 grid. Random dispersion is also robust, since agents rarely sense neighbors and a few false sensor readings have only minor effects on the overall fitness. In most cases, a move forward or turn caused by a false sensor reading will not lead to the formation of a different structure or a big change in the sensor values. We find that random dispersion is favored on the 20×20 grid with increasing sensor noise, which is caused by the lower swarm density compared to the 15×15 grid.



(a) aggregation ($L = 15$)

(b) random disperison ($L = 20$)

Figure 5.2: Examples of robust structures forming in our self-assembly scenario with 15 % sensor noise. Agents are represented by circles, their color and the lines give their headings.

Triangular lattices seem to emerge rather independently of sensor noise, which may be due to variation in the experiments. The formed structure is rotation symmetric as in all dispersion behaviors (i.e., triangular lattices, random dispersion, and squares) and agents turn constantly to stay on their grid cell. Here, sensor values might already change slightly with the change of heading when turning, since perfectly assembled triangular lattices are quite sophisticated requiring the correct positioning of nine agents on 21 grid cells (see Sec. 4.2.2). Thus, the behaviors are maybe always evolved to withstand small variations in sensor readings, such as those additionally caused by sensor noise in this scenario.

Less robust to disturbances are lines and pairs, since they rely on the correct positioning and heading of at least three agents.

Algorithm 1: Remove agents from rectangular area

Input: set $\mathcal{S}_N$ of all agents with positions $P_n = (x_n, y_n)$,
rectangular area defined by $(x_{\min}, y_{\min})$ and $(x_{\max}, y_{\max})$
Output: set $\mathcal{S}_N$ reduced by agents in specified rectangular area

for $n \in \mathcal{S}_N$ **do**
  if $x_{min} \leq x_n \leq x_{max}$ *and* $y_{min} \leq y_n \leq y_{max}$ **then**
    $\mathcal{S}_N \leftarrow \mathcal{S}_N \backslash \{n\}$

Algorithm 2: Reposition agents outside of rectangular area

Input: set $\mathcal{S}_N$ of all agents with positions $P_n = (x_n, y_n)$ and
headings $H_n = (h_n^x, h_n^y)$,
rectangular area defined by $(x_{\min}, y_{\min})$ and $(x_{\max}, y_{\max})$
Output: set $\mathcal{S}_N$ with all agents being positioned outside specified rectangular area

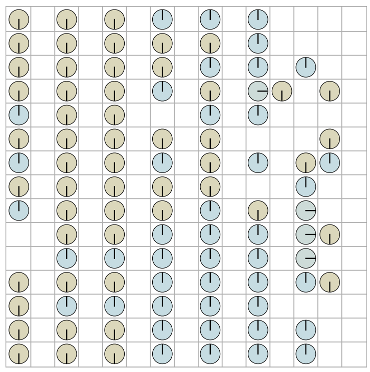
for $n \in \mathcal{S}_N$ **do**
  while $x_{min} \leq x_n \leq x_{max}$ *and* $y_{min} \leq y_n \leq y_{max}$ **do**
    generate new random agent position P_{new} on an unoccupied grid cell
    generate new random heading H_{new}
    $P_n \leftarrow P_{\text{new}}$
    $H_n \leftarrow H_{\text{new}}$

Agents may turn based on false negative sensor readings and immediately destroy the structure formation.

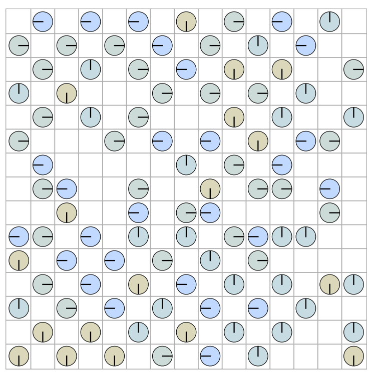
In summary, our minimize surprise approach is robust to sensor noise and even adapts by selecting for swarm behaviors that are robust to sensor errors. Thus, the approach still reaches high fitness even with high sensor noise levels. Still, we find a dependence of the emergent behavior distributions on swarm density.

5.1.2 Damage and Repair

In our self-assembly scenario no external disturbances exist while emergent behaviors let agents assemble into structures. Thus, we test in the next step if the best evolved individuals are robust against damage to the formed structure. As representative examples, we use two individuals leading to the formation of lines, see Fig. 5.3a, and triangular lattices, see Fig. 5.3b. First, we show that the structures form independent of the initial agent poses by rerunning the ANN pairs with new random starting poses in 20 independent runs of 500 time steps each (hereafter referred to as reruns). Afterwards, we damage the initially formed structures (see Fig. 5.3) in two ways: In the first variant, we remove agents from a rectangular area of the initially assembled structure completely, which leads to a decrease in swarm density, see Alg. 1. In the second variant, we uniformly randomly reposition all agents from a rectangular area of the initially



(a) lines ($L = 15$)



(b) triangular lattices ($L = 15$)

Figure 5.3: Initially formed lines and triangular lattices by the self-assembly behaviors used to study the robustness of the emergent behaviors against damage to the formed structure [38, 39].

assembled structure outside this area, see Alg. 2. We remove or reposition agents from three different areas in both initial structures. The respective actor-predictor pair is evaluated for another 500 time steps after damage (hereafter referred to as repair runs). We do one run when removing agents for variant one, since our completely deterministic simulation environment will lead to the same result in each repetition, and 20 independent runs when repositioning agents for variant two. A video illustrating our experiments is online.[1]

1: https://youtu.be/KWJIgPZd060

Metrics

We measure fitness F (Eq. 3.1) over the full evaluation length and solution quality Q (Eq. 4.11) at the start (i.e., $t = 0$) and the end (i.e., $t = T$) of an evaluation. As an additional metric, we introduce similarity S to the initial structure measuring the amount of agents with equal poses in the last time step T of the initial and the repair run. We define similarity S to the initial structure as

$$S(\mathcal{S}_{\text{repair}}, \mathcal{S}_{\text{initial}}) = \frac{1}{N} \sum_{\mathcal{U}_i \in \mathcal{S}_{\text{repair}}} \text{match}(\mathcal{U}_i)\,, \tag{5.1}$$

with lists of final agent poses $\mathcal{S}_{\text{repair}}$ for the repair run and $\mathcal{S}_{\text{initial}}$ for the initial run, initial swarm size N, and matching between agent poses before ($\mathcal{H}_j \in \mathcal{S}_{\text{initial}}$) and after ($\mathcal{U}_i \in \mathcal{S}_{\text{repair}}$) the damage

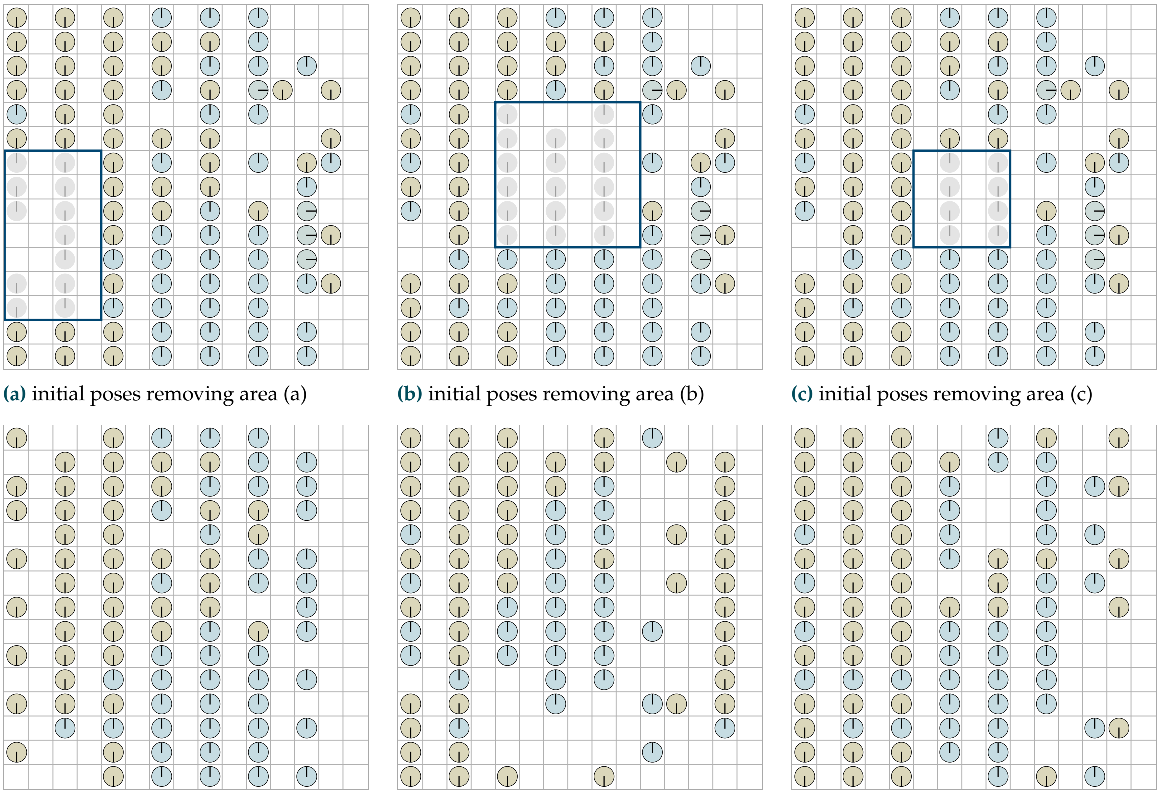
$$\text{match}(\mathcal{U}_i) = \begin{cases} 1, & \text{if } \exists \mathcal{H}_j \in \mathcal{S}_{\text{initial}} : \mathcal{U}_i = \mathcal{H}_j \\ 0, & \text{otherwise} \end{cases}. \tag{5.2}$$

For line structures, agent positions P and headings H are important as the parallel orientation of agents guarantees the formation of a stable structure. For triangular lattices, we consider only positions P of the agents as they constantly turn to stay on the grid cell (see Sec. 4.2.2). Consequently, agent headings H change in every time step and are irrelevant for structure formation here.

Line Formation Behavior

We first analyze the resilience of the line formation behavior shown in Fig. 5.3a. The formed line structure is not completely stable at the end of the initial run as some agents still move. However, 82 % of agents assemble into lines.

First, we rerun the ANN pair using new random agent starting poses in 20 independent runs. As before, almost all swarm members assemble into line structures leading to a mean solution quality of 0.86, see Tab. 5.1. We measure an



(a) initial poses removing area (a) **(b)** initial poses removing area (b) **(c)** initial poses removing area (c)

(d) final poses removing area (a) **(e)** final poses removing area (b) **(f)** final poses removing area (c)

Figure 5.4: Initial and final agent poses when removing agents from areas (a), (b), or (c) of the initially formed line structure in our self-assembly scenario. (a) - (c) give the initial agent poses of the repair run after removing agents from the area marked by the blue rectangle. (d) - (f) give the agent poses at the end of the repair run. Agents are represented by circles, their color and the lines give their headings [39].

increase in fitness F (Eq. 3.1) compared to the initial run. The fitness of the reruns is higher because we include the fitness measured in all 20 independent runs, while the fitness of the initial run is the minimum fitness out of ten evaluations, see Sec. 4.1.

Next, we completely remove agents from three different areas of the structure, see Fig. 5.4. We remove 12 (Fig. 5.4a), 17 (Fig. 5.4b) and eight (Fig. 5.4c) agents leading to swarm sizes N of 88, 83, and 92 agents, respectively. In all three cases, solution quality decreases by at least 13 percentage points (pp) due to the removal of agents. We observe that solution quality increases again as lines reform in the damaged area over the repair run, resulting in higher solution qualities $Q(T)$ than in the initial structure in two out of three cases. Line structures form again in the removed area, see Figs. 5.4d - 5.4f, and we find high similarity to the initial structure in two out of three runs ($S > 0.7$).

Last, we randomly reposition agents from three different areas outside of the respective area instead of removing them completely. Consequently, swarm density stays the same in this scenario. Repositioning agents leads not only to the destruction of line formations on the now empty part of the grid, but randomly repositioned agents may also corrupt

Table 5.1: Mean fitness F (Eq. 3.1), mean solution qualities $Q(0)$ and $Q(T)$ (Eq. 4.11) at the start and the end of the run, respectively, and mean similarity S to the initial structure (Eq. 5.1) studying robustness against damage in our self-assembly scenario using a line formation behavior and a triangular lattice formation behavior. Median values are in parentheses [39].

	lines				triangular lattices			
	F	$Q(0)$	$Q(T)$	S	F	$Q(0)$	$Q(T)$	S
initial run	0.81	0	82.0	1.0	0.76	0	63.0	1.0
rerun	0.86	1.3	83.3	0.10	0.78	0.0	77.0	0.47
	(0.85)	(0.0)	(86.0)	(0.02)	(0.781)	(0.0)	(78.0)	(0.46)
remove area (a)	0.96	69.3	79.5	0.76	0.76	60.9	58.6	0.39
remove area (b)	0.94	49.4	90.4	0.57	0.76	67.1	65.9	0.52
remove area (c)	0.94	65.2	89.1	0.72	0.79	66.3	76.1	0.67
reposition area (a)	0.92	47.9	83.5	0.59	0.79	21.8	75.2	0.564
	(0.92)	(54.0)	(86.5)	(0.6)	(0.79)	(22.5)	(80.5)	(0.62)
reposition area (b)	0.91	32.2	83.9	0.39	0.79	16.4	78.2	0.55
	(0.91)	(36.5)	(86.0)	(0.42)	(0.79)	(18.0)	(80.0)	(0.61)
reposition area (c)	0.90	56.2	86.2	0.59	0.80	38.4	79.3	0.571
	(0.93)	(60.0)	(87.0)	(0.59)	(0.90)	(39.5)	(80.5)	(0.61)

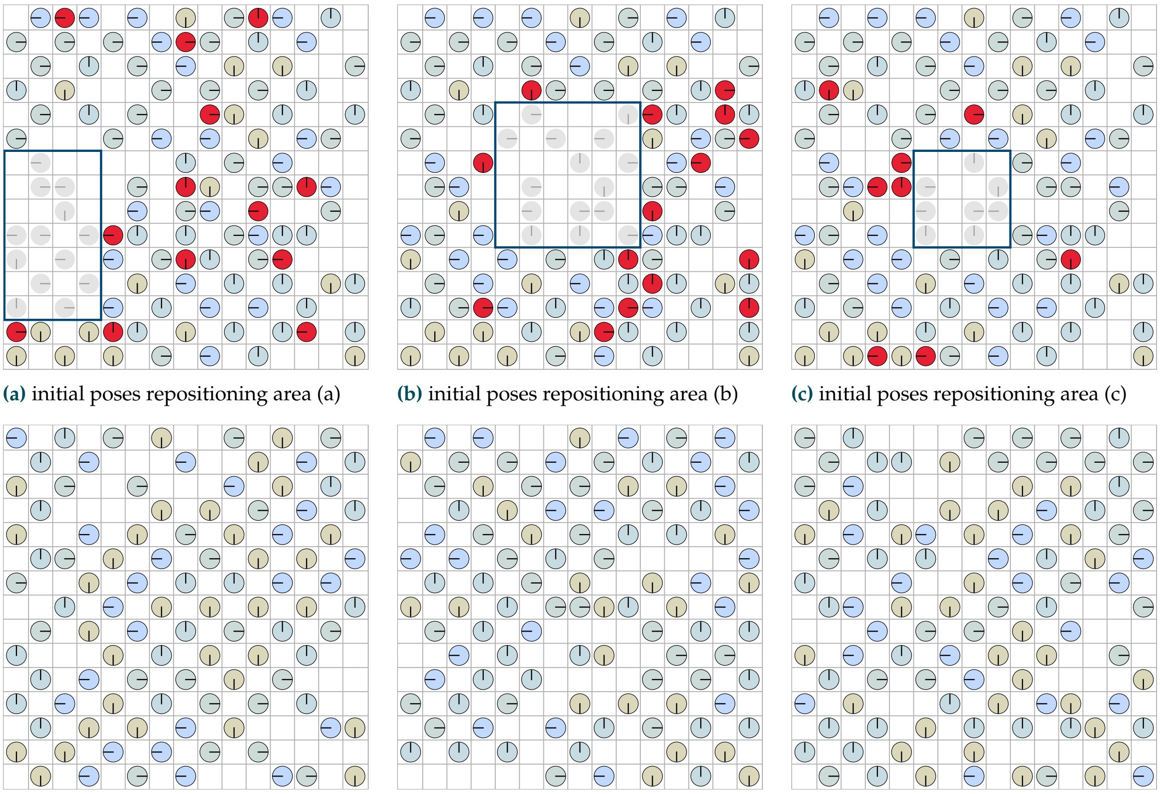
lines in other parts of the grid. Hence, we find a decrease of solution quality by at least 25 pp, see Tab. 5.1. But at the end of the repair runs, solution quality $Q(T)$ is at least 1.5 pp higher than at the end of the initial run. However, similarity S to the initial structure is always lower than when removing agents completely. We therefore assume that the self-assembled structure is more intensively disturbed by the randomly placed agents than by the completely removed agents.

We reach higher fitness F for the repair runs than for the initial run. In the repair runs, agents have the advantage of already being mostly positioned in the structure at the beginning of the runs compared to the random starting poses in the initial runs. Thus, sensor predictions of the predictor ANNs match more closely from the start of the repair runs and higher fitness can be reached.

Triangular Lattice Formation Behavior

As a second example, we show the resilience of the triangular lattice formation behavior illustrated in Fig. 5.3b. We observe an increase in fitness F rerunning the ANN pairs with new initial random starting poses as for the line structures, see Tab. 5.1. Furthermore, we find higher solution quality $Q(T)$ at the end of the reruns than after the initial run.

Again, we remove agents from the three different areas of the structure. Thus, 13, 15 and eight agents are removed from the triangular lattice structure leading to an immediate increase in solution quality for two of the three cases, see Tab. 5.1. Swarm size N is reduced by removing agents from the structure, and hence the percentage of agents being assembled into the structure (i.e., solution quality) can change. An increase in solution quality indicates that we removed



(a) initial poses repositioning area (a) **(b)** initial poses repositioning area (b) **(c)** initial poses repositioning area (c)

(d) final poses repositioning area (a) **(e)** final poses repositioning area (b) **(f)** final poses repositioning area (c)

Figure 5.5: Initial and final agent poses of one run each repositioning agents from areas (a), (b), or (c) of the initially formed triangular lattice structure in our self-assembly scenario. (a) - (c) give the initial agent poses of the repair run after repositioning all agents from within the area marked by the blue rectangle. Randomly repositioned agents are visualized in red. (d) - (f) give the final agent poses of the repaired structure. Agents are represented by circles, their color and the lines give their headings.

agents that were not part of the triangular lattice structure. At the end of the repair run, solution quality $Q(T)$ increased for one case while the other two cases have lower solution quality than at the start of the repair run. This is probably caused by the reduced swarm size. On the 15×15 grid, 112 agents are required to form a repetitive triangular lattice pattern over the whole grid. Our original swarm size of $N = 100$ is lower, but still close to the ideal number of agents. Reducing swarm size slightly makes the formation of triangular lattices harder, but it is still possible. Removing a larger amount of agents, as in our examples removing 13 or 15 agents, leads to a too low swarm density that may prevent the formation of a repetitive triangular lattice pattern. In consequence, we find lower solution quality $Q(T)$ at the end of the repair run and low similarities S to the initial structure when removing agents from areas (a) and (b) ($S \leq 0.52$).

Next, we reposition the agents positioned in the three different areas outside of the respective area, see Fig. 5.5, which leads to a major decrease in solution quality. But at the end of the repair runs, we observe an increase in solution quality $Q(T)$ by at least 12 pp compared to the initial structure and intermediate similarity S to the initial structure, see Tab. 5.1. Thus, agents reassemble to form the same pattern type (see Sec. 4.2.2) as in the initial run but only partly position themselves on the same grid cells.

In total, we find that the emergent self-assembly behaviors are resilient to damage. We have shown, using a line formation behavior and a triangular lattice formation behavior as examples, that removing or repositioning agents triggers the reassembly of the initial pattern.

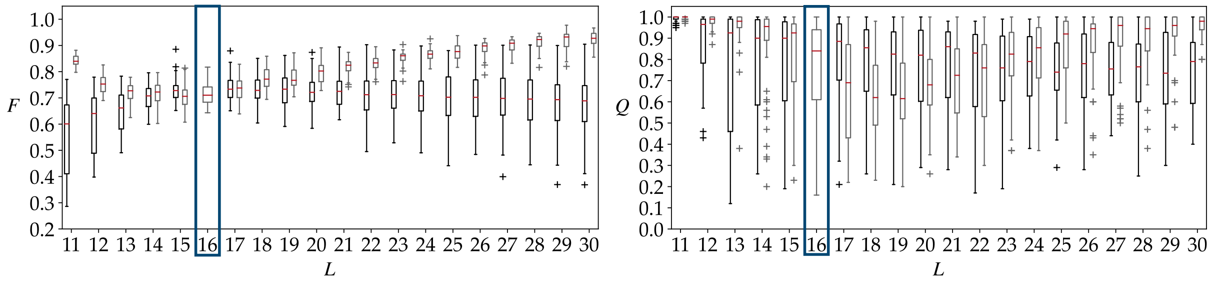
5.2 Scalability with Swarm Density

In Sec. 4.4, we found that swarm density influences the emergence of behaviors. High swarm densities lead to more grouping behaviors (i.e., aggregation, clustering, loose grouping) while low densities lead mainly to dispersion. In this section, we analyze the scalability of the emergent behaviors with swarm density [42]. As before, we keep a fixed swarm size N of 100 while altering the side length L of our square torus grid to change swarm density D_N (Eq. 4.3). We rerun individuals (i.e., actor-predictor ANN pairs) that were specialized for a specific swarm density and grid size L_s on different not-trained grid sizes $L \neq L_s$ (hereafter referred to as reruns). First, we compare fitness F (Eq. 3.1), solution quality Q (Eq. 4.11), and behavior distributions of the reruns with the original (specialized) evolutionary runs on L_s to determine if the performance of the best evolved individuals is independent of swarm density. Afterwards, we compare the reruns with the ANN pairs that are specialized for L to evaluate the added value of evolving ANN pairs specialized for given swarm densities. Since we have found the greatest behavioral diversity on $L_s = 16$, we exemplarily rerun each of the 50 best evolved individuals of the 16×16 grid for 500 time steps on each grid size $L \in [11 .. 30]$ and analyze their scalability.

5.2.1 Comparison of Specialized Individuals with their Reruns in Different Swarm Densities

First, we compare the specialized best evolved individuals of the 16×16 grid (i.e., $L_s = 16$) with their reruns on grid sizes $L \in [11 .. 30]$. We find statistically significantly lower fitness for the reruns on grid sizes $L \in [11 .. 13]$ (black boxes) than for the original runs on the 16×16 grid (gray box framed by blue rectangle for $L = 16$; MW-U with BC, $p < 0.001$), see Fig. 5.6a. For all other grid sizes, there are no significant differences in fitness between the reruns and the specialized run on $L_s = 16$.

Furthermore, the reruns on grid sizes $L \in \{11, 12\}$ have statistically significantly better solution quality than the original runs on the 16×16 grid, while we do not find



(a) fitness F **(b)** solution quality Q

Figure 5.6: Fitness F (Eq. 3.1) and solution quality Q (Eq. 4.11) of the reruns of the best evolved individuals of the 16×16 grid (framed in blue) on grid sizes $L \times L, L \in [11 .. 30]$ (black boxes) and of the best evolved individuals of each grid size (gray boxes, cf. Figs. 4.15b, 4.17) per grid size in the self-assembly scenario. Medians are given by the red bars. [42]

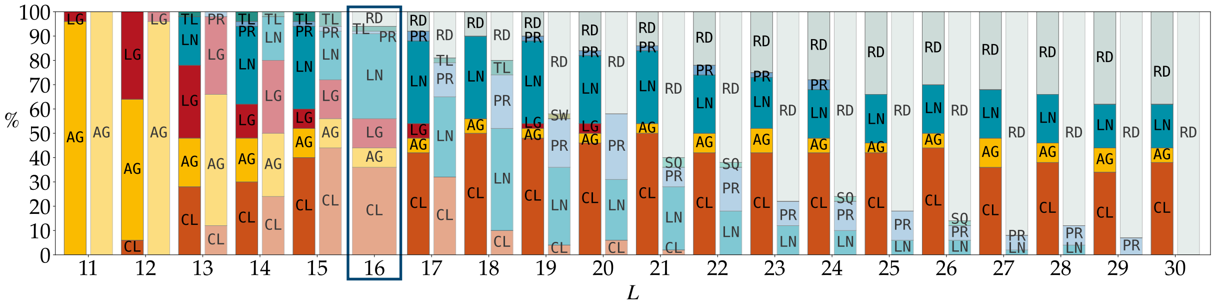


Figure 5.7: Percentage of resulting structures of the reruns of the best evolved individuals specialized for grid size $L_s \times L_s, L_s = 16$ (framed in blue) on grid sizes $L \times L, L \in [11 .. 30]$ (left, darker colored bars) and of the best evolved individuals of each grid size (right, lighter colored bars) with clustering (CL), aggregation (AG), loose grouping (LG), lines (LN), pairs (PR), triangular lattices (TL) and random dispersion (RD).

significant differences for the reruns on other grid sizes (MW-U with BC, $p < 0.05$). The high swarm densities on the small grid sizes $L \in [11 .. 13]$ allow only for grouping behaviors (i.e., clustering, aggregation, loose grouping). Due to the limited amount of free grid cells, most agents are forced to be part of the forming grouping structure and thus high solution quality is reached. Behaviors that require free grid cells between some or all agents, such as pairs, lines, and triangular lattices, cannot form during the reruns in such high swarm densities. For those behaviors, the predictor outputs do not match to the formed structures anymore and we thus find lower fitness.

Next, we study the influence on the behavior distributions when running ANN pairs in swarm densities that they are not evolved for, see Fig. 5.7. We do not find statistically significantly different behavior distributions for the reruns on grid sizes $L \in [13 .. 24]$ compared to the behavior distribution on the 16×16 grid (FE with BC, $p < 0.05$). The reruns on even smaller (i.e., $L < 13$) and even larger grids (i.e., $L > 24$), that is, in higher and lower swarm densities, lead to significantly different behavior distributions. As described before, high swarm densities only allow for grouping behaviors and thus the original variety of patterns cannot form. Low densities, on the other hand, lead to agents likely being randomly dispersed and far apart from each other by the initial uniformly random positioning. Here, a combination of the initial agent positions, the evaluation length and the behavior determined by the best evolved actor may influence whether the structure formed on the 16×16 grid reforms in the rerun. An indicator for different structure formation approaches of the actors gives the individual analysis of the scalability of each best evolved individual of the 16×16 grid. Half of the best evolved individuals lead to the formation of the same pattern or a pattern of the same group (e.g., to aggregation in the rerun when clustering on the 16×16 grid) in the reruns on grid sizes $L \in [13 \mathinner{..} 30]$. The other half of the best evolved individuals only forms the initial pattern in some of the other swarm densities. In these cases, the reruns in higher swarm densities mostly lead to grouping behaviors while the reruns in lower densities frequently result in random dispersion. In both groups, we find the same variety of formed structures on the 16×16 grid. Thus, the best evolved individuals seem to vary in their approach to structure formation by more or less adaptation to and exploitation of the swarm density during evolution. In summary, many ANN pairs keep their clustering, aggregation, and line formation behaviors even in low swarm densities they are not specialized for. Hence, this scaling ability can, for example, be exploited to provoke desired behaviors (see also Ch. 6) in low-density situations, which would otherwise not lead to the emergence of those behaviors.

Overall, we find similar fitness, solution quality, and behavior distributions for the reruns on grid sizes $L \in [14 \mathinner{..} 24]$ and the original runs of the best evolved individuals on the 16×16 grid. Thus, the best evolved individuals scale well with swarm density, but the initial performance cannot be maintained for all behaviors for high and low densities. In general, this allows for the reuse of the best evolved individuals in other swarm densities.

5.2.2 Comparison of Reruns with the Specialized Individuals of the Respective Swarm Density

Previously, we compared the best evolved individuals of the 16×16 grid with their reruns on other grid sizes $L \in [11 \mathinner{..} 30]$. Next, we compare the reruns of the best evolved individuals specialized for the 16×16 grid on grid sizes $L \in [11 \mathinner{..} 30]$ with the best evolved individuals specialized for the respective grid size. We compare fitness F, see Fig. 5.6a, and solution quality Q, see Fig. 5.6b, of the reruns (black boxes) to the specialized runs of the respective grid size (gray boxes).

We find statistically significantly greater fitness for the individuals specialized for a grid size than for the reruns on that grid size except for $L \in [14 \mathinner{..} 17]$ (MW-U with BC, $p < 0.05$), see Fig. 5.6a. There are no statistically significant differences in solution quality for the specialized individuals and the reruns on grid sizes $L \in [11 \mathinner{..} 15]$ and $L \in [20 \mathinner{..} 24]$ (MW-U with BC, $p < 0.05$), see Fig. 5.6b. For grid sizes $L \in [17 \mathinner{..} 19]$, we find significantly better solution quality in the reruns than for the specialized individuals of the respective grid size and for grid sizes $L \in [25 \mathinner{..} 30]$, solution quality is significantly lower for the reruns. Furthermore, there is no significant difference in the behavior distributions of the reruns and the best evolved individuals specialized for the respective grid size for $L \in \{11, 14, 15, 17\}$ (FE with BC, $p < 0.001$), see Fig. 5.7. All other behavior distributions are statistically significantly different.

Overall, we find that for grid sizes $L \in [14 \mathinner{..} 17]$ the reruns and the best evolved specialized individuals for these grid sizes are competitive in fitness, solution quality, and behavior distributions. This was expected, since we have also not found any significant differences between the evolutionary runs for those swarm densities (see Sec. 4.4). Thus, the best evolved individuals of one swarm density can easily be reused in similar swarm densities. For grid sizes $L \in [17 \mathinner{..} 19]$, the reruns lead to higher solution quality and similar or lower fitness than the best evolved individuals specialized for the respective grid sizes. The evolutionary runs on these grid sizes reached the lowest solution quality of all scenarios (see Sec. 4.4). Thus, the evolution of behaviors leading to the formation of the defined patterns (see Sec. 4.2) seems especially challenging for these swarm densities. Here, the rerun of the best evolved individuals of the lower swarm density (i.e., in this scenario of the 16×16 grid) is beneficial when high solution quality is required and prediction accuracy (i.e., fitness) is unimportant. Prediction accuracy is irrelevant when we use the minimize surprise approach in an offline manner to generate swarm controllers [13], that is, we separate the optimization phase from the operational phase. In the optimization phase, prediction accuracy is used as the means to guide evolution towards interesting swarm behaviors. In the operational phase, the actor-predictor pairs are no longer optimized. As the actor of an ANN pair determines the agent's next action independently from the predictor, the latter is not required for the pure execution of an evolved swarm behavior. For significantly higher or lower swarm densities, the best evolved actor-predictor ANN pairs of the respective grid size are better adapted to each other and to the environment and thus reach higher fitness and solution quality. Nevertheless, the rerun of the best individuals of another swarm density can be beneficial for structures that may not or only rarely emerge by evolution in the respective density, such as lines on the 30×30 grid.

5.3 Behavioral Diversity

As we have seen in our previous analysis (see Sec. 4.4), minimize surprise leads to diverse high-quality behaviors across independent evolutionary runs by rewarding prediction accuracy. By contrast, divergent search algorithms and quality-diversity algorithms (see Sec. 2.3.2) explicitly aim for behavioral diversity within one evolutionary run. While divergent search algorithms cannot guarantee solution quality because they do not use performance-related rewards, the more sophisticated quality-diversity algorithms output several maximally diverse and well-performing solutions [296]. Novelty search [24] and MAP-Elites [44] are competitive and easy-to-implement representatives of divergent search algorithms and quality-diversity algorithms, respectively, that have already been successfully applied to swarm scenarios [134, 300]. In this section, we compare our standard minimize surprise approach (see Ch. 3) with novelty search and MAP-Elites to evaluate whether it is competitive in terms of the evolved behavioral diversity [40, 41, 43]. For a fair comparison between the approaches, we introduce task-independent variants for both novelty search and MAP-Elites that are conceptually related to our minimize surprise approach. As before, we do our experiments in the self-assembly scenario presented in Sec. 4.1.

5.3.1 Evolutionary Approaches

We introduce our task-independent novelty search and Minimize Surprise MAP-Elites approaches, that we compare with standard minimize surprise, first.[2] In addition, we introduce evaluation metrics. All three evolutionary approaches do not optimize for a predefined task, but behaviors are classified during post-evaluation to measure the emergent behavioral diversity. As we use our self-assembly scenario, we again categorize our behaviors using the nine patterns defined in Sec. 4.2.2. To keep this study reasonably short, we restrict ourselves to grid sizes 15×15 and 20×20 as sample scenarios for denser and sparser swarm density settings.

2: An extensive comparison of standard minimize surprise, minimize surprise with predefined predictions (see Ch. 6), novelty search with a task-specific behavioral characteristic, novelty search with a task-independent characteristic, and a standard genetic algorithm as baseline can be found in Kaiser and Hamann [41]. A comparison of our task-independent Minimize Surprise MAP-Elites approach and MAP-Elites with a task-dependent performance measure is future work.

Novelty Search

Novelty search promotes behavioral (i.e., phenotypic) diversity by scoring individuals on how different, or novel, their behavior is compared to other individuals instead of driving the evolutionary process to a fixed goal [24]. The novelty η of an individual is calculated by a novelty metric that determines the sparseness at a point in behavior space taking into account the current population and an archive of past individuals. These samples of individuals represent previously visited regions of search space and the current list of solution candidates. Novelty η is defined as

$$\eta(i) = \frac{1}{K} \sum_{k=0}^{K-1} \text{b\_dist}(i, v_k) \ , \tag{5.3}$$

where v_k is individual i's kth-nearest neighbor with respect to the behavioral distance metric b_dist$(\cdot, \cdot)$ and considering K nearest neighbors. Here, we calculate the Euclidean distance between the behavioral feature vector of individual i and its ten nearest neighbors (i.e., $K = 10$) as our behavioral distance b_dist$(\cdot, \cdot)$.

The behavioral feature vector characterizes the behavior of an individual and is supposed to capture relevant aspects of an assumed task. Key is to carefully select these features, such that useful and diverse behaviors are found and distinguished. Here, we define a potentially task-independent behavioral feature vector (i.e., behavioral characteristic) as a R-dimensional vector

$$\text{b\_char}(i) = \frac{1}{N} \left[\sum_{n=0}^{N-1} s_0^n(T), \ \ldots \ , \sum_{n=0}^{N-1} s_{R-1}^n(T) \right] \tag{5.4}$$

of R sensor values in the last time step of the evaluation averaged over swarm size N. Similarly to our concept in minimize surprise, sensor values can serve as local templates of forming patterns. Novelty search tries to popularize all variants of these vectors, which should enable the evolution of a variety of behaviors leading to the assembly of different patterns by agents. We average the behavioral vectors obtained in ten repetitions per individual.

To limit the algorithm's computational complexity, only a subset of explored individuals is added to the archive of past individuals. Several strategies of when to add individuals to the archive exist [361], for example, adding all individuals with a novelty value η above a minimum threshold η_{min} (i.e., $\eta > \eta_{\text{min}}$) or adding each individual with low probability. Always adding individuals of high novelty may limit the local search in newly explored regions of search space. Thus, we use the second approach, as also promoted by Lehman and Stanley [362], and add individuals with a probability of 2 % to the archive of past individuals.

Otherwise our novelty search implementation (see Alg. 3) is based on the evolutionary algorithm used in our minimize surprise self-assembly scenario (see Sec. 4.1), but we do not apply elitism here. Since we reward novelty instead of prediction accuracy, the predictor ANN is superfluous and not used in novelty search. Thus, we evolve only the actor ANN (see Fig. 4.2a). We do 50 independent runs per grid

Algorithm 3: Novelty search

generate random initial population $\mathcal{P}(0)$
$g \leftarrow 0$ // current generation
$\mathcal{A} \leftarrow \emptyset$ // archive of past individuals
/* until maximum number of generations is reached */
while $g < g_{max}$ **do**
  /* evaluate each individual in current population */
  for $i \in \mathcal{P}(g)$ **do**
    b_char(i) $\leftarrow$ evaluate individual i // behavioral characteristic
  for $i \in \mathcal{P}(g)$ **do**
    calculate novelty $\eta(i)$ w.r.t. $\mathcal{P}(g)$ and $\mathcal{A}$
    generate uniformly random value $\mathcal{X} \in [0, 1]$
    if $\mathcal{X} < 0.02$ **then**
      add individual i to $\mathcal{A}$
  generate population $\mathcal{P}(g + 1)$ by selection and mutation
  $g \leftarrow g + 1$

size $L \times L, L \in \{15, 20\}$ and consider two sets of potential solutions: (i) all 250,000 evaluated individuals (50 individuals, 100 generations, 50 experiments; `N-SVA`), and (ii) the 50 individuals with the best solution quality out of all 250,000 individuals evaluated in the 50 independent evolutionary runs (`N-SVQ`). The latter equals the resulting number of best individuals out of 250,000 evaluated individuals in 50 independent evolutionary runs with our standard minimize surprise approach in the self-assembly scenario. So we create solution set N-SVQ especially to compare the evolutionary approaches.

Minimize Surprise MAP-Elites

MAP-Elites [44] generates diverse high-quality solutions by filling a discrete behavior-performance map over generations with solutions, retaining the highest performing individual per cell. We propose Minimize Surprise MAP-Elites [43] that evolves actor-predictor ANN pairs using our task-independent minimize surprise reward for prediction accuracy (Eq. 3.1) as the performance measure in standard MAP-Elites and sensor values as the dimensions of the behavior-performance map. The approach is fully task-independent, potentially allowing for behavioral diversity within behavior categories (i.e., patterns in our self-assembly scenario) and across behavior categories within one evolutionary run.

For our self-assembly scenario, we define a behavior-performance map $\mathcal{M}$ with five dimensions $X_0 - X_4$ based on the swarm's discretized mean sensor values in the last time step of an evaluation. Mean sensor values greater than 0.5 are mapped to 1, otherwise to 0. We aggregate two sensors each

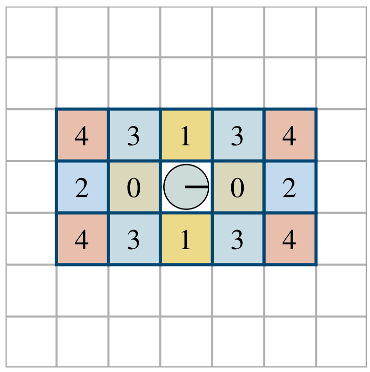


Figure 5.8: Visualization of the dimensions of the behavior-performance map $\mathcal{M}$ of Minimize Surprise MAP-Elites in our self-assembly scenario. Each dimension is based on mean sensor values of two to four sensors (i.e., colored grid cells). Grid cells in the same color and with the same index i form dimension X_i. The circle represents an agent and the line gives its heading.

Algorithm 4: MAP-Elites

```
Output: behavior-performance map M
e ← 0                                          // current evaluation
M ← ∅                                          // initialize empty map
/* until maximum number of evaluations is reached             */
while e < e_max do
    /* initialization                                          */
    if e < e_init then
        generate random individual i′
    /* main loop                                               */
    else
        apply random selection and mutation to generate
          individual i′
    /* determine performance and behavioral descriptor         */
    F′, b_des ← evaluate individual i′
    /* store individual i′ if the respective map cell is
       empty or i′ has the better performance                  */
    if M(b_des) = ∅ or F(b_des) < F′ then
        M(b_des) ← (i′, F′)
    e ← e + 1
```

for dimensions $X_0 - X_2$ and four sensors each for dimensions X_3 and X_4 (see Fig. 5.8) resulting in $3^3 \times 5^2 = 675$ map cells. This aggregation of sensor values is based on symmetries in the nine defined patterns (see Sec. 4.2.2 and Fig. 5.9), which keeps the solution set concise. A potential other option would be using each of our $R = 14$ sensors as one dimension, resulting in $2^{14} = 16{,}384$ map cells. This would also increase computational load considerably because potentially more individuals have to be evaluated to fill the behavior-performance map with high-quality solutions. Since we are confident that our nine defined patterns form an extensive solution set, we do not expect more interesting behaviors to emerge by drastically increasing the size of the behavior-performance map in this way and rely on our computationally less expensive discretization of the behavior-performance map. In each evaluation, both an individual's performance F (Eq. 3.1) and its behavioral descriptor b_des = $[X_0, X_1, X_2, X_3, X_4]$ along the five defined map dimensions are determined.

We initialize Minimize Surprise MAP-Elites (see Alg. 4) by evaluating $e_{\text{init}} = 2{,}500$ individuals and placing them into their respective map cells $\mathcal{M}$(b_des). Per individual, we do three independent evaluations of 500 time steps each, assigning the minimum fitness reached in these three evaluations as the overall fitness. After initialization, a random cell of the map is chosen and offspring created by mutation, that is, we add a uniformly random number in $[-0.4, 0.4]$ to each value of the genome, that represents individual i, with a probability p_{mut} of 0.1. In total, we do $e_{\text{max}} = 250{,}000$ evaluations per independent evolutionary run, equaling the number of evaluations in 50 independent evolutionary runs with standard minimize surprise. We do 10 independent MAP-Elites runs per grid size $L \times L, L \in \{15, 20\}$.

5.3.2 Metrics

As before, we classify emergent behaviors based on the nine defined patterns presented in Sec. 4.2.2 and measure their solution quality Q (Eq. 4.11) to compare behavioral diversity. Additionally, we introduce pattern coverage C_{pat}. Pattern coverage C_{pat} is the amount of different emergent patterns normalized by the nine defined patterns, that is, the percentage of found patterns, per evolutionary run. It is a measure for the emergent behavioral diversity within one evolutionary run. In standard minimize surprise each run leads to one best evolved individual and thus always to a pattern coverage C_{pat} of 0.11. But novelty search and Minimize Surprise MAP-Elites explicitly push for behavioral diversity within one evolutionary run and can output several patterns per run (i.e., $C_{\text{pat}} \geq 0.11$).

Furthermore, we quantify the quality of the Minimize Surprise MAP-Elites runs using three metrics proposed by Mouret and Clune [44]: (i) coverage C_{map}, (ii) precision $\bar{F}$, and (iii) global performance F_{max}.

Coverage C_{map} and precision $\bar{F}$ are both calculated with respect to the number of filled cells in the behavior-performance map $\mathcal{M}$. We define the number of filled map cells m_{filled} as

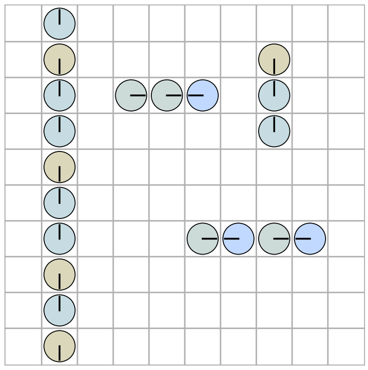
$$m_{\text{filled}} = \sum_{X_0=0}^{2} \sum_{X_1=0}^{2} \sum_{X_2=0}^{2} \sum_{X_3=0}^{4} \sum_{X_4=0}^{4} \text{filled}(X_0, X_1, X_2, X_3, X_4)\,, \tag{5.5}$$

with map dimensions $X_0 - X_2 \in [0\,..\,2]$, map dimensions $X_3, X_4 \in [0\,..\,4]$, and measure $\text{filled}(X_0, X_1, X_2, X_3, X_4)$ being 1 if map cell $\mathcal{M}(X_0, X_1, X_2, X_3, X_4)$ is filled with a solution and 0 otherwise.

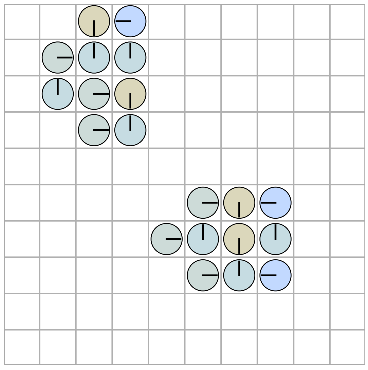
Coverage C_{map} is the percentage of cells in the behavior-performance map that are filled with solutions and is defined as

$$C_{\text{map}} = \frac{m_{\text{filled}}}{m_{\text{total}}}\,, \tag{5.6}$$

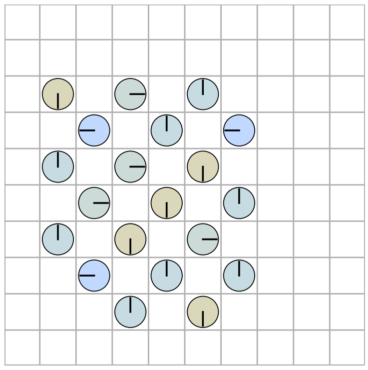
with number m_{total} of map cells that theoretically could be filled. Based on preliminary experiments, we assume that all 675 cells theoretically can be filled in our self-assembly scenario (i.e., $m_{\text{total}} = 675$).



(a) lines



(b) clustering



(c) triangular lattices

Figure 5.9: Example illustrations of lines, clustering, and triangular lattices to exemplify symmetries in the defined patterns (reprinted from Sec. 4.2.2). Agents in line structures detect neighbors directly in front and behind them (dimensions X_0 and X_2), clustered agents detect neighbors at least in their Moore neighborhood (dimensions X_0, X_1, X_3), and agents in triangular lattices detect neighbors two grid cells in front and behind them (dimension X_2). Circles represent agents, their color and the lines give their headings.

Precision $\bar{F}$ is the mean performance (i.e., prediction accuracy; Eq. 3.1) of the filled map cells. We define precision $\bar{F}$ as

$$\bar{F} = \frac{1}{m_{\text{filled}}} \sum_{X_0=0}^{2} \sum_{X_1=0}^{2} \sum_{X_2=0}^{2} \sum_{X_3=0}^{4} \sum_{X_4=0}^{4} F(X_0, X_1, X_2, X_3, X_4)\,, \tag{5.7}$$

with $F(X_0, X_1, X_2, X_3, X_4)$ being the performance of the solution in map cell $\mathcal{M}(X_0, X_1, X_2, X_3, X_4)$.

Global performance $F_{\max}$ is the maximum performance found in the behavior-performance map as given by

$$F_{\max} = \max_{X_0, X_1, X_2 \in [0..2], X_3, X_4 \in [0..4]} F(X_0, X_1, X_2, X_3, X_4)\,. \tag{5.8}$$

5.3.3 Results

First, we analyze the results of our novelty search variant with a task-independent behavioral characteristic and Minimize Surprise MAP-Elites individually. Afterwards, we compare standard minimize surprise, novelty search, and Minimize Surprise MAP-Elites, particularly in terms of behavioral diversity and solution quality.

Novelty Search

As previously discussed, we consider two sets of potential solutions in our novelty search runs: (i) all 250,000 individuals (`N-SVA`), and (ii) the 50 individuals with the best solution quality out of all 250,000 evaluated individuals (`N-SVQ`). Prediction accuracy F (Eq. 3.1) is not measured in novelty search. Thus, we only evaluate pattern coverage C_{pat}, solution quality Q (Eq. 4.11), and behavior distributions.

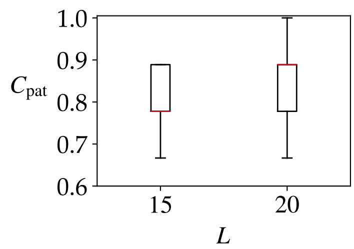
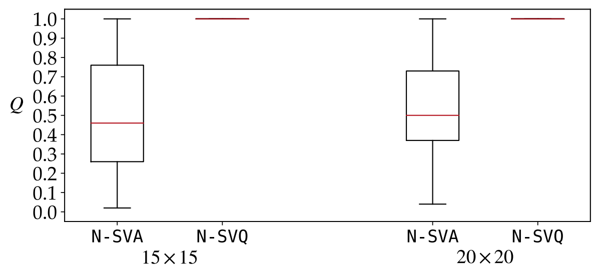
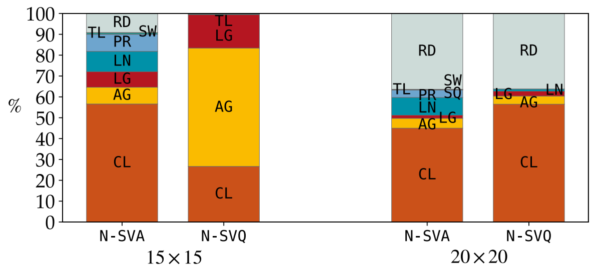


Figure 5.10: Pattern coverage C_{pat} for 50 independent novelty search runs per grid side length $L \in \{15, 20\}$ in our self-assembly scenario.

Pattern coverage C_{pat} is measured per evolutionary run and we consider all 5,000 evaluated individuals per independent novelty search run as potential solutions for the calculation. We find a median pattern coverage C_{pat} of 0.77 for the 15×15 grid and of 0.88 for the 20×20 grid, see Fig. 5.10. This means that we find a median of seven and eight different patterns, respectively, out of the nine possible patterns per evolutionary run. The difference in pattern coverage C_{pat} for the two grid sizes is partly caused by swarm density as already found for minimize surprise (see Sec. 4.4). Squares cannot form on the smaller grid due to the high swarm density of 0.44; the pattern can only perfectly form in swarm densities up to 0.25 (i.e., grid side length $L \geq 20$). However, squares and swirls generally rarely emerge on both grid sizes and are probably



(a) solution quality Q



(b) behavior distributions

Figure 5.11: Solution quality Q (Eq. 4.11) and behavior distributions of 50 independent novelty search runs with behavioral characteristic b_char (Eq. 5.4) with solution set `N-SVA` containing all 250,000 evaluated individuals and solution set `N-SVQ` containing the 50 individuals with best solution quality as potential solutions for grid sizes 15×15 and 20×20 in our self-assembly scenario. Behavior distributions give the percentage of clustering (`CL`), aggregation (`AG`), loose grouping (`LG`), lines (`LN`), pairs (`PR`), triangular lattices (`TL`), squares (`SQ`), swirls (`SW`) and random dispersion (`RD`).

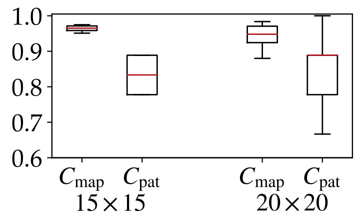
hard to form. On the larger grid, swarm density is almost too low for the formation of repetitive triangular lattices and thus the pattern does not always emerge. All other six patterns (i.e., clustering, aggregation, loose grouping, lines, pairs, and random dispersion) are found in all runs. Although we find high pattern coverage, it is not guaranteed that solution quality Q is high for all formed patterns and that a post-selected solution set is equally diverse.

For solution set N-SVA, we find a median solution quality of 0.46 on the 15×15 grid and of 0.5 on the 20×20 grid, see Fig. 5.11a. By contrast, solution set N-SVQ leads to a median solution quality of 1.0 on both grid sizes, which is intuitive since we selected this solution set based on solution quality. Consequently, selecting individuals based on solution quality results in a set of high-performing solutions. However, we find statistically significant differences in the behavior distributions of N-SVA and N-SVQ on both grid sizes (FE, $p < 0.001$), see Fig. 5.11b. For N-SVA, we find more grouping behaviors on the smaller grid and more dispersion behaviors, lines, and pairs on the larger grid. For N-SVQ, grouping behaviors prevail on both grid sizes and structures that require correct positioning, such as triangular lattices, are rare. It is probably harder to reach high solution quality Q for patterns that require correct positioning of several agents and thus the easier-to-form grouping and random dispersion patterns dominate in N-SVQ. In consequence, N-SVQ leads to four pattern types less than N-SVA on both grid sizes.

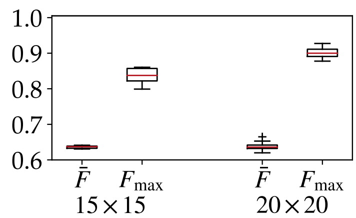
Overall, novelty search successfully pushes towards behavioral diversity as we find high pattern coverage C_{pat}. But not all solutions lead to high-quality behaviors and post-selecting high-quality solutions may reduce behavioral diversity.

Minimize Surprise MAP-Elites

Next, we analyze the quality of our Minimize Surprise MAP-Elites runs. We find a median coverage C_{map} of 0.97



(a) coverage C_{map} and pattern coverage C_{pat}



(b) precision $\bar{F}$ and global performance $F_{\max}$

Figure 5.12: Coverage C_{map} (filled map cells; Eq. 5.6), pattern coverage C_{pat} (percentage of emergent pattern types), precision $\bar{F}$ (mean prediction accuracy; Eq. 5.7), and global performance $F_{\max}$ (maximum prediction accuracy; Eq. 5.8) for 10 independent Minimize Surprise MAP-Elites runs per grid size $L \times L, L \in \{15, 20\}$ in our self-assembly scenario.

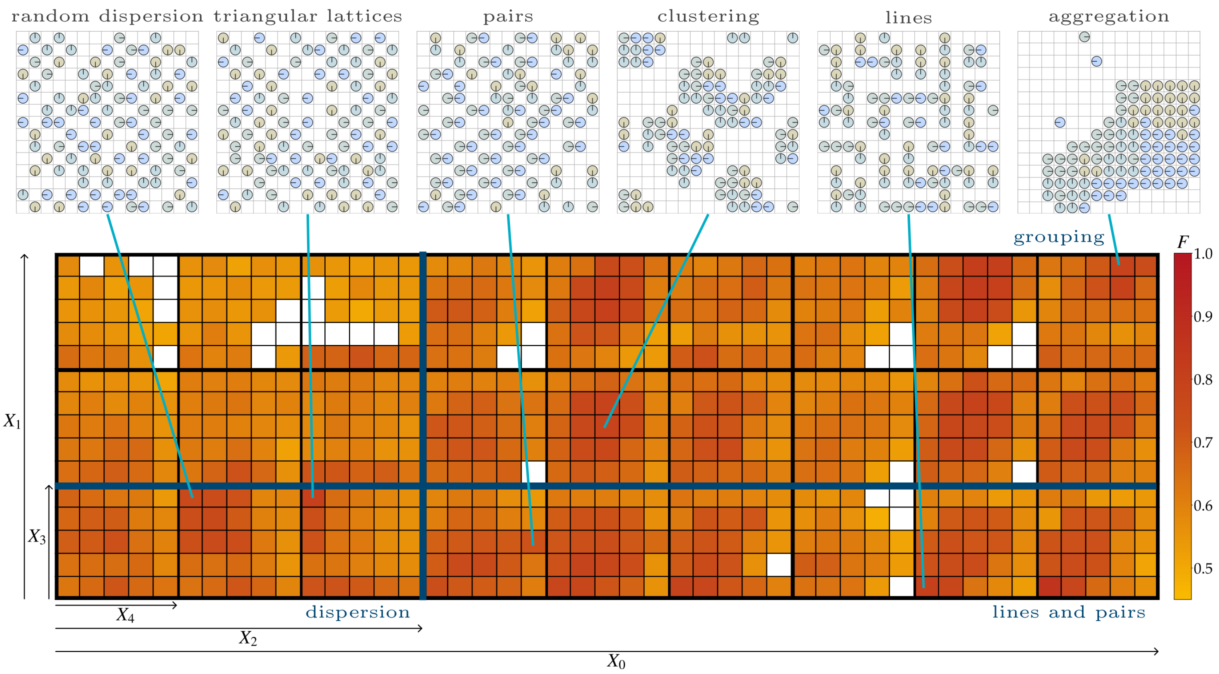
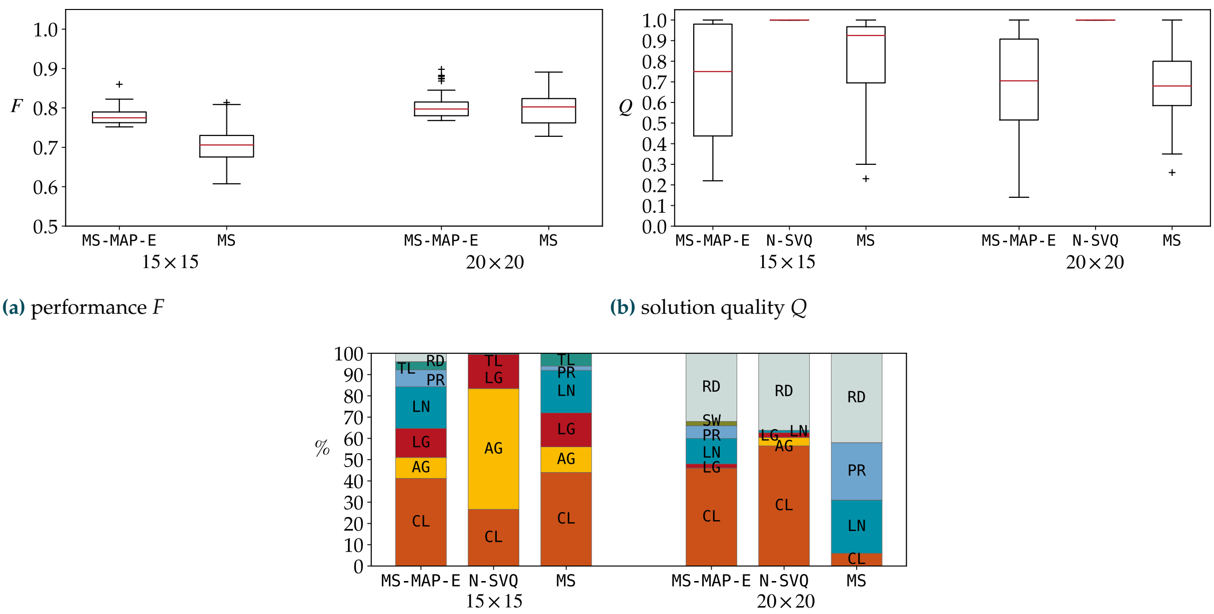


Figure 5.13: Behavior-performance map of one representative Minimize Surprise MAP-Elites run on the 15×15 grid and examples for several emergent structures. Dimensions $X_0 - X_4$ represent two to four sensors each (see Fig. 5.8) with $X_0 - X_2 \in [0\,..\,2]$ and $X_3, X_4 \in [0\,..\,4]$. Cell colors give the performance $F \in [0, 1]$, white map cells are empty (i.e., no solution was found). Dark blue lines mark areas where certain pattern types perform well.

on the 15×15 grid and of 0.95 on the 20×20 grid, see Fig. 5.12a. Thus, nearly all (i.e., $\geq 0.95 \times 675 = 621$) cells of the behavior-performance map are filled with solutions. Pattern coverage C_{pat} is also high with a median of 0.83 on the 15×15 grid and of 0.88 on the 20×20 grid. That means, we find a median of 7.5 patterns on the smaller grid and of eight patterns on the larger grid. As discussed in the previous section for novelty search, the difference in pattern coverage C_{pat} is partly caused by swarm density. As before, clustering, aggregation, loose grouping, lines, pairs, and random dispersion are found in all runs, but squares, swirls, and triangular lattices generally rarely emerge. Solutions are also high performing with a median precision $\bar{F}$ of 0.64 and a median global performance $F_{\max}$ of at least 0.84 on both grid sizes, see Fig. 5.12b. In total, we find high-quality behaviors for the formation of a variety of different patterns.

Fig. 5.13 visualizes the behavior-performance map of one Minimize Surprise MAP-Elites run on the 15×15 grid as representative example. Areas of high performance (i.e., red areas) each match the criteria of one of the nine defined patterns closely. Structure formations that deviate slightly from the defined classification criteria (i.e., $Q < 1.0$) can still reach high performance (i.e., prediction accuracy F; Eq. 3.1), since each predicted sensor accounts only for $\frac{1}{14}$th of the total performance. Consequently, several map cells can contain well performing solutions for the formation of the same pattern. We always find more than one solution for a pattern type, if any. Map cells with worse performance



(a) performance F

(b) solution quality Q

(c) behavior distributions

Figure 5.14: Performance F (Eq. 3.1), solution quality Q (Eq. 4.11), and behavior distributions of the 50 best solutions of Minimize Surprise MAP-Elites (`MS-MAP-E`; 1 independent run), minimize surprise (`MS`; 50 independent runs), and novelty search (`N-SVQ`; 50 independent runs) per grid size $L \times L, L \in \{15, 20\}$ with clustering (`CL`), aggregation (`AG`), loose grouping (`LG`), swirls (`SW`), lines (`LN`), pairs (`PR`), triangular lattices (`TL`), and random dispersion (`RD`).

usually contain grouping or random dispersion behaviors because these patterns are not based on the exact positioning and heading of several agents unlike, for example, lines and pairs. Overall, this shows that our Minimize Surprise MAP-Elites approach finds high-quality solutions for the self-assembly of different patterns and diverse solutions to the assembly of each pattern.

Comparison of Minimize Surprise, Novelty Search, and Minimize Surprise MAP-Elites

We compare performance F (Eq. 3.1) of standard minimize surprise and Minimize Surprise MAP-Elites as well as the resulting behavioral diversity and solution quality Q of standard minimize surprise, novelty search, and Minimize Surprise MAP-Elites. For a fair comparison, we select 50 best solutions out of the same number of evaluations per approach. From minimize surprise, we take the best evolved individuals of 50 independent evolutionary runs from our previous experiments (see Sec. 4.4). One Minimize Surprise MAP-Elites run has 250,000 evaluations, which is as many as 50 independent minimize surprise and novelty search runs. Thus, we pick the 50 best solutions based on performance F (Eq. 3.1) of one representative Minimize Surprise MAP-Elites run for our comparison. As prediction accuracy (Eq. 3.1) is not and cannot be measured in novelty search, we compare minimize surprise and Minimize Surprise MAP-Elites with our solution set N-SVQ, which contains the 50 best solutions of 50 independent novelty search runs selected by solution quality Q. These high-quality solutions generally would allow for high prediction accuracy.

First, we compare the performance F (Eq. 3.1) of the 50 best solutions of Minimize Surprise MAP-Elites and minimize surprise, see Fig. 5.14a. We find statistically significantly better performance for Minimize Surprise MAP-Elites on the 15×15 grid but no significant differences on the 20×20 grid (MW-U, $p < 0.001$). Next, we compare solution quality Q (Eq. 4.11) of the three approaches, see Fig. 5.14b. Novelty search reaches statistically significantly better solution quality than Minimize Surprise MAP-Elites and minimize surprise on both grid sizes (KW, $p < 0.001$; MW-U with BC, $p < 0.001$). However, novelty search has a median solution quality of 1.0, since we use solution set N-SVQ that contains the best solutions based on solution quality Q. Thus, the comparison is biased. Selecting solutions based on solution quality in Minimize Surprise MAP-Elites would lead to a competitive median solution quality of 0.97. For Minimize Surprise MAP-Elites and minimize surprise, we do not find significant differences in solution quality. Last, we compare behavioral diversity and find statistically significant differences on both grid sizes (FE, $p < 0.001$), see Fig. 5.14c. On the 15×15 grid, novelty search has a significantly different behavior distribution than Minimize Surprise MAP-Elites and minimize surprise. By contrast, minimize surprise has a significantly different behavior distribution than the other two approaches on the 20×20 grid. This is intuitive, since we find the least number of different patterns (i.e., four) for the significantly different behavior distributions, but the maximum number of different patterns (i.e., seven) on both grid sizes for Minimize Surprise MAP-Elites.

Comparing minimize surprise, novelty search, and Minimize Surprise MAP-Elites, the differences between the three evolutionary approaches become apparent. Novelty search creates a large number of diverse solutions within one run without putting weight on improving those over generations by only aiming for novel behaviors. Minimize surprise produces behavioral diversity across independent evolutionary runs while aiming to improve prediction accuracy (Eq. 3.1) of its individuals over generations in a single run. And Minimize Surprise MAP-Elites both improves solutions over generations and produces behavioral diversity. Hence, novelty search requires post-evaluation to select the best performing individuals, while the other two approaches already provide high-quality solutions. Even when the solution set of Minimize Surprise MAP-Elites is to be reduced, it requires less post-evaluation effort than novelty search because it outputs 675 high-quality solutions instead of 250,000 potential solutions of varying quality.

Overall, we find that Minimize Surprise MAP-Elites outperforms the other two approaches. It leads to high-quality solutions and the greatest behavioral diversity (i.e., the most different pattern types) on both grid sizes as intended by quality-diversity algorithms. But standard minimize surprise reaches similar solution quality on both grid sizes and a competitively diverse behavior distribution on the 15×15 grid. Thus, the standard, and rather simple, minimize surprise approach is also promising. In particular, the lower number of evaluations in one evolutionary run makes standard minimize surprise more suitable for applications where the number of evaluations should be kept to a minimum, such as real robot experiments where battery life is limited and wear and tear of the hardware must be avoided. We show that our standard minimize surprise approach can be applied to real robots in Chs. 10 and 11. For the rest of this thesis, we focus on the standard minimize surprise approach and explore its potential in different scenarios and settings. However, we will investigate the very promising combination of quality-diversity algorithms and our task-independent minimize surprise approach further in future work.

5.4 Hyperparameter Optimization

In all our previous experiments, we have used a fixed set of hyperparameters to evolve self-assembly behaviors with minimize surprise (see Sec. 4.1). Here, we test how the number of simulation runs per fitness evaluation (repetitions), evaluation length T, population size μ, number of generations $g_{\max}$, and mutation rate p_{mut} influence the resulting fitness (Eq. 3.1) [42]. We do 20 independent evolutionary runs per tested parameter combination on grid sizes 15×15 and 20×20. The two grid sizes are examples for denser and sparser settings, since swarm density affects the emergence of behaviors and thus different parameter sets may be optimal. Our experiments on hyperparameter optimization are divided into two parts. First, we do experiments where we vary a single hyperparameter at a time while fixing the rest to the values used in our previous experiments (see Tab. 5.2). Based on the results of these first experiments, we test different hyperparameter combinations afterwards, since also the combination of the different parameters can influence the resulting fitness.

Variation of One Hyperparameter First, we vary one hyperparameter while fixing the rest to their initial values used in our previous experiments. Tab. 5.2 summarizes initial and tested values. For one to ten repetitions, we do not find significantly different fitness for both grid sizes (KW, $p < 0.05$).[3] Evaluation lengths T between 50 and 500 time steps do not lead to significant differences in fitness on the 20×20 grid, but this only holds true for 200 to 500 time steps on the 15×15 grid (MW-U with BC, $p < 0.05$). This is in line with

3: To keep this study concise, we do not show the data for the experiments where we vary a single hyperparameter.

Table 5.2: Initial hyperparameter values used in our previous experiments and values tested for hyperparameter optimization in our self-assembly scenario.

hyperparameter	initial value	tested values
# of simulation runs per fitness evaluation	10	$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
evaluation length T (time steps)	500	$\{50, 100, 200, 300, 400, 500\}$
population size μ	50	$\{10, 20, 30, 40, 50, 60, 70, 80\}$
number of generations $g_{\max}$	100	$\{50, 60, 70, 80, 90, 100\}$
mutation rate p_{mut}	0.1	$\{0.01, 0.02, 0.03, 0.04, 0.05, 0.07, 0.1, 0.2, 0.3\}$

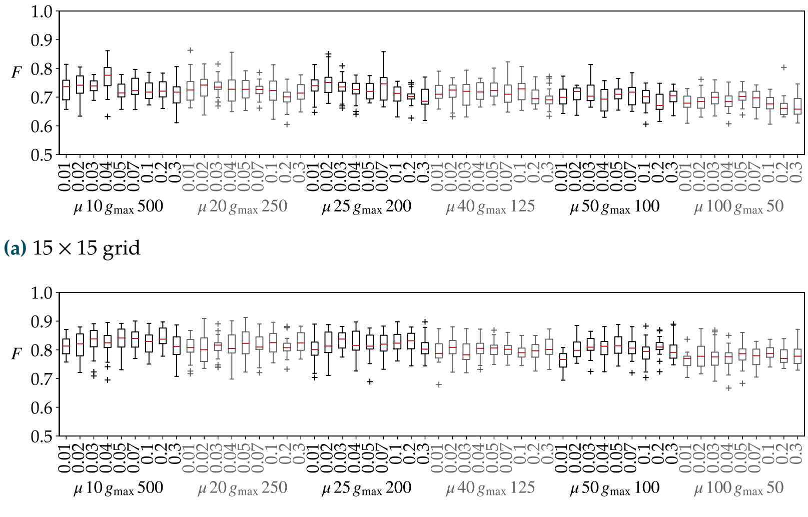
our findings on the temperature of the system (see Fig. 4.19) showing that the swarm quickly assembles into the structure. Numbers of generations $g_{\max}$ between 50 and 100 do not lead to significantly different fitness on the 15×15 grid (KW, $p < 0.05$). On the 20×20 grid, we find significantly better fitness for 90 generations than for numbers of generations between 50 and 70 but no significant differences for numbers of generations between 80 and 100 (MW-U with BC, $p < 0.05$). Population sizes μ between ten and 80 do not lead to significant differences on the 15×15 grid. On the 20×20 grid, population sizes of 50 and more have significantly better fitness than population size ten (MW-U with BC, $p < 0.05$). We do not find statistically significant differences for mutation rates p_{mut} between 0.01 and 0.3 on both grid sizes (MW-U with BC, $p < 0.05$). Other mutation rates, however, may lead to significantly lower fitness.

In total, we find that we can speed up evolution by reducing repetitions and evaluation length without loss in fitness. The total number of evaluations (i.e., population size $\mu \times$ number of generations $g_{\max}$) does not impact fitness on the 15×15 grid significantly, but 4,000 or more evaluations should be done on the 20×20 grid.

Variation of Hyperparameter Combinations Based on our previous results, we test different combinations of mutation rate, population size, and number of generations next. Thereby, population size μ influences the width (i.e., exploration of the solution space) and the number of generations $g_{\max}$ the depth of the search. We do 5,000 evaluations divided between population sizes μ of 10 to 100 and 50 to 500 generations $g_{\max}$, see Tab. 5.3. For all combinations of population size and generations, we vary mutation rates between 0.01 and 0.3, that is, we have in total 54 hyperparameter combinations per grid size. We do two simulation runs of 200 time steps each per fitness evaluation. Fig. 5.15 visualizes best fitness for the different hyperparameter combinations on the 15×15 and the 20×20 grids. On both grid sizes, we find statistically significantly different fitness for the different hyperparameter combinations (KW, $p < 0.01$). We find the maximum median best fitness for population size 10 and 500 generations, that is, 0.78 for mutation rate 0.04 on the 15×15 grid and 0.84 for mutation rate 0.05 on the 20×20 grid. In total, we find no

Table 5.3: Tested combinations of population size μ and number of generations $g_{\max}$ for hyperparameter optimization in our self-assembly scenario.

μ	$g_{\max}$
10	500
20	250
25	200
40	125
50	100
100	50



(a) 15×15 grid

(b) 20×20 grid

Figure 5.15: Best fitness per combination of mutation rate p_{mut}, population size μ and number of generations g_{max} with $\mu \times g_{max} = 5{,}000$ evaluations per setup. Medians are indicated by red bars [42].

significant differences for population sizes of ten to 40 and 125 to 500 generations for all mutation rates (MW with BC, $p < 0.01$). Higher population sizes and thus fewer generations may lead to significantly lower fitness than some other hyperparameter settings.

In summary, we find that for both grid sizes (i.e., swarm densities) low population sizes and many generations are best for evolving self-assembly behaviors with minimize surprise. Mutation rate and number of simulation runs per fitness evaluation do not significantly impact fitness. Evaluation length requires a greater minimum time with increasing swarm density to not affect fitness negatively, probably because agents may need more time to self-assemble into a structure due to the sparser swarm density.

5.5 Discussion and Conclusion

Our in-depth analysis has shown that we can evolve robust, scalable, and diverse high-quality behaviors with our task-independent minimize surprise approach. The evolutionary process is robust to noise, which is an important prerequisite for experiments with real robots. In addition, emergent behaviors are robust to disturbances, such as the damage of the assembled structure in our self-assembly scenario. Since behaviors are scalable with swarm density, the effort to generate swarm behaviors for different densities can be reduced. Furthermore, the comparison with novelty search and MAP-Elites shows that our standard minimize surprise approach leads to high behavioral diversity and solution quality. However, the combination of our minimize surprise reward with quality-diversity algorithms instead of simple evolutionary algorithms allows for similar or even higher behavioral diversity and competitive solution quality. Thus, we will put more focus on it in future work. Last but not least, we found that fitness (i.e., prediction accuracy) is best when running evolution for many generations using a small population size, while short evaluation lengths and few repetitions are sufficient. Thus, carefully selecting hyperparameters can lead to higher prediction accuracy and speed up evolution.

Engineered Self-Organization | 6

Chapter Contents

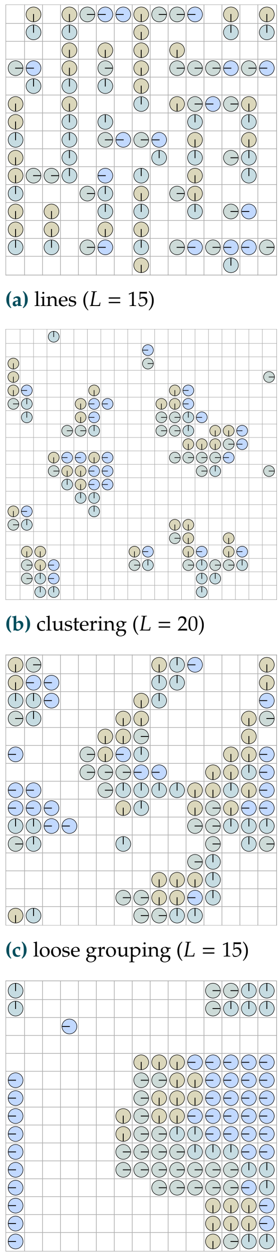
In this chapter, we engineer self-organization to bias emergence with minimize surprise towards desired behaviors and exemplify the approach in our self-assembly scenario. We...

- *Sec. 6.1:* partially predefine predictions to bias emergence towards grouping behaviors and lines,
- *Sec. 6.2:* fix all predictions to predefine emergence to grouping behaviors or lines, and
- *Sec. 6.3:* draw a conclusion.

Parts of this chapter are based on [38–42].

In our previous self-assembly experiments (see Ch. 4), we have used minimize surprise with complete freedom, that is, we relied completely on the innate motivation to maximize prediction accuracy. Consequently, we had no direct influence on the emergent behaviors. In this chapter, we present variants of our standard minimize surprise approach (see Ch. 3) that enable us to push evolution towards the emergence of desired behaviors (research question Q2, Sec. 1.2) and show its competitiveness to evolutionary algorithms with task-specific fitness functions (research question Q3). Assuming that a certain behavior is associated with a specific sensor input pattern, we can influence the evolution of behaviors with minimize surprise by predefining what we want an agent to predict and thus also to perceive. This means that agents have prior beliefs about what they expect to perceive in their environment (cf. the 'Dark-Room Problem' in Sec. 3.1). To elaborate, we engineer self-organization by predefining some or all of the agents' predictor outputs $\tilde{s}_0, \ldots, \tilde{s}_{R-1}$ (see Fig. 4.2b) to fixed, desired values while still rewarding high prediction accuracy (Eq. 3.1). High fitness can then be achieved by good predictions of the unfixed outputs $\tilde{s}_r$ (if any) and by appropriate behaviors that create sensor input satisfying the fixed and unfixed sensor predictions. Thus, we push evolution towards the emergence of desired behaviors without having to tailor a task-specific fitness function [20].

Without the loss of generality, we exemplify our approach to engineer self-organization by (partially) predefining sensor predictions in minimize surprise by aiming for the emergence of grouping behaviors and lines on the 15×15 and 20×20 grids in our self-assembly scenario (see Sec. 4.1). If not indicated otherwise, we keep the setup for our self-assembly scenario including all parameters (see Tab. 4.1) as before. We do 50 independent evolutionary runs per setting and evaluate



(a) lines (L = 15)

(b) clustering (L = 20)

(c) loose grouping (L = 15)

(d) aggregation (L = 15)

Figure 6.1: Resulting behaviors in the self-assembly scenario using minimize surprise with partially predefined predictions aiming for grouping behaviors and lines. Agents are represented by circles, their color and the lines indicate their headings.

the resulting behaviors based on our metrics presented in Sec. 4.2.2.

6.1 Partially Predefining Sensor Predictions

First, we partially predefine predictions, that is, we fix some of the agent's predictor outputs $\tilde{s}_r$ to desired values while leaving all other predictor outputs unfixed. As mentioned before, we still reward high prediction accuracy (Eq. 3.1). Consequently, agents need good predictions of the unfixed outputs and behaviors matching both the fixed and the unfixed predictor outputs to reach high fitness.

Experimental Setup We exemplify our approach of minimize surprise with partially predefined predictions (`MS-PP`) by aiming to bias emergence towards lines and grouping behaviors (i.e., aggregation, clustering, and loose grouping here[1]) in our self-assembly scenario, see Fig. 6.1. For this, we set the predictions of the sensors in front and behind the agent to 1 (i.e., $\tilde{s}_0 = \tilde{s}_3 = \tilde{s}_8 = \tilde{s}_{11} = 1$, see Fig. 6.2). The predictor ANN still has to predict the other ten sensor values. It has ten output neurons, as well as 15 input neurons (i.e., 14 sensor values and one action value, see Sec. 4.1) and 12 hidden neurons. The actor ANN is left as before. Since we reward high prediction accuracy for both the fixed and the unfixed predictor outputs, our minimize surprise fitness function (Eq. 3.1) adapts to

1: Swirls are not considered, since they require exact positioning and headings of groups of four agents. In contrast, aggregation, clustering, and loose grouping are all based on clusters of at least seven agents with less strict rules regarding positioning and heading (see Sec. 4.2.2).

$$F_{\text{MS-PP}} = \frac{1}{TNR}\sum_{t=0}^{T-1}\sum_{n=0}^{N-1}\Bigg(\overbrace{\sum_{r\in\{0,3,8,11\}}(1-|\underbrace{1}_{\tilde{s}_r^n(t)=1}-s_r^n(t)|)}^{\text{predefined predictions: occupied grid cells}} + \underbrace{\sum_{r\in\{1,2,4,5,6,7,9,10,12,13\}}(1-|\underbrace{\tilde{s}_r^n(t)}_{\text{predictor output}}-s_r^n(t)|)}_{\text{unfixed predictions}}\Bigg), \tag{6.1}$$

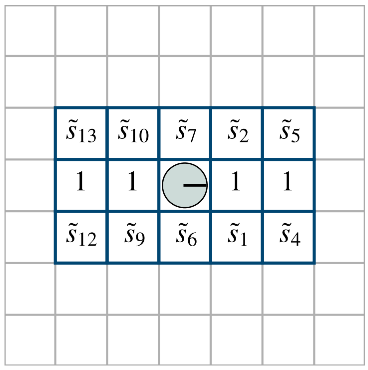
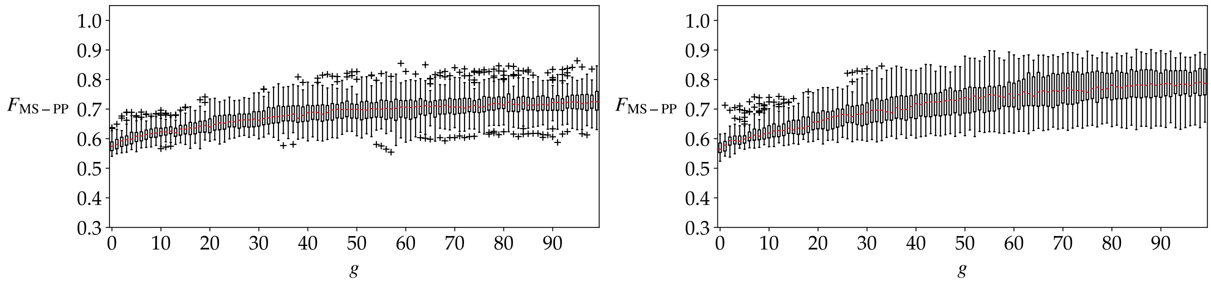


Figure 6.2: Partially predefined sensor value predictions aiming to bias emergence towards grouping behaviors and lines in our self-assembly scenario. Unfixed sensor predictions are given by $\tilde{s}_r$, the circle represents the agent, and the line indicates its heading.

with evaluation length T in time steps, swarm size N, number of sensors per swarm member R, and prediction $\tilde{s}_r^n(t)$ for and actual value $s_r^n(t)$ of sensor r of agent n at time step t. By partially predefining sensor predictions in this way, we facilitate the emergence of lines and grouping behaviors as predicting no more neighbors than the predefined ones (i.e., predicting 0 for the ten unfixed predictor outputs) matches exactly the predictions for line structures (see Fig. 4.22f) while predicting several more neighbors matches the predictions for grouping behaviors (see Figs. 4.22a – 4.22c). All other



(a) 15 × 15 grid **(b)** 20 × 20 grid

Figure 6.3: Best fitness $F_{\text{MS-PP}}$ (Eq. 6.1) over generations g of 50 independent evolutionary runs using minimize surprise with partially predefined predictions aiming for lines and grouping behaviors on grid sizes $L \in \{15, 20\}$ in our self-assembly scenario. Medians are indicated by the red bars.

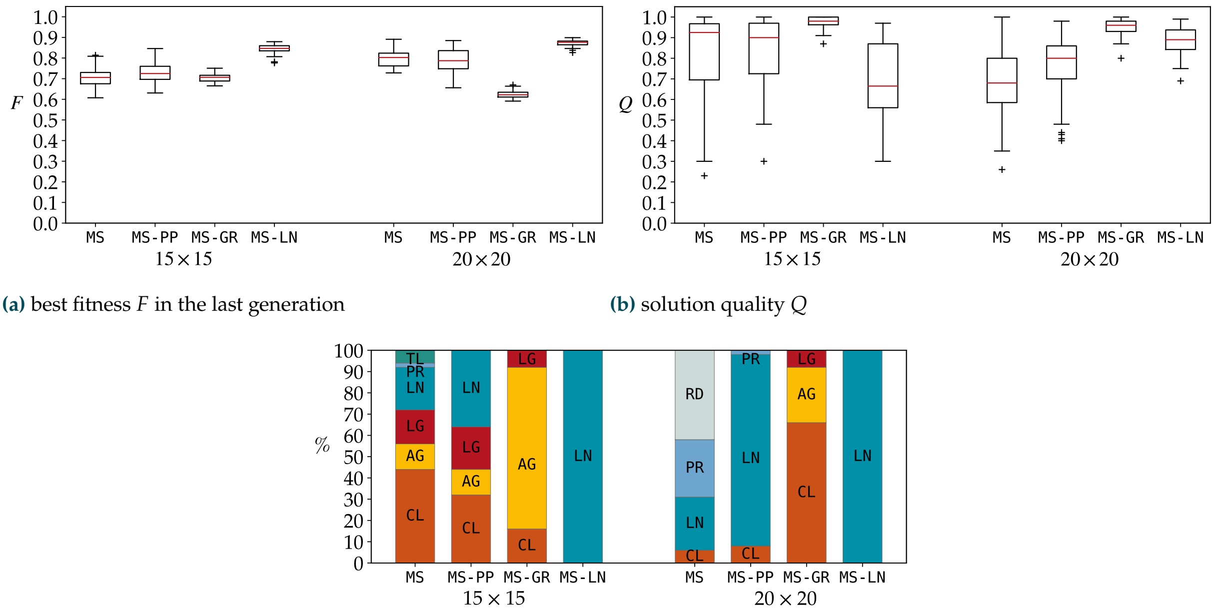
behaviors need some of the predefined predictions to be 0 (instead of 1), see Fig. 4.22.

In almost all other respects, we keep the setup for our self-assembly scenario as before except for the mutation rate for the 20×20 grid, which we had to set to 0.3 to obtain a converging fitness curve. This was probably necessary because task difficulty may be increased on the 20 × 20 grid. To determine the success of our approach, we investigate, in particular, the impact of partially predefining sensor predictions on the resulting behavior distributions.

Results The increase in best fitness over generations for both grid sizes is visualized in Fig. 6.3. The median best fitness (Eq. 6.1) of the last generation is 0.72 for the 15 × 15 grid and 0.78 for the 20 × 20 grid. Compared to our standard minimize surprise approach (`MS`), we find significantly higher fitness on the 15 × 15 grid and similar fitness on the 20 × 20 grid when partially predefining predictions (MW-U with BC, $p < 0.05$), see Fig. 6.4a.[2] Furthermore, we find median solution qualities (Eq. 4.11) of 0.83 on the 15 × 15 grid and of 0.76 on the 20×20 grid when partially predefining predictions (see Fig. 6.4b). This means that at least 76 % of the swarm assembles into the dominant structure. Compared to our standard minimize surprise, we find similar solution quality on the 15 × 15 grid and statistically significantly greater solution quality on the 20×20 grid when partially predefining predictions (MW-U with BC, $p < 0.01$).

2: The results for minimize surprise with predefined predictions aiming for grouping behaviors (`MS-GR`) and lines (`MS-LN`) will be discussed in detail in Sec. 6.2.

Fig. 6.4c visualizes the resulting behavior distributions for the standard minimize surprise approach and for minimize surprise with partially predefined predictions. As we have discussed in Sec. 4.4, we already find a majority of lines and grouping behaviors on the 15 × 15 grid when running our standard minimize surprise approach with complete freedom, but only 31 % of the runs lead to those behaviors on the 20 × 20 grid. Thus, we push evolution towards the search for solutions that are otherwise rarely found by partially predefining predictions. Accordingly, the behavior distributions



(a) best fitness F in the last generation

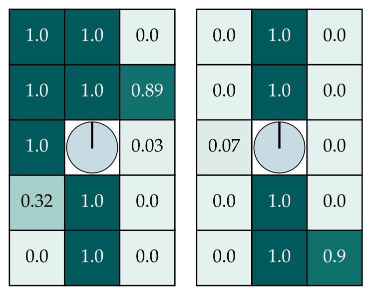
(b) solution quality Q

(c) behavior distributions

Figure 6.4: Fitness F (Eqs. 3.1, 6.1, 6.2, and 6.4, respectively), solution quality Q (Eq. 4.11), and behavior distributions of the best evolved individuals of 50 independent evolutionary runs using minimize surprise (`MS`), minimize surprise with partially predefined predictions (`MS-PP`; Sec. 6.1), and minimize surprise with predefined predictions aiming for grouping behaviors (`MS-GR`; Sec. 6.2.1) and lines (`MS-LN`; Sec. 6.2.2) per grid size $L \in \{15, 20\}$ in our self-assembly scenario. The behavior distributions give the percentage of resulting structures with clustering (`CL`), aggregation (`AG`), loose grouping (`LG`), lines (`LN`), pairs (`PR`), triangular lattices (`TL`), and random dispersion (`RD`). The data for minimize surprise (`MS`) is reprinted from Ch. 4 for easier comparison. Medians are indicated by the red bars in the box plots.

for minimize surprise and minimize surprise with partially predefined predictions are similar on the 15×15 grid but statistically significantly different on the 20×20 grid (FE with BC, $p < 0.001$). For minimize surprise with partially predefined predictions, we find 64 % grouping behaviors (i.e., clustering, aggregation, loose grouping) and 36 % lines on the 15×15 grid. Compared to the runs with the standard minimize surprise approach, the formation of line structures increases by 16 percentage points (pp) and grouping behaviors decrease by 8 pp while pairs and triangular lattices do not emerge anymore. On the 20×20 grid, we find 90 % lines, 2 % pairs, and 8 % clustering when partially predefining predictions. Compared to our standard minimize surprise approach, we notice an increase of 2 pp in clustering and of 65 pp in the formation of lines. Pairs form only in one run and no random dispersion as well as no triangular lattices emerge anymore.

As before (see Sec. 4.4), we study the mean sensor predictions of the best evolved individuals when partially predefining sensor predictions and compare them to the formed structures. In our study, we focus on the ten sensors that are still predicted by the predictor ANN here. In the emergent grouping behaviors, we observe that the mean sensor predictions of all agents are above 0.5 for at least three of the ten unfixed sensor predictions on the 15×15 grid and for at least four sensor predictions on the 20×20 grid (see Fig. 6.5a). A



(a) clustering (see Fig. 6.1b)

(b) lines (see Fig. 6.1a)

Figure 6.5: Mean sensor predictions in the self-assembly scenario with partially predefined predictions aiming for grouping behaviors and lines. The four sensor predictions directly in front and behind the agent are predefined to 1, all other sensor values are still predicted by the predictor ANN. Agents are represented by circles and the lines indicate their headings.

mean sensor prediction above 0.5 states that a grid cell is predicted by all agents to be occupied in at least half of the time steps. Similarly, for line structures maximally two of the sensors on the 15×15 grid and maximally three sensors on the 20×20 grid are on average predicted above 0.5, since the predefined predictions already match the occupied grid cells in the structure (see Fig. 6.5b).

Overall, we find a decrease in the variety of resulting structures by partially predefining sensor predictions while emergent behaviors still have high fitness and solution quality. As expected, we could push emergence towards lines and grouping behaviors by predefining some of the sensor values even on the 20×20 grid where these behaviors are little found when using the standard minimize surprise approach. Thus, we successfully engineered an influence on the emergence of structures. However, we cannot avoid a dependence on swarm density as in our previous experiments (see Sec. 4.4). Grouping behaviors generally rarely form in low swarm densities and thus lines prevail on the 20×20 grid when partially predefining predictions.

6.2 Predefining All Sensor Predictions

Next, we predefine all sensor predictions, that is, we fix all of the agents' predictor outputs $\tilde{s}_r$ to desired values but continue to reward prediction accuracy (Eq. 3.1). Consequently, we remove the predictor network because it has become superfluous. High fitness can be achieved by behaviors that create sensor input satisfying the fixed sensor predictions. Thus, we predefine the emergence of desired behaviors similarly to when using an evolutionary algorithm with a task-specific reward that relies on local information only.

Since we biased emergence towards lines and grouping behaviors in our previous experiment on minimize surprise with partially predefined predictions, we also aim for grouping behaviors and for line structures in the following experiments. The predefined predictions then serve as local templates for the targeted structures in our self-assembly scenario, which is similar to approaches used in cellular automata research [173] (see Sec. 2.2.1). In our first experiment, we aim for grouping behaviors and show that minimize surprise with predefined predictions is competitive to an evolutionary algorithm with a task-specific reward. In the second experiment, we aim for lines and show how the emergence of behaviors can be influenced further by changing the geometry of the environment.

6.2.1 Grouping Behaviors

First, we predefine emergence to grouping behaviors (i.e., aggregation, clustering, loose grouping) and show the competitiveness of our minimize surprise approach with predefined predictions to an evolutionary algorithm with a task-specific fitness function [40, 41].

Experimental Setup

We evolve grouping behaviors (i.e., clustering, aggregation, and loose grouping) in our self-assembly scenario using the previously presented evolutionary algorithm (see Sec. 4.1) with two different fitness functions: (i) the minimize surprise fitness function with predefined predictions (`MS-GR`), and (ii) a task-specific fitness function as a baseline.

Minimize Surprise with Predefined Predictions In our first experiment on minimize surprise with predefined predictions, we fix the predictions of all sensor values to 1 (i.e., $\tilde{s}_r = 1$, $r \in [0 \,..\, R-1]$; see Fig. 6.6) while still rewarding high prediction accuracy to predefine emergence to grouping behaviors (`MS-GR`). Thus, our minimize surprise fitness function (Eq. 3.1) adapts to

$$F_{\text{MS-GR}} = \frac{1}{TNR} \sum_{t=0}^{T-1} \sum_{n=0}^{N-1} \sum_{r=0}^{R-1} (1 - | \overbrace{\underset{\uparrow\, \tilde{s}^n_r(t)\,=\,1}{\boxed{1}}}^{\substack{\text{predefined predictions:} \\ \text{occupied grid cells}}} - s^n_r(t)|) \,, \qquad (6.2)$$

with evaluation length T in time steps, swarm size N, number of sensors per swarm member R, and value $s^n_r(t)$ of sensor r of swarm member n at time step t. In comparison to our fitness function with partially predefined predictions (Eq. 6.1), all sensor predictions $\tilde{s}^n_r(t)$ are set to fixed values here. High fitness is reached when agents detect many neighbors as common in grouping behaviors.

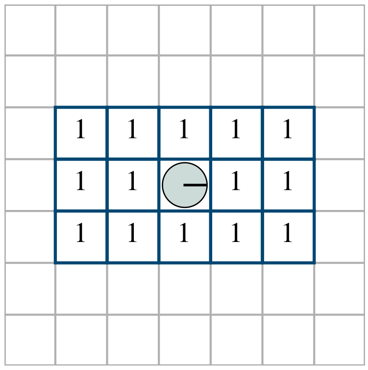
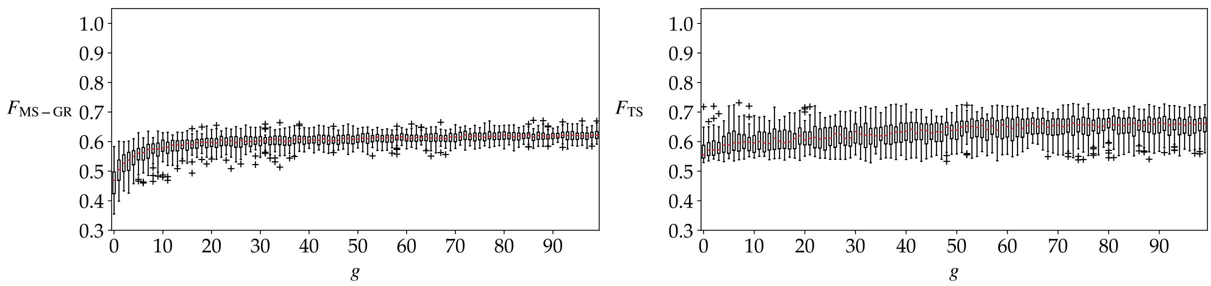


Figure 6.6: Predefined sensor value predictions for grouping behaviors in our self-assembly scenario. All sensor predictions are predefined to 1 here. The circle represents the agent and the line indicates its heading.

Task-Specific Fitness Function As a baseline, we use our evolutionary algorithm (see Sec. 4.1) with a task-specific fitness function aiming for the emergence of grouping behaviors. Our task-specific fitness function rewards shorter mean distances to the center of (agent) mass in the last time step T of an evaluation, which is a common choice for evolving aggregation behaviors [133, 134]. We define fitness F_{TS} as

$$F_{\text{TS}} = 1 - \frac{1}{LN} \sum_{n=0}^{N-1} d_M(\text{CoM}(T), P_n(T)) \,, \qquad (6.3)$$



(a) minimize surprise with predefined predictions

(b) evolutionary algorithm with the task-specific fitness function

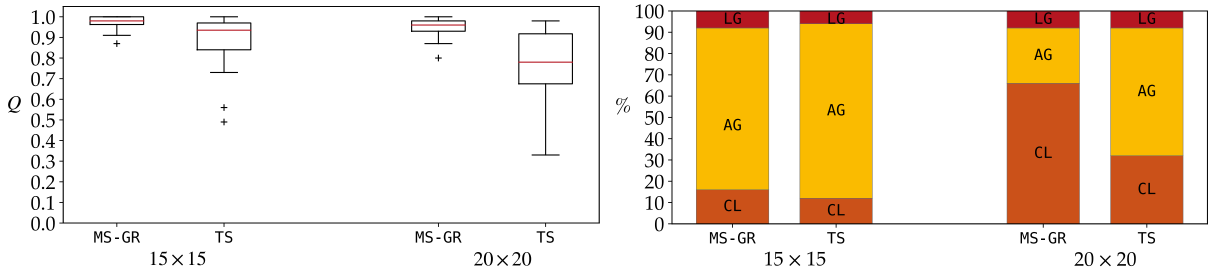
Figure 6.7: Best fitness over generations g of 50 independent runs each of minimize surprise with predefined predictions for grouping behaviors ($F_{\text{MS-GR}}$; Eq. 6.2) and an evolutionary algorithm with the task-specific fitness function F_{TS} (Eq. 6.3) on the 20×20 grid in our self-assembly scenario. Medians are indicated by the red bars.

with swarm size N, grid size L, Manhattan distance d_M (Eq. 4.5), center of agent mass $\mathrm{CoM}(T)$, and position $P_n(T)$ of agent n in the last time step T of the evaluation. We calculate the center of mass on our 2D grid with periodic boundary conditions using the algorithm by Bai and Breen [363]. Distances between center of mass and agents are calculated using the Manhattan distance as agents cannot move diagonally in our self-assembly scenario (see Sec. 4.2). We normalize the Manhattan distance between agent and center of mass by the maximum possible distance, that is, grid side length L for square grids.[3] Therefore, fitness is theoretically normalized to $[0, 1]$, but since each grid cell can be occupied by maximally one agent, the maximum possible fitness is 0.69 on the 15×15 grid and 0.76 on the 20×20 grid.

3: The maximum Manhattan distance between two points on a torus is $\frac{L_x}{2} + \frac{L_y}{2}$.

Results

The increase in best fitness over generations for both minimize surprise with predefined predictions and the evolutionary algorithm with the task-specific fitness function on the 20×20 grid are visualized in Fig. 6.7. For the approach with the task-specific fitness function, the median best fitness (Eq. 6.3) of the last generation is 0.63 for the 15×15 grid and 0.66 for the 20×20 grid. Thus, fitness is optimized successfully by evolution. For minimize surprise with predefined predictions, we reach median best fitness (Eq. 6.2) of 0.71 on the 15×15 grid and of 0.63 on the 20×20 grid. We do not compare the fitness of both approaches due to the differing fitness functions. But when comparing minimize surprise with predefined predictions and the standard minimize surprise approach, we find similar fitness on the 15×15 grid and significantly higher fitness for the standard minimize surprise approach on the 20×20 grid (MW-U with BC, $p < 0.001$), see Fig. 6.4a. Due to the low swarm density, agents may need more time to assemble into groups on the 20×20 grid, making the prediction task harder and thus leading to lower fitness. This is also in line with our finding that grouping behaviors only



(a) solution quality Q

(b) behavior distributions

Figure 6.8: Solution quality Q (Eq. 4.11) and behavior distributions of the best evolved individuals of 50 independent evolutionary runs each using minimize surprise with predefined predictions (MS-GR) and an evolutionary algorithm with a task-specific fitness function (TS) aiming for grouping behaviors per square grid size $L \in \{15, 20\}$ in the self-assembly scenario. The behavior distributions give the percentage of resulting structures with clustering (CL), aggregation (AG), and loose grouping (LG). Medians are indicated by the red bars in the box plots.

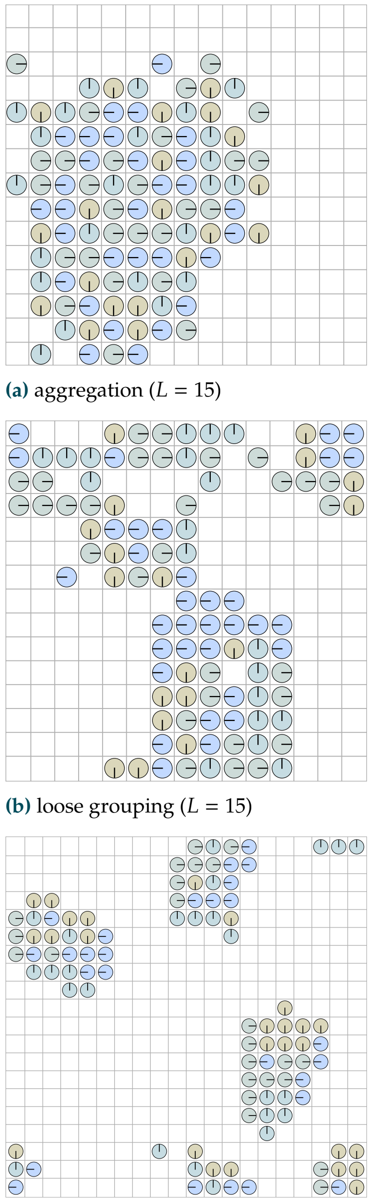
rarely emerge for standard minimize surprise and minimize surprise with partially predefined predictions on the 20×20 grid (see Fig. 6.4c).

We find median solution qualities Q (Eq. 4.11) of 0.98 on the 15×15 grid and of 0.94 on the 20×20 grid when predefining predictions. For the approach with the task-specific fitness function, we find median solution qualities of 0.96 on the 15×15 grid and of 0.78 on the 20×20 grid, see Fig. 6.8a. On both grids, the solution quality of the emergent behaviors is significantly higher for minimize surprise with predefined predictions than for the evolutionary algorithm with the task-specific fitness function (MW-U with BC, $p < 0.001$). Thus, more agents assemble into the dominant structure in the behaviors evolved with minimize surprise with predefined predictions.

As expected, both the evolutionary algorithm with the task-specific fitness function and minimize surprise with predefined predictions lead only to grouping behaviors (i.e., clustering, aggregation, loose grouping, see Fig. 6.9), see Fig. 6.8b. Thus, both approaches lead to statistically significantly different behavior distributions than the standard minimize surprise approach (FE with BC, $p < 0.01$). However, the behavior distributions for the evolutionary algorithm with the task-specific fitness function and minimize surprise with predefined predictions are similar on the 15×15 grid but statistically significantly different on the 20×20 grid. The approach with the task-specific fitness function leads to a majority of aggregation behaviors on both grid sizes. By contrast, in minimize surprise with predefined predictions, we mainly find aggregation on the 15×15 grid, but clustering prevails on the 20×20 grid. The different nature of the fitness functions seems to cause this difference. The task-specific fitness function has a global view on aggregation by calculating the distance to the center of mass (Eq. 6.3). Contrarily, the fitness function of minimize surprise with predefined sensor predictions (Eq. 6.2) equates to using a local template for behaviors, here the grouping behaviors. It does not differentiate between aggregation, clustering, and loose grouping, and even distant clusters can have high fitness. Clustering emerges more easily on the larger grid due to the low swarm density while aggregation can easily form in the high swarm density of the smaller grid. Consequently, the combination of swarm density and fitness function leads to the differences in the behavior distributions.

In the last part of our comparison, we compare the runtime of minimize surprise with predefined predictions to the runtime of the evolutionary algorithm with the task-specific fitness function. We run each approach five times on a MacBook Pro (2017) with a 3.1 GHz Intel Core i5 processor (7th generation) and 16 GB RAM for the 15×15 grid case. Both the evolutionary algorithm with the task-specific fitness function and minimize surprise with predefined predictions have a runtime of approximately 23 min each and are thus competitive in that regard.

Overall, predefining predictions in our minimize surprise approach allows us to predefine emergence to desired behaviors and leads to high-quality solutions. As a result, minimize surprise with predefined predictions offers a more intuitive way to target specific behaviors than defining a task-specific fitness function. Task-specific fitness functions may even require global information, such as the center of agent mass in our scenario here, while minimize surprise with predefined predictions relies on local information only. In consequence, swarm density has a greater impact in our minimize surprise approach than in the evolutionary algorithm with the task-specific fitness function.



(a) aggregation ($L = 15$)

(b) loose grouping ($L = 15$)

(c) clustering ($L = 20$)

Figure 6.9: Resulting behaviors in the self-assembly scenario using minimize surprise with predefined predictions for grouping. Agents are represented by circles, their color and the lines indicate their headings.

6.2.2 Lines

In a second experiment with minimize surprise with predefined predictions, we predefine the emergence of lines (`MS-LN`). Furthermore, we investigate the influence of the geometry of the environment and the initial agent poses on the emergent structures. A video illustrating our experiments is online.[4]

4: `https://youtu.be/XYZgqPYt-kY`

Experimental Setup

We fix the sensor predictions for the sensors directly in front and behind the agent to 1 (i.e., $\tilde{s}_0 = \tilde{s}_3 = \tilde{s}_8 = \tilde{s}_{11} = 1$) and all other predictor outputs to 0, see Fig. 6.10. This matches the real and predicted sensor values of agents assembled into line structures, see Fig. 4.22f. Consequently, our minimize surprise fitness function (Eq. 3.1) adapts to

$$F_{\text{MS-LN}} = \frac{1}{TNR}\sum_{t=0}^{T-1}\sum_{n=0}^{N-1}\Bigg(\overbrace{\sum_{r\in\{0,3,8,11\}} \left(1-|\underbrace{1}_{\tilde{s}_r^n(t)=1} - s_r^n(t)|\right)}^{\text{predefined predictions: occupied grid cells}} + \underbrace{\sum_{r\in\{1,2,4,5,6,7,9,10,12,13\}} \left(1-|\underbrace{0}_{\tilde{s}_r^n(t)=0} - s_r^n(t)|\right)}_{\text{predefined predictions: empty grid cells}} \Bigg), \tag{6.4}$$

with evaluation length T in time steps, swarm size N, number of sensors per swarm member R , and value $s_r^n(t)$ of sensor r of agent n at time step t.

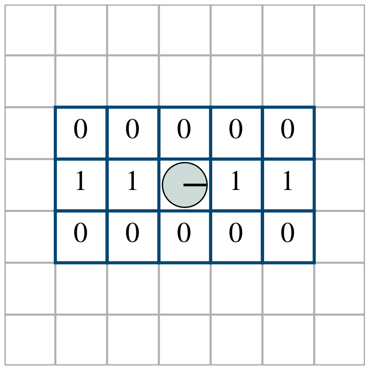
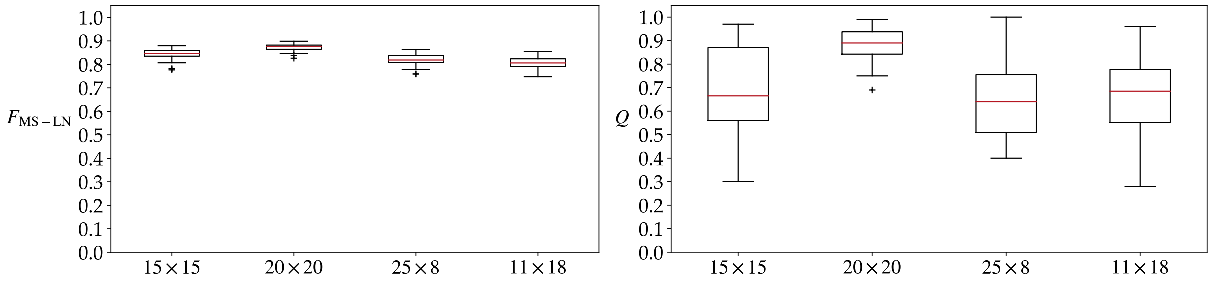


Figure 6.10: Predefined sensor value predictions for lines. Agents predict to sense neighboring agents directly in front and behind them while they do not expect neighbors an any other grid cell in their sensor view. The circle represents the agent and the line indicates its heading.

As before, we test our approach on square grids with side lengths $L \in \{15, 20\}$. Additionally, we investigate the influence of the geometry of our torus grids (i.e., the ratio of one diameter to the other) by using rectangular fundamental polygons. Thereby, we bias emergence towards the formation of horizontal or vertical lines. First, we use a 25×8 grid, which leads to a swarm density of 0.5 (i.e., 50 % of the grid cells are occupied by agents). The agents can either self-assemble into horizontal lines (in x-direction along the longer diameter of the torus) or vertical lines (in y-direction along the shorter diameter of the torus) to reach maximum fitness values in this scenario. Second, we use a 11×18 grid resulting in a swarm density of approximately 0.51. Theoretically, the best possible fitness can be reached if the agents self-assemble into nine horizontal lines (in x-direction along the shorter diameter of the torus) with 11 agents each and thus one agent would be left without a proper spot. Thus, we increase the bias towards horizontal lines here. We then study the effect of the environment's geometry on the emergent line structures by comparing the quantity of behaviors leading to horizontal, vertical, and maze-like line structures on all four grid sizes (i.e., 15×15, 20×20, 25×8, 11×18). Formed line structures are categorized as mostly horizontal or mostly vertical when more than two thirds of the resulting lines are formed horizontally or vertically, respectively. Otherwise the formed line structure is categorized as maze-like.

Results

As expected, lines emerge in all runs on all four grid sizes when predefining predictions as described above. Consequently, we successfully predefined emergence by setting sensor predictions to fixed values as in our previous experiment when aiming for grouping behaviors. In the following, we study the resulting line structures in more detail.



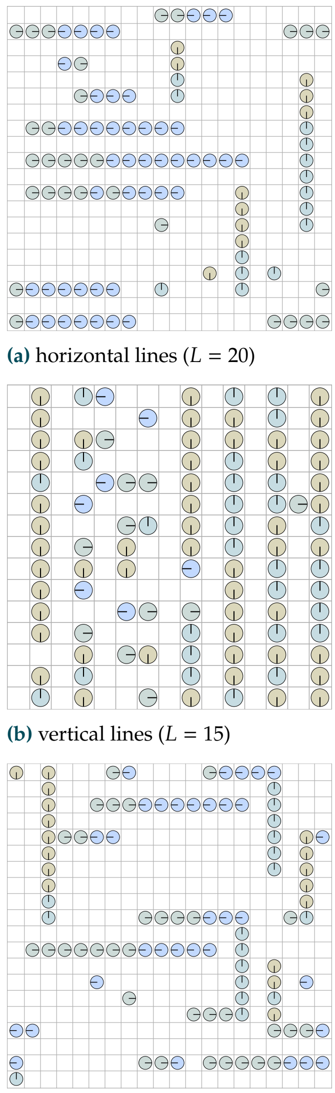
(a) best fitness $F_{\text{MS-LN}}$ in the last generation

(b) solution quality Q

Figure 6.12: Fitness $F_{\text{MS-LN}}$ (Eq. 6.4) and solution quality Q (Eq. 4.11) of the best evolved individuals of 50 independent evolutionary runs using minimize surprise with predefined predictions aiming for lines for grid sizes 15×15, 20×20, 25×8, and 11×18 in the self-assembly scenario. Medians are indicated by the red bars.

Square Grids On the square grids, we find a median best fitness $F_{\text{MS-LN}}$ (Eq. 6.4) of 0.85 for the 15×15 grid and of 0.88 for the 20×20 grid in the last generation, see Fig. 6.12a. Furthermore, we find median solution qualities Q (Eq. 4.11) of 0.67 on the 15×15 grid and of 0.89 on the 20×20 grid (see Fig. 6.12b).

On the 15×15 grid, the best evolved individuals let agents self-assemble into mostly horizontal lines and into mostly vertical lines in 28 % of the cases each (see Fig. 6.11). The remaining 44 % of runs result in maze-like line structures. On the 20×20 grid, agents self-assemble in 22 % of the runs into mostly horizontal lines and in 22 % of the runs into mostly vertical lines. In the remaining 56 % of the runs, agents form maze-like line structures. Thus, the ANN pairs evolved on the square torus grids lead to the formation of a variety of line structures.



(a) horizontal lines ($L = 20$)

(b) vertical lines ($L = 15$)

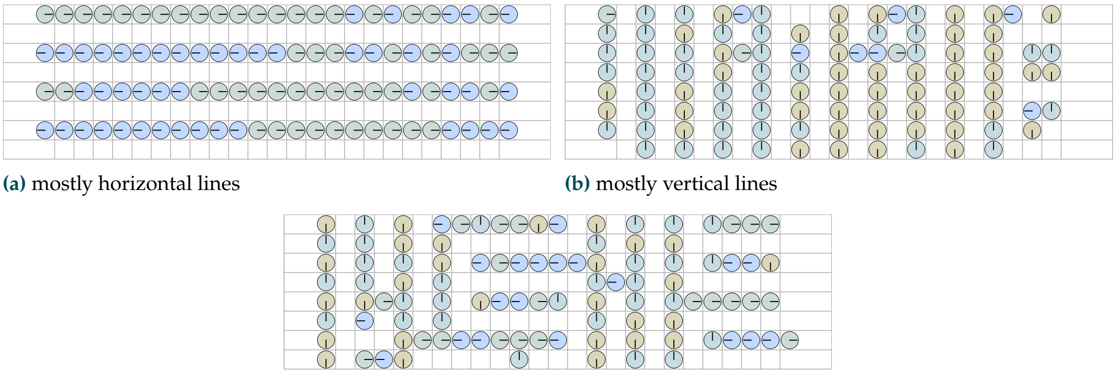
(c) maze-like lines ($L = 20$)

Figure 6.11: Resulting behaviors in the self-assembly scenario on square grids using minimize surprise with predefined predictions for lines. Agents are represented by circles, their color and the lines indicate their headings.

25×8 Grid We find a median best fitness $F_{\text{MS-LN}}$ (Eq. 6.4; see Fig. 6.12a) of 0.82 in the last generation and a median solution quality Q (Eq. 4.11; see Fig. 6.12b) of 0.64 on the 25×8 grid.

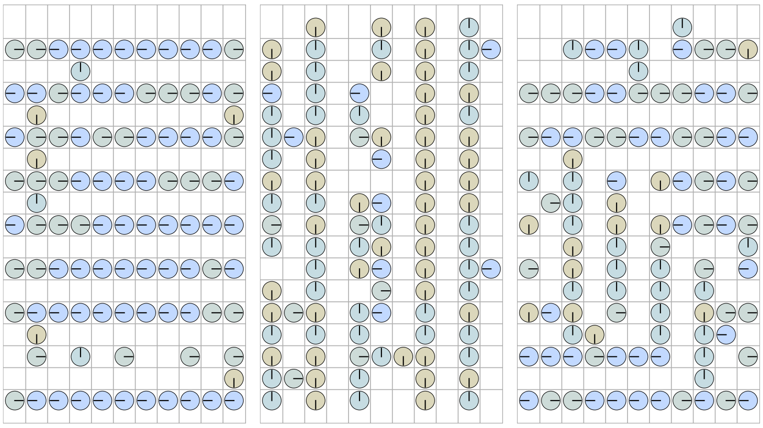
The best evolved individuals let agents self-assemble into mostly vertical lines in 32 % and into mostly horizontal lines in 16 % of the cases, see Figs. 6.13a and 6.13b. The remaining 52 % of runs result in maze-like line structures, see Fig. 6.13c. Compared to the 15×15 grid, which has a similar swarm density, we observe an increase in the formation of vertical lines and a decrease in the formation of horizontal lines on the 25×8 grid. The overall formation of horizontal and vertical lines decreases slightly from 56 % to 48 %. Thus, we find only a small effect of the rectangular grid size on the formation of horizontal and vertical lines overall, but it leads to an increase in the formation of vertical lines.

11×18 grid We find a median best fitness $F_{\text{MS-LN}}$ (Eq. 6.4; see Fig. 6.12a) of 0.81 in the last generation and a median



(a) mostly horizontal lines (b) mostly vertical lines (c) maze-like line structures

Figure 6.13: Resulting behaviors in the self-assembly scenario on the 25×8 grid using minimize surprise with predefined predictions for lines. Agents are represented by circles, their color and the lines indicate their headings.



(a) mostly horizontal lines (b) mostly vertical lines (c) maze-like line structures

Figure 6.14: Resulting behaviors in the self-assembly scenario on the 11×18 grid with predefined predictions for lines. Agents are represented by circles, their color and the lines indicate their headings.

solution quality Q (Eq. 4.11; see Fig. 6.12b) of 0.69 on the 11×18 grid.

The best evolved individuals let agents self-assemble into mostly horizontal lines in 42 % and into mostly vertical lines in 4 % of the runs, see Figs. 6.14a and 6.14b. The remaining 54 % of runs result in maze-like line structures, see Fig. 6.14c. Consequently, we have successfully biased evolution towards horizontal lines, since they emerge frequently here.

Grid-Spanning Lines We find that the geometry of the environment influences the number of runs in which lines spanning the whole grid length are formed. On the 15×15 grid, we find grid-spanning lines in 20 % of the runs in which lines form with the standard minimize surprise approach, in 67 % when partially predefining predictions, and in 68 % when predefining all predictions. By contrast, we do not observe any grid-spanning lines on the 20×20 grid for all three minimize surprise variants. There are two potential reasons that are also related: (i) the swarm density is too low or (ii) the side length of the grid is too large to allow the formation of grid-spanning lines even when predefining emergence

grid size	total	vertical	horizontal
15×15	68	28	40
20×20	0	-	-
25×8	94	90	4
11×18	88	12	76

Table 6.1: Total percentage of grid-spanning lines and percentage of vertically and horizontally grid-spanning lines using minimize surprise with predefined predictions for lines in our self-assembly scenario.

to lines. The runs on the rectangular grids provide more insight.

While the 15×15, 25×8, and 11×18 grids have comparable swarm densities (i.e., 0.44 to 0.51), we find more runs in which grid-spanning lines are formed for the rectangular grids, see Tab. 6.1. On the 25×8 grid, 94 % of the runs lead to grid-spanning lines and 88 % of the runs on the 11×18 grid. We find both horizontally and vertically grid-spanning lines on the 25×8 grid and thus large grid side lengths do not generally prevent the formation of grid-spanning lines. However, the majority of the runs leads to vertically and only few runs to horizontally grid-spanning lines. As the vertical side length is much smaller, grid-spanning lines may be formed more easily along this shorter side. On the 11×18 grid, the majority of runs leads to horizontally grid-spanning lines while few runs lead to vertically grid-spanning lines. Here, the formation of vertical lines does not allow for maximum fitness and thus horizontal lines prevail in general. Overall, we conclude that grid-spanning lines form easier along shorter sides of the grid but also swarm density and the geometry of the environment influence their formation.

Influence of the Initial Agent Poses on Structure Orientation Last, we analyze the influence of the agents' initial poses on the orientation of the formed line structures. We re-evaluate the best evolved individuals using minimize surprise with predefined predictions for lines on the two square grids 15×15 and 20×20, and the two rectangular grids 25×8 and 11×18. We use new random starting poses in 20 independent runs per best evolved individual, that is a total of 1,000 runs per grid size.[5] An ANN pair is considered to form mostly vertical, mostly horizontal or mostly maze-like lines, respectively, if more than half of these 20 evaluations with new random initial poses lead to the formation of such line structures. Otherwise, the ANN pair is classified as forming diverse line structures.

5: Please note that this re-evaluation of the genomes with new random starting poses led to the classification of 0.7 % (7 out of 1000 runs) as random dispersion on the 15×15 grid and to pairs in 0.1 % (1 out of 1000 runs) on both the 25×8 grid and the 11×18 grid.

Both on the 15×15 grid and on the 20×20 grid, we observe a decrease in the quantity of ANN pairs that lead to the formation of lines in a certain orientation, that is, vertical, horizontal or maze-like. Diverse line structures are formed by 70 % of the best evolved individuals on the 15×15 grid and by 60 % on the 20×20 grid, see Tab. 6.2. Thus, the behaviors evolved on the square torus grids do not lead to the formation of specifically oriented line structures, but the initial agent poses influence the formation of the final structure.

Table 6.2: Percentage of mostly horizontal, mostly vertical and mostly maze-like line structures when predefining all sensor predictions for lines. Values of the 50 initial runs (one random starting position) and of the re-evaluations (20 random starting positions/initial run, 1,000 runs in total) per grid size are given.

grid size		vertical	horizontal	maze-like	diverse
15×15	initial run	28	28	44	
	re-evaluations	2	14	14	70
20×20	initial run	22	22	56	
	re-evaluations	2	2	36	60
25×8	initial run	32	16	52	
	re-evaluations	38	6	30	26
11×18	initial run	4	42	52	
	re-evaluations	0	52	36	12

By contrast, the amount of ANN pairs forming mostly vertical and mostly horizontal lines stays similar to the initial runs for the 25×8 and the 11×18 grids. In line with our previous findings, the best evolved individuals have a tendency towards vertical lines in the initial run and in the re-evaluations on the 25×8 grid. In this setting, maximum fitness can be reached only when agents assemble into grid-spanning horizontal or vertical lines. The tendency towards vertical lines probably arises because their formation of is easier; they are much shorter than grid-spanning horizontal lines. The initial agent poses have thus only a small impact on the resulting line structures in this case. On the 11×18 grid, the percentage of horizontal lines even increases by 10 pp in the re-evaluations. In this setting, maximum fitness can be reached only when agents assemble into grid-spanning horizontal lines. Thus, as expected, the majority of best evolved individuals leads to the formation of horizontal lines independent of the initial agent pose. Overall, we find that although the initial agent poses influence the orientation of the final line structure, the geometry of the environment has a stronger influence.

6.3 Discussion and Conclusion

Minimize surprise enables us to engineer self-organization by (partially) predefining predictions. We have shown that there is a gradient from running our minimize surprise approach with complete freedom (i.e., no sensor predictions are predefined; see Ch. 4) to simplifying it to a special kind of evolutionary algorithm with a task-specific fitness function (i.e., all sensor predictions are predefined). In the latter case, all outputs of the predictor are fixed, which makes the respective ANN obsolete. Fitness is then directly defined via the desired sensor pattern and the predefined predictions serve as a local template for the behavior. As shown in Fig. 6.4c, the standard minimize surprise approach has the freedom to evolve a diversity of behaviors. By partially predefining some sensor predictions, we can limit the variety of emergent behaviors to a subset. Predefining all sensor predictions can be seen as a special way of defining a task-specific fitness function that leads to the evolution of a desired behavior without requiring global information. Additionally, changing the environment, such as its geometry, enables us to bias the evolutionary process even more. In total, we can trigger the emergence of desired behaviors by engineering self-organization.

Evolution of Dynamic Behaviors in Complex Environments

7

Chapter Contents

In this chapter, we study the evolution of dynamic behaviors in complex environments with minimize surprise. We...

- *Sec. 7.1:* introduce the experimental setup and
- *Sec. 7.2:* evaluation metrics,
- *Sec. 7.3:* present our results, and
- *Sec. 7.4:* draw a conclusion.

In all previous studies using our standard minimize surprise approach (see Ch. 3), we found repetitive behaviors that are easy to predict due to constant sensor input. Consequently, the resulting behaviors are limited in behavioral plasticity, which is the ability of agents to react to variations in their sensor input with different behaviors that are suitable for the given context [364]. In this chapter, we aim to push evolution towards more dynamic behaviors (research question Q4, Sec. 1.2) by modifying environment, agent capabilities, and fitness function. We draw inspiration from studies focusing on adaptation to environmental influences and survivability. In robot ecology, Egerstedt et al. [26] focus on the survivability of agents to enable long duration autonomy. We adopt their idea of introducing ecological constraints by limiting the agent's battery making recharging necessary, which corresponds to an animal's need to find nutrition to survive. Similar to Miras et al. [365], we investigate the effect of environmental conditions by testing both a static environment (except for the behavior of the swarm itself) and an environment that is dynamic independent from agent behavior. In addition, we divide our environment into two different zones introducing spatial variability. Last, we extend our fitness function with additional task-specific and task-independent rewards to push evolution towards more dynamic behaviors.

7.1 Experimental Setup

In this section, we introduce the simulation environment and the adaptation of our minimize surprise approach for our scenario aiming for dynamic behaviors in complex environments.

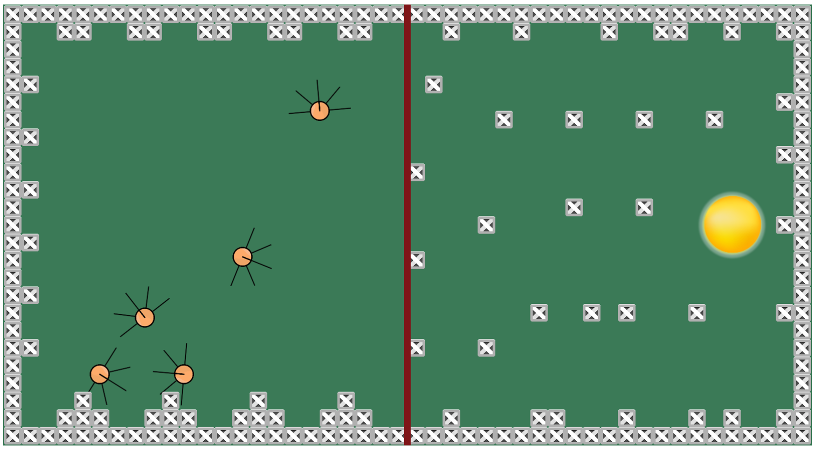


Figure 7.1: Experimental arena in our scenario evolving dynamic behaviors in complex environments. Gray squares mark immovable blocks and the yellow circle is the light source. Orange circles represent agents, black lines in the circle give their heading, and black lines around the agents give orientation and sensor range of their proximity sensors. The red line marks the division of the arena into the safe zone and the risk zone.

7.1.1 Simulation Environment

First, we present our experimental arena, the agent model, and the tested environment and agent modifications.

Arena

We use a 2D continuous simulation environment[1] that is implemented using the Python Arcade[2] library. Arcade is a framework for creating 2D video games providing an inbuilt physics system and graphics. Thus, we make a first step towards more realistic simulation environments here.

1: `https://gitlab.iti.uni-luebeck.de/minimize-surprise/dynamic-behaviors`

2: `https://arcade.academy/`

The experimental arena has a size of 1540 px × 805 px and is bounded by walls built of blocks, see Fig. 7.1. The arena is divided into two zones: a safe zone and a risk zone. The safe zone is mostly empty, but blocks along the walls are placed in unique patterns per compass direction. The risk zone contains 34 randomly placed blocks of size 35 px × 35 px that serve as obstacles. A light source of radius 120 px is placed between these obstacles. Light intensity $\mathcal{J}$ decreases with distance to the light source as given by

$$\mathcal{J} = \frac{(L_x - d_{LS})^2}{{L_x}^2}, \tag{7.1}$$

where d_{LS} is the distance to the light source along the x-axis and L_x is the arena width. We normalize light intensity $\mathcal{J}$ to be between 0 and 1, resulting in maximum light intensity $\mathcal{J} = 1$ in the center of the light source and a light intensity of approximately zero at the western arena boundary. The light source serves as a charging station for the agents' batteries.

Agents

We use circular agents with a radius $\mathscr{R}$ of 20 px, see Fig. 7.2. Agents have a battery that lasts for $T_{\text{battery}} = 2{,}000$ time steps, that is, the battery is initially fully charged to a value of $\mathscr{b}(0) = 1$ and discharges by 0.0005 per time step. When agents have contact with the light source, their batteries are recharged by 0.01 per time step (i.e., charging is 20× faster than discharging). If an agent's battery is completely discharged, the agent is stopped on its current position.

Agents have a differential drive with a maximum linear speed $v_{\max}$ of 4 px per time step and a maximum angular velocity $\mathscr{w}_{\max}$ of ±3° per time step. We restrict agents to forward movement and rotation here to simplify the setting (e.g., reducing the number of required sensors).

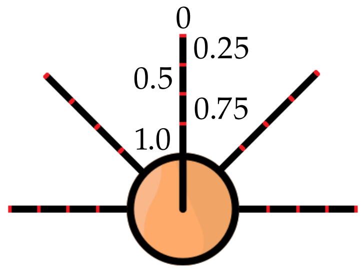


Figure 7.2: Agent model in our scenario aiming for dynamic behaviors in complex environments. The orange circle represents the agent, the black line gives its heading. Black lines with red stripes represent the agent's five proximity sensors and the gradations of the sensor values are given by the red stripes and the numbers.

Each agent has $R_{\text{sen}} = 8$ sensors: five proximity sensors $s_0, \ldots, s_4$, one light intensity sensor s_5, one compass s_6, and one sensor s_7 detecting battery level $\mathscr{b}$. Thus, we use exteroceptive and proprioceptive sensors in this setup. All sensor values s_r are in range $[0, 1]$. The proximity sensors are distributed equidistantly on the front half of the agent's circumference with the outermost sensors being along the wheel axis. Agents have no proximity sensors at their back, since linear movement is restricted to the forward direction. Each proximity sensor has a range of 105 px and outputs one of five discrete values in $\{0.0, 0.25, 0.5, 0.75, 1.0\}$. These discrete values are inversely proportional to the distance to the detected obstacles, as visualized in Fig. 7.2. We include a simple hardware protection layer to avoid collisions between agents. If an agent's forward movement with its current speed v would result in a collision with an obstacle or agent, it performs a rotation on the spot with its current angular velocity $\mathscr{w}$.

In all experiments, we use a swarm size N of 5 agents resulting in a swarm density D_N of approximately 0.01.[3]

3: $D_N = \frac{N\times\mathscr{R}^2\times\pi}{L_x\times L_y} = \frac{5\times(20\text{ px})^2\times\pi}{1540\text{ px}\times 805\text{ px}} \approx 0.00506$ with swarm size N, agent radius $\mathscr{R}$, arena width L_x, and arena length L_y.

Modifications

We test the effect of three different agent and environment modifications, and combinations thereof, on the emergent behaviors in our experiments: energy sharing (`ES`), risk of sudden discharge (`RoSD`), and changing environment (`CE`).

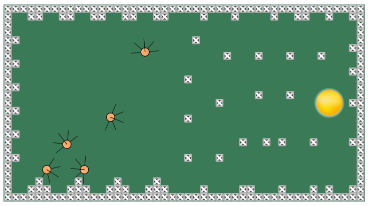
Energy Sharing (`ES`) Energy sharing enables agents to equally share their battery levels when in contact. Consequently, battery level $\mathscr{b}_n(t)$ of agent n at time step t is updated to

$$\mathscr{b}_n(t) = \frac{1}{|\mathcal{S}_n^{\text{contact}}(t)| + 1}\Big(\mathscr{b}_n(t) + \sum_{i \in \mathcal{S}_n^{\text{contact}}(t)} \mathscr{b}_i(t)\Big)\,, \tag{7.2}$$

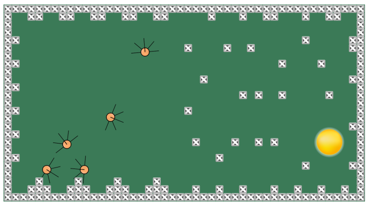
with $\mathcal{S}_n^{\text{contact}}(t)$ being the set of agents that agent n is currently in contact with. Energy sharing has two benefits: (i) agents with completely discharged batteries can be recharged, and (ii) not all agents need to move through the risk zone to keep the batteries of the entire swarm charged.

Risk of Sudden Discharge (RoSD) We increase the hazard in the risk zone by introducing a probability that the battery of an agent that is currently in the risk zone suddenly discharges completely, rendering the agent useless. Consequently, it is advantageous to stay in the risk zone only as short as needed to recharge. When risk of sudden discharge is applied in a setting, batteries of agents located in the risk zone discharge completely with a probability p_{RoSD} of 0.0013 per time step. If an agent stays for the full evaluation length in the risk zone, the total probability of sudden battery discharge is 0.4. We use this rather low total probability of sudden battery discharge in our first experiments, since we aim to increase the pressure to leave the risk zone quickly without complicating the evolution of collective behaviors too severely. Using higher probabilities could make it hard to find behaviors that allow swarms to survive over the full evaluation in the evolutionary process.

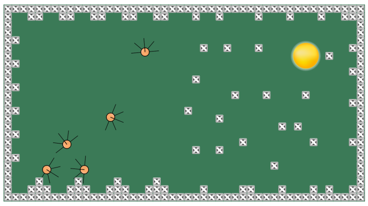
Changing Environment (CE) In our standard setup, the environment is static (i.e., fixed arena layout). By automatically changing the layout of the risk zone several times during an evaluation, we create an environment that is dynamic independent of the swarm behavior.[4] We change the positions of obstacles and the light source in equal time intervals, that is, every $\frac{1}{5}$th of the total evaluation length T, see Fig. 7.3. This increases the difficulty of predicting the risk zone further.



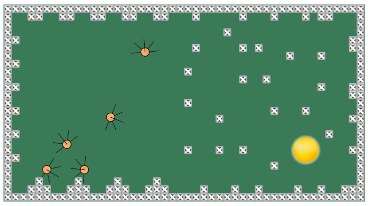
(a) layout for $t \in [0, \frac{1}{5}T)$



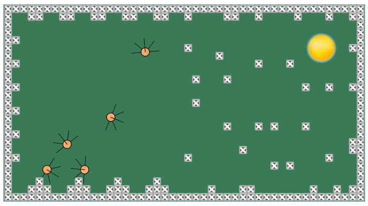
(b) layout for $t \in [\frac{1}{5}T, \frac{2}{5}T)$



(c) layout for $t \in [\frac{2}{5}T, \frac{3}{5}T)$



(d) layout for $t \in [\frac{3}{5}T, \frac{4}{5}T)$



(e) layout for $t \in [\frac{4}{5}T, T)$

Figure 7.3: Arena layouts over an evaluation of T time steps in the changing environment (CE) setup in our scenario aiming for dynamic behaviors in complex environments. Layout (a) is also the layout used over the full evaluation in experiments with a static environment.

7.1.2 Minimize Surprise

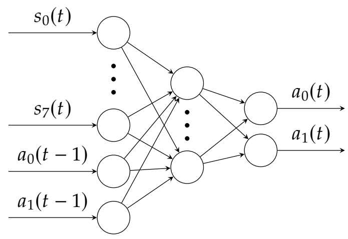
Following our minimize surprise approach (see Ch. 3), we equip each agent with an actor-predictor ANN pair. Actor and predictor are both three-layer ANNs with an input, one hidden and an output layer.

In this scenario, the actor has ten input, seven hidden, and two output neurons, see Fig. 7.4a. It receives an agent's $R_{\text{sen}} = 8$ current sensor values and its action values $a_0(t-1)$ and $a_1(t-1)$ of the previous time step $t-1$ as input and outputs two action values $a_0(t) \in [0, 1]$ and $a_1(t) \in [-1, 1]$.

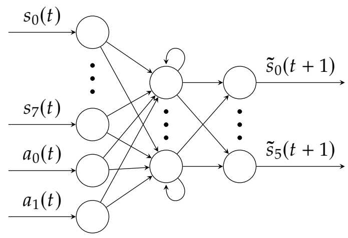
4: We investigate potentially dynamic environments in which agents move passive objects in Chs. 9 and 11.

Action value $a_0(t)$ gives the linear speed v and $a_1(t)$ the angular velocity ω. The action values are multiplied by the maximum possible linear speed $v_{\max}$ and angular velocity $\omega_{\max}$, respectively.

The predictor has ten input, eight hidden, and six output neurons, see Fig. 7.4b. It receives an agent's $R_{\text{sen}} = 8$ current sensor values and its next action values $a_0(t)$ and $a_1(t)$ as input and outputs predictions for the agent's five proximity sensor values $\tilde{s}_0(t+1), \ldots, \tilde{s}_4(t+1)$ and its light sensor value $\tilde{s}_5(t+1)$ of the next time step $t+1$. The battery level $\mathscr{b}_n$ detected by sensor s_7 linearly decreases or, when charging, increases over time and the compass value s_6 linearly changes with the rotation of the agent. Thus, both sensors are trivial to predict and excluded here, since they do not provide an incentive for more interesting behaviors.



(a) actor



(b) predictor

Figure 7.4: Actor-predictor ANN pair in the scenario aiming for dynamic behaviors in complex environments. $a_0(t-1)$ and $a_1(t-1)$ are the agent's last, and $a_0(t)$ and $a_1(t)$ are its next action values determining linear speed v and angular velocity ω, respectively. $s_0(t), \ldots, s_7(t)$ are the agent's eight sensor values at time step t. The predictor outputs predictions $\tilde{s}_0(t+1), \ldots, \tilde{s}_4(t+1)$ for an agent's five proximity sensors and $\tilde{s}_5(t+1)$ for its light sensor for time step $t+1$. Compass s_6 and battery level s_7 are trivial to predict and thus excluded here.

Charged agents propagate inputs through actor and predictor every time step. By contrast, agents with completely discharged batteries are stopped and do not predict their sensor values, which in turn prevents an increase in fitness.

In our experiments, we use two extended minimize surprise fitness functions to push evolution towards dynamic behaviors. Based on the first works in 2D environments by Borkowski and Hamann [37], we do not expect that using our standard minimize surprise fitness function would lead to more dynamic behaviors than the previously found repetitive basic swarm behaviors in our scenario here. Next to rewarding prediction accuracy (see Ch. 3), fitness function F_{MSC} has an additional task-independent reward for curiosity, while fitness function F_{MSH} additionally includes a task-specific reward for homing.

Fitness function F_{MSC} rewards curiosity in addition to prediction accuracy and thus is fully task-independent. Curiosity is the desire to know or learn [366], motivating humans [367] and robots [316, 368] to seek out for novel stimuli. Here, we push agents to seek new sensory experiences by reducing fitness when sensor values are constant for more than 150 time steps. We limit the maximum achievable fitness per time step to $1 - \xi$, with penalty factor $\xi \in [0, 1]$ linearly increasing between 150 and 350 time steps with constant sensor values (i.e., an increase of 0.005 per time step). The penalty factor is reset to zero as soon as at least one sensor value differs from its value in the previous time step. We define fitness function F_{MSC} as

$$F_{\text{MSC}} = \frac{1}{TN} \sum_{t=0}^{T-1} \sum_{n=0}^{N-1} f_n(t)\,, \tag{7.3}$$

with evaluation length T in time steps, swarm size N, and fitness of agent n in time step t

$$f_n(t) = \min\left(1 - \xi_n(t), \frac{1}{R_{\text{pred}}} \sum_{r=0}^{R_{\text{pred}}-1} (1 - |\tilde{s}_r^n(t) - s_r^n(t)|)\right), \tag{7.4}$$

with penalty factor $\xi_n(t) \in [0, 1]$ for agent n at time step t, and prediction $\tilde{s}_r^n(t)$ for and actual value $s_r^n(t)$ of sensor r of agent n in time step t for each of the five predicted sensors (see Fig. 7.4b).

Fitness function F_{MSH} combines our task-independent reward for prediction accuracy with a task-dependent reward for homing. Thus, we increase the evolutionary pressure that agents reach the light source for recharging their batteries, which can be seen as a survival instinct. We define fitness F_{MSH} as

$$F_{\text{MSH}} = \frac{1}{TN} \sum_{t=0}^{T-1} \sum_{n=0}^{N-1} \left(\mathcal{b}_n(t) \overbrace{\left(\frac{1}{R_{\text{pred}} - 1} \sum_{r=0}^{R_{\text{pred}}-1} (1 - |\tilde{s}_r^n(t) - s_r^n(t)|) \right)}^{\text{prediction accuracy}} + (1 - \mathcal{b}_n(t)) \underbrace{s_5^n(t)}_{\text{homing}} \right), \tag{7.5}$$

with evaluation length T in time steps, swarm size N, battery level $\mathcal{b}_n(t)$ of agent n in time step t, light intensity sensor value $s_5^n(t)$, and prediction $\tilde{s}_r^n(t)$ for and actual value $s_r^n(t)$ of sensor r of agent n in time step t for each of the five predicted sensors. Thus, fitness function F_{MSH} is a weighted sum of prediction accuracy and detected light intensity as a measure of the agent's distance to the light source where the weighting depends on the battery level. Proximity to the light source is rewarded the higher, the lower the battery level.

As in our previous experiments, we use a simple evolutionary algorithm (see Sec. 2.3.2) to evolve the actor-predictor ANN pairs using the two defined fitness functions F_{MSC} (Eq. 7.3) and F_{MSH} (Eq. 7.5). We run the evolutionary algorithm for $g_{\text{max}} = 150$ generations (i.e., termination criterion) and evaluate each genome in one independent simulation run for $T = 3{,}000$ time steps each. The potential overall fitness of a genome decreases with the number of completely discharged agents, since discharged agents do not output sensor predictions. An evaluation is terminated early if all agents are completely discharged before all 3,000 time steps have passed. Genomes encode the synaptic weights of both neural networks (see Sec. 3.2) and we randomly generate the initial population $\mathcal{P}(0)$ by drawing the weights from a uniform distribution in $[-1, 1]$. We place agents with a uniformly random heading uniformly random in the safe zone at the

fitness function	RoSD	ES	CE
F_{MSH}	-	-	-
F_{MSC}	-	-	-
	x	-	-
	-	x	-
	x	x	-
	-	x	x

Table 7.1: Tested settings in our scenario aiming for dynamic behaviors in complex environments with agent and environment modifications (see Sec. 7.1.1) risk of sudden death (`RoSD`), energy sharing (`ES`), and changing environment (`CE`), and fitness functions F_{MSH} (Eq. 7.5) rewarding prediction accuracy and homing and F_{MSC} (Eq. 7.3) rewarding prediction accuracy and curiosity.

parameter	value
arena size $L_x \times L_y$	1540 px × 805 px
swarm size N	5
# of sensors R_{sen}	8
# of predictor outputs R_{pred}	5
sensor values s_r	$\{0, 1\}$
action value a_0	$[0, 1]$
action value a_1	$[-1, 1]$
max. linear speed $v_{\max}$	$4\ \frac{\text{px}}{t}$
max. angular velocity $w_{\max}$	$\pm 3° \frac{1}{t}$
population size μ	50
number of generations $g_{\max}$	150
evaluation length T (time steps)	3,000
# of simulation runs per fitness evaluation	1
elitism	1
mutation rate p_{mut}	0.4

Table 7.2: Parameters for our scenario aiming for dynamic behaviors in complex environments.

beginning of each evaluation. Thus, agents cannot be initially placed on the light source, which would bring advantages for survival, since the battery does not last for the full evaluation length. For the evolutionary algorithm, we use a population size μ of 50, proportionate parent selection, age-based survivor selection, and elitism of one. We generate $\lambda = \mu - 1$ offspring for the population of the next generation. We apply mutation only, that is, we do not use recombination. Each value v of a genome is mutated with a probability p_{mut} of 0.4 by adding a uniformly random number from $[-0.4, 0.4]$. We test six different settings by differently combining our two fitness functions and the environment and agent modifications explained above as specified in Tab. 7.1. For each setting, we do ten independent evolutionary runs. Tab. 7.2 summarizes all parameters.

7.2 Evaluation Metrics

As before, we analyze the overall success of our evolutionary runs based on fitness (Eqs. 7.3 and 7.5). Since we aim for dynamic behaviors, we analyze the best evolved individuals and categorize the resulting behaviors also based on (i) time $\overline{T}_N^{\text{risk}}$ agents spent in the risk zone, (ii) time $\overline{T}_N^{\text{enter}}$ until agents first enter the risk zone, and (iii) time $\overline{T}_N^{\text{empty}}$ agents have empty batteries.

We define mean time $\overline{T}_N^{\text{risk}}$ agents spent in the risk zone as

$$\overline{T}_N^{\text{risk}} = \frac{1}{N} \sum_{n=0}^{N-1} t_n^{\text{risk}}, \tag{7.6}$$

with swarm size N and time steps t_n^{risk} agent n has spent in the risk zone.

Time $\overline{T}_N^{\text{enter}}$ is the mean time until agents first enter the risk zone by crossing the zone border visualized in Fig. 7.1.

We define $\overline{T}_N^{\text{empty}}$ as the mean time swarm members have empty batteries, as given by

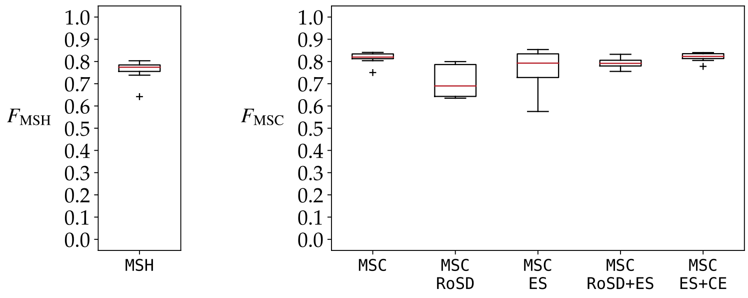
$$\overline{T}_N^{\text{empty}} = \frac{1}{N} \sum_{t=T_{\text{battery}}}^{T} \sum_{n=0}^{N} c_n(t), \tag{7.7}$$

with swarm size N, evaluation length T, maximum battery life T_{battery}, and count of charged agents

$$c_n(t) = \begin{cases} 1, & \text{if } b_n(t) = 0.0 \\ 0, & \text{otherwise} \end{cases}, \tag{7.8}$$

where $b_n(t)$ is the battery level of agent n at time step t. In our experiments, we evaluate each genome for $T = 3{,}000$ time steps, but a fully charged battery only lasts for $T_{\text{battery}} = 2{,}000$ time steps. Thus, we measure mean time $\overline{T}_N^{\text{empty}}$ agents have empty batteries over the last 1,000 time steps of an evaluation. A value of 1,000 indicates that all agents have empty batteries and were not able to recharge, while a value of 0 indicates that all agents were able to recharge their batteries.

We use all three metrics for an overall comparison of the best evolved individuals of the different setups (see Tab. 7.1). Furthermore, we categorize the resulting behaviors based on time $\overline{T}_N^{\text{risk}}$ agents spent in the risk zone, and time $\overline{T}_N^{\text{enter}}$ until agents first enter the risk zone. As explained before, agents need to recharge their batteries to survive for the full evaluation. The point of time at which agents first enter the risk zone can thus serve as an indicator for the swarm behavior because a single recharge is only potentially sufficient if agents recharge at the earliest after 1,000 time steps. In this case, agents may enter risk zone already a few time steps earlier, since the light source serving as the charging station is located at the end of the risk zone. Consequently,



(a) MSH **(b)** MSC

Figure 7.5: Best fitness in the last generation rewarding prediction accuracy and homing (MSH; F_{MSH}, Eq. 7.5) and prediction accuracy and curiosity (MSC; F_{MSC}, Eq. 7.3) with environment and agent modifications risk of sudden discharge (RoSD), energy sharing (ES), and changing environment (CE) for ten independent evolutionary runs per setting in our scenario aiming for dynamic behaviors in complex environments. Medians are indicated by the red bars.

we differentiate four different behavior categories (see also Fig. 7.7). The first category are swarms entering the risk zone quickly ($\overline{T}_N^{\text{enter}} < 750$) and staying there for the rest of the run (i.e., $T - \overline{T}_N^{\text{enter}} \approx \overline{T}_N^{\text{risk}}$). The second category are swarms that enter the risk zone quickly ($\overline{T}_N^{\text{enter}} < 750$), but leave it again (i.e., $T - \overline{T}_N^{\text{enter}} \gg \overline{T}_N^{\text{risk}}$). Agents potentially enter the risk zone several times, since the first recharge may not be sufficient to survive for the full evaluation length. The third category are swarms that enter the risk zone late enough ($\overline{T}_N^{\text{enter}} > 750$) that one battery recharge is sufficient to survive for the remainder of the evaluation. The fourth category are swarms that do not enter the risk zone at all (i.e., $\overline{T}_N^{\text{enter}} > 2{,}000$). Agents are not able to recharge their batteries and the run terminates before the maximum possible evaluation length T of 3,000 time steps. The second and third categories, in particular, have the potential for interesting, dynamic behaviors.

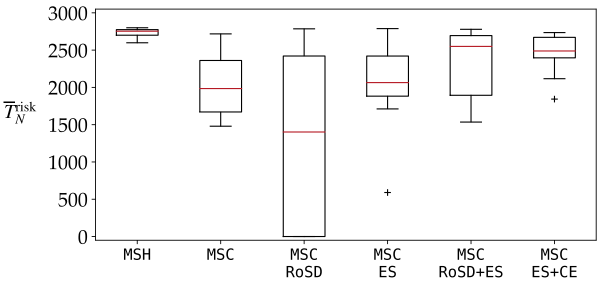
7.3 Results

We first compare the best evolved individuals of our six tested settings on an overall level, followed by a more detailed analysis of the resulting behaviors.

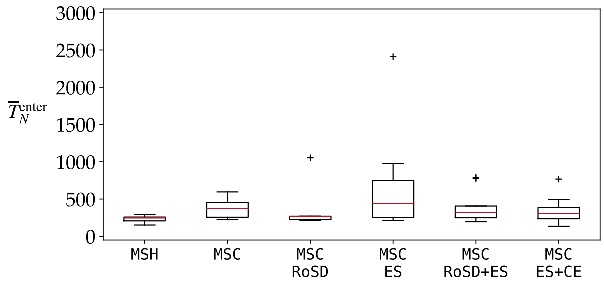
7.3.1 Overall Comparison of the Different Settings

First, we analyze the best fitness in the last generation of the six different settings specified in Tab. 7.1. We find a median best fitness of 0.77 in the setting using fitness function F_{MSH} (Eq. 7.5) rewarding prediction accuracy and homing, see Fig. 7.5a. In the five settings using fitness function F_{MSC} (Eq. 7.3) rewarding prediction accuracy and curiosity, we find median best fitnesses between 0.69 (setting with risk of sudden discharge (RoSD)) and 0.82 (setting with energy sharing (ES) and changing environment (CE)), see Fig. 7.5b. These five settings[5] lead to statistically significant differ-

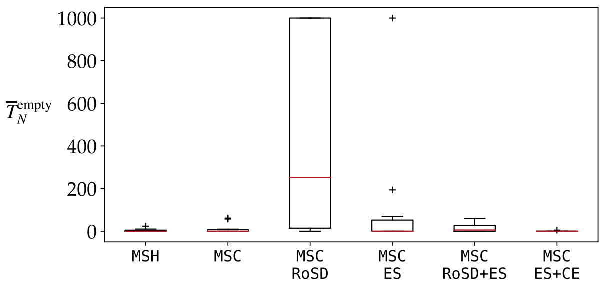
5: We do not test for statistically significant differences between settings using different fitness functions, since the results would have only limited informative value.



(a) time $\overline{T}_N^{\text{risk}}$ agents spent in the risk zone



(b) time $\overline{T}_N^{\text{enter}}$ until agents first enter the risk zone



(c) time $\overline{T}_N^{\text{empty}}$ agents have empty batteries

Figure 7.6: Time $\overline{T}_N^{\text{risk}}$ agents spent in the risk zone (Eq. 7.6), time $\overline{T}_N^{\text{enter}}$ until agents first enter the risk zone (only runs with at least one agent entering the risk zone are included), and time $\overline{T}_N^{\text{empty}}$ agents have empty batteries (Eq. 7.7) rewarding prediction accuracy and homing (`MSH`; Eq. 7.5) and prediction accuray and curiosity (`MSC`; Eq. 7.3) with environment and agent modifications risk of sudden discharge (`RoSD`), energy sharing (`ES`), and changing environment (`CE`) for ten independent evolutionary runs per setting in our scenario aiming for dynamic behaviors in complex environments. Medians are indicated by the red bars.

ences in fitness (KW, $p < 0.001$), indicating an influence of our agent and environment modifications on fitness. The setting with risk of sudden discharge (`RoSD`) leads to statistically significantly worse fitness than the setting without any modifications and the setting with energy sharing (`ES`) and changing environment (`CE`) (MW-U with BC, $p < 0.01$).

Next, we compare time $\overline{T}_N^{\text{risk}}$ agents spent in the risk zone (Eq. 7.6), time $\overline{T}_N^{\text{enter}}$ until agents first enter the risk zone, and time $\overline{T}_N^{\text{empty}}$ agents have empty batteries (Eq. 7.7), see Fig. 7.6. We find statistically significant differences in the time $\overline{T}_N^{\text{risk}}$ spent in the risk zone for the six different settings (KW, $p < 0.001$). In the setting using fitness function F_{MSH}, agents spent a median of 2,754 time steps in the risk zone, that is, they are in the risk zone for nearly the complete run. In the setting using fitness function F_{MSC}, agents spent median times between 1,402 time steps (`RoSD`) and 2,550 time steps (`RoSD` + `ES`) in the risk zone. Except for the setting using fitness function F_{MSC} with modifications risk of sudden death (`RoSD`) and energy sharing (`ES`), all settings using fitness function F_{MSC} lead to significantly lower times spent in the risk zone than the setting using fitness function F_{MSH} (MW-U with BC, $p < 0.05$). Agents enter the risk zone after a median time $\overline{T}_N^{\text{enter}}$ between 245 time steps (F_{MSH}) and 438 time steps

(F_{MSC} + `ES`). We do not find statistically significant differences here (KW, $p < 0.05$), since agents enter the risk zone in all settings quickly. We measure time $\overline{T}_N^{\mathrm{empty}}$ of agents having empty batteries. We find median values ranging from 0.0 time steps[6] to 252.2 time steps.[7] The differences in time $\overline{T}_N^{\mathrm{empty}}$ agents have empty batteries are statistically significant (KW, $p < 0.05$). The setting using fitness function F_{MSC} with energy sharing (`ES`) and changing environment (`CE`) leads to significantly lower values than the setting using fitness function F_{MSC} with risk of sudden discharge `RoSD` (MW-U with BC, $p < 0.05$). This is expected, since the risk of sudden discharge modification can lead to immediate discharges of agent batteries, which in turn increases the time $\overline{T}_N^{\mathrm{empty}}$ agents have empty batteries.

6: setting with fitness function F_{MSH}; settings with fitness function F_{MSC} with no modifications, with energy sharing (`ES`), and with energy sharing (`ES`) and changing environment (`CE`)

7: setting with fitness function F_{MSC} with risk of sudden discharge (`RoSD`)

In total, evolution successfully optimizes the actor-predictor pairs in all six settings. The best evolved individuals lead to behaviors that enter the risk zone quickly and stay there for most of the run. Consequently, agents are close to the light source to recharge their batteries and succeed in surviving for the full evaluation in most runs. However, the scenario using fitness function F_{MSC} and environment modification risk of sudden discharge (`RoSD`) leads to several runs where agents do not enter the risk zone at all, leading to an early termination of these runs. Consequently, the `RoSD` setting is probably harder than the other settings. Since all settings lead to long times spent in the risk zone, we do not find a significant bias towards dynamic behaviors that let agents switch between the zones based on battery level. Nevertheless, there may be some best individuals leading to dynamic behaviors. Thus, we investigate the resulting behaviors in more detail in the next step.

7.3.2 Emergent Behaviors

Fig. 7.7 visualizes the categorization of the best evolved individuals based on time $\overline{T}_N^{\mathrm{enter}}$ until agents first enter the risk zone and time $\overline{T}_N^{\mathrm{risk}}$ agents spent in the risk zone (see Sec. 7.2).

Homing The majority of the best evolved individuals of all settings leads to homing behaviors, that is, agents enter the risk zone quickly (i.e., $\overline{T}_N^{\mathrm{enter}} < 750$) and stay for the rest of the run in the risk zone (i.e., $T - \overline{T}_N^{\mathrm{enter}} \approx \overline{T}_N^{\mathrm{risk}}$) moving more or less closely around the light source. For the setting using fitness function F_{MSH} (Eq. 7.5), all runs lead to homing behaviors, which is probably due to the included reward for reaching the light source. Fig. 7.8a visualizes one homing behavior as representative example. Although the risk zone

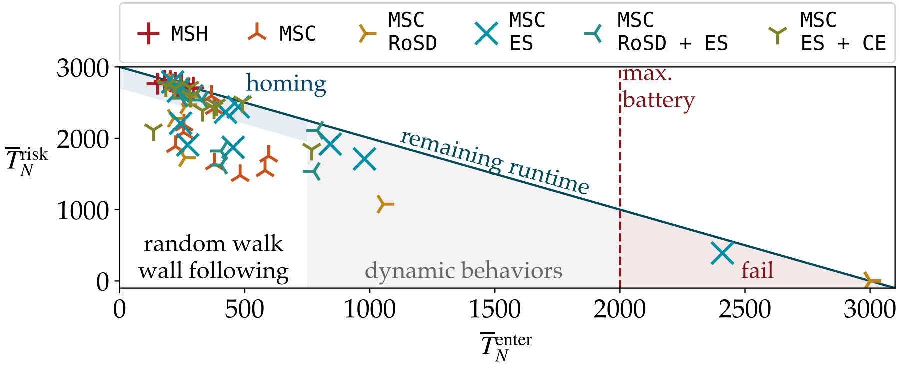
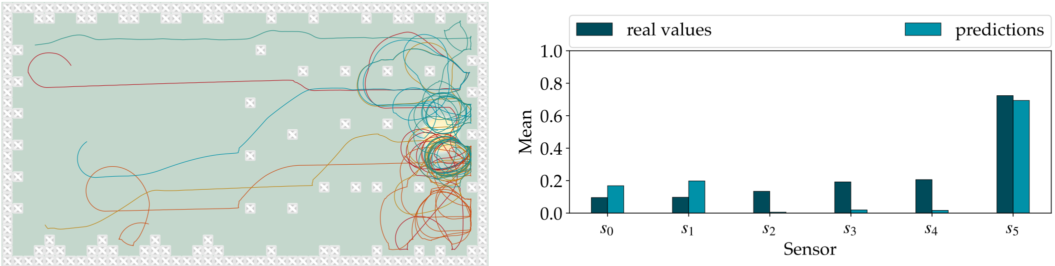


Figure 7.7: Categorization of the best evolved individuals based on time $\overline{T}_N^{\text{enter}}$ until agents first enter the risk zone and time $\overline{T}_N^{\text{risk}}$ agents spent in the risk zone in our scenario aiming for dynamic behaviors in complex environments. The diagonal indicates the remaining runtime (i.e., $T - \overline{T}_N^{\text{enter}}$) and thus the maximum time agents can still spent in the risk zone. The dashed line marks the time at which the initial battery charge is depleted, that is, after 2,000 time steps. We include the data for ten best evolved individuals per setting with `MSH` and `MSC` indicating that, respectively, prediction accuracy and homing (Eq. 7.5) and prediction accuracy and curiosity (Eq. 7.3) are rewarded, as well as environment and agent modifications risk of sudden discharge (`RoSD`), energy sharing (`ES`), and changing environment (`CE`).

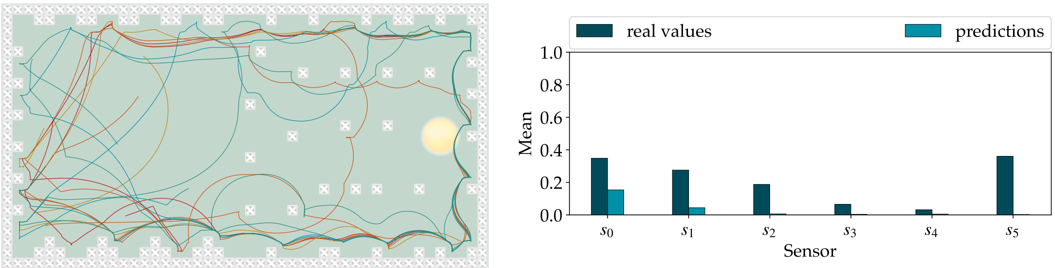
includes several obstacles, which complicate the prediction task, the proximity sensor values are still low enough to allow for high prediction accuracy when constantly predicting low sensor values. Light intensity is high in the risk zone as correctly predicted by the predictor ANN (see Fig. 7.9a).

Random Walk and Wall Following Several best evolved individuals in all settings using fitness function F_{MSC} result in behaviors where agents enter the risk zone quickly (i.e., $\overline{T}_N^{\text{enter}} < 750$) but leave it again (i.e., $T - \overline{T}_N^{\text{enter}} \gg \overline{T}_N^{\text{risk}}$). By qualitative analysis, we find wall following and random walk behaviors causing agents to enter and leave the risk zone several times in this category. Swarms probably exploit the hardware protection to prevent collisions with obstacles here. Fig. 7.8b visualizes a wall following behavior allowing trivial predictions for the proximity sensors. Although light intensity changes linearly with distance to the light source, the predictor does not predict it well, see Fig. 7.9b. However, agents can recharge their batteries when passing over the light source, which guarantees survival for the full evaluation length T.

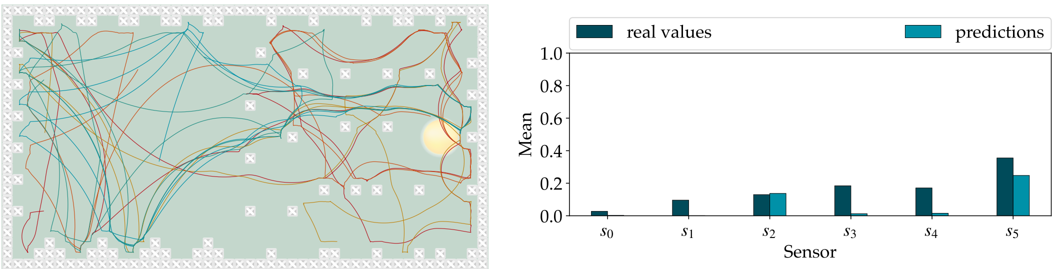
Dynamic Behaviors A few best evolved individuals of settings using fitness function F_{MSC} with agent and environment modifications lead to potentially dynamic behaviors, that is, agents enter the risk zone late enough (i.e., $\overline{T}_N^{\text{enter}} > 750$) that one recharge can be sufficient to survive for the rest of the evaluation. Here, we find that agents switch the location where they exhibit a random walk (Fig. 7.8c) or circling behavior (Fig. 7.8d) based on battery level. As before, the predictions for the proximity sensors are trivial for these behaviors. But light intensity changes with the switch between safe zone



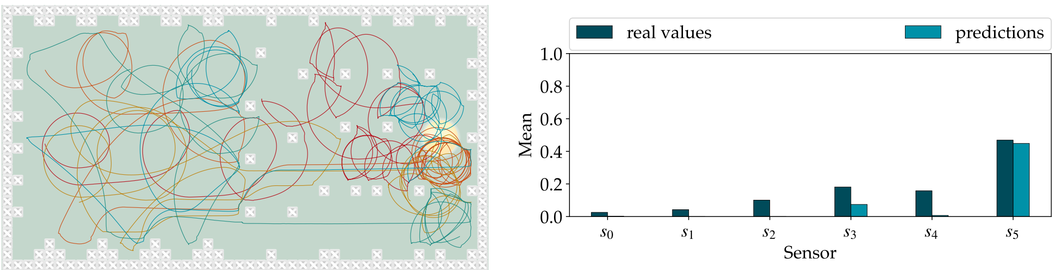
(a) homing (F_{MSH})



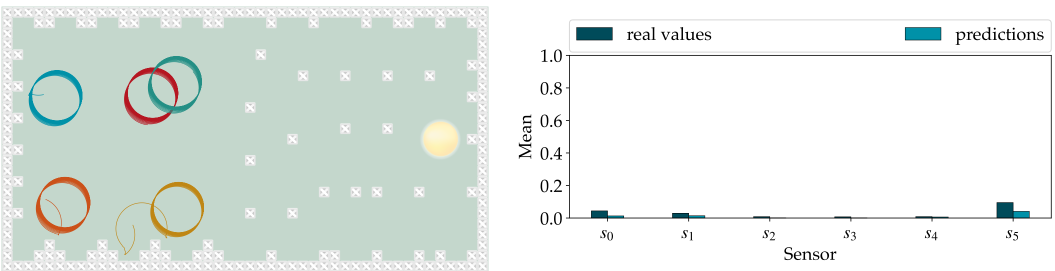
(b) wall following (F_{MSC} with `RoSD` and `ES`)



(c) dynamic behavior: random walk (F_{MSC} with `RoSD` and `ES`)

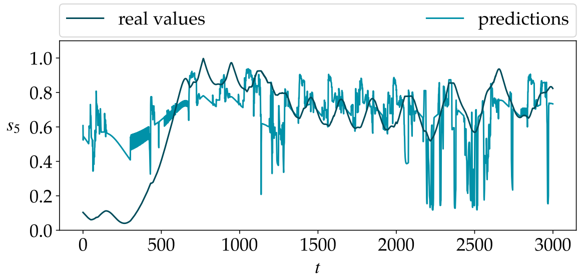


(d) dynamic behavior: circling (F_{MSC} with `ES`)

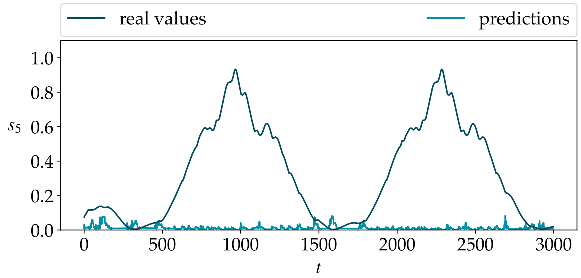


(e) fail: circling without recharging (F_{MSC} with `RoSD`)

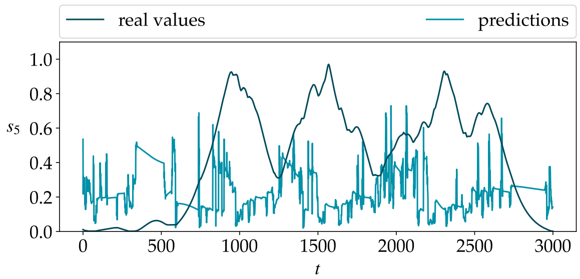
Figure 7.8: Sample agent trajectories and mean sensor values and predictions for the agents' five proximity sensors $s_0, \ldots, s_4$ and light sensor s_5 over the full evaluation length $T = 3{,}000$ time steps of the respective best evolved individual in our scenario aiming for dynamic behaviors in complex environments. The trajectories of the five agents in the swarm are represented by different colors. Settings are specified in parenthesis with fitness function F_{MSH} (Eq. 7.5) rewarding prediction accuracy and homing, fitness function F_{MSC} (Eq. 7.3) rewarding prediction accuracy and curiosity, risk of sudden discharge (`RoSD`), and energy sharing (`ES`).



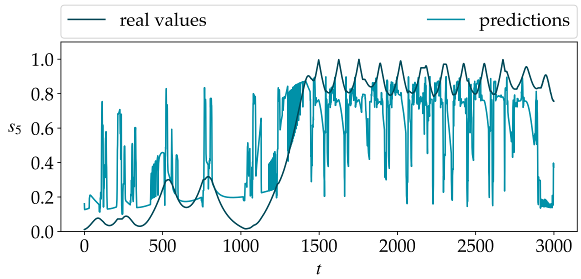
(a) homing (F_{MSH}; see Fig. 7.8a)



(b) wall following (F_{MSC} with `RoSD` and `ES`; see Fig. 7.8b)



(c) dynamic behavior: random walk (F_{MSC} with `RoSD` and `ES`; see Fig. 7.8c)



(d) dynamic behavior: circling (F_{MSC} with `ES`; see Fig. 7.8d)

Figure 7.9: Predictions for and real values of light sensor s_5 of one representative swarm member per behavior over evaluation length $T = 3{,}000$ time steps in our scenario aiming for dynamic behaviors in complex environments. Settings are specified in parenthesis with fitness function F_{MSH} (Eq. 7.5) rewarding prediction accuracy, fitness function F_{MSC} (Eq. 7.3) rewarding homing and prediction accuracy, risk of sudden discharge (`RoSD`), and energy sharing (`ES`).

and risk zone. The predictors for the dynamic circling behaviors are able to adjust their light sensor predictions with the distance to the light source (see Fig. 7.9d), but the predictors for the dynamic random walk behaviors fail to do so (see Fig. 7.9c). The best evolved individual of the setting with risk of sudden discharge (`RoSD`) that is categorized as a dynamic behavior leads to a very short time $\overline{T}_N^{\text{risk}}$ agents spent in the risk zone and a long time $\overline{T}_N^{\text{enter}}$ until agents enter the risk zone. However, this is caused by several agents not entering the risk zone at all or failing to recharge, indicating a poor performance of this behavior.

Fail We find five runs that lead to behaviors failing to recharge agent batteries (i.e., $\overline{T}_N^{\text{enter}} > 2{,}000$) in total, all for settings using fitness function F_{MSC}. The setting with energy sharing (`ES`) leads to one failed run, while the setting with risk of sudden discharge (`RoSD`) leads to four failed runs. In most cases, agents drive in circles in the safe zone until they run out of battery as visualized in Fig. 7.8e. This behavior allows for easily predictable, low sensor values but prevents that the swarm survives.

7.4 Discussion and Conclusion

Our experiments aiming for dynamic behaviors in complex environments lead to mixed results. Our fitness function F_{MSH} combines our task-independent reward for prediction accuracy with a task-specific reward for homing, resulting in homing behaviors only. By contrast, fitness function F_{MSC} combines the reward for prediction accuracy with a task-independent reward for curiosity. We find a variety of repetitive behaviors, including homing, wall following, and random walk. But in the settings including environment and agent modifications, we also find a small set of more dynamic behaviors that let agents change the location of behavior execution based on battery level. Some of these more dynamic behaviors let agents adjust their sensor predictions, in particular for the light sensors, based on the current position. That is, light sensor predictions are low while agents are in the safe zone and switch to higher values when the agents move to the risk zone. Including environmental influences and more agent capabilities is thus a promising first step to enable the evolution of dynamic behaviors but still leads only to a limited amount of behaviors with behavioral plasticity [364]. For this reason, we mainly focus on our standard minimize surprise approach for the rest of this thesis. In future work, we will intensify our studies on evolving dynamic behaviors with minimize surprise to determine how evolutionary pressure towards dynamic behaviors can be further increased. In particular, our environment modifications changing environment (`CE`) and risk of sudden discharge (`RoSD`) in combination with the agent modification energy sharing (`ES`) seem promising (see Sec. 3). Both environment modifications increase the entropy of the environment and make it more dynamic independent from the swarm behavior. By further increasing environment entropy and dynamics, we can potentially also increase evolutionary pressure for dynamic behaviors. In future work, we will study the effects of increasing the so far rather low risk of sudden discharge on the emergent behaviors. In addition, we will include random disruptive perturbations [328], similar to changing weather in the real world, in our environment. For example, the obstacle positions could change more frequently and agents that are not part of the swarm could move through the environment.

Scenarios

Collective Perception 8

Chapter Contents

In this chapter, we evolve collective decision-making behaviors in the collective perception scenario. We...

- *Sec. 8.1:* introduce the experimental setup, and
- *Sec. 8.2:* the evaluation metrics and the setup for benchmarks,
- *Sec. 8.3:* evaluate the emergent behaviors individually and in benchmark experiments, and
- *Sec. 8.4:* draw a conclusion.

In all other scenarios, we evolve spatial organization and navigation behaviors with minimize surprise. In theses cases, the sensor values of the robots vary with the evolved collective behavior. In this chapter, we aim for the evolution of collective decision-making behaviors with minimize surprise (research question Q5, Sec. 1.2). Here, the coupling between evolved collective behaviors and sensor input is weaker. Sensors related to the current opinions of the swarm members vary with the collective decision-making behavior, but the values of sensors perceiving other aspects of the environment (e.g., the quality of available options) are not influenced by the decision-making behavior. Consequently, the evolutionary process can only simplify the prediction of the sensors related to the current opinions of the swarm members by adapting the actor ANN in our minimize surprise approach. This can potentially lead to simple collective behaviors that always choose a fixed option independent of the actual best option. Thus, evolving collective decision-making behaviors with minimize surprise is probably challenging.

We use collective perception [220], that is, swarm members have to collectively decide which of two environmental feature is more frequent, as our sample scenario. Several researchers studied collective decision-making in this scenario, for example, evolving collective decision-making behaviors with a task-specific fitness function [231], using an Ising-based approach that takes into account learned preferences of agents [229], making decisions in a sparse swarm using a Bayesian algorithm [228], and implementing multi-feature collective decision-making [224, 225] (see Sec. 2.2.3). Here, we use the basic collective perception setup by Valentini et al. [220] to evolve decision-making mechanisms with our minimize surprise approach in a realistic simulation environment. Robots concurrently execute low-level motion routines and a probabilistic state machine for decision-making that can be used with different decision-making mechanisms. Valentini et al. [220] use the majority rule [222] and the

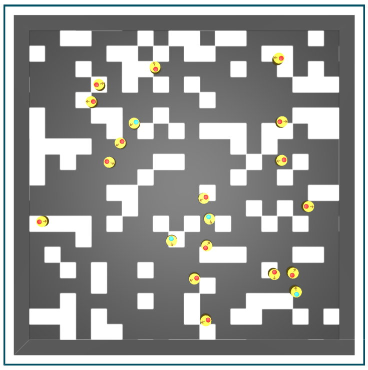


Figure 8.1: Screenshot of the simulation environment for the collective perception scenario in the BeeGround simulator. The arena is 2 m×2 m, bounded by walls, and has black and white grid cells on its surface. The yellow robots indicate their current opinion via the LED on top (red represents black, blue represents white).

voter model [223] in their experiments. In this chapter, we evolve decision-making mechanisms that are applied in the probabilistic finite state machine by using the actor of our actor-predictor ANN pair as a decision-making mechanism rather than a controller for robot motion.

8.1 Experimental Setup

In this section, we introduce our custom robot model, the simulation environment, the adaptation of minimize surprise for the collective perception scenario, and the setup for our benchmark experiments.[1] We base our general experimental setup (i.e., arena and robot control) on Valentini et al. [220].

1: `https://gitlab.iti.uni-luebeck.de/minimize-surprise/collective-perception`

8.1.1 Simulation Environment

We use the Unity-based BeeGround simulator [45], which was specifically developed for swarm robotics (see Ch. 2.1.2). Our simulation environment is a square 2 m × 2 m arena that is bounded by walls, see Fig. 8.1. There are two features in the environment, which are represented by black and white grid cells of 10 cm × 10 cm each on the arena surface. Robots have to collectively reach a consensus[2] which of the two features is the best option with the better option quality (i.e., best-of-2 problem [221]). Option quality ρ_i is the frequency of feature $i \in \{\text{black}, \text{white}\}$ as defined by

2: A consensus is reached when all N robots agree on the same opinion (i.e., environmental feature here).

$$\rho_i = \frac{\mathcal{M}_i}{\mathcal{M}_{\text{black}} + \mathcal{M}_{\text{white}}}, \tag{8.1}$$

where $\mathcal{M}_i$ is the number of i = black or i = white grid cells, respectively. Normalized option quality ρ_i^* is defined as

$$\rho_i^* = \frac{\rho_i}{\max\{\rho_{\text{black}}, \rho_{\text{white}}\}}, \tag{8.2}$$

evaluating to $\rho_i^* = 1$ for the feature with the better option quality (i.e., higher frequency) and to $\rho_i^* \in [0, 1)$ for the feature with the worse option quality. Problem difficulty is given by

$$\rho_{\min}^* = \min\{\rho_{\text{black}}^*, \rho_{\text{white}}^*\}, \tag{8.3}$$

that is, the normalized option quality of the worse feature. The more similar the option qualities of both features, the higher the problem difficulty.

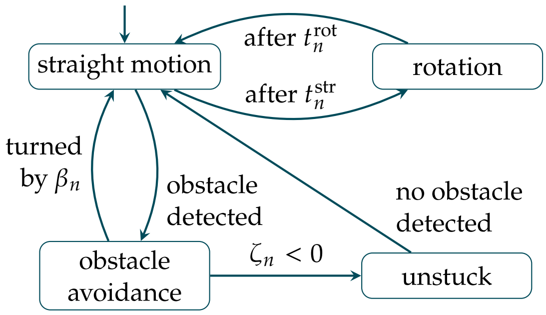


Figure 8.2: Finite state machine for robot motion adapted from Valentini et al. [220]. The robot executes a random walk behavior by switching between *straight motion* and *rotation*. Time periods t_n^{str} and t_n^{rot} specify the time robot n spends in *straight motion* and *rotation*, respectively. When the robot detects obstacles with its proximity sensors, it switches to *obstacle avoidance*, that is, it turns by angle β_n away from the obstacle. Buffer values ζ_n less than zero indicate that the robot got stuck between obstacles and the robot turns until it does not detect any obstacles anymore to get *unstuck*.

8.1.2 Robot

Platform

We use a custom robot model that is similar to popular robot platforms, such as the Thymio II [77] and the e-puck [78]. Our custom robot has a diameter of 7 cm, a differential drive with a maximum speed of 10 $\frac{\text{cm}}{\text{s}}$, and a LED on top indicating its current opinion. We initialize the swarm members in equal parts with the current opinion black and white. The robot has five horizontal frontal proximity sensors[3] with a range of 10 cm that are updated every 0.15 s. In addition, it has a binary ground sensor that measures every 0.2 s whether the arena surface below the robot is black (0) or white (1). Robots can broadcast messages containing their ID and their current opinion to their neighbors in a 70 cm radius (i.e., local communication). We use a swarm of $N = 20$ robots resulting in a swarm density D_N of approximately 0.02.

3: The sensors are located at 0°, that is, in the robot's heading direction, as well as at ±25°, and ±50°.

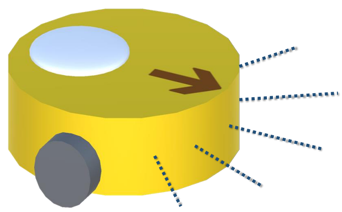


Figure 8.3: Robot model in the collective perception scenario. The robot has a differential drive, five frontal proximity sensors (dashed lines), a ground sensor (not visible), and a LED indicating its current opinion on top. The arrow indicates its heading.

Control

The behavior of our robots is based on the low-level motion routines and the decision-making behavior of the direct modulation of majority-based decision (DMMD) [222] and direct modulation of voter-based decision (DMVD) [223] approaches used by Valentini et al. [220]. Different decision-making mechanisms can be applied in the decision-making behavior. We implement the motion routines and the decision-making behavior in two separate finite state machines that are executed concurrently.

Robot Motion The state machine for robot motion implements a random walk behavior, see Fig. 8.2. Robot n moves straight (*straight motion* state) for a random time period t_n^{str} that is sampled from an exponential distribution with a mean of 40 s. Afterwards, the robot turns on the spot in a random turning direction (*rotation* state) for a random time period t_n^{rot} that is sampled from a uniform distribution with bounds [0 s, 4.5 s]. The robot then moves straight again.

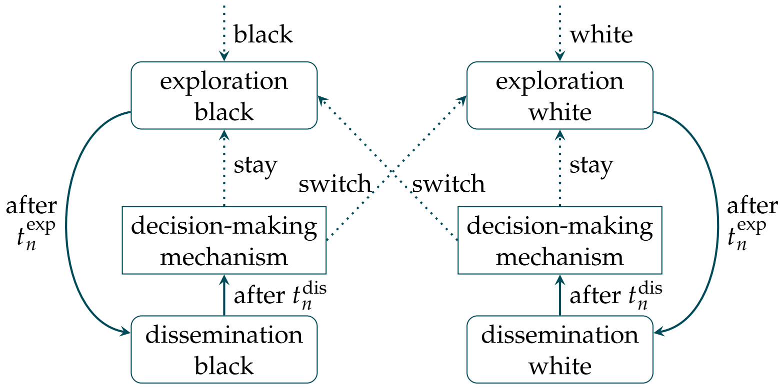


Figure 8.4: Probabilistic finite state machine for decision-making based on Valentini et al. [220]. The state machine has an *exploration* and a *dissemination* for each of the two features in the environment (i.e., black and white). Time periods t_n^{exp} and t_n^{dis} specify how long robot n stays in the *exploration* and *dissemination* state, respectively. The decision-making mechanism updates the robot's current opinion after *dissemination* and the robot switches to the respective exploration state. Dotted lines represent stochastic and solid lines deterministic transitions.

We implement an enhanced variant of the obstacle avoidance routine by Valentini et al. [220] that is triggered when a robot detects an obstacle (i.e., wall or other robot) with its proximity sensors while moving straight. The robot then turns until its heading points to the opposite direction of the nearest detected obstacle (*obstacle avoidance* state). We add a random value sampled from a uniform distribution with bounds $[-25°, 25°]$ to the rotation angle $\beta_n = 180°$ to avoid thrashing. As this approach cannot completely prevent thrashing, we additionally introduce the *unstuck* state in which the robot rotates into a random direction until no obstacles are detected by the proximity sensors. The robot switches from *obstacle avoidance* to *unstuck* when buffer ζ_n indicates that the robot got stuck (i.e., $\zeta_n < 0$). The buffer value ζ_n decreases when the robot is rotating (i.e., the active state is *rotation* or *obstacle avoidance*) and increases up to a maximum of 7.5 s when the robot is moving straight (i.e., the active state is *straight motion*). Buffer values less than zero ($\zeta_n < 0$) indicate that the robot has mainly rotated recently and is thus probably stuck between obstacles. After the robot has avoided the obstacles, it switches back to random walk behavior.

Decision-Making Behavior The decision-making behavior is implemented as a probabilistic finite state machine (PFSM) that can be used with different decision-making mechanisms (e.g., the majority rule in the DMMD [222] strategy and the voter model in the DMVD [223] strategy). The PFSM has four states: an *exploration* and a *dissemination* state for each of the two environmental features. As the state machine for decision-making and the state machine for robot motion are executed concurrently, robots always perform random walk and obstacle avoidance in parallel to these four states.

In the *exploration* state, robot n explores its environment by sampling the local ground color and determining a quality estimate $\hat{\rho}_n$ of its current opinion. The quality estimate $\hat{\rho}_n$ is the proportion of measurements where the sampled local ground color matches a robot's current opinion. After time period t_n^{exp}, which is drawn from an exponential distribution with a mean of 10 s, the robot switches to the *dissemination* state.

In the *dissemination* state, the robot broadcasts its current opinion to its local neighborhood for a time period t_n^{send} that is sampled from an exponential distribution with a mean of $T_{\mathrm{send}}\hat{\rho}_n$. We set the design parameter T_{send} to 10 s and robots broadcast their current opinion with 1 Hz. Robots modulate positive feedback by scaling T_{send} with their current quality estimate $\hat{\rho}_n$ as this results in longer dissemination of higher quality opinions. Subsequent to sending its opinion, the robot records for $t_n^{\mathrm{receive}} = 3$ s its neighbors' opinions in a message queue with a maximum length $\mathbb{Q}_{\mathrm{max}}$ of four and maximally one opinion per neighbor. After time period $t_n^{\mathrm{dis}} = t_n^{\mathrm{send}} + t_n^{\mathrm{receive}}$, the robot applies a decision-making mechanism to update its current opinion and switches to the respective exploration state.

8.1.3 Minimize Surprise

We use our minimize surprise approach (see Ch. 3) to evolve decision-making mechanisms. As before, each robot is equipped with an actor-predictor ANN pair. Actor and predictor are both three-layer ANNs with an input, one hidden and an output layer. We use the sigmoid function as the transfer function. Inputs are propagated through the ANN pair at the end of each robot n's dissemination phase. As explained above, the time periods t_n^{exp} and t_n^{dis} determining the length of the exploration and dissemination phases are randomly drawn and thus the number of ANN propagations per robot may vary. Consequently, we need to adapt our standard minimize surprise fitness function (Eq. 3.1) slightly. We adjust fitness F to

$$F = \frac{1}{RT} \sum_{n=0}^{N-1} \sum_{t=0}^{t_n-1} \sum_{r=0}^{R-1} (1 - |\tilde{s}_r^n(t) - s_r^n(t)|), \tag{8.4}$$

with swarm size N, number of sensors per swarm member R, number of ANN propagations t_n of robot n, total number of ANN propagations $T = \sum_{n=0}^{N-1} t_n$ of the swarm, robot n's prediction $\tilde{s}_r^n(t)$ for the real value of sensor r at ANN propagation t and the real value $s_r^n(t)$ of sensor r at ANN propagation t.

In addition, we introduce a task-specific variant of this standard minimize surprise fitness function that includes additional evolutionary pressure similar as in Ch. 7. We define penalized fitness F_{p} as

$$F_{\mathrm{p}} = \left(\frac{1 - \varphi}{\kappa} + \varphi \right) F\,, \tag{8.5}$$

with minimize surprise fitness F (Eq. 8.4), penalty factor $\kappa > 1$, and percentage $\varphi \in [0,1]$ of robots having the best option as their opinion at the end of the evaluation. Thus, fitness is reduced when the swarm did not reach a consensus for the best option by the end of the evaluation. The maximum penalty of $1 - \frac{1}{\kappa}$ is applied if the swarm reaches a consensus for the lower quality option. By reducing fitness this way, we introduce a task dependence to the fitness function that results in a stronger coupling between the collective perception scenario and the desired behavioral outcome. We use a penalty factor κ of two in our experiments.

For the usage in our minimize surprise approach, we aggregate the information in a robot's message queue at the end of the dissemination phase to serve as sensor input at ANN propagation t. We define $s_0(t)$ as the percentage of neighbor opinions with value white as

$$s_0(t) = \frac{\mathbb{Q}_{\text{white}}}{\mathbb{Q}_{\text{obs}}} \in [0,1], \tag{8.6}$$

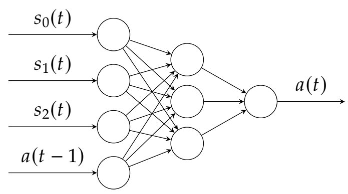
with number of received messages with opinion white $\mathbb{Q}_{\text{white}}$ and the current length of the message queue $\mathbb{Q}_{\text{obs}}$. Additionally, $s_1(t)$ gives the number of received messages normalized by the maximum queue length $\mathbb{Q}_{\text{max}}$. We define $s_1(t)$ as

$$s_1(t) = \frac{\mathbb{Q}_{\text{obs}}}{\mathbb{Q}_{\text{max}}} \in [0,1]. \tag{8.7}$$

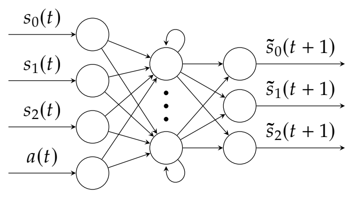
The actor (see Fig. 8.5a) serves as the decision-making mechanism that is used at the end of the dissemination phase (see Sec. 8.1.2). The feedforward ANN has four input neurons, three hidden neurons, and one output neuron. We input the aggregated neighbor opinions $s_0(t)$ and $s_1(t)$, the current binary ground sensor reading $s_2(t)$, and the robot's current opinion determined by its last decision $a(t-1)$. We map the actor output to the binary decision value $a(t)$ representing the robot's updated opinion on the best option.

The predictor (see Fig. 8.5b) is a recurrent ANN with four input neurons, four hidden recurrent neurons, and three output neurons. It receives the aggregated neighbor opinions $s_0(t)$ and $s_1(t)$, the current binary ground sensor reading $s_2(t)$, and the robot's updated opinion $a(t)$ as inputs. We introduce a coupling with the environment and the collective perception task by including the ground sensor reading $s_2(t)$. The predictor outputs predictions in $[0,1]$ for the aggregated neighbor opinions $\tilde{s}_0(t+1)$ and $\tilde{s}_1(t+1)$ and the ground sensor value $\tilde{s}_2(t+1)$.

As before (see Sec. 4.1), we use a simple evolutionary algorithm to evolve the ANN pairs. We run evolution for 300 generations using the standard minimize surprise fitness



(a) actor



(b) predictor

Figure 8.5: Actor-predictor ANN pair in the collective perception scenario. The actor serves as a decision-making mechanism here. $s_0(t)$ and $s_1(t)$ are the robot's aggregated neighbor opinions and $s_2(t)$ is the robot's ground sensor value at the current ANN propagation t. $a(t-1)$ is the robot's last decision or current opinion and $a(t)$ is the robot's current decision or updated opinion on the best option. $\tilde{s}_0(t+1)$ and $\tilde{s}_1(t+1)$ are the predictions for the robot's aggregated neighbor opinions and $\tilde{s}_2(t+1)$ is the robot's prediction for its ground sensor value at the next ANN propagation $t+1$.

parameter	value
arena side length L	2 m
swarm size N	20
task difficulty $\rho^*_{\min}$	$\{0.25, 0.52\}$
number of sensors R	3
aggregated neighbor opinions s_0, s_1	$[0, 1]$
ground sensor s_2	$\{0, 1\}$
decision a	$\{0, 1\}$
population size μ	50
number of generations $g_{\max}$	$\{300, 600\}$
evaluation length	200 s
# of simulation runs per fitness evaluation	6
elitism	1
mutation rate p_{mut}	0.2

Table 8.1: Parameters for the evolutionary runs in the collective perception scenario.

function (Eq. 8.4) and for 600 generations using the penalized fitness function (Eq. 8.5). Each genome is evaluated in six independent evaluations for 200 s. Genomes encode the synaptic weights of both neural networks (see Sec. 3.2) and we randomly generate the initial population $\mathcal{P}(0)$ by drawing the weights from a uniform distribution in $[-0.5, 0.5]$. The fitness of a genome is the minimum fitness observed in its six independent evaluations. We aim to prevent that actor-predictor pairs are optimized to output a fixed opinion independent from the actual best option by evaluating each individual using three different black and white patterns and the inverse of these patterns. That is, we evaluate each genome in settings with differing best options but the same problem difficulty $\rho^*_{\min} \in \{0.25, 0.52\}$. For the evolutionary algorithm, we use a population size μ of 50, proportionate parent selection, age-based survivor selection, and elitism of one. We generate $\lambda = \mu - 1$ offspring for the population of the next generation. For variation, we apply mutation only, that is, we do not use recombination. Each value of a genome is mutated with a probability p_{mut} of 0.2 and by adding a random number drawn from the uniform distribution on the interval $[-0.4, 0.4]$. For each setting, we do ten independent evolutionary runs per fitness function and problem difficulty $\rho^*_{\min} \in \{0.25, 0.52\}$. All parameters are summarized in Tab. 8.1.

8.2 Evaluation Metrics and Benchmarks

We validate our approach by comparing best evolved individuals to state-of-the-art decision-making mechanisms in benchmark experiments. In this section, we introduce the metrics used for quantitative comparison and the settings for the benchmark experiments.

8.2.1 Metrics

As before, we evaluate the success of evolution with our minimize surprise approach based on the best fitness (Eqs. 8.4 and 8.5). In addition, we analyze the decision-making mechanisms based on two common metrics for the best-of-2 problem [220, 221]: (i) consensus time T_N and (ii) exit probability E_N.

Consensus time T_N is the minimum time a swarm needs to reach a first consensus and thus quantifies decision speed.[4] We calculate the mean consensus time $\overline{T}_N$ as the average over all runs leading to consensus.

4: Since consensus is reached when all N robots agree on the same opinion, a swarm may lose consensus if one or more robots switch their opinion after the first consensus. The consensus time T_N does not take into account whether the swarm loses its first consensus. While it is very unlikely for voter model and majority rule to lose consensus or even switch consensus to the opposite option, we cannot guarantee this for the evolved decision-making mechanisms. But in the experiments for all decision-making mechanisms, we only observed in one voter model run that consensus switched from one option to the other. Thus, we consider consensus time to be a useful indicator for decision speed.

Exit probability E_N is the percentage of runs in which the swarm successfully reached a consensus for the best option and thus quantifies decision accuracy.

Speed and accuracy are known to be often conflicting goals in (collective) decision-making, that is, faster decision speed results in loss of decision accuracy (i.e., speed versus accuracy trade-off [222, 369, 370]).

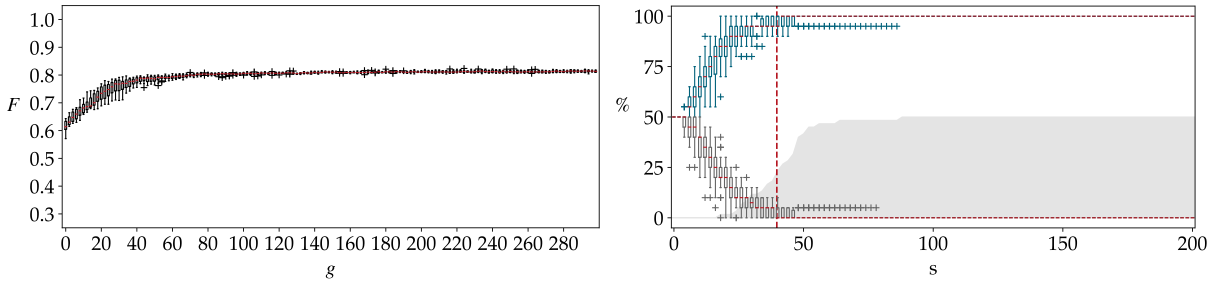
8.2.2 Benchmarks

The probabilistic finite state machine for decision-making (see Fig. 8.4) can be used with different decision-making mechanisms. Thus, we compare our evolved decision-making mechanisms with the voter model and the majority rule. These two state-of-the-art mechanisms were also used in the initial collective perception experiments by Valentini et al. [220]. The voter model lets robots take the opinion of a random neighbor. By contrast, when applying the majority rule, robots adopt the opinion endorsed by the majority of their neighbors and themselves. From each minimize surprise setting (i.e., fitness function and problem difficulty used to evolve the controllers), we select the ANN pair with fitness closest to the median of the best fitness of the last generation as a representative example for our benchmark experiments.

We benchmark the decision-making mechanisms in the two problem difficulties $\rho^*_{\text{min}} \in \{0.25, 0.52\}$ used in evolution as well as in two harder problem difficulties $\rho^*_{\text{min}} \in \{0.67, 0.82\}$ to test the scalability of the decision-making mechanisms in problem difficulty (see Tab. 8.2). Each decision-making mechanism is run 1,000 times for 400 s per problem difficulty ρ^*_{min}. To ensure maximal comparability, we use the same evaluation settings (i.e., initial robot poses, initial robot opinions, ground pattern of black and white tiles) for the different mechanisms. Without loss of generality, the best option in all settings is black.

Table 8.2: Problem difficulty ρ^*_{min} (Eq. 8.3) and corresponding feature ratio (lower quality option : better quality option).

ρ^*_{min}	feature ratio
0.25	20 : 80
0.52	34 : 66
0.67	40 : 60
0.82	45 : 55



(a) best fitness F over generations g **(b)** decision-making process

Figure 8.6: Best fitness F (Eq. 8.4) over generations g and decision-making process over time in seconds s of the best evolved individuals using our standard minimize surprise approach in the collective perception scenario for 10 independent evolutionary runs. The decision-making process is represented by the percentage of swarm members with the best option as opinion over the evaluation run. Blue boxes represent evaluations leading to the consensus for the best option and gray boxes represent evaluations leading to an incorrect consensus. The gray area gives the exit probability E_N over time. The mean consensus time $\overline{T}_N$ is given by the dashed red line. Medians are given by the red bars in the boxes. For clearer illustration, we only plot the data of every second generation or time step.

8.3 Results

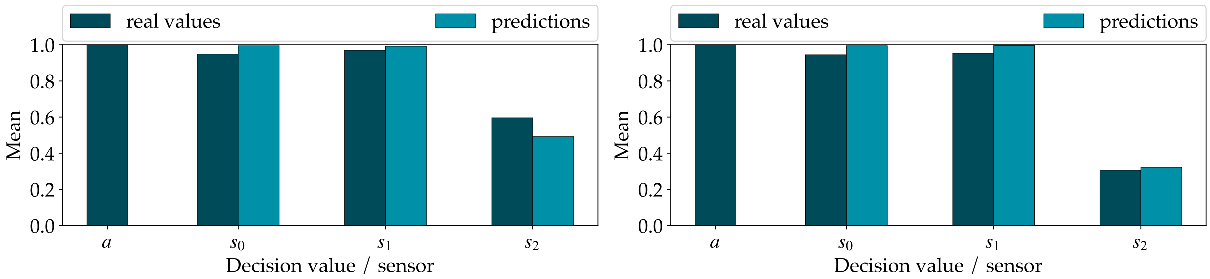
First, we analyze if decision-making mechanisms emerge in the evolutionary runs with our standard minimize surprise approach and with minimize surprise with penalized fitness (see Sec. 8.1.3). Afterwards, we compare the evolved decision-making mechanisms in benchmark experiments with the majority rule and the voter model.

8.3.1 Standard Minimize Surprise Approach

In the first step, we aim to evolve decision-making mechanisms with our standard minimize surprise approach. We do ten independent evolutionary runs for 300 generations each using problem difficulty $\rho^*_{\text{min}} = 0.52$.

Fig. 8.6a visualizes the increase in best fitness F (Eq. 8.4) over generations reaching a median of 0.81 in the last generation. The predictor (see Fig. 8.5b) has thus a median prediction accuracy of 81 %. We find that our minimize surprise approach successfully optimizes the ANN pairs over generations.

Fig. 8.6b shows the decision-making process over the 200 s long evaluations for the best evolved individuals. All evaluations lead to consensus resulting in a mean consensus time $\overline{T}_N$ of 41.7 s. But only half of the evaluations reach consensus for the best option, that is, we find an exit probability E_N of 50 %. As explained in Sec. 8.1.3, each genome is evaluated in six independent evaluations whereof three have black as the best option and three white. We find that all best evolved individuals lead to a consensus for one of these two options independent of the actual best option. Part of the best evolved individuals always lead to a consensus for black, the other individuals always for white. Consequently, each best evolved individual reaches the correct consensus only in half of its evaluations. High fitness can still be reached as



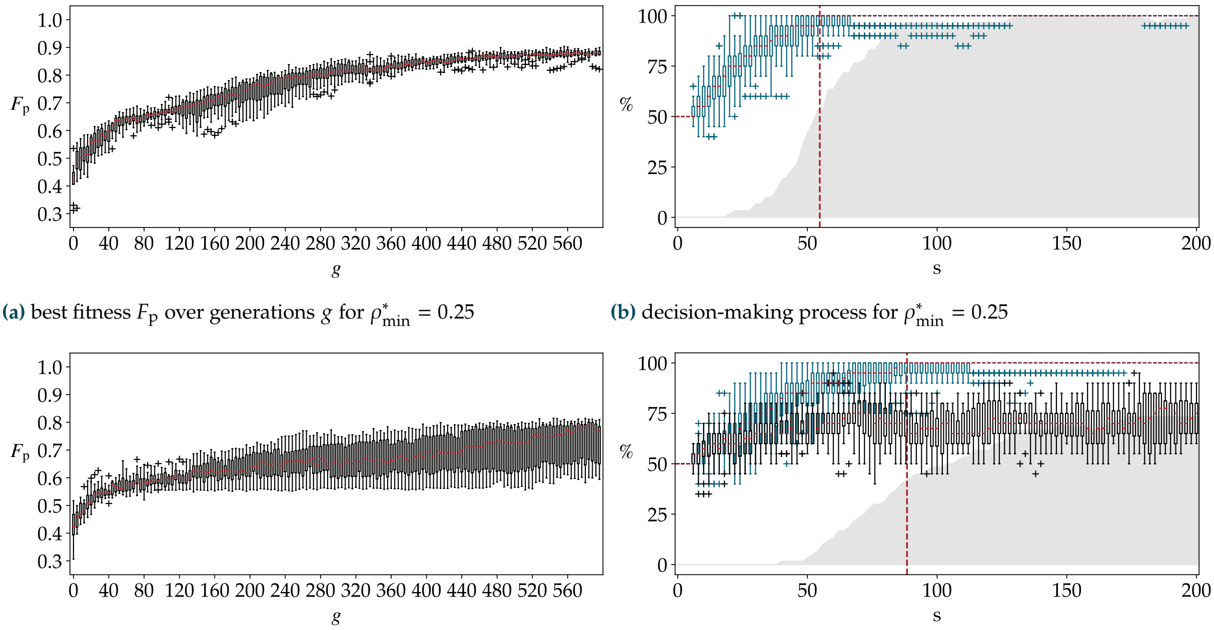
(a) best option: white

(b) best option: black

Figure 8.7: Mean actor decision output a (cf. Fig. 8.5a) and mean inputs to and outputs of the predictor (cf. Fig. 8.5b) for one representative best evolved individual in our collective perception scenario using our standard minimize surprise approach with (a) white and (b) black as the actual best option. s_0 (Eq. 8.6) is the percentage of neighbors with opinion white, s_1 (Eq. 8.7) the percentage of neighbors relative to the maximum possible number of neighbors, and s_2 the ground sensor reading.

the prediction task is trivial. We illustrate this based on the mean real and predicted values for a best evolved individual always leading to a consensus for white as representative example, see Fig. 8.7. The percentage of neighbors relative to the maximum possible number of neighbors s_1 is easy to predict as it is almost always at the maximum ($s_1 \approx 1.0$). The actors output a fixed opinion a independent from the actual best option. Thus, each swarm member switches to the consensus opinion after its first dissemination phase when the decision-making mechanism is executed the first time. This leads to consensus as quickly as possible and high prediction accuracy as the neighbor opinions s_0 are easy to predict. The ground sensor values s_2 depend on the actual best option. We find higher mean values showing a run with white as the best option in Fig. 8.7a than in Fig. 8.7b visualizing a run with black as the best option. The mean predictions for the ground sensor match the mean real values closely. Thus, evolution successfully optimized the predictor to output ground sensor predictions matching the actual frequencies of the ground colors. In total, two of the three predictor outputs are trivial to predict due to the fixed opinion output of the actor. The results of this exemplary run are representative for all runs using the standard minimize surprise fitness function (Eq. 8.4).

Overall, we find that the optimization of the ANN pairs with our standard minimize surprise approach is too detached from the collective perception task in the presented experimental setup. Only the ground sensor introduces a coupling between the swarm and the environment with the collective perception task, which is too weak to lead to the emergence of decision-making mechanisms here. Evolution exploits the easiest possible solution leading to high fitness, which is making neighbor opinions easily predictable by setting the actor's decision output to a fixed opinion independent from the actual best option. While evolution successfully optimizes fitness, the resulting behaviors are of no practical use and we refrain from investigating this setup further. Instead, we



(a) best fitness F_p over generations g for $\rho^*_{min} = 0.25$

(b) decision-making process for $\rho^*_{min} = 0.25$

(c) best fitness F_p over generations g for $\rho^*_{min} = 0.52$

(d) decision-making process for $\rho^*_{min} = 0.52$

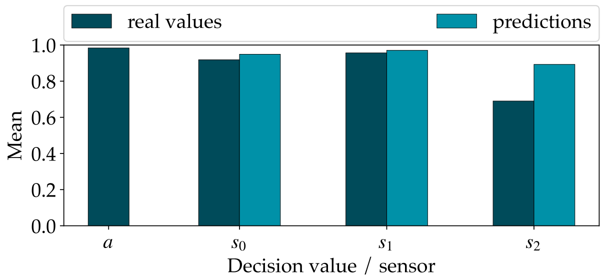
Figure 8.8: Best fitness F_p (Eq. 8.5) over generations g and decision-making process over time in seconds s of the best evolved individuals using our minimize surprise approach with penalized fitness in the collective perception scenario for problem difficulties $\rho^*_{min} \in \{0.25, 0.52\}$. The decision-making process is represented by the percentage of swarm members with the best option as opinion over the evaluation run. Blue boxes represent evaluations leading to the consensus for the best option and black boxes represent evaluations leading to no consensus. The gray areas give the exit probability E_N over time. The mean consensus times $\overline{T}_N$ are given by the dashed red lines. Medians are given by the red bars in the boxes. For clearer illustration, we only plot the data of every fourth generation and of every second time step.

add an additional evolutionary pressure to push emergence towards decision-making mechanisms in the next step.

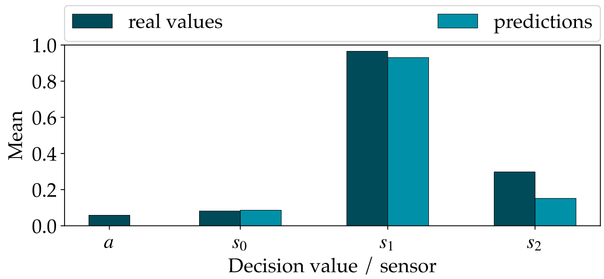
8.3.2 Minimize Surprise with Penalized Fitness

Next, we aim to evolve decision-making mechanisms using minimize surprise with penalized fitness as described in Sec. 8.1.3. We do ten independent evolutionary runs for 600 generations per problem difficulty $\rho^*_{min} \in \{0.25, 0.52\}$.

Figs. 8.8a and 8.8c show the increase in best fitness F_p (Eq. 8.5) over generations for both problem difficulties. We find a median best fitness of 0.88 for problem difficulty 0.25 and of 0.77 for problem difficulty 0.52 in the last generation. The harder problem difficulty of 0.52 leads to more variability in fitness in the higher generations of the run. This has probably two related reasons: (i) the problem difficulty makes reaching consensus hard and (ii) the penalized fitness rigorously punishes ANN pairs that do not lead to the correct consensus or a consensus at all. As visualized in Fig. 8.8d, not all evaluations in the harder problem difficulty setting lead to a consensus. Since the fitness of an ANN pair is the minimum fitness of six evaluations, even a single evaluation not leading to consensus significantly worsens its fitness. Increasing the number of generations and the evaluation length for the



(a) best option: white



(b) best option: black

Figure 8.9: Mean actor decision output a (cf. Fig. 8.5a) and mean inputs to and outputs of the predictor (cf. Fig. 8.5b) for one representative best evolved individual in our collective perception scenario with problem difficulty $\rho^*_{\text{min}} = 0.52$ using our minimize surprise approach with penalized fitness with (a) white and (b) black as the best option. s_0 (Eq. 8.6) is the percentage of neighbors with opinion white, s_1 (Eq. 8.7) the percentage of neighbors relative to the maximum possible number of neighbors, and s_2 the ground sensor reading.

harder problem difficulty probably leads to less variability and better convergence of the fitness curve.

Figs. 8.8b and 8.8d show the decision-making process over the 200 s long evaluation runs for the best evolved individuals for both problem difficulties. For problem difficulty $\rho^*_{\text{min}} = 0.25$, a consensus for the best option is reached in all evaluations (i.e., $E_N = 100\,\%$). We find a mean consensus time $\overline{T}_N$ of 56.8 s in this setting. For problem difficulty $\rho^*_{\text{min}} = 0.52$, the mean consensus time $\overline{T}_N$ is 90.4 s. We find an exit probability E_N of 73 %, that is, 73 % of the evaluations reach the correct consensus. The remaining 27 % of the evaluations do not reach consensus. We find that two of the ten best evolved individuals do not lead to consensus in any of their six evaluations each, and one best evolved individual leads to consensus in only two of its six evaluations. However, there is a tendency towards correct consensus in all runs in which the swarm has not reached consensus, see Fig. 8.8d. Again, increasing the length of the evaluations or running evolution for more generations could enable evolution to find a decision-making mechanism in every run.

Fig. 8.9 visualizes the real and predicted values for the best evolved individual of a minimize surprise run with penalized fitness in problem difficulty $\rho^*_{\text{min}} = 0.52$ serving as a representative example for all best evolved individuals reaching consensus. Similar to our standard minimize surprise approach, the actual mean ground sensor s_2 values and their predictions differ based on the best option and the percentage of neighbors relative to the maximum possible number of neighbors s_1 is always close to 1.0. But in contrast to the previous case, the percentage of neighbors with opinion white s_0 and the actor's decision output a are high when the best option is white and low when the best option is black. The decision output is furthermore neither exactly zero nor exactly one, indicating that the actor does not constantly output a fixed value. Overall, we find that minimize surprise with penalized fitness leads to the emergence of decision-making mechanisms.

Table 8.3: Mean consensus times $\overline{T}_N$ and exit probabilities E_N (see Sec. 8.2) for the benchmark runs in our collective perception scenario. We compare voter model (`VM`), majority rule (`MR`), an ANN pair evolved with minimize surprise with penalized fitness in the easier task difficulty $\rho^*_{\mathrm{min}} = 0.25$ (`MS-E`), and an ANN pair evolved in the harder task difficulty $\rho^*_{\mathrm{min}} = 0.52$ (`MS-H`). Values are calculated based on 1,000 runs of 400 s per decision-making mechanism and problem difficulty ρ^*_{min}.

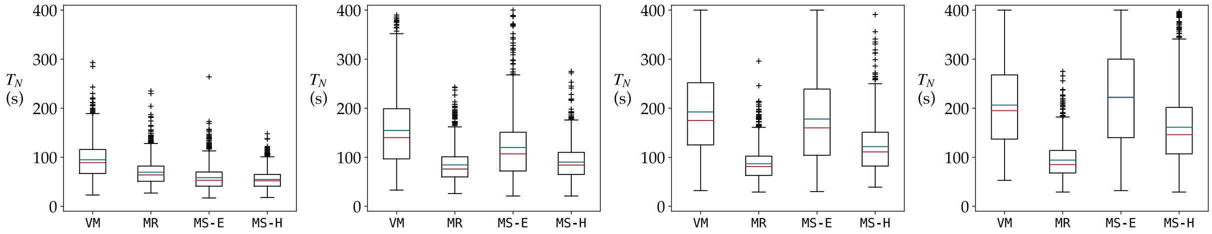
decision-making mechanism	$\rho^*_{\mathrm{min}} = 0.25$		$\rho^*_{\mathrm{min}} = 0.52$		$\rho^*_{\mathrm{min}} = 0.67$		$\rho^*_{\mathrm{min}} = 0.82$	
	$\overline{T}_N$	E_N	$\overline{T}_N$	E_N	$\overline{T}_N$	E_N	$\overline{T}_N$	E_N
`VM`	94.9 s	100.0 %	154.7 s	97.1 %	192.5 s	83.4 s	206.3 s	60.2 %
`MR`	69.5 s	96.7 %	84.5 s	83.5 %	86.8 s	72.8 %	94.1 s	61.1 %
`MS-E`	58.4 s	100.0 %	119.8 s	99.2 %	178.0 s	90.5 %	222.4 s	55.6 %
`MS-H`	54.8 s	100.0 %	90.1 s	100.0 %	121.6 s	98.5 %	161.1 s	87.6 %

8.3.3 Benchmarks

Last, we evaluate the competitiveness of the decision-making mechanisms evolved with our minimize surprise approach in benchmark experiments. We compare voter model (`VM`), majority rule (`MR`), and one best evolved ANN pair per problem difficulty $\rho^*_{\mathrm{min}} = 0.25$ (`MS-E`) and $\rho^*_{\mathrm{min}} = 0.52$ (`MS-H`) used in the evolutionary settings with minimize surprise with penalized fitness (see Sec. 8.2.2). We do not compare against standard minimize surprise (see Sec. 8.3.1), since no decision-making mechanisms emerged in this case. As described in Sec. 8.2.2, we run 1,000 independent runs with black as the best option per setting (i.e., problem difficulty $\rho^*_{\mathrm{min}} \in \{0.25, 0.52, 0.67, 0.82\}$ and decision-making mechanism). The performance of the four decision-making mechanisms under investigation is compared based on mean consensus time $\overline{T}_N$ and exit probability E_N (see Sec. 8.2). We conducted control experiments with white as the best option and obtained comparable results; we do not show the data here.

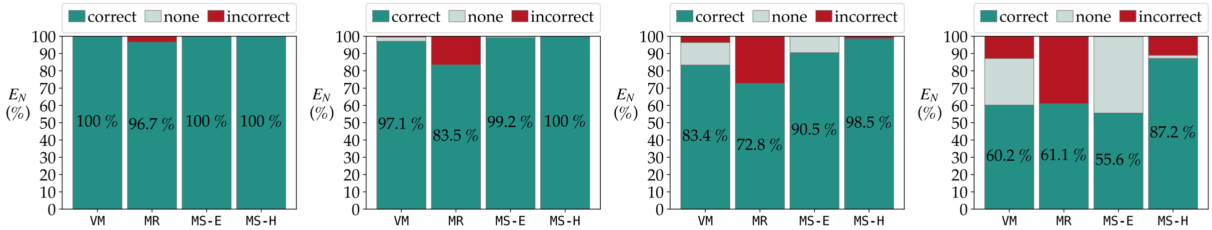
In general, we find increasing mean consensus times $\overline{T}_N$ and decreasing exit probabilities E_N with increasing problem difficulty ρ^*_{min} for all four decision-making mechanisms, see Tab. 8.3.

As already discussed by Valentini et al. [220, 222], the voter model is more accurate (i.e., higher E_N) while the majority rule is faster (i.e., shorter $\overline{T}_N$). This result is in line with the speed versus accuracy tradeoff (see Sec. 8.2). For the hardest problem difficulty (i.e., $\rho^*_{\mathrm{min}} = 0.82$), we find that the majority rule leads to a slightly higher exit probability E_N than the voter model. As visualized in Fig. 8.11d, the majority rule always leads to consensus – either a correct or an incorrect one. In contrast, the swarm did not reach a consensus in 27 % of the runs for problem difficulty $\rho^*_{\mathrm{min}} = 0.82$ with the voter model. We expect that the exit probability E_N is also higher for the voter model than for the majority rule in this problem difficulty when increasing runtime with the expense of an even higher mean consensus time $\overline{T}_N$.



(a) $\rho^*_{\text{min}} = 0.25$ **(b)** $\rho^*_{\text{min}} = 0.52$ **(c)** $\rho^*_{\text{min}} = 0.67$ **(d)** $\rho^*_{\text{min}} = 0.82$

Figure 8.10: Consensus time T_N (see Sec. 8.2) per problem difficulty ρ^*_{min} for the benchmark runs in our collective perception scenario. We compare voter model (`VM`), majority rule (`MR`), an ANN pair evolved with minimize surprise with penalized fitness in the easier task difficulty $\rho^*_{\text{min}} = 0.25$ (`MS-E`), and an ANN pair evolved in the harder task difficulty $\rho^*_{\text{min}} = 0.52$ (`MS-H`). Each box summarizes the data of 1,000 independent runs. Red bars give the median consensus time, blue bars the mean consensus time $\overline{T}_N$.



(a) $\rho^*_{\text{min}} = 0.25$ **(b)** $\rho^*_{\text{min}} = 0.52$ **(c)** $\rho^*_{\text{min}} = 0.67$ **(d)** $\rho^*_{\text{min}} = 0.82$

Figure 8.11: Exit probability E_N (see Sec. 8.2; green part of the bars) per problem difficulty ρ^*_{min} for the benchmark runs in our collective perception scenario. We compare voter model (`VM`), majority rule (`MR`), an ANN pair evolved with minimize surprise with penalized fitness in the easier task difficulty $\rho^*_{\text{min}} = 0.25$ (`MS-E`), and an ANN pair evolved in the harder task difficulty $\rho^*_{\text{min}} = 0.52$ (`MS-H`). Values are calculated based on 1,000 runs per decision-making mechanism and problem difficulty ρ^*_{min}. The gray parts of the bars give the percentage of runs in which no consensus was reached and the red parts indicate the percentage of runs in which an incorrect consensus (i.e., for the worse option) was reached.

The decision-making mechanism evolved in the easier problem difficulty setting (`MS-E`) results in comparable decision speed and accuracy as the voter model, see Figs. 8.10 and 8.11. While not all runs lead to consensus, none leads to a wrong consensus. We expect an increase in the exit probability E_N when increasing run time.

The decision-making mechanism evolved in the harder problem difficulty setting (`MS-H`) results in the best decision accuracy of the four compared mechanisms, see Fig. 8.11. Except for the easiest task difficulty $\rho^*_{\text{min}} = 0.25$, the majority rule outperforms `MS-H` in decision speed, but `MS-H` is faster than the voter model and `MS-E`. Overall, `MS-H` is a fast and accurate decision-making mechanism. This is probably caused by the penalized fitness F_p (Eq. 8.5) used to evolve this behavior. The penalized fitness explicitly rewards decision accuracy by penalizing wrong opinions and implicitly rewards decision speed by minimizing surprise. Minimize surprise intrinsically rewards fast decisions, since a consensus is easy to predict for the swarm members. In all problem difficulties ρ^*_{min}, `MS-H` outperforms `MS-E`. This suggests that optimizing controllers in harder problem difficulties leads to the emergence of better performing decision-making mechanisms. In future work, we will investigate whether this is generally true or whether there is an ideal problem difficulty for optimizing the decision-making mechanisms.

A big difference between voter model, majority rule, and the decision-making mechanisms evolved using minimize surprise with penalized fitness (`MS-E` and `MS-H`) is the data used to make decisions. Voter model and majority rule make their decision solely based on their neighbors' opinions. By contrast, the evolved decision-making mechanisms use both their neighbors' opinions and their ground sensor readings. This enables swarm members to determine the best option independently of the rest of the swarm, for example, when they receive sparse opinions from their neighbors. We assume that this additional input contributes to the better performance of the `MS-H` decision-making mechanism. Detailed analysis to verify this assumption is left for future work.

8.4 Discussion and Conclusion

Collective decision-making is essential to make a swarm autonomous on the macro-level (i.e., the global or swarm level) [6] by enabling swarms, for example, to synchronize or to allocate tasks [119]. But, as mentioned in the introduction, collective decision-making is more complex than our previously studied spatial organization and navigation behaviors, since the swarm's decision does not directly affect the environmental features it should potentially be based on. Thus, it is not surprising that our study on evolving collective decision-making mechanisms with minimize surprise in the collective perception scenario shows limitations of our approach. The standard minimize surprise approach exploits the easiest possible way to reach high prediction accuracy, which may lead to undesired behaviors for the system designer. In our case, the coupling between possible collective behaviors and the sensors perceiving the quality of the environmental features was too weak to evolve desired collective decision-making mechanisms. Instead, we found behaviors that let the swarm quickly reach a consensus for one option – regardless of the actual best option. Several options to push evolution with our standard minimize surprise approach to desired behaviors can be explored, such as increasing the weight of the ground sensors in the prediction task.

Here, we chose to add task-specific evolutionary pressure to evolve the desired decision-making mechanisms. We found competitive decision-making mechanisms both regarding decision speed and accuracy. However, as we have introduced an evolutionary pressure towards reaching a correct consensus, the influence of minimize surprise on the resulting behaviors is unknown. We expect that minimize surprise increases decision speed as fast decisions allow for easy predictions of neighbor opinions. We will validate this assumption in additional experiments in future work. While the mean inputs and outputs of the ANN pairs give a first glimpse into the evolved decision-making mechanisms, an in-depth study of the resulting behaviors could be informative and is future work.

We found that the decision-making mechanisms evolved in the harder problem difficulty have better scalability in problem difficulty. We aim to find the ideal problem difficulty for evolving controllers in future experiments and want to test whether increasing problem difficulty over generations may be beneficial.

Overall, this study is a first step towards the evolution of decision-making mechanisms with minimize surprise. Our first results are promising, but further analysis is necessary to fully understand the dynamics of the evolutionary process and the resulting decision-making mechanisms.

Collective Construction 9

Chapter Contents

In this chapter, we study the evolution of behaviors for collective construction in 2D torus grid worlds with minimize surprise. We...

- *Sec. 9.1:* introduce the experimental setup and
- *Sec. 9.2:* the evaluation metrics,
- *Sec. 9.3:* study the impact of the agent-block ratio on the emergent behaviors,
- *Sec. 9.4:* engineer self-organized construction, and
- *Sec. 9.5:* draw a conclusion.

Parts of this chapter are based on [47].

Approaches to object clustering and collective construction (see Sec. 2.2.1) differ in their complexity ranging from simple reactive control [193, 201] to calculating local rules for the robots offline [195]. We aim for the evolution of object clustering and collective construction behaviors using our minimize surprise approach in simple 2D torus grid worlds in this scenario (research question Q5, Sec. 1.2). By providing manipulable objects (blocks) to the agents (see Fig. 9.1), we increase task complexity compared to our previous works in which we generated behaviors requiring only agent-agent interaction, such as collective motion [28, 37] (see Ch. 7) and self-assembly [38, 39] (see Ch. 4), or minimal agent-environment interaction, such as collective decision-making (see Ch. 8). Agents can then push these blocks around to form different structures and thereby change their environment, that is, the environment is potentially dynamic beyond the dynamics of the swarm itself.

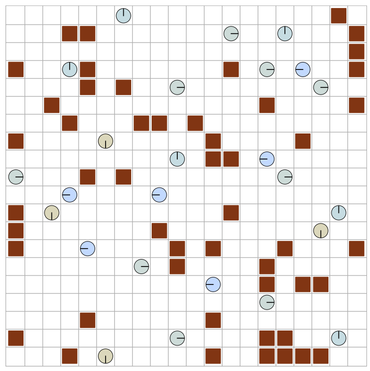


Figure 9.1: Illustration of the collective construction scenario. Agents are represented by circles, their color and the lines give their headings. Blocks are represented by brown squares.

9.1 Experimental Setup

For our collective construction scenario, we use a simple simulation of a 2D torus grid world with a homogeneous swarm of N simulated agents as in our self-assembly scenario (see Ch. 4). We use two different side lengths $L \in \{16, 20\}$ (here, number of grid cells) of the square that serves as the fundamental polygon of the torus. Additionally, we distribute B blocks of building material in the environment. These blocks can be moved by agents and thus agents can change their environment. But in contrast to our dynamic environments in Ch. 7, blocks do not change position without being actively manipulated by agents.

Table 9.1: Experimental setups for the collective construction scenario with number of agents N, number of blocks B, agent-block ratio $N : B$, and grid size $L \times L$.

N	B	$N : B$	$L \times L$
10	32	5 : 16	16×16
16	32	1 : 2	16×16
32	32	1 : 1	16×16
20	50	2 : 5	20×20
25	50	1 : 2	20×20
50	50	1 : 1	20×20
25	75	1 : 3	20×20

Both agents and blocks occupy one grid cell each and each grid cell can be occupied by either one agent or one block. Changeable quantities in this scenario are swarm density and block density, and hence the agent-block ratio. We keep a constant block density ($D_B = \frac{B}{L \times L}$) of 0.13 in all, except one, experimental setups for our collective construction scenario. Swarm density ($D_N = \frac{N}{L \times L}$) and agent-block ratio are varied by using different swarm sizes N. Thereby, we can investigate the effects of the agent-block ratio on the emergent behaviors. All studied setups are summarized in Tab. 9.1.

Our simulated swarm consists of simple agents. Each agent has two sets of binary sensors (agent sensors, block sensors) covering each the six grid cells in front of it, see Fig. 9.2, that is, a total of $R = 12$ sensors. Sensors $s_0, \ldots, s_5$ (see Fig. 9.2a) enable agents to sense other agents while sensors $s_6, \ldots, s_{11}$ (see Fig. 9.2b) enable the observation of blocks. Each agent n has a position $P_n(t) = (x_n(t), y_n(t))$ and a discrete heading $H_n(t) = (h_n^x(t), h_n^y(t))$, that is either North $H = (0, 1)$, East $H = (1, 0)$, South $H = (0, -1)$, or West $H = (-1, 0)$, at time step t. In each time step, agents execute one of two possible actions: moving one grid cell forward, that is,

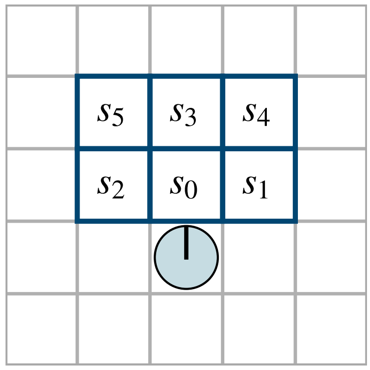
$$\begin{aligned} P_n(t+1) &= ((x_n(t) + h_n^x(t)) \bmod L_x, \\ &\qquad (y_n(t) + h_n^y(t)) \bmod L_y) \,, \\ H_n(t+1) &= H_n(t) \,, \end{aligned} \tag{9.1}$$

or rotating $\pm$ 90° on the spot, that is,

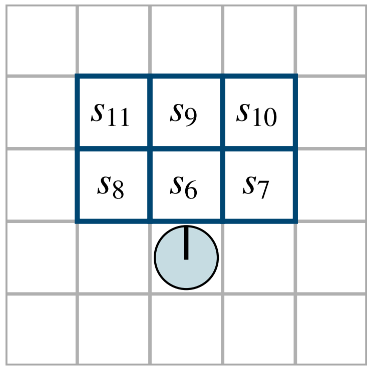
$$\begin{aligned} P_n(t+1) &= P_n(t), \\ H_n(t+1) &= (-h_n^y(t), h_n^x(t)) \text{ or} \\ H_n(t+1) &= (h_n^y(t), -h_n^x(t)) \,. \end{aligned} \tag{9.2}$$

A move forward is only possible if the grid cell in front is not occupied by another agent or if the agent attempts to push maximally one block to an empty grid cell. If an agent attempts to move on a grid cell already occupied by another agent, to push a block to an already occupied grid cell or to push more than one block, the move forward is prevented and the agent stays on its current grid cell.

Following our minimize surprise approach (see Ch. 3), we equip each agent with an actor-predictor ANN pair as visualized in Fig. 9.3. Actor and predictor are both three-layer ANNs with an input, one hidden and an output layer. We use the hyperbolic tangent `tanh` as the transfer function and map the network outputs to our discrete action values and sensor value predictions. The actor network (see Fig. 9.3a)

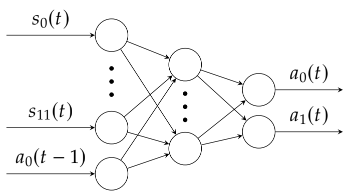


(a) agent sensors

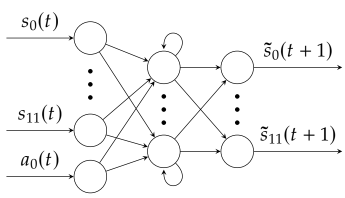


(b) block sensors

Figure 9.2: Sensor model in the collective construction scenario with labels for each sensor. Each agent has 12 binary sensors covering the six grid cells in front of it: one set of six sensors senses agents, another set of six sensors senses blocks. The blue circle represents the agent and the black line indicates its heading.



(a) actor



(b) predictor

Figure 9.3: Actor and predictor networks in the collective construction scenario. $a_0(t-1)$ is the agent's last action value and $a_0(t)$ is its next action determining whether to move or turn. $a_1(t)$ is its turning direction. $s_0(t), \ldots, s_{11}(t)$ are the agent's 12 sensor values at time step t, $\tilde{s}_0(t+1), \ldots, \tilde{s}_{11}(t+1)$ are its sensor predictions for time step $t+1$ [47].

parameter	value
grid side length L	$\{16, 20\}$
swarm size N	$\{10, 16, 20, 25, 32, 50\}$
# of blocks B	$\{32, 50, 75\}$
# of sensors and predictor outputs R	12
sensor values s_r	$\{0, 1\}$
action value a_0	{straight,turn}
action value a_1	±90°
population size μ	50
number of generations $g_{\max}$	100
evaluation length T (time steps)	1,000
# of simulation runs per fitness evaluation	10
elitism	1
mutation rate p_{mut}	0.1

Table 9.2: Parameters for the collective construction scenario.

has 13 input neurons, seven hidden neurons, and two output neurons. It determines the agent's next action by outputting two action values: $a_0(t)$ decides whether to move or turn and $a_1(t)$ determines the turning direction (i.e., ±90°). The actor receives the agent's 12 current sensor values $s_0(t), \ldots, s_{11}(t)$ and its last action $a_0(t-1)$ as inputs. As in our self-assembly scenario (see Ch. 4), we use only $a_0(t)$ as input to the ANNs as the turning direction $a_1(t)$ is solely informative when $a_0(t)$ selects to turn. The predictor network (see Fig. 9.3b) has 13 input neurons, 12 hidden recurrent neurons, and 12 output neurons. It outputs predictions $\tilde{s}_0(t+1), \ldots, \tilde{s}_{11}(t+1)$ for the $R = 12$ sensor values of the next time step $t+1$. That is, agents predict whether they may see other agents and also whether they may see blocks. The predictor receives the agent's 12 current sensor values $s_0(t), \ldots, s_{11}(t)$ and its next action $a_0(t)$ as inputs.

As in all scenarios, we evolve the actor-predictor ANN pairs using a simple evolutionary algorithm and reward high prediction accuracy as defined by our minimize surprise fitness function (Eq. 3.1). Consequently, agents can simplify the prediction of whether they may see blocks by creating 'boring environments' with areas of few blocks and areas of many blocks. Our swarm is homogeneous both related to the agent model and the controller, that is, each swarm member has an instance of the same genome in a given evaluation. Genomes encode the synaptic weights of both neural networks (see Ch. 3.2) and we randomly generate the initial population $\mathcal{P}(0)$ by drawing the weights from a uniform distribution in $[-0.5, 0.5]$. We run the evolutionary algorithm for $g_{\max} = 100$ generations and evaluate each genome in ten independent simulation runs for $T = 1{,}000$ time steps each using uniformly random initial agent and block positions. The fitness of a genome is the minimum fitness (Eq. 3.1) observed in those ten evaluations. For the evolutionary algorithm, we use a population size μ of 50, proportionate parent selection, age-based survivor selection, and elitism of one. We generate $\lambda = \mu - 1$ offspring for the population of the next generation. Mutation adds a uniformly random number in $[-0.4, 0.4]$ to each value v of a genome with a probability p_{mut} of 0.1. We evaluate all experimental setups (see Tab. 9.1) in 20 independent evolutionary runs and post-evaluate the best evolved individuals (i.e., the ANN pair with the highest fitness in the last generation of an evolutionary run). Tab. 9.2 summarizes all parameters.

9.2 Evaluation Metrics and Methods

We validate our approach by conducting post-evaluation of the best evolved individuals. In this section, we define several metrics for the quantitative evaluation and classification of the emergent behaviors.

9.2.1 Metrics

We define two metrics next to fitness (Eq. 3.1) for the evaluation of the best evolved individuals in our collective construction scenario: (i) similarity S of block positions at the start and the end of the run and (ii) agent movement M_N and block movement M_B.

The similarity S of the block positions is defined as the quantity of grid cells that are both occupied by blocks at the start (i.e., time step $t = 0$) and at the end (i.e., time step $t = T$) of the run normalized by the total number of blocks. It serves as an indicator to assess how much the block structure was changed by the agents. We define similarity S as

$$S(\mathcal{S}(0), \mathcal{S}(T)) = \frac{1}{B} \sum_{P_b \in \mathcal{S}(T)} \text{match}(P_b) \tag{9.3}$$

with number of blocks B, sets $\mathcal{S}(t) = \{P_b(t) : b \in [0 \mathinner{..} B-1]\}$ containing the position $P_b(t) = (x_b(t), y_b(t))$ of each block b at time step t, and match

$$\text{match}(P_i) = \begin{cases} 1, & \text{if } \exists U_i \in \mathcal{S}(0) : P_i = U_i \\ 0, & \text{otherwise} \end{cases} . \tag{9.4}$$

In addition, we measure the movement of agents M_N and of blocks M_B, that is, the mean distance covered by agents or the mean distance blocks were pushed. It is the mean accumulated displacement of agents or blocks over a time period of $\tau = \frac{L \times L}{2}$ time steps as in our previous work [28]. We calculate displacement using the Manhattan distance, since our grid world environment and the discrete agent headings do not allow diagonal movement. The Manhattan distance $d_M(P_n, P_m)$ between two positions $P_n = (x_n, y_n)$

and $P_m = (x_m, y_m)$ on a torus with side length L is defined as

$$d_M(P_n, P_m) = \min(|x_n - x_m|, L - |x_n - x_m|) + \min(|y_n - y_m|, L - |y_n - y_m|)\,. \tag{9.5}$$

We define agent movement M_N or block movement M_B as

$$M_N = M_B = \frac{1}{\tau\Psi} \sum_{t=T-\tau}^{T-1} \sum_{\psi=0}^{\Psi-1} d_M(P_\psi(t), P_\psi(t+1)), \tag{9.6}$$

with Ψ being the swarm size N in the case of measuring agent movement M_N and Ψ being the number of blocks B in the case of measuring block movement M_B; $P_\psi(t)$ and $P_\psi(t+1)$ are the positions of agent or block ψ at time steps t and $t+1$, respectively. Block movement indicates how much agents have manipulated their environment.

9.2.2 Classification of Emergent Block Structures

In our collective construction scenario, we classify the resulting behaviors by the best evolved individuals based on the structures formed by blocks. We differentiate between four different types of block structures: pairs (`PR`), lines (`LN`), clustering (`CL`), and random dispersion (`RD`). Block structures formed by the best evolved individuals at the start ($t = 0$) and the end ($t = T$) of a run are automatically classified based on their highest resemblance to one of the four structure types using Python scripts.[1] The highest resemblance to a structure type is determined by measuring the solution quality q_Z for each of the four possible block structures Z at the respective time step t.[2] The formed block structure is then labeled according to its highest solution quality (i.e., $\max(\{q_{\mathrm{PR}}, q_{\mathrm{LN}}, q_{\mathrm{CL}}, q_{\mathrm{RD}}\})$). We define solution quality q_Z of structure type Z as

$$q_Z = \frac{1}{B} \sum_{b=0}^{B-1} c_Z^b\,, \tag{9.7}$$

with number of blocks B and criterion c_Z^b of structure type $Z \in \{\mathrm{PR}, \mathrm{LN}, \mathrm{CL}, \mathrm{RD}\}$ that evaluates to one ($c_Z^b = 1$) if block b fulfills the structure criterion and to zero ($c_Z^b = 0$) otherwise. Thus, solution quality measures how many blocks from the set of all blocks $\mathcal{S}_B$ fulfill the structure criterion.

1: `https://gitlab.iti.uni-luebeck.de/minimize-surprise/collective-construction-torus`

2: For conciseness, we denote by P_b the position $P_b(t)$ of block b in time step t of the post-evaluation run, that is, we omit the time step in our notation in the following.

Pairs (PR) and lines (LN) are formed by blocks that are horizontally or vertically placed next to each other as illustrated in Figs. 9.4 and 9.5. The criteria for pairs and lines differ only in the structure length: pairs consist of two blocks, while lines are at least three blocks long. An individual pair or line k is formed out of a set $\mathcal{S}^k_{\mathrm{PL}}$ of adjacent blocks $b \in \mathcal{S}_B$ where $\forall b, m \in \mathcal{S}^k_{\mathrm{PL}} \exists r \in \mathbb{Z} : P_b = (P_m + r\gamma) \bmod L$, and $\forall b, m \in \mathcal{S}^k_{\mathrm{PL}} : d_M(P_b, P_m) < |\mathcal{S}^k_{\mathrm{PL}}|$ with block positions $P_b = (x_b, y_b)$ and $P_m = (x_m, y_m)$, structure orientation $\gamma = (\gamma_x, \gamma_y) = (1, 0)$ for horizontal lines and pairs and $\gamma = (0, 1)$ for vertical lines and pairs, and Manhattan distance $d_M(\cdot, \cdot)$ (Eq. 4.5) hold. That is, each block in a pair or line structure has at least one neighboring block on an adjacent grid cell in the direction of the structure's orientation. Lines and pairs can both have up to half of their length of neighboring blocks on each side next to them (i.e., $|\mathcal{S}^k_{\mathrm{NB},1}| \le \frac{|\mathcal{S}^k_{\mathrm{PL}}|}{2}$, $|\mathcal{S}^k_{\mathrm{NB},2}| \le \frac{|\mathcal{S}^k_{\mathrm{PL}}|}{2}$ with $\mathcal{S}^k_{\mathrm{NB},1}$ and $\mathcal{S}^k_{\mathrm{NB},2}$ being sets of neighboring blocks per side of the structure[3]), whereby no two adjacent grid cells parallel to the structure are allowed to be occupied by blocks. The latter can be determined using the Manhattan distance (Eq. 9.5) as all neighboring blocks have to be at least one grid cell apart, which equals a Manhattan distance of at least two. Several pairs or lines $\mathcal{S}^k_{\mathrm{PL}}$ can be assembled out of all blocks $\mathcal{S}_B$ on the grid. The set $\mathcal{S}_{\mathrm{PR}}$ of all assembled pairs is given by

$$\begin{aligned}\mathcal{S}_{\mathrm{PR}} = \{\mathcal{S}^k_{\mathrm{PL}} : |\mathcal{S}^k_{\mathrm{PL}}| = 2, |\mathcal{S}^k_{\mathrm{NB},1}| \le \frac{|\mathcal{S}^k_{\mathrm{PL}}|}{2}, |\mathcal{S}^k_{\mathrm{NB},2}| \le \frac{|\mathcal{S}^k_{\mathrm{PL}}|}{2},\\ \forall i, j \in (\mathcal{S}^k_{\mathrm{NB},1} \cup \mathcal{S}^k_{\mathrm{NB},2}), i \ne j : d_M(P_i, P_j) \ge 2,\\ 0 \le k < K\},\end{aligned} \tag{9.8}$$

with K being the number of sets of adjacent blocks $\mathcal{S}^k_{\mathrm{PL}}$. We define the criterion c^b_{PR} for block b being part of a pair structure as

$$c^b_{\mathrm{PR}} = \begin{cases} 1, & \text{if } \exists \mathcal{S}_{\mathrm{PL}} \in \mathcal{S}_{\mathrm{PR}} : b \in \mathcal{S}_{\mathrm{PL}} \\ 0, & \text{otherwise} \end{cases}. \tag{9.9}$$

3: All neighboring blocks b above a horizontal line or pair structure (i.e., $\gamma = (1, 0)$) or, respectively, to the right of a vertical line or pair structure (i.e., $\gamma = (0, 1)$) are given by set $\mathcal{S}^k_{\mathrm{NB},1} = \{b : (x_b, y_b) = (x_m + \gamma_y, y_m + \gamma_x), b \in \mathcal{S}_B, m \in \mathcal{S}^k_{\mathrm{PL}}\}$. Similarly, all neighboring blocks b below a horizontal line or pair structure (i.e., $\gamma = (1, 0)$) or, respectively, to the left of a vertical line or pair structure (i.e., $\gamma = (0, 1)$) are given by set $\mathcal{S}^k_{\mathrm{NB},2} = \{b : (x_b, y_b) = (x_m - \gamma_y, y_m - \gamma_x), b \in \mathcal{S}_B, m \in \mathcal{S}^k_{\mathrm{PL}}\}$.

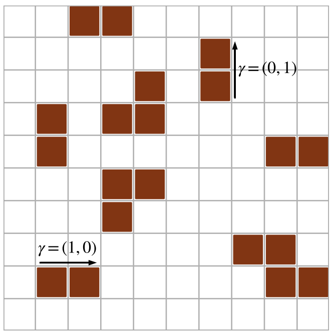


Figure 9.4: Illustration of pairs (PR) in our collective construction scenario. Blocks are represented by brown squares.

Accordingly, the set S_{LN} of all assembled lines is given by

$$\mathcal{S}_{\mathrm{LN}} = \{\mathcal{S}^k_{\mathrm{PL}} : |\mathcal{S}^k_{\mathrm{PL}}| > 2, |\mathcal{S}^k_{\mathrm{NB},1}| \leq \frac{|\mathcal{S}^k_{\mathrm{PL}}|}{2}, |\mathcal{S}^k_{\mathrm{NB},2}| \leq \frac{|\mathcal{S}^k_{\mathrm{PL}}|}{2},$$
$$\forall i, j \in (\mathcal{S}^k_{\mathrm{NB},1} \cup \mathcal{S}^k_{\mathrm{NB},2}|), i \neq j : d_M(P_i, P_j) \geq 2,$$
$$0 \leq k < K\} . \tag{9.10}$$

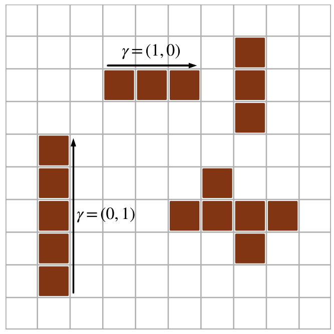


Figure 9.5: Illustration of lines (LN) in our collective construction scenario. Blocks are represented by brown squares.

As already stated above, lines differ only by the length of the structure ($|\mathcal{S}^k_{\mathrm{PL}}| > 2$) from pairs. We define the criterion c^b_{LN} for block b being part of a line structure as

$$c^b_{\mathrm{LN}} = \begin{cases} 1, & \text{if } \exists \mathcal{S}_{\mathrm{PL}} \in \mathcal{S}_{\mathrm{LN}} : b \in \mathcal{S}_{\mathrm{PL}} \\ 0, & \text{otherwise} \end{cases} . \tag{9.11}$$

We define clustering (CL) and random dispersion (RD) based on the Moore (see Fig. 9.6b), von Neumann (see Fig. 9.6a), and diagonal (see Fig. 9.6c) neighborhoods. The set $\mathcal{S}^b_{\mathrm{VN}}$ of blocks in the von Neumann neighborhood of a block b is given by

$$\mathcal{S}^b_{\mathrm{VN}} = \{m : d_M(P_m, P_b) = 1, m \in \mathcal{S}_B\} , \tag{9.12}$$

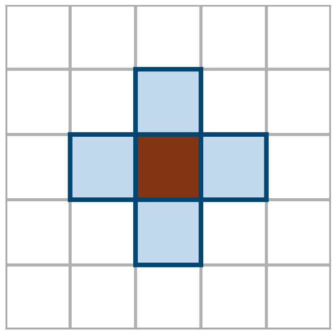
that is, all neighboring blocks with a Manhattan distance of one. The set $\mathcal{S}^b_M$ of blocks in the Moore neighborhood of a block b is given by

$$\mathcal{S}^b_{\mathrm{M}} = \{m : \min(|x_m - x_b|, L - |x_m - x_b|) \leq 1,$$
$$\min(|y_m - y_b|, L - |y_m - y_b|) \leq 1, \tag{9.13}$$
$$m \in \mathcal{S}_B, m \neq b\} .$$

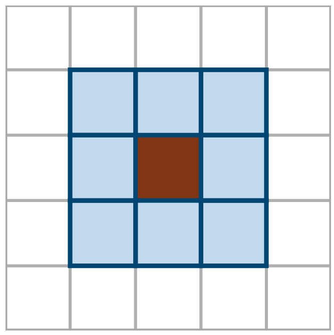
The set $\mathcal{S}^b_D$ of blocks in the diagonal neighborhood of a block b is given by

$$\mathcal{S}^b_D = \{m : \min(|x_m - x_b|, L - |x_m - x_b|) = 1,$$
$$\min(|y_m - y_b|, L - |y_m - y_b|) = 1, m \in \mathcal{S}_B\} . \tag{9.14}$$

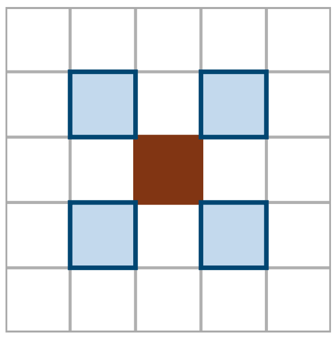
Clusters (Fig. 9.7) are formed by blocks that have at least four blocks in their Moore neighborhood and at least three blocks in their von Neumann neighborhood, as well as their neighbors. Blocks that are part of a line (i.e., $c^b_{\mathrm{LN}} = 1$) cannot



(a) von Neumann neighborhood



(b) Moore neighborhood



(c) diagonal neighborhood

Figure 9.6: Moore, von Neumann, and diagonal neighborhoods of a block. Blocks are represented by brown squares, and blue grid cells give the neighboring grid cells.

be part of a cluster and are excluded from the calculation of the cluster criterion. We define c^i_{BNH} as the measurement if block i has at least four blocks in its Moore neighborhood and at least three blocks in its von Neumann neighborhood. c^i_{BNH} is defined as

$$c^i_{\mathrm{BNH}} = \begin{cases} 1, & \text{if } |\mathcal{S}^i_{\mathrm{M}}| \geq 4 \land |\mathcal{S}^i_{\mathrm{VN}}| \geq 3 \\ 0, & \text{otherwise} \end{cases} . \qquad (9.15)$$

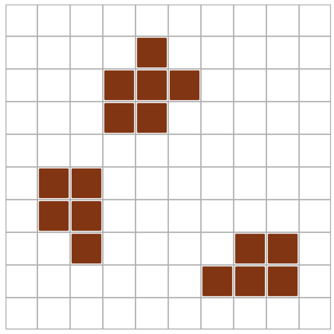


Figure 9.7: Illustration of clustering (CL) in our collective construction scenario. Blocks are represented by brown squares.

The criterion c^b_{CL} for block b to be classified as a part of a cluster is then defined as

$$c^b_{\mathrm{CL}} = \begin{cases} 1, & \begin{array}{l} \text{if } c^b_{\mathrm{BNH}} = 1 \\ \quad \lor (\exists m \in \mathcal{S}_B : (c^m_{\mathrm{BNH}} = 1 \land b \in \mathcal{S}^m_M)) \end{array} \\ 0, & \text{otherwise} \end{cases} , \qquad (9.16)$$

that is, block b has either itself sufficiently many neighbors as defined by c^i_{BNH} to be classified as a part of a cluster or is in the neighborhood of a block m that fulfills this criterion.

Randomly dispersed blocks (Fig. 9.8) have maximally one direct diagonal neighbor. This means that these blocks have no neighbors in their von Neumann neighborhood $\mathcal{S}^b_{\mathrm{VN}}$ and maximally one neighbor in their diagonal neighborhood $\mathcal{S}^b_{\mathrm{D}}$. We thus define the criterion c^b_{RD} for block b to be classified as randomly dispersed as

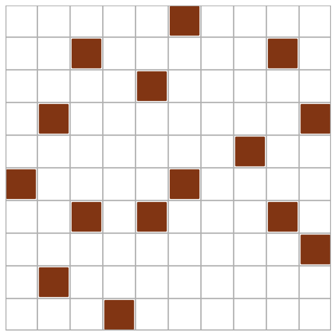


Figure 9.8: Illustration of random dispersion (RD) in our collective construction scenario. Blocks are represented by brown squares.

$$c^b_{\mathrm{RD}} = \begin{cases} 1, & \text{if } |\mathcal{S}^b_{\mathrm{D}}| \leq 1 \land |\mathcal{S}^b_{\mathrm{VN}}| = 0 \\ 0, & \text{otherwise} \end{cases} . \qquad (9.17)$$

9.3 Impact of the Agent-Block Ratio

In our first experiments, we study the impact of the agent-block ratio using the standard minimize surprise approach (see Ch. 3) in the seven different experimental setups (see Tab. 9.1). For six experimental setups, we use a constant block density of 0.13 by setting either 32 blocks on a 16×16 grid or 50 blocks on a 20×20 grid. On both grids, we do experiments with agent-block ratios of 1 : 1 (high swarm density) and 1 : 2 (intermediate swarm density). In addition, we use a ratio of 5 : 16 on the smaller grid and of 2 : 5 on the larger grid (low swarm densities), see Tab. 9.3. Furthermore, we increase the block density to 0.19 in a setup with 25 agents and 75 blocks (i.e., a 1 : 3 agent-block ratio) on the 20×20 grid to show the effects on the resulting structures.

Table 9.3: Evaluation metrics (see Sec. 9.2) of 20 independent evolutionary runs in the collective construction scenario with N agents and B blocks: median best fitness F (Eq. 3.1) in the last time step of all runs; quantity, similarity S (Eq. 9.3), mean block movement M_B, and mean agent movement M_N (Eq. 9.6) for runs with block movement, that is, with a similarity S smaller one; percentages of formed block structures (i.e., lines LN, pairs PR, clustering CL, and random dispersion RD) at the start ($t = 0$) and end ($t = T$) of the runs. Median values in parentheses [47].

N	B	$N-B$ ratio	grid size	median fitness F	qty.	similarity $S<1.0$ S	M_B	M_N	t	structures LN	PR	CL	RD
10	32	5 : 16	16×16	0.91	11	0.78	0.0	0.43	0	0.0	20.0	0.0	80.0
						(0.88)	(0.0)	(0.47)	T	0.0	27.5	5.0	67.5
16	32	1 : 2	16×16	0.90	11	0.65	0.0	0.49	0	0.0	15.0	0.0	85.0
						(0.78)	(0.0)	(0.48)	T	2.5	15.0	12.5	70.0
32	32	1 : 1	16×16	0.89	14	0.418	0.0	0.38	0	0.0	7.5	0.0	92.5
						(0.34)	(0.0)	(0.42)	T	5.0	47.5	10.0	37.5
20	50	2 : 5	20×20	0.90	10	0.85	0.0	0.33	0	0.0	2.5	0.0	97.5
						(0.87)	(0.0)	(0.42)	T	0.0	7.5	0.0	92.5
25	50	1 : 2	20×20	0.90	10	0.82	0.0	0.43	0	0.0	7.5	0.0	92.5
						(0.83)	(0.0)	(0.45)	T	0.0	12.5	0.0	87.5
50	50	1 : 1	20×20	0.87	7	0.49	0.0	0.24	0	0.0	12.5	0.0	87.5
						(0.42)	(0.0)	(0.29)	T	5.0	27.5	0.0	67.5
25	75	1 : 3	20×20	0.86	6	0.78	0.0	0.20	0	5.0	50.0	0.0	45.0
						(0.82)	(0.0)	(0.20)	T	15.0	47.5	0.0	37.5

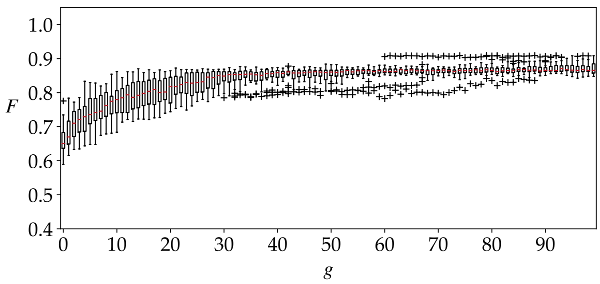


Figure 9.9: Best fitness F over generations g of 20 independent evolutionary runs on the 20×20 grid with 50 agents and 50 blocks in the collective construction scenario. Medians are indicated by the red bars [47].

A median best fitness (Eq. 3.1) of at least 0.86 in the last generation is reached across all experiments, see Tab. 9.3, meaning that a median of 86 % of the sensor values are predicted correctly by the prediction networks. We infer that the prediction task is easy as high fitness values are reached in all runs. Fig. 9.9 shows the increase of the best fitness over generations of 20 independent evolutionary runs using 50 agents and 50 blocks on a 20×20 grid. It is representative for the fitness curves observed in all experiments.

Since we are aiming for collective construction, we study the runs with altered block positions, that is, with a similarity (Eq. 9.3) lower than 1.0, in more detail. First, we compare the quantity of runs with altered block positions in the six different experimental setups with a block density of 0.13. For the smaller agent-block ratios, half of the 20 runs on the 20×20 grid and 11 runs on the 16×16 grid lead to the alteration of block positions. The number rises to 14 on the 16×16 grid and decreases to seven on the 20×20 grid for a 1 : 1 ratio. The mean and median similarities in these runs with altered block positions decrease (i.e., more moved blocks) with increasing agent-block ratios. Precisely, runs with a 1 : 1 ratio have about 40 percentage points (pp) lower median similarities than smaller ratios and thus block positions are altered most. In the scenario with increased block density (i.e., 75 blocks on a 20×20 grid), six runs lead to the alteration of block positions. The mean similarity is high, that is, few blocks are moved.

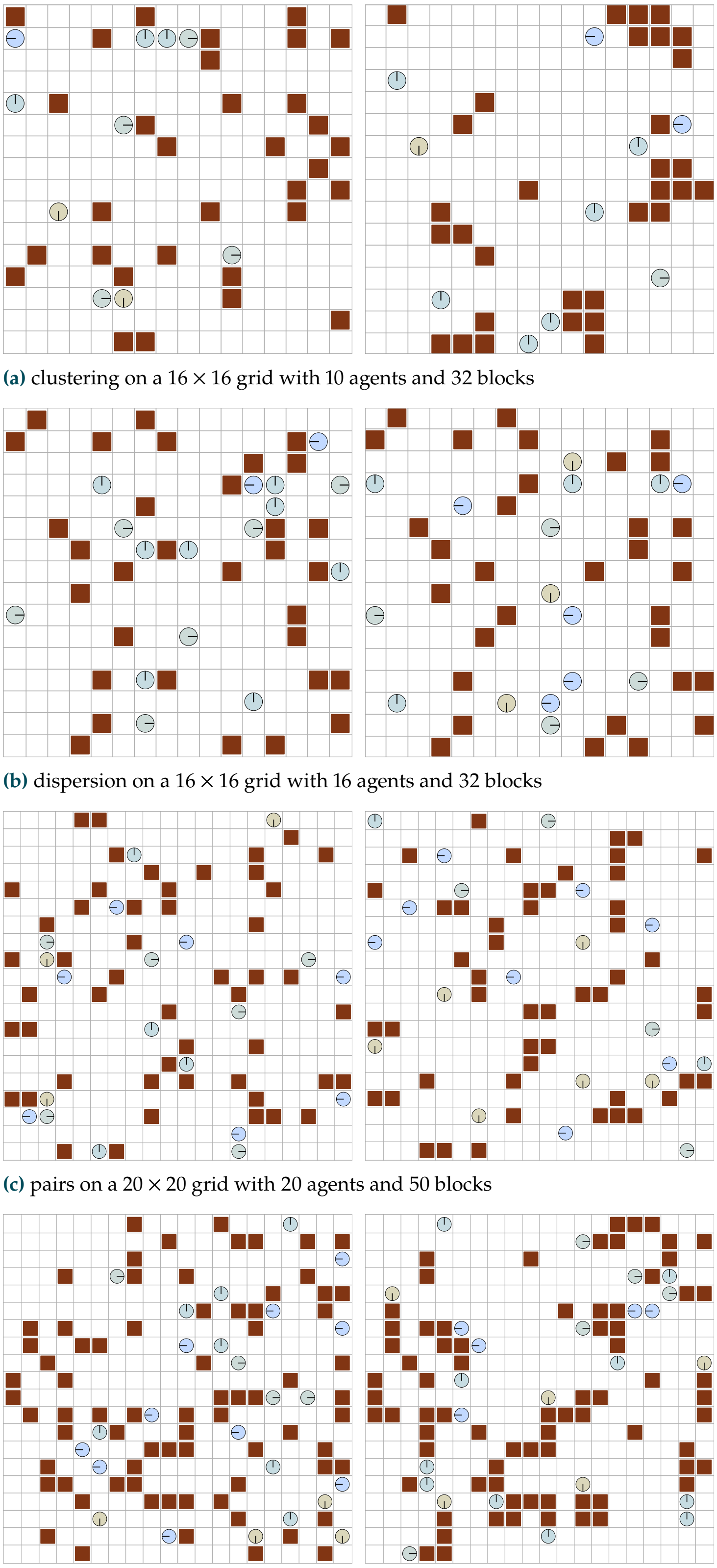
For all runs of the seven experimental setups, we measure no block movement (Eq. 9.6) during the last τ time steps (i.e., $\tau = 128$ for the 16×16 grid and $\tau = 200$ for the 20×20 grid). Consequently, block pushing happens, if any, at the beginning of the runs. The system then converges, that is, blocks have fixed positions and form stable structures. We find agent movement in all runs with block movement, that is, runs with a similarity below 1.0. In contrast, agent movement (Eq. 9.6) is mostly zero in runs without altered block positions (i.e., similarity S is 1.0) indicating that agents mostly turn in this case. Nevertheless, in a few runs agents move constantly without pushing any blocks or self-assemble into structures.

We classify the block structures formed by the best evolved behaviors at the end ($t = T$) of the run using our metrics defined in Sec. 9.2.2, see Tab. 9.3. The best evolved behaviors most frequently lead to the random dispersion of blocks (Fig. 9.10b) in all experiments except for the 1 : 3 agent-block ratio setup. The formation of pairs of blocks prevails in the latter scenario. Pairs of blocks (Fig. 9.10c) also form frequently in all other experiments while lines emerge rarely. Clusters (Fig. 9.10a) form only on the smaller grid, maybe because agents need to push blocks more grid cells forward to group them on the larger grid. Overall, we find a variety of swarm behaviors emerging due to our task-independent reward for high prediction accuracy. A video of emergent behaviors is online.[4]

4: `https://youtu.be/T9s5669ypXM`

Next we compare the distribution of formed block structures at the start ($t = 0$) and the end ($t = T$) of the runs of the best evolved individuals to estimate to which extend agents changed their environment, see Fig. 9.11. In the experimental setups with a block density of 0.13, the uniformly random initialization of block positions leads mostly to randomly dispersed block structures. In addition, in less than 20 % of the runs pairs of blocks form due to the random initialization. The denser block distribution in the experimental setup with the higher block density (i.e., block density of 0.19 with 75 blocks on a 20×20 grid) reduces the probability of dispersion and increases the probability of pairs during the initial random block placement.

In the scenarios with a block density of 0.13, we find that the best evolved behaviors decrease dispersion by 5 to 15 pp for the lower agent-block ratios. For the 1 : 1 agent-block ratio, dispersion decreases by 55 pp on the 16×16 grid and by 20 pp on the 20×20 grid during the runs. The percentage



(a) clustering on a 16 × 16 grid with 10 agents and 32 blocks

(b) dispersion on a 16 × 16 grid with 16 agents and 32 blocks

(c) pairs on a 20 × 20 grid with 20 agents and 50 blocks

(d) lines on a 20 × 20 grid with 25 agents and 75 blocks

Figure 9.10: Block structures at the start (left) and end (right) of a run in the collective construction scenario. Agents are represented by circles, their color and the lines give their headings. Blocks are represented by brown squares [47].

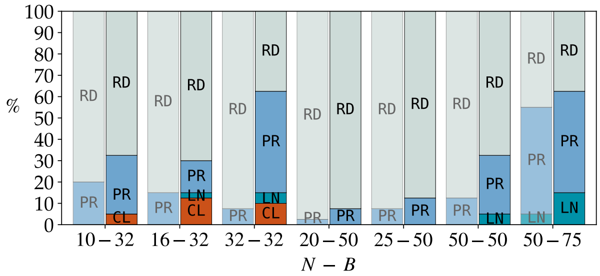


Figure 9.11: Distributions of formed block structures at the start ($t = 0$, left) and end ($t = T$, right) of the post-evaluation runs of the 20 best evolved individuals for the different experimental setups with swarm size N and block quantity B (see Tab. 9.1) in our collective construction scenario with clustering (CL), lines (LN), pairs (PR), and random dispersion (RD).

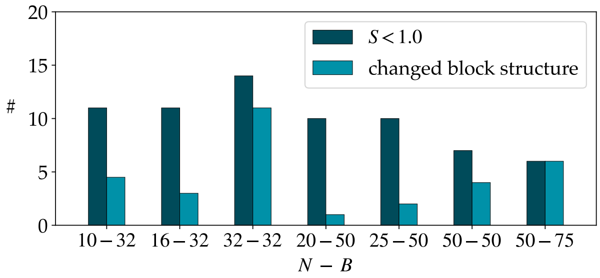


Figure 9.12: Quantity # of runs of the 20 best individuals with a similarity $S < 1.0$ (Eq. 9.3, left) and with different block structure classifications at the start ($t = 0$) and the end ($t = T$) of the runs (changed block structure, right) in our collective construction scenario for the different experimental setups with swarm size N and block quantity B (see Tab. 9.1).

drop in dispersion is always less than the initial percentage of randomly dispersed block structures. We find that agents do not push blocks into a new structure in all runs with altered block positions. The scenarios with lower agent-block ratios lead to ten to eleven runs with altered block positions (i.e., similarity $S < 1.0$), but in only one to four runs the block structure changes from start to end of the run, see Fig. 9.12. By contrast, the scenarios with a 1 : 1 agent-block ratio lead to different structure classifications at the end of the run than at the beginning of the run in four out of seven runs on the 20×20 grid and 11 out of 14 runs on the 16×16 grid. The scenario with a block density of 0.19 resulted in six runs with altered block positions (i.e., similarity $S < 1.0$), all of which have different block structure classifications at the start and end of the run. Nevertheless, the structure distribution does only slightly change with an increase of 10 pp in lines and a decrease of 2.5 pp and 7.5 pp, respectively, in pairs and random dispersion. Consequently, it is the only setup in which the amount of pairs decreases during the runs.

Overall, we find that the scenarios with a 1 : 1 agent-block ratio and a block density of 0.13 lead to the greatest alteration of block structures and thus the most active collective construction behaviors. Furthermore, different structures form on the two grid sizes; clustering, for example, can only be found on the smaller grid. Using a higher block density of 0.19 with a 1 : 3 agent-block ratio did not improve our results. Consequently, we focus on the first six experimental setups with the lower block density of 0.13 in the following.

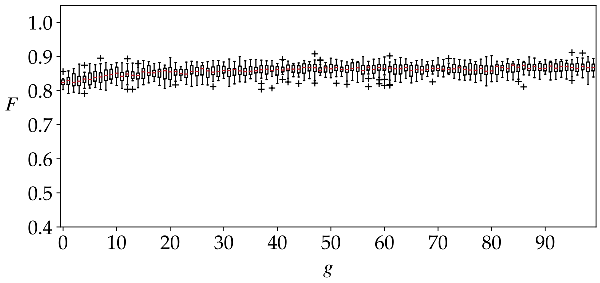


Figure 9.13: Best fitness F (Eq. 9.18) over generations g of 20 independent evolutionary runs on the 20×20 grid with 20 agents and 50 blocks in the collective construction scenario with predefined predictions aiming for pairs and lines. Medians are indicated by the red bars [47].

9.4 Engineered Self-Organized Construction

In the next experiments, we set the sensor predictions to fixed values to predefine that the resulting behaviors of the evolutionary process lead to the formation of desired block structures, while still rewarding high prediction accuracy (see Ch. 6). To elaborate, we still use the fitness function of our minimize surprise approach as defined in Eq. 3.1, but predefine the sensor predictions to values matching our targeted block structures. As before, actors are only indirectly rewarded by being paired with a predictor. High fitness can only be reached if the actors lead to 'predictable' behavior, that is, to the formation of the desired block structure here. We set the agent sensor predictions to 0 (i.e., $\tilde{s}_0 = \tilde{s}_1 = \tilde{s}_2 = \tilde{s}_3 = \tilde{s}_4 = \tilde{s}_5 = 0$, see Fig. 9.14) and vary the block sensor predictions in three different experiments.

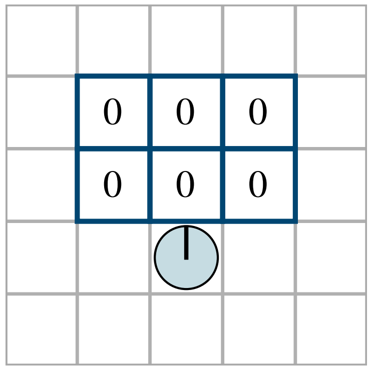


Figure 9.14: Agent sensor predictions for the collective construction scenario with predefined predictions. All six sensor predictions are set to 0, that is, agents predict to sense no other agents. The circle represents the agent and the line indicates its heading.

In the first experiment, we aim for pairs and lines by predefining the sensor predictions for the two block sensors in front of the agent to 1 (i.e., $\tilde{s}_6 = \tilde{s}_9 = 1$), while all other predictions are set to 0 (i.e., $\tilde{s}_7 = \tilde{s}_8 = \tilde{s}_{10} = \tilde{s}_{11} = 0$, see Fig. 9.15). Consequently, we require all agents to have a pair or line of blocks directly in front of them to maximize their fitness. Our minimize surprise fitness function (Eq. 3.1) adapts to

$$F = \frac{1}{TNR}\sum_{t=0}^{T-1}\sum_{n=0}^{N-1}\Bigg(\overbrace{\sum_{r\in\{0,1,2,3,4,5,7,8,10,11\}}(1-|\underbrace{\boxed{0}}_{\tilde{s}_r^n(t)=0}-s_r^n(t)|)}^{\text{predefined predictions: empty grid cells}} + \underbrace{\sum_{r\in\{6,9\}}(1-|\underbrace{\boxed{1}}_{\tilde{s}_r^n(t)=1}-s_r^n(t)|)}_{\text{predefined predictions: grid cells occupied by blocks}}\Bigg), \tag{9.18}$$

with evaluation length T in time steps, swarm size N, number of sensors per swarm member R, and prediction $\tilde{s}_r^n(t)$ for and value $s_r^n(t)$ of sensor r of swarm member n at time step t.

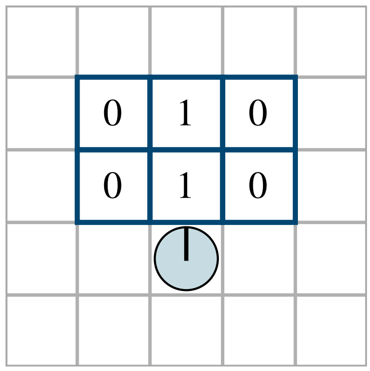
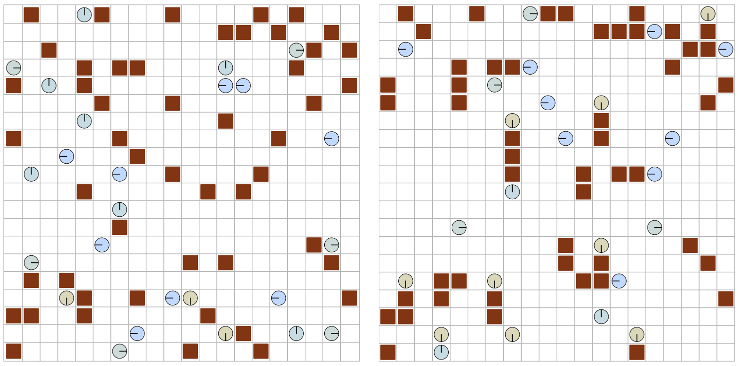


Figure 9.15: Block sensor predictions for the collective construction scenario with predefined predictions for pairs and lines of blocks. The circle represents the agent and the line indicates its heading.

We find a mean best fitness of about 0.8 in all six experimental setups, see Tab. 9.4. This is up to 0.08 lower than in our first

Table 9.4: Evaluation metrics (see Sec. 9.2) of 20 independent evolutionary runs in the collective construction scenario with predefined predicted pairs of blocks, N agents and B blocks: median best fitness (Eq. 3.1) in the last time step of all runs, similarity S (Eq. 9.3), mean block movement M_B, mean agent movement M_N (Eq. 9.6), and percentages of formed block structures (i.e., lines (LN), pairs (PR), clustering (CL), and random dispersion (RD)) at the start ($t = 0$) and end ($t = T$) of the runs. Median values in parentheses [47].

N	B	$N-B$ ratio	grid size	median fitness F	S	M_B	M_N	t	structures LN	PR	CL	RD
10	32	5 : 16	16 × 16	0.87	0.66	0.03	0.25	0	0.0	20.0	0.0	80.0
					(0.69)	(0.03)	(0.24)	T	0.0	85.0	2.5	12.5
16	32	1 : 2	16 × 16	0.85	0.56	0.02	0.21	0	0.0	12.5	0.0	87.5
					(0.56)	(0.0)	(0.22)	T	10.0	72.5	5.0	12.5
32	32	1 : 1	16 × 16	0.81	0.44	0.0	0.26	0	0.0	12.5	0.0	87.5
					(0.42)	(0.0)	(0.26)	T	15.0	55.0	15.0	15.0
20	50	2 : 5	20 × 20	0.87	0.55	0.01	0.18	0	0.0	0.0	0.0	100.0
					(0.56)	(0.0)	(0.20)	T	15.0	75.0	5.0	5.0
25	50	1 : 2	20 × 20	0.86	0.52	0.01	0.18	0	0.0	7.5	0.0	92.5
					(0.53)	(0.0)	(0.18)	T	5.0	85.0	5.0	5.0
50	50	1 : 1	20 × 20	0.82	0.42	0.0	0.29	0	0.0	7.5	0.0	92.5
					(0.43)	(0.0)	(0.30)	T	5.0	55.0	15.0	25.0



(a) initial positions **(b)** final positions

Figure 9.16: Resulting pair structure with predefined sensor predictions using 25 agents and 50 blocks on a 20 × 20 grid. Agents are represented by circles, their color and the lines give their headings. Blocks are represented by squares.

experiments without predefined predictions (see Tab. 9.3). Fig. 9.13 visualizes the increase of the best fitness over generations of 20 independent evolutionary runs with 20 agents and 50 blocks on a 20 × 20 grid with predefined predictions pushing emergence towards lines and pairs of blocks. It is representative for the fitness curves observed in all experiments with predefined predictions.

Block positions were altered in all runs, that is, we always have similarities S (Eq. 9.3) smaller than 1.0. For the lower agent-block ratios, the median similarity decreased by roughly 20 pp on the 16 × 16 grid and by 30 pp on the 20 × 20 grid compared to our initial experiments without predefined predictions (see Tab. 9.3). We thus find a greater alteration of the block structures using low agent-block ratios than before. This finding is supported by the formed structures as the best evolved behaviors decrease random dispersion during the runs by at least 67 pp. The setups with a 1 : 1 agent-block ratio reach similar median similarities in both the experiments without predefined predictions and the experiments with predefined predictions aiming for pair and line structures. Nevertheless, random dispersion decreases also by at least 67 pp in the runs with a 1:1 agent-block ratio when predefining predictions aiming for pairs and lines compared to a maximum of 55 pp in the experiments without predefining predictions. In all six experimental setups, the majority of formed structures are pairs (Fig. 9.16) and we also observe lines. Consequently, predefining predictions successfully pushes emergence towards the formation of pairs and lines of blocks.

In the second experiment, we want to provoke that the agent swarm forms clusters of blocks. We predefine that all block sensors predict that a block will be sensed (i.e., $\tilde{s}_6 = \tilde{s}_7 = \tilde{s}_8 = \tilde{s}_9 = \tilde{s}_{10} = \tilde{s}_{11} = 1$, Fig. 9.17). Consequently, our minimize surprise fitness function (Eq. 3.1) adapts to

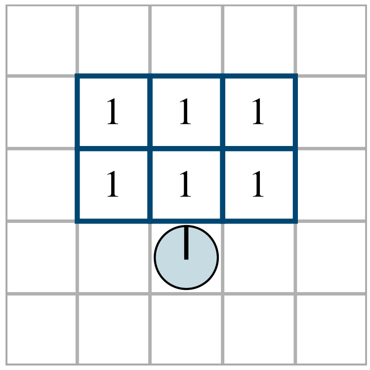


Figure 9.17: Block sensor predictions for the collective construction scenario with predefined predictions for clustering. The circle represents the agent and the line indicates its heading.

$$F = \frac{1}{TNR} \sum_{t=0}^{T-1} \sum_{n=0}^{N-1} \Bigg(\overbrace{\sum_{r \in \{0,1,2,3,4,5\}} (1 - |\underbrace{\boxed{0}}_{\tilde{s}_r^n(t) = 0} - s_r^n(t)|)}^{\text{predefined predictions: unoccupied grid cells}} + \underbrace{\sum_{r \in \{6,7,8,9,10,11\}} (1 - |\underbrace{\boxed{1}}_{\tilde{s}_r^n(t) = 1} - s_r^n(t)|)}_{\text{predefined predictions: grid cells occupied by blocks}} \Bigg), \tag{9.19}$$

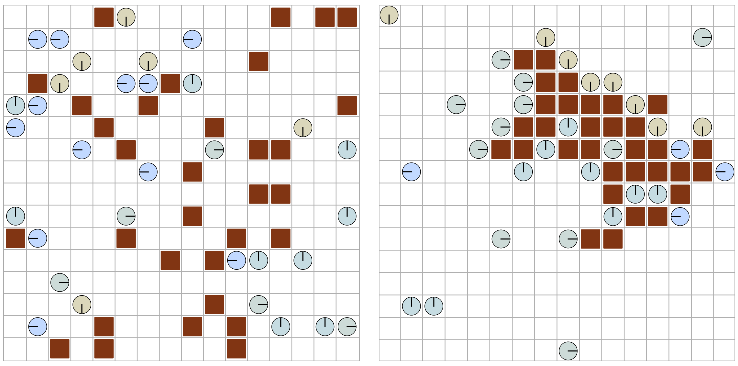
with evaluation length T in time steps, swarm size N, number of sensors per swarm member R, and prediction $\tilde{s}_r^n(t)$ for and value $s_r^n(t)$ of sensor r of swarm member n at time step t.

We find a median best fitness of 0.63 to 0.69 and thus around 0.2 lower values than in our previous experiments. We infer that the task complexity increased.

As when aiming for pairs and lines, block positions were altered in all runs (i.e., similarity $S < 1.0$). The median similarity of block positions decreases with increasing agent-block ratio on both grids from around 50 % to 22 %. Compared to all previous experiments, we reach the lowest similarities in this experiment and thus the most intense pushing of blocks. The percentage of random dispersion decreases by at least 65 pp during the runs and in three experimental setups (see Tab. 9.1) agents push all initially randomly dispersed block structures into other structure formations. We observe that the amount of emerging clusters varies with swarm density. Mainly pairs emerge for the two lower agent-block ratios on both grids. While no clusters form for the lowest agent-block ratio, one run on the 16×16 grid and roughly half of the runs on the 20×20 grid result in clusters for the $1:2$ agent-block ratio setups. The runs with the highest swarm density and a $1:1$ agent-block ratio mainly result in clustering on both grid sizes (Fig. 9.18) but the number is 25 pp higher on the larger grid. We conclude that the task is

Table 9.5: Evaluation metrics (see Sec. 9.2) of 20 independent evolutionary runs in the collective construction scenario with predefined predicted block clusters, N agents and B blocks: median best fitness (Eq. 3.1) in the last time step of all runs, similarity S (Eq. 9.3), mean block movement M_B, mean agent movement M_N (Eq. 9.6), and percentages of formed block structures (i.e., lines (LN), pairs (PR), clustering (CL), and random dispersion (RD)) at the start ($t = 0$) and end ($t = T$) of the runs. Median values in parentheses [47].

N	B	$N-B$ ratio	grid size	median fitness F	S	M_B	M_N	t	structures LN	PR	CL	RD
10	32	5 : 16	16 × 16	0.66	0.57	0.05	0.24	0	0.0	20.0	0.0	80.0
					(0.55)	(0.06)	(0.29)	T	2.5	82.5	0.0	15.0
16	32	1 : 2	16 × 16	0.67	0.38	0.06	0.23	0	0.0	10.0	0.0	90.0
					(0.34)	(0.05)	(0.19)	T	5.0	80.0	5.0	10.0
32	32	1 : 1	16 × 16	0.64	0.20	0.04	0.18	0	0.0	5.0	0.0	95.0
					(0.22)	(0.03)	(0.20)	T	0.0	40.0	55.0	5.0
20	50	2 : 5	20 × 20	0.68	0.45	0.03	0.11	0	0.0	5.0	0.0	95.0
					(0.48)	(0.02)	(0.10)	T	0.0	90.0	0.0	10.0
25	50	1 : 2	20 × 20	0.69	0.32	0.02	0.14	0	0.0	0.0	0.0	100.0
					(0.35)	(0.02)	(0.10)	T	0.0	55.0	45.0	0.0
50	50	1 : 1	20 × 20	0.64	0.24	0.01	0.15	0	0.0	15.0	0.0	85.0
					(0.24)	(0.0)	(0.14)	T	0.0	20.0	80.0	0.0



(a) initial positions **(b)** final positions

Figure 9.18: Resulting block cluster with predefined sensor predictions using 32 agents and 32 blocks on a 16 × 16 grid. Agents are represented by circles, their color and the lines give their headings. Blocks are represented by squares [47].

harder for smaller grids and lower agent-block ratios. Since there is still agent and block movement in the last τ time steps[5] in all six experimental setups, an increased runtime may lead to more clusters.

5: $\tau = 128$ for the 16×16 grid and $\tau = 200$ for the 20×20 grid

In the third experiment, we predefine that agents predict no blocks in front of them (i.e., $\tilde{s}_6 = \tilde{s}_7 = \tilde{s}_8 = \tilde{s}_9 = \tilde{s}_{10} = \tilde{s}_{11} = 0$, Fig. 9.19). Consequently, no specific block structure is predefined in this case as long as agents do not detect blocks. Our minimize surprise fitness function (Eq. 3.1) adapts to

$$F = \frac{1}{TNR} \sum_{t=0}^{T-1} \sum_{n=0}^{N-1} \overbrace{\sum_{r=0}^{R-1} (1 - |\underbrace{\boxed{0}}_{\tilde{s}_r^n(t) = 0} - s_r^n(t)|)}^{\text{predefined predictions: unoccupied grid cells}}, \qquad (9.20)$$

with evaluation length T in time steps, swarm size N, number of sensors per swarm member R, and prediction $\tilde{s}_r^n(t)$ for and value $s_r^n(t)$ of sensor r of swarm member n at time step t.

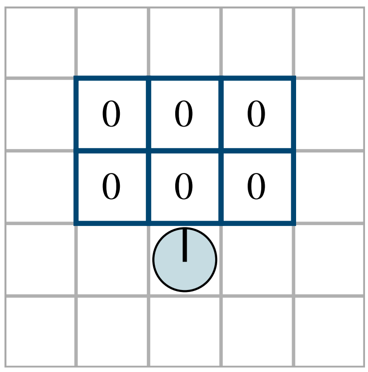


Figure 9.19: Block sensor predictions for the collective construction scenario with predefined empty predictions. The circle represents the agent and the line indicates its heading.

Table 9.6: Evaluation metrics (see Sec. 9.2) of 20 independent evolutionary runs in the collective construction scenario with predefined zero predictions, N agents and B blocks: median best fitness F (Eq. 3.1) in the last time step of all runs, similarity S (Eq. 9.3), mean block movement M_B, mean agent movement M_N (Eq. 9.6), and percentages of formed block structures (i.e., lines (LN), pairs (PR), clustering (CL), and random dispersion (RD)) at the start ($t = 0$) and end ($t = T$) of the runs. Median values in parentheses [47].

N	B	$N-B$ ratio	grid size	median fitness F	S	M_B	M_N	t	structures LN	PR	CL	RD
10	32	5 : 16	16 × 16	0.93	0.77 (0.81)	0.0 (0.0)	0.46 (0.43)	0	0.0	10.0	0.0	90.0
								T	5.0	15.0	5.0	75.0
16	32	1 : 2	16 × 16	0.93	0.58 (0.59)	0.0 (0.0)	0.52 (0.48)	0	0.0	0.0	0.0	100.0
								T	0.0	35.0	15.0	50.0
32	32	1 : 1	16 × 16	0.91	0.30 (0.30)	0.0 (0.0)	0.42 (0.40)	0	0.0	5.0	0.0	95.0
								T	7.5	40.0	42.5	10.0
20	50	2 : 5	20 × 20	0.93	0.754 (0.82)	0.0 (0.0)	0.45 (0.45)	0	0.0	10.0	0.0	90.0
								T	2.5	2.5	5.0	90.0
25	50	1 : 2	20 × 20	0.93	0.57 (0.63)	0.0 (0.0)	0.52 (0.46)	0	0.0	10.0	0.0	90.0
								T	0.0	45.0	15.0	40.0
50	50	1 : 1	20 × 20	0.907	0.351 (0.360)	0.0 (0.0)	0.37 (0.36)	0	0.0	0.0	0.0	100.0
								T	5.0	45.0	35.0	15.0

We observe that blocks are moved around in almost all runs (i.e., similarity $S < 1.0$) except for one run on the 20×20 grid and for two runs on the 16×16 grid with the lowest swarm density. The median best fitness is above 0.9 for all runs (see Tab. 9.6) and between 0.02 and 0.06 higher than for the runs without predefined predictions.

We find various structures in this experiment. Agents try to disperse themselves to neither perceive agents nor blocks. Grouping blocks in clusters, pairs or lines may be beneficial but especially for lower swarm densities the initial structures may already allow all agents to disperse and find an isolated position. Furthermore, we find the highest agent movement in the last τ time steps compared to all previous experiments of the collective construction scenario, but there is no block movement in these last time steps. Thus, agents form block structures in the beginning of the runs as in the experiments without predefined predictions while still moving but avoiding those blocks in the end of the runs.

In summary, we can engineer self-organized construction by predefining sensor value predictions. Some structures, such as clustering or pairs, can easily be engineered while predefining not to sense any blocks provides rather an additional intrinsic driver to group blocks. This is due to the fact that block structures form independently from the agents and their sensor model in contrast to the self-assembly scenario (see Ch. 6). Lines and pairs, for example, can only be differentiated by agents that are positioned perpendicular to the structure as they can only sense more than two adjacent blocks in this alignment to the structure. In a similar way, an agent can sense up to three blocks in randomly dispersed block structures.

9.5 Discussion and Conclusion

Our study on self-organized collective construction with minimize surprise shows that our approach can also be used to evolve swarm behaviors that require agent-environment interactions. Furthermore, we are able to engineer several collective construction behaviors by predefining sensor predictions. Nevertheless, the emergence of behaviors that form desired structures remains challenging. Preliminary investigations with seeds of block formations placed initially in the environment showed that they can effectively trigger the grouping of blocks at designated spots. However, seeds were not as effective as predefining predictions. In future work, we want to investigate the combination of predefined predictions and seeds. This may allow us to evolve collective construction behaviors with minimize surprise that build desired structures at designated positions as realized in other approaches, for example, by using blueprints of the target structures that can be used for the calculation of local rules for the robots [195] or in form of a scalar field guiding the agents [201]. Based on this initial study of collective construction with minimize surprise in a simple simulation environment, we are confident that it can be realized in real world settings, too. In Ch. 11, we make the first step towards evolving collective construction behaviors on real robots by aiming for object manipulation in a site clearance scenario in realistic simulations and with real Thymio II robots.

Basic Swarm Behaviors in Real-World Settings

10

Chapter Contents

In this chapter, we study the evolution of basic swarm behaviors using our minimize surprise approach in realistic simulations and with real robots. We...

- *Sec. 10.1:* present the general setup of our approach for real-world settings,
- *Sec. 10.2:* introduce the experimental setup for evolving basic swarm behaviors,
- *Sec. 10.3:* discuss our results in realistic simulations and
- *Sec. 10.4:* with real robots, and
- *Sec. 10.5:* draw a conclusion.

Parts of this chapter are based on [42, 48].

In all previous scenarios, we used custom robot models in simple simulations (Chs. 4, 9) and more complex, game engine-based simulators (Chs. 7, 8). While those studies show that our minimize surprise approach can be used to evolve a variety of collective behaviors, the evolutionary algorithm used so far is not suitable for the use with real robots. It requires thousands of evaluations per evolutionary run, which would wear down our hardware. In this chapter, we make the step to evolve swarm robot controllers for real-world applications with minimize surprise (see Fig. 10.1). First, we address research question Q6 (see Sec. 1.2) by presenting the general adaptation of our minimize surprise approach for real-world settings. Afterwards, we evolve basic swarm behaviors with minimize surprise in realistic simulations and with real robots, which is a transfer of earlier works in simple simulation environments by Hamann [28] and Borkowski and Hamann [37] (see Sec. 3.3) to the real world. This also contributes to our study on evolving collective behaviors for scenarios with different environmental complexities and agent capabilities with minimize surprise (research question Q5, Sec. 1.2).

(a) realistic simulator Webots

(b) real robot experiments

Figure 10.1: Illustration of the scenario aiming for basic swarm behaviors in real-world settings.

10.1 Adaptation of Minimize Surprise for Real-World Settings

The evolution of robot controllers for the application in the real world is challenging, since there is a tradeoff between optimization speed and avoiding the reality gap (see Sec. 5).

parameter	simulation	real robots
mutation rate p_{mut}	0.1	0.1
max. evaluations e_{max}	1,000	400
time step length (sensor update rate)	10 ms	100 ms
evaluation length T (time steps)	1,000	100
post-evaluation length T_P (time steps)	10,000	1,000
re-evaluation probability p_{reeval}	0.2	0.2
re-evaluation weight α	0.2	0.2

Table 10.1: Parameters for the evolutionary algorithm with (1+1)-selection used in realistic simulations and real robot experiments.

We adapt our minimize surprise approach for real-world settings by relying on a centralized online and onboard evolutionary approach, which allows us to avoid the reality gap. Consequently, we lose the advantages of optimization in simulation, namely speeding up the search process and avoiding wear and tear of the robot hardware. Furthermore, we have to prevent costly damage to the real hardware due to potential harmful robot behavior [371]. We address these problems in a twofold way [48]: (i) by restricting the number of evaluations [13] and (ii) by integrating a hardware protection layer [372]. The former is addressed by our choice of the evolutionary algorithm and the evolution architecture that we present in this section. The hardware protection layer is adjusted to our two real-world settings using Thymio II robots [77] (see Sec. 2.1.2) in the Webots[1] [46] simulator and in real robot experiments and is presented with the respective experimental setups (see Secs. 10.2 and 11.1).

1: Webots R2020a is licensed under the Apache License, Version 2.0 (`https://www.apache.org/licenses/LICENSE-2.0`). The Thymio II Webots model is licensed under the open source Webots Assets license agreement (`https://cyberbotics.com/doc/guide/webots-license-agreement`).

In our previous experiments in simulation (Chs. 4 - 9), we used an evolutionary algorithm with proportionate parent selection and age-based survivor selection conducting several thousand evaluations per evolutionary run. In online evolution, this would be too time consuming and, as already mentioned, would wear down our hardware [13]. We therefore use an evolutionary algorithm with $(1+1)$-selection[2] for our experiments in real-world settings [16, 373]. In $(1+1)$-selection, the current best individual forms the parent population. We randomly initialize the initial parent population by drawing weights from a uniform distribution in $[-1, 1]$. With a 20 % chance, this current best individual is re-evaluated to test its suitability for a possibly changed environment. The updated fitness of the best individual F_e is calculated as an exponentially weighted mean

2: $(1+1)$-selection is a form of $(\mu+\lambda)$-selection with number of parents $\mu = 1$ and number of offspring $\lambda = 1$ (see Sec. 2.3.2) [16].

$$F_e = \alpha \hat{f}_e + (1-\alpha) F_{e-1}\,, \tag{10.1}$$

with weighting $\alpha = 0.2$, fitness value $\hat{f}_e$ reached during re-evaluation, and previous best fitness F_{e-1}. Otherwise, offspring is created by adding a uniformly random value in the range $[-0.4, 0.4]$ with a 10 % probability (i.e., $p_{\text{mut}} = 0.1$) to each value of the genome and evaluated. In survivor selection, the best of these two individuals (i.e., current best individual or offspring) is selected to form the next parent

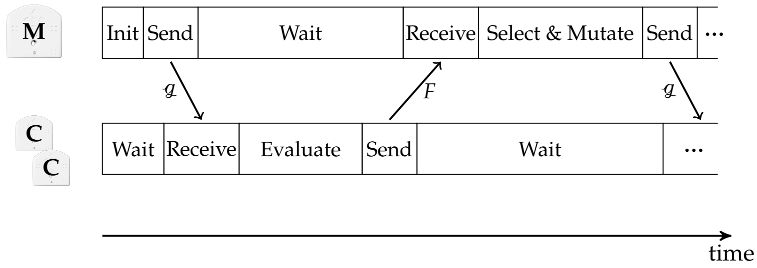


Figure 10.2: Schematic interplay of the robots in our centralized online evolution architecture. The external master robot (M) runs the evolutionary process: it initializes the first genome (g) and sends it to the clients (C). Clients evaluate the received genome and send their individual fitnesses (F) back. The master determines the overall fitness, selects the current best individual, decides if it will be re-evaluated or creates offspring by mutation, and sends the respective genome to the clients. The process continues until terminated, that is, after $e_{\max}$ evaluations here.

population. This approach has already proven to be suitable for online evolution, for example, by Heinerman et al. [278] in a collective foraging scenario with real Thymio II robots. Online evolution allows for infinitely continued adaptation, but we stop evolution after 1,000 evaluations in simulation and after 400 evaluations on the real robots due to limitations in battery life. We do a forward pass through the actor-predictor ANN pair used in our minimize surprise approach (see Fig. 3.1) with every sensor update, which is every 100 ms on the real Thymio II robots and every 10 ms (simulated time) in the Webots simulator. Thus, time is discretized into steps of 100 ms and 10 ms, respectively, and we can calculate fitness (Eq. 3.1) based on time steps as before. Individuals are evaluated for 10 s each, which is 1,000 time steps in simulation and 100 time steps on the real robots. Robots start an evaluation at the last position of the previous evaluation. To evaluate the variety of behaviors that emerges due to our task-independent fitness function (Eq. 3.1), we re-run the best evolved individual at the end of the evolutionary run (i.e., the parent individual after $e_{\max}$ evaluations). We post-evaluate the best evolved individual for 100 s to store sensor values, predictions, and, in simulation, the robot trajectories. The robot trajectories give each robot n's pose $\mathcal{H}_n(t) = (P_n(t), \theta_n(t))$ in the global coordinate frame of the Webots simulator with position $P_n(t) = (x_n(t), y_n(t))$ (i.e., the coordinates of the robot's rotation center) and heading $\theta_n(t)$ per time step t over the full duration of the post-evaluation. Table 10.1 summarizes the parameters of the evolutionary algorithm.

Fig. 10.3 shows the centralized online evolution architecture [16] that we use in our experiments in real-world settings in the Webots simulator and with real Thymio II mobile robots. For the implementation of this evolution architecture, we extend the real Thymio II robots with Raspberry Pis (RPi) (see Sec. 2.1.2) for inter-robot communication capabilities via Wi-Fi, increased processing power, and the programmability in Python 3. In the following, we always refer to this extended version with Thymio II. All of the functionalities provided by the RPi are available by default in the Webots simulator. In our online evolution architecture, one robot serves as a central master. This master robot guides the evolutionary

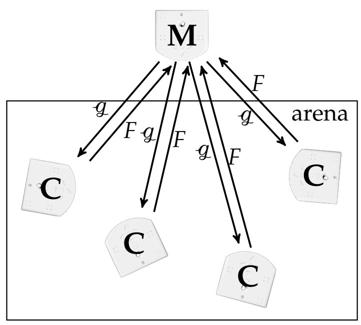
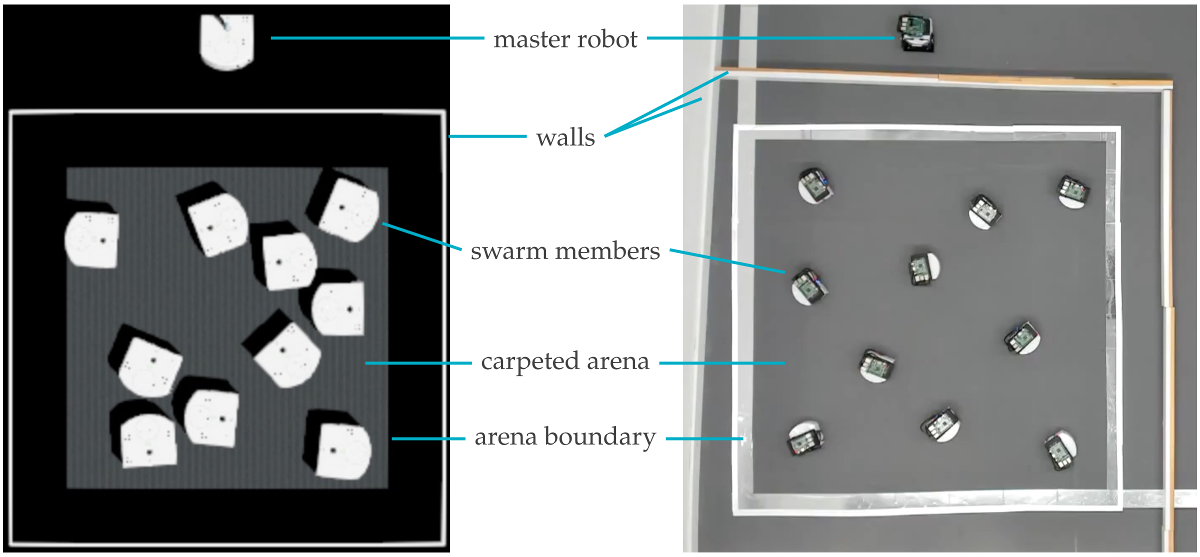


Figure 10.3: Centralized online evolution architecture for our experiments in real-world settings. One robot serves as the master (M) and coordinates the evolutionary process. This master robot is placed outside the experimental arena. The swarm members, or clients (C), receive a genome (g) from the master robot, evaluate it and send their individual fitnesses (F) back.



(a) arena in the Webots simulator with side length $L = 0.7$ m

(b) real arena with side length $L = 1.5$ m

Figure 10.4: Experimental arenas used in the scenario aiming for basic swarm behaviors in real-world settings. We use square, carpeted arenas with a swarm of ten robots placed in the arena evaluating genomes. The master robot handling the evolutionary process is placed outside the arena. The arena boundaries are the edges of the carpeted floor in simulation and mirror film in the real arena. The low walls in distance to the arenas' boundaries prevent robots from leaving the arena completely.

process, that is, it distributes genomes, collects individual fitnesses, calculates the overall fitness, selects the best individual, and creates offspring all onboard the Raspberry Pi when using the real robots and otherwise in the separate process of this simulated robot in Webots. We place the master outside the arena (i.e., it is not evaluating genomes itself) to prevent experiment abortions due to hardware defects of the master in the real robot experiments. Still, evolution is running fully onboard on the real robots. The swarm members (clients) receive the genomes from the master, evaluate them, and send their individual fitnesses back. Fig. 10.2 illustrates this interplay schematically. The communication between master and clients is realized in the simulation with Webots' built-in Emitter and Receiver nodes.[3] On the real robots, we use Wi-Fi and a TCP[4] connection. In the case of transmission errors, we evaluate genomes again in the next evaluation and do not count the failed evaluation towards the total number of evaluations.

3: The Emitter and Receiver nodes in Webots can be used to model radio, serial or infrared communication. We use simulated radio communication here, since this resembles the communication via Wi-Fi on the real robots.

4: Transmission Control Protocol

10.2 Experimental Setup

In our first real-world scenario, we aim for the evolution of basic swarm behaviors.[5] We use a swarm of $N = 10$ Thymio II [77] robots as clients and one Thymio II robot as the master (see Sec. 10.1). The Thymio II has a differential drive with a maximum velocity of 20 $\frac{\text{cm}}{\text{s}}$, but we restrict maximum speed v_{max} to 12.6 $\frac{\text{cm}}{\text{s}}$ to reduce wear of the motors. We use the Thymio II's seven horizontal infrared (IR) proximity

5: `https://gitlab.iti.uni-luebeck.de/minimize-surprise/basic-swarm-behaviors-thymio` sensors, whereof five are located at the front ($s_0, \ldots, s_4$) and two are located at the back (s_5 and s_6), and its two IR ground sensors (s_7 and s_8), see Fig. 10.5. Thus, we have $R = 9$ sensor values. The proximity sensors have a reach of about 10 cm and are updated every 100 ms on the real robot and every 10 ms (simulated time) in Webots. We normalize all sensors by their maximum value (i.e., $s_r \in [0, 1]$).

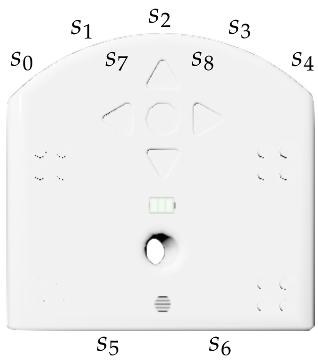


Figure 10.5: Thymio II robot in the Webots simulator and the positions of the robot's five horizontal IR sensors at the front ($s_0, \ldots, s_4$), two horizontal IR sensors at the back (s_5, s_6), and two ground IR sensors (s_7, s_8).

Our square arenas have a side lengths of L. We vary swarm density[6] by placing our robot swarm into arenas with different side lengths as in our self-assembly scenario (see Ch. 4). In the Webots simulator, we use side lengths $L \in \{0.7\text{ m}, 0.8\text{ m}, 0.9\text{ m}, 1.0\text{ m}, 1.5\text{ m}\}$ while we restrict ourselves to side lengths $L \in \{0.8\text{ m}, 1.0\text{ m}, 1.5\text{ m}\}$ in our real robot experiments. This results in swarm densities D_N between 0.06 and 0.25, see Tab. 10.2. As shown in Fig. 10.4, the arenas have carpeted floor and robots can detect the arena boundaries, that is, the edges of the carpeted floor in simulation and mirror film in the real arena, with their ground IR sensors. Since the ground IR sensors are located in the front of the robots, backward driving robots detect the arena boundaries only when they are already almost completely out of the arena. Therefore, we add low walls in distance to the real arena boundaries that prevent robots to leave the arena completely. These walls are not detectable by a robot's horizontal IR sensors while the robot is completely inside the arena. Overall, robots can differentiate between other robots and the arena boundaries due to the different used sensors. Other robots are detected by the horizontal IR sensors ($s_0, \ldots, s_6$) while the arena boundaries are detected by the ground IR sensors (s_7, s_8).

6: $D_N = \frac{N \times \mathscr{L}_n \times \mathscr{W}_n}{L \times L}$ with robot length $\mathscr{L}_n$, robot width $\mathscr{W}_n$, and arena side length L.

We prevent damages to robots by adding a simple hardware protection layer (see Sec. 10.1). A robot is stopped if:

1. The robot detects other robots with its front IR sensors ($s_0, \ldots, s_4$) and currently aims to drive rather straight forward.
2. The robot detects other robots with its back IR sensors (s_5, s_6) and currently aims to drive rather straight backward.
3. The robot detects the arena boundaries with its ground IR sensors (s_7, s_8) and currently aims to drive rather straight forward or backward.

Table 10.2: Swarm densities D_N per arena side length L for the scenario aiming for basic swarm behaviors in real-world settings. All specified side lengths are used in simulation; side lengths used in simulation and in real robot experiments are marked with a '*'.

L	D_N
0.7 m	0.25
0.8 m *	0.19
0.9 m	0.15
1.0 m *	0.12
1.5 m *	0.06

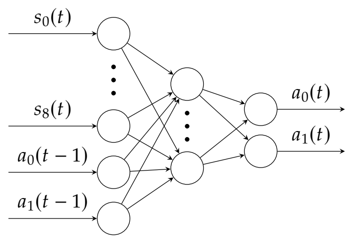
Driving rather straight means that the robot attempts to drive in a circle with a radius larger than 5 cm. Consequently, robots are still allowed to turn on the spot in these cases and can thus avoid deadlocks. Robots could also exploit being stopped in these cases to position themselves in structures and create easily predictable environments. Overall, we keep hardware protection simple and thus do not guarantee that all crashes between robots are prevented. But this simple hardware protection gives robots the freedom to evolve a

Table 10.3: Parameters of the experimental setup for the scenario aiming for basic swarm behaviors in real-world settings. All specified side lengths are used in simulation; side lengths used in simulation and in real robot experiments are marked with a '*'.

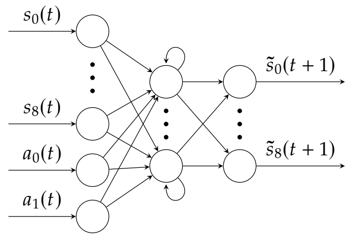
parameter	value
arena side length L	$\{0.7\text{ m}, 0.8\text{ m}^*, 0.9\text{ m}, 1.0\text{ m}^*, 1.5\text{ m}^*\}$
swarm size N	10
# of sensors and predictor outputs R	9
sensor values s_r	$[0, 1]$
action values a_0, a_1	$[-1, 1]$
max. speed $v_{\max}$	$12.6\ \frac{\text{cm}}{\text{s}}$

variety of behaviors, including their own obstacle avoidance strategy.

Each robot is equipped with an actor-predictor ANN pair as defined in our minimize surprise approach (see Sec. 3.2) and rewarded for high prediction accuracy (Eq. 3.1). The actor (see Fig. 10.6a) is a three-layer feedforward ANN with eleven input neurons, seven hidden neurons, and two output neurons. We use the hyperbolic tangent as the transfer function. The network outputs two normalized speeds $a_0(t), a_1(t) \in [-1.0, 1.0]$ for the Thymio II's two differential drive motors with every sensor update. These normalized speeds $a_0(t)$ and $a_1(t)$ are scaled with the maximum speed $v_{\max}$ when sent to the robot's motors. The previously described hardware protection layer may overwrite the actor outputs to prevent robot damage; fitness evaluation is not affected by these interventions and continues as normal. The actor receives the current $R = 9$ normalized sensor values and the last set of normalized speeds $a_0(t-1)$ and $a_1(t-1)$ as input. The predictor (see Fig. 10.6b) is a three-layer ANN with one recurrent hidden layer. The network has eleven input neurons, ten hidden neurons, and nine output neurons. We use the logistic sigmoid function as the transfer function. The predictor outputs normalized prediction values $\tilde{s}_0(t+1), \dots, \tilde{s}_8(t+1)$ ($\tilde{s}_r \in [0, 1]$) for the nine used sensors. We input the normalized current sensor values and the next normalized speeds $a_0(t)$ and $a_1(t)$.



(a) actor



(b) predictor

Figure 10.6: Actor and predictor networks in the scenario aiming for basic swarm behaviors in real-world settings. $a_0(t-1)$ and $a_1(t-1)$ are a robot's last normalized speeds per wheel of the Thymio's differential drive, and $a_0(t)$ and $a_1(t)$ are its next speeds. $s_0(t), \dots, s_8(t)$ are the robot's $R = 9$ sensor values at time step t, $\tilde{s}_0(t+1), \dots, \tilde{s}_8(t+1)$ are its sensor predictions for time step $t+1$.

We do 20 independent evolutionary runs per arena size in simulation and eight independent evolutionary runs per arena size in the real robot experiments. To study the effectiveness of our approach, we additionally generate 20 random ANN pairs per arena size in simulation drawing weights from a uniform distribution in $[-1, 1]$. We randomly place the robots in the arena at the beginning of an evolutionary run. As mentioned in the previous section, robots start an evaluation at the last position of the previous evaluation afterwards. For post-evaluation of the best evolved individual at the end of the evolutionary run, we reset the arena in simulation by placing the robots at random positions. In our real robot experiments, the robots start post-evaluation at their last position after $e_{\max}$ evaluations. Tab. 10.1 gives the hyperparameters used for the evolutionary algorithm and

Tab. 10.3 summarizes the parameters for the experimental setup.

10.3 Experiments in Realistic Simulations

First, we evolve basic swarm behaviors in the Webots simulator with our minimize surprise approach. Running evolution in simulation is fast and enables easier tracking of experimental data, facilitating the analysis of the best evolved individuals. A video of the resulting behaviors is online.[7]

7: https://youtu.be/Hai7fzyb_RA

10.3.1 Metrics

We measure best fitness (Eq. 3.1) over evaluations to determine the success of evolution as in previous scenarios. In addition, we analyze and classify the behaviors of the randomly generated individuals and the best evolved individuals based on two metrics: (i) the robots' mean change in heading σ and (ii) the mean cluster size. We calculate both metrics over the last $\tau = 5{,}000$ time steps of the post-evaluation run giving robots time in the first half of the run to cover some distance and initiate their dominant behavior. We do not consider the distance covered by robots as it is low for all runs.[8]

8: The majority of runs leads to mean covered distances of less than 1.0 m, which is low compared to the theoretically possible maximum of 6.3 m. The mean covered distance is maximally 2.02 m for the randomly generated individuals and 2.87 m for the best evolved individuals.

We define the robots' mean change in heading σ as the absolute accumulated angular displacement over the last τ time steps of the post-evaluation run as given by

$$\sigma = \frac{1}{N} \sum_{n=0}^{N-1} \left| \sum_{t=T_P-\tau}^{T_P-1} (\theta_n(t+1) - \theta_n(t)) \right| , \qquad (10.2)$$

with N robots, post-evaluation length of T_P time steps, and headings $\theta_n(t)$ and $\theta_n(t+1)$ of robot n at time steps t and $t+1$, respectively.[9] We use the absolute of the change in heading per robot, since the turning direction is irrelevant for behavior classification. Robots can reach a maximum change in heading σ of 126 rad when constantly turning in the same direction with maximum angular velocity during the last 5,000 post-evaluation time steps. Using an empirical approach, we differentiate between stopped ($\sigma < 5$), diverse ($5 < \sigma < 55$), and spinning ($\sigma > 55$) behaviors. Robots turn on the spot in spinning behaviors. Diverse behaviors can range from robots turning slowly or in large circles to swarms that are partially stopped and partially circling.

9: The robot's maximum rate of rotation about the ICC is $w_{\max} = \frac{v_r - v_l}{\mathcal{W}_n} = \frac{12.6\text{ cm/s} + 12.6\text{ cm/s}}{10\text{ cm}} = 2.52\,\frac{\text{rad}}{\text{s}}$ with right and left wheel speeds along the ground v_r and v_l, and distance between wheels $\mathcal{W}_n$. We assume that the robot always turns in the direction that leads to the smallest absolute difference between headings $\theta_n(t+1)$ and $\theta_n(t)$. That means, if $\theta_n(t+1) - \theta_n(t)$ is less than zero, we take the maximum of $\theta_n(t+1) - \theta_n(t)$ and $-2\pi - (\theta_n(t+1) - \theta_n(t))$. Otherwise, we take the minimum of $\theta_n(t+1) - \theta_n(t)$ and $2\pi - (\theta_n(t+1) - \theta_n(t))$.

Our second metric is the mean cluster size in the last 5,000 post-evaluation time steps. We define that clusters are formed out of robots that have at maximum twice the robot radius[10] distance to a minimum of one other cluster member. We

10: The radius of the Thymio II is approximately 8.1 cm.

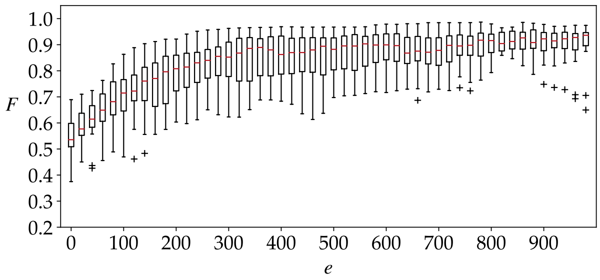


Figure 10.7: Best fitness F of 20 independent evolutionary runs over evaluations e for the 1.0 m × 1.0 m arena in simulation aiming for basic swarm behaviors in real-world settings. Only every 20th evaluation is printed for clearer illustration. Medians are indicated by red bars [343].

differentiate between behaviors leading to grouped and dispersed robots based on a threshold for mean cluster size of 1.43, which corresponds to five grouped robots.[11]

11: That is, a group of two robots, a group of three robots, and five dispersed robots.

10.3.2 Results

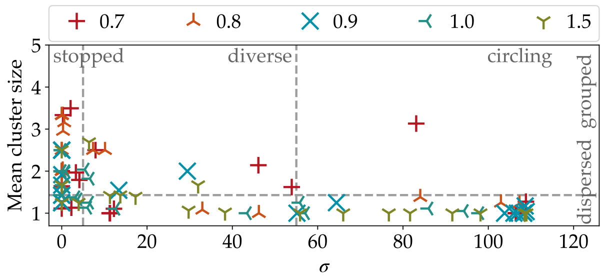
Fig. 10.7 visualizes the increase of best fitness of 20 independent evolutionary runs over evaluations in the 1.0 m × 1.0 m arena as representative example for all obtained fitness curves. We reach a median best fitness of at least 0.88 for the best evolved individuals and of maximum 0.58 for the randomly generated ANN pairs for all arena sizes, see Tab. 10.4. Consequently, our approach outperforms pure random search in fitness, that is, best evolved individuals have better prediction accuracy than randomly generated individuals.

Table 10.4: Median best fitness F (Eq. 3.1) for the best evolved individuals (`evo`) and for the randomly generated ANN pairs (`random`) per arena size $L \times L$ for the scenario aiming for basic swarm behaviors in realistic simulations.

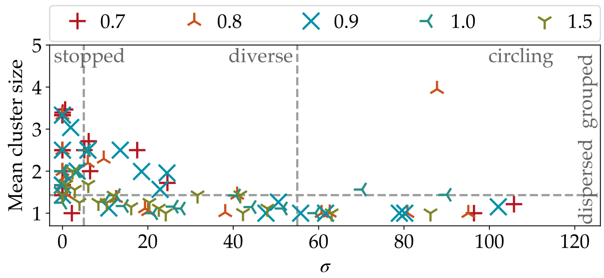
L	F	
	`evo`	`random`
0.7 m	0.92	0.58
0.8 m	0.88	0.58
0.9 m	0.92	0.53
1.0 m	0.93	0.58
1.5 m	0.95	0.57

Next, we analyze the behavior diversity by post-evaluating the best evolved individuals and the randomly generated individuals. Over all arena sizes, we find more behaviors that are categorized as diverse for randomly generated ANN pairs than for best evolved individuals, see Fig. 10.8. The majority of diverse behaviors leads to partially stopped and partially circling swarms. For those behaviors, we assume that predicting sensor inputs is hard as sensor values are time-variant and vary for the individual swarm members. Otherwise, most randomly generated ANN pairs lead to behaviors with stopped robots that are either grouped or dispersed and a few randomly generated individuals lead to spinning dispersed robots. By contrast, evolution mostly leads to stopped robots that are grouped or dispersed, or spinning robots that are dispersed. All of these behaviors are easy to predict as they have constant sensor values.

Grouped robots are stopped by hardware protection in most cases guaranteeing constant sensor input. Rare cases lead to robots staying on their current position by quickly switching turning direction or robots constantly turning on the spot while still remaining close enough to be considered as grouped. Grouped robots are positioned close enough to detect each other with their horizontal proximity sensors $(s_0, \ldots, s_6)$. Fig. 10.9a visualizes one run leading to stopped grouped robots and their mean sensor values and predictions over the post-evaluation run. In this run, robots detect

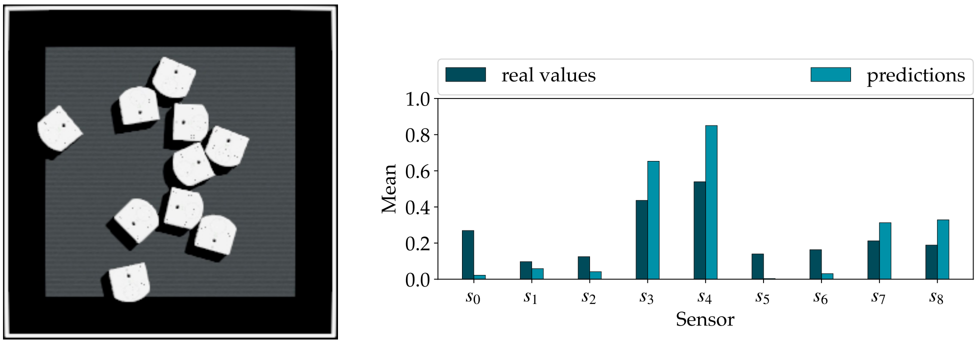


(a) best evolved individuals

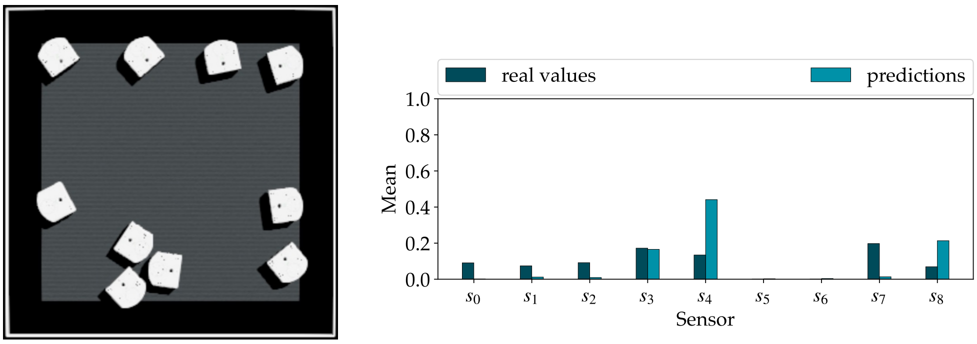


(b) randomly generated ANN pairs

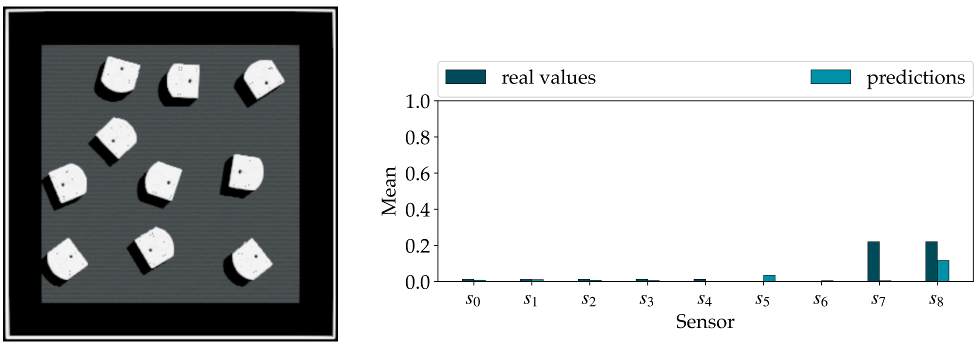
Figure 10.8: Mean change in heading σ (Eq. 10.2) and mean cluster size over the last 5,000 time steps for the 20 best evolved individuals and 20 randomly generated individuals per arena size $L \times L, L \in \{0.7\text{ m}, 0.8\text{ m}, 0.9\text{ m}, 1.0\text{ m}, 1.5\text{ m}\}$ in our scenario aiming for basic swarm behaviors in realistic simulations. Gray lines categorize the resulting behaviors [343].



(a) stopped grouped robots ($L = 0.8$ m)



(b) stopped dispersed robots ($L = 0.9$ m)



(c) spinning dispersed robots ($L = 0.9$ m)

Figure 10.9: Robot positions at the end of the post-evaluation run and the swarm's mean sensor values and predictions over the post-evaluation run of three best evolved individuals leading to stopped robots that are dispersed or grouped, or to spinning dispersed robots in our scenario aiming for basic swarm behaviors in realistic simulations. $s_0, \dots, s_4$ give the frontal horizontal proximity sensors, s_5 and s_6 the back horizontal proximity sensors, and s_7 and s_8 the ground IR sensors.

neighbors mainly with their front right proximity sensors (s_3, s_4) and in few cases also with their outer left and back proximity sensors (s_1, s_5, s_6). The ground IR sensors (s_7, s_8) lead to real and predicted values matching the reflected light from the arena's carpet. Mean real sensor values and predictions match closely in general. But slightly larger deviations between actual and predicted values for a few sensors do not affect the total fitness drastically, since each sensor contributes only one ninth to the total fitness.

Stopped dispersed robots (see Fig. 10.9b) and spinning dispersed robots (see Fig. 10.9c) frequently distribute over the arena by employing an emergent obstacle avoidance behavior. In the case of stopped dispersed swarms, the robots drive to different parts of the arena boundary where they are stopped by hardware protection. Dispersed robots have constant low horizontal proximity sensor values and predictions ($s_0, \ldots, s_6 \approx 0$), that is, they neither detect nor predict other swarm members. In our example for spinning dispersed robots, see Fig. 10.9c, such low sensor values and predictions for all horizontal proximity sensors are found. By contrast, we find slightly higher proximity sensor values for the stopped dispersed swarm, see Fig. 10.9b, which are caused by the three grouped robots. As before, we find ground IR sensor values that match the reflected light from the arena's carpet in both dispersion variants ($s_7, s_8 \approx 0.2$). The swarm stopped at the arena's boundary (Fig. 10.9b) has lower right ground IR sensor values ($s_8 \approx 0.1$) since this sensor detects the edge of the carpeted floor, which leads to IR sensor values of zero. The predictions for the ground sensors appear to be reversed in this run. This may be caused by robots having detected the arena boundary with their left ground IR sensors during the evolutionary run and having optimized the predictor accordingly. Since we randomly reposition the robots at the beginning of the post-evaluation run, they may detect the arena boundary with the other ground sensor than before. High fitness values can still be reached as the difference between predictions and actual sensor values is small.

10.4 Experiments with Real Thymio II Robots

Next, we evolve basic swarm behaviors in real robot experiments with our minimize surprise approach. Each of the eight independent evolutionary runs per arena size (see Sec. 10.2) took approximately 80 min.

Fig. 10.10 visualizes the increase of best fitness over evaluations in the 1.0 m × 1.0 m arena as representative example for all obtained fitness curves. We reach a median best fitness of at least 0.81 for the best evolved individuals for all arena sizes, see Tab. 10.5.

Table 10.5: Median best fitness F for the best evolved individuals per arena side length L for the scenario aiming for basic swarm behaviors in real robot experiments.

L	F
0.8 m	0.81
1.0 m	0.87
1.5 m	0.94

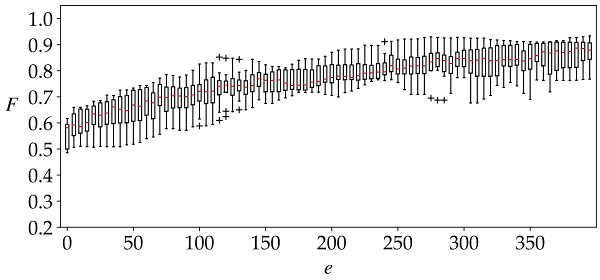


Figure 10.10: Best fitness F of eight independent evolutionary runs over evaluations e for the 1.0 m × 1.0 m arena in real robot experiments aiming for basic swarm behaviors. Only every 5th evaluation is printed for clearer illustration. Medians are indicated by red bars.

Unlike the runs in simulation, we do not have exact data for the robot trajectories in our real robot experiments. However, we can estimate the robots' mean change in heading based on the robots' angular velocities determined by the set speeds of its two differential drive wheels (i.e., actor outputs scaled by $v_{\max}$ or zero if hardware protection is active).[12] This calculation does not take into account influences of the real world, such as friction or production-related differences between the motors, which affect the actual change in heading. But the calculated values are still a good estimate. We can determine the approximate mean cluster size by hand, that is, we count how many robot clusters form using the video footage. This enables us to classify the behaviors of the best evolved individuals based on mean change in heading and mean cluster size in the last half of the post-evaluation run (i.e., $\tau = 500$ time steps here) as before.

12: The rotation between two time steps t and $t+1$ ($\Delta t = 100$ ms) is $\Delta\theta(t, t+1) = w(t)\Delta t$ with $w(t)$ being the angular velocity set at time step t.

As in our runs in simulation, we mainly find stopped robots that are grouped or dispersed, or spinning dispersed swarms, see Fig. 10.11. We thus have comparable results in simulation and real robot experiments. It is noticeable that half of the runs in the 0.8 m × 0.8 m arena are classified as diverse. We assume that the high swarm density in combination with increased noise in the real world make it difficult to evolve behaviors leading to stopped or spinning swarms. We find low sensor values and predictions for the horizontal proximity sensors ($s_0, \ldots, s_6 \approx 0$) in dispersed swarms, see Figs. 10.12b and 10.12c. Our grouped swarm, shown in Fig. 10.12a, has high frontal proximity sensors values ($s_0, \ldots, s_4 > 0.4$) and low back proximity sensors values ($s_5, s_6 < 0.2$). The ground IR sensor values of the grouped swarm are low ($s_7, s_8 \approx 0.25$) matching the reflected light from the arena's carpet. By contrast, we find high ground IR sensor values ($s_7, s_8 \approx 0.62$) for the robots that are stopped at the arena's boundary because the mirror film reflects light well (see Fig. 10.12b). The spinning dispersed robots (see Fig. 10.12c) have intermediate ground IR sensor values ($s_7, s_8 \approx 0.4$), since a few robots are close to the arena's boundary.

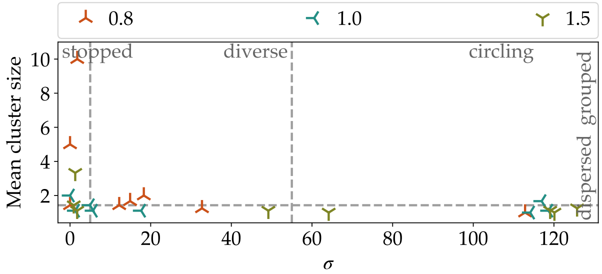
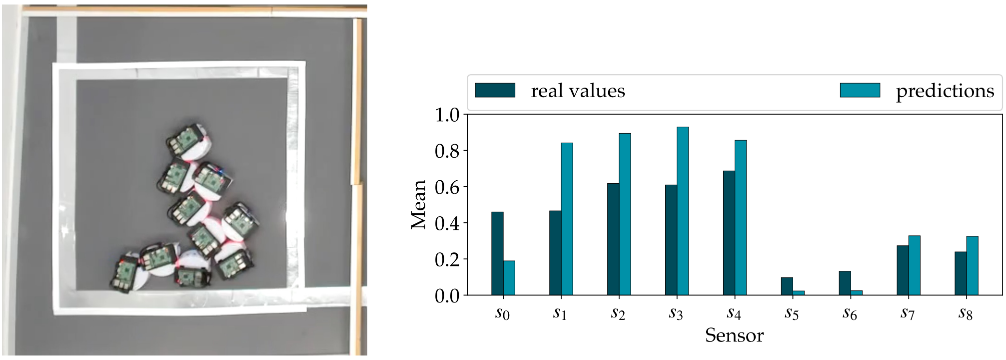
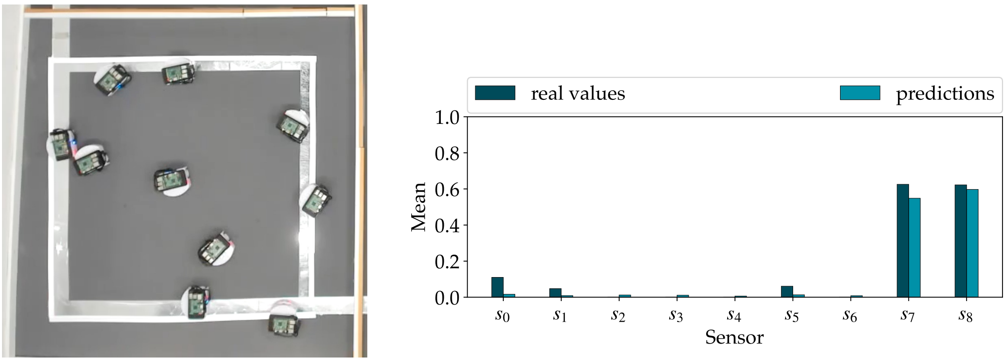


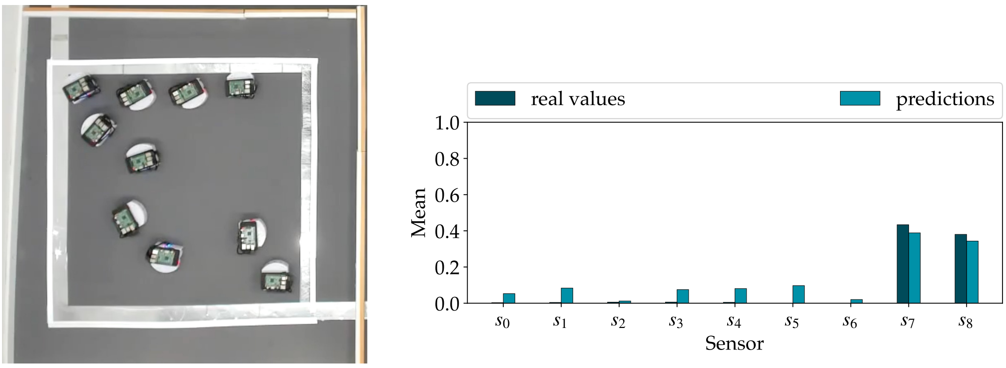
Figure 10.11: Approximated mean change in heading σ and mean cluster size over the last 5,000 time steps for the eight best evolved individuals per arena size $L \times L, L \in \{0.8\,\text{m}, 1.0\,\text{m}, 1.5\,\text{m}\}$ in our real robot experiments aiming for basic swarm behaviors. Gray lines categorize the resulting behaviors.



(a) stopped grouped robots ($L = 0.8$ m)



(b) stopped dispersed robots ($L = 1.0$ m; the two robots in the arena's center are circling)



(c) spinning dispersed robots ($L = 1.0$ m)

Figure 10.12: Robot positions at the end of the post-evaluation run and the swarm's mean sensor values and predictions over the post-evaluation run of three best evolved individuals leading to stopped robots that are dispersed or grouped, or to spinning dispersed robots in real robot experiments aiming for basic swarm behaviors. $s_0, \ldots, s_4$ give the frontal horizontal proximity sensors, s_5 and s_6 the back horizontal proximity sensors, and s_7 and s_8 the ground IR sensors.

10.5 Discussion and Conclusion

Our initial study of evolving swarm behaviors with minimize surprise in realistic simulations and real robot experiments has proven that our approach is suitable for online evolution in real-world settings. In first experiments, we have evolved a variety of simple swarm behaviors, that is, mainly dispersion and grouping behaviors with either stopped or spinning robots. The implemented hardware protection probably has a huge influence on the resulting behaviors as swarm members exploit being stopped at the arena's boundary or when getting too close to each other to stay on their current positions. We want to investigate the effect of the chosen hardware protection on the emergent behaviors in detail in future work. Another influencing factor on the emergent behaviors is the environment. In the scenario presented in this chapter, we used a simple, empty environment in which swarm members can only interact with each other. In the next chapter, we make a first step towards more sophisticated and dynamic environments by distributing blocks in the environment that can be manipulated by the swarm.

Object Manipulation Behaviors in Real-World Settings 11

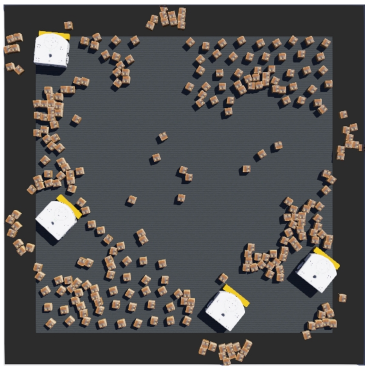
Chapter Contents

In this chapter, we study the evolution of object manipulation behaviors using our minimize surprise approach in realistic simulations and with real robots. We...

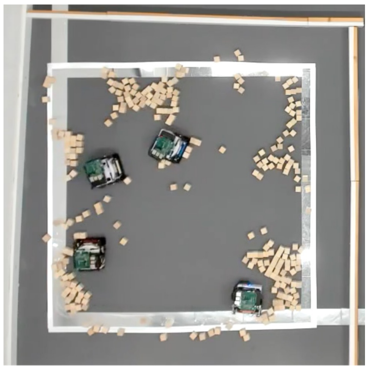
- *Sec. 11.1:* introduce the experimental setup,
- *Sec. 11.2:* discuss our results in realistic simulations and
- *Sec. 11.3:* with real robots, and
- *Sec. 11.4:* draw a conclusion.

Parts of this chapter are based on [42, 48].

Previously, we have studied, among other things, the evolution of collective construction behaviors in simulated 2D torus grid environments (see Ch. 9) as well as basic swarm behaviors in real-world settings (see Ch. 10). In this chapter, we combine both previous approaches aiming for object manipulation behaviors in real-world settings (research question Q5, Sec. 1.2). We distribute manipulable objects (blocks; see Fig. 11.1) in the environment as in our previous study of collective construction in simple simulations, but here we do experiments in realistic simulators and with real robots using the approach presented in Ch. 10. For this purpose, we extend the Thymio II mobile robot with a bulldozer blade that enables it to push blocks, for example, to form clusters or to clear an area from blocks as in blind bulldozing [193] (see Sec. 2.2.1). The latter is a necessary behavior for the preparation of construction sites and thus object manipulation behaviors that are forms of collective construction [192] can potentially emerge.



(a) realistic simulator Webots



(b) real robot experiments

Figure 11.1: Illustration of the scenario aiming for object manipulation behaviors in real-world settings.

11.1 Experimental Setup

Our setup for the evolution of object manipulation behaviors with minimize surprise in real-world settings is similar to our previous scenario, see Ch. 10.[1] We evolve the robot controllers using the same online and onboard evolutionary approach as before, see Sec. 10.1. The experimental setup is similar to the one presented in Sec. 10.2, but we make several changes to the arena, the robot platform, and the hardware protection layer. Unchanged is the setup of the actor-predictor ANN pairs.[2] We present the modifications in detail and refer the reader to the previous chapter for all other details.

1: `https://gitlab.iti.uni-luebeck.de/minimize-surprise/object-manipulation-thymio`

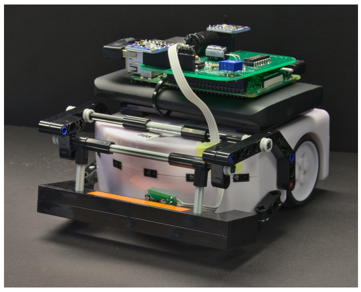
2: However, the number of input, hidden, and output neurons changes, since we extend the robots with additional sensors in this scenario. When using one additional sensor (i.e., total of $R = 10$), the actor has 12 input, 7 hidden and 2 output neurons, and the predictor has 12 input, 11 hidden and 10 output neurons. When using three additional sensors (i.e., total of $R = 12$), the actor has 14 input, 8 hidden and 2 output neurons, and the predictor has 14 input, 13 hidden and 12 output neurons.

parameter	value
arena side length L	1.1 m
swarm size N	4
block densities D_B	$\{0.036, 0.14\}$
# of sensors and predictor outputs R	$\{10, 12\}$
sensor values s_r	$[0, 1]$
action values a_0, a_1	$[-1, 1]$
max. speed $v_{\max}$	12.6 $\frac{\text{cm}}{\text{s}}$

Table 11.1: Parameters of the experimental setup for the scenario aiming for object manipulation behaviors in real-world settings.

In our object manipulation scenario, we use a swarm of $N = 4$ Thymio II robots as clients and one Thymio II robot as the master. As before (see Sec. 10.2), we restrict the robots' maximum speed $v_{\max}$ to 12.6 $\frac{\text{cm}}{\text{s}}$ to reduce wear of the motors, and use its seven horizontal proximity sensors $(s_0, \dots, s_6)$ and its two IR ground sensors (s_7, s_8). For this scenario, we extend the simulated and real robots with a bulldozer blade in a bumper style, two light sensors (s_{10}, s_{11}), and a pressure sensor (s_9). The light sensors are on top of the robot and can detect light gradients. The force sensor measures forces when the Thymio II pushes objects with its bulldozer blade. In Webots, we modify the open-source PROTO-files of the Thymio II Webots model to add sensors and the bulldozer blade, see Fig. 11.2a. For the real robots, we connect the force sensor (HSFPAR303A) and the light sensors (TSL45315; not used in our real robot experiments here) to the RPi (see Sec. 2.1.2). The bulldozer blade is built out of LEGO® parts and mounted to the LEGO® attachment points on the Thymio II, see Fig. 11.2b. Fig. 11.2c indicates the sensor positions on the extended Thymio II robot.

(a) extended Thymio II in Webots



(b) real Thymio II with bulldozer blade

(c) sensor positions

Figure 11.2: Extended Thymio II for the object manipulation scenario (a) in the Webots simulator and (b) in reality, and (c) the position of the robot's seven horizontal proximity sensors $(s_0, \dots, s_6)$, two ground IR sensors (s_7, s_8), two light sensors $(s_{10}, s_{11}$; invisible in simulation) and one force sensor (s_9) [374].

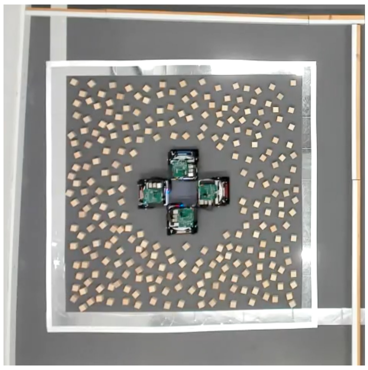
We initially place the robots at the center of arenas of size 1.1 m × 1.1 m, see Fig. 11.3. We use arenas with uniform light conditions (*standard arena*) in simulation and real robot experiments, see Figs. 11.3a and 11.3b. To investigate the influence of light on emerging behaviors, we use a second arena (*gradient arena*) in simulation that has a light bulb above its center resulting in gradually decreasing light intensity towards the arena boundaries, see Fig. 11.3c. The light sensors are irrelevant in the standard arena and are only used in the gradient arena. As before, the arenas have carpeted floor, boundaries that can be detected by the robots with their ground IR sensors, and walls in distance to the real arena boundaries preventing robots to leave the arena completely. Similar as in our collective construction scenario (see Ch. 9), we randomly distribute blocks in the arena. In our real robot experiments, wooden cubes of size 2.5 cm × 2.5 cm × 2.5 cm that weigh about 10 g serve as the blocks. In simulation, we use blocks of the same size but weighing 2 g to compensate for differences between simulation and real world that we determined in preliminary investigations. The blocks are too small to be detected by the horizontal proximity sensors and are only detected by the pressure sensors. This enables robots to discriminate between other robots $(s_0, \dots, s_6)$, blocks (s_9), and the arena boundaries (s_7, s_8). We use two different block densities D_B in our experiments: a low block density of ca. 0.04 (i.e., $\approx 55 \frac{\text{blocks}}{\text{sqm}}$) and a high block density of ca. 0.14 (i.e., $\approx 220 \frac{\text{blocks}}{\text{sqm}}$). Tab. 11.1 summarizes the parameters for the experimental setup.

To prevent damage to the robots, we add a hardware protection layer (see Sec. 10.1) that is adapted to our object manipulation scenario. Hardware protection intervenes as follows:

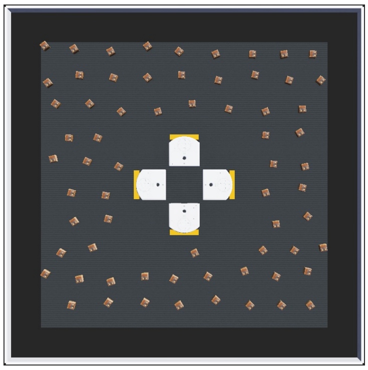
1. If a robot detects a too close obstacle with its horizontal proximity sensors ($s_0, \dots, s_6$), an escape behavior is executed.
2. If a robot detects the arena boundaries with its ground IR sensors (s_7, s_8), the robot is prevented to completely leave the arena by being forced to turn on the spot.
3. If a robot detects a pushing force with its pressure sensor (s_9) exceeding an equivalent of ca. 10 blocks, the robot is forced to turn away from the blocks on the spot to avoid motor damage.
4. If a robot constantly drives backward for more than 9 s, the robot is stopped to avoid motor damage as pushing force cannot be measured in this driving direction. The 9 s limit is reset once positive motor values occur.

Backward driving robots detect the arena boundaries only when they are already almost completely out of the arena, since the ground IR sensors are in the front of the robot, see Fig. 11.2c. In this case, the walls in distance to the arena's real boundaries trigger the robot's escape behavior (1.) and the robot drives back into the arena.

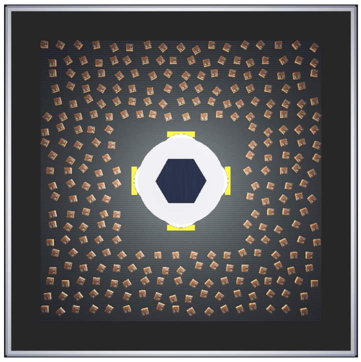
In simulation, we do 20 independent evolutionary runs per block density in the standard arena and the gradient arena. Additionally, we generate 20 random ANN pairs per arena and block density in simulation to study the effectiveness of our approach. In our real robot experiments, we do eight independent evolutionary runs per block density. As before, we post-evaluate the best evolved individual at the end of the evolutionary run. In simulation, we reset the arena for post-evaluation by placing the robots at their initial positions and randomly distributing the blocks in the arena. We store start and final positions of the blocks during the post-evaluation runs in simulation in addition to the tracked data as described in Sec. 10.1. In our real robot experiments, the robots start post-evaluation at their last position after $e_{\max}$ evaluations.



(a) real arena, high block density ($D_B \approx 0.14$)



(b) simulated standard arena, low block density ($D_B \approx 0.04$)



(c) simulated gradient arena, high block density ($D_B \approx 0.14$)

Figure 11.3: Arenas used in our real robot experiments and in the Webots simulator in our scenario aiming for object manipulation behaviors in real-world settings. The initial pose (i.e., position and heading) of each robot is fixed for all setups while blocks are distributed randomly at the beginning of the evolutionary run. For each arena, we use two different block densities $D_B \in \{0.04, 0.14\}$. Not shown is the master robot guiding the evolutionary process that is positioned outside the experimental arena, see Fig. 10.4 [374].

11.2 Experiments in Realistic Simulations

In the first step, we evolve object manipulation behaviors in the Webots simulator with our minimize surprise approach, which allows a detailed study of the emergent behaviors.

11.2.1 Metrics

We measure best fitness (Eq. 3.1) over evaluations to determine the success of evolution as in previous scenarios. In addition, we analyze and classify the behaviors of the best evolved individuals and the randomly generated individuals during the post-evaluation runs based on two metrics that are based on robot and block positions: (i) distance d_N covered by all robots and (ii) block displacement d_B.

We define the distance d_N covered by all robots as mean accumulated robot displacement over post-evaluation runtime T_P as given by

$$d_N = \frac{1}{N} \sum_{n=0}^{N-1} \sum_{t=0}^{T_P-1} ||P_n(t+1) - P_n(t)||_2 \,, \tag{11.1}$$

with N robots, and positions $P_n(t)$ and $P_n(t+1)$ of robot n at time steps t and $t+1$, respectively. Robots can cover a theoretical maximum distance d_N of 12.6 m when constantly driving with maximum linear speed of 0.126 $\frac{\mathrm{m}}{\mathrm{s}}$ during the 10,000 post-evaluation time steps of 10 ms each.

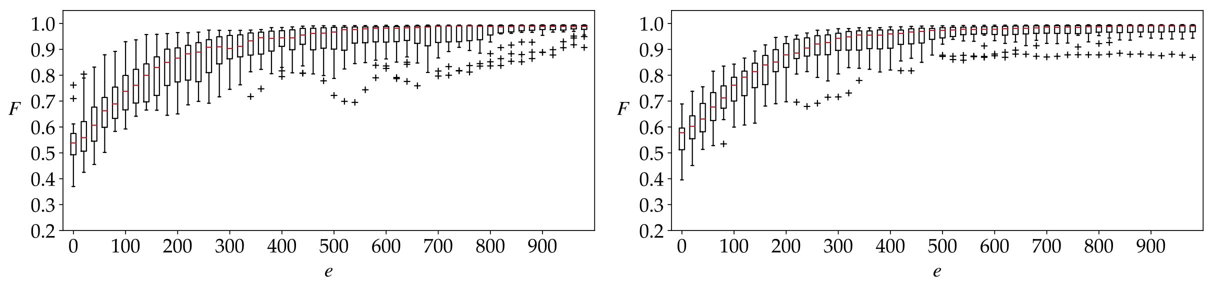
We define block displacement

$$d_B = \frac{1}{B} \sum_{b=0}^{B-1} ||P_b(T_P) - P_b(0)||_2 \tag{11.2}$$

as the mean Euclidean distance between the starting positions $P_b(0)$ and final positions $P_b(T_P)$ of the B blocks. The theoretical maximum displacement of a block is the arena's diagonal, but we obviously expect a lower effective mean distance. We find by qualitative analysis that a threshold of 0.1 in block displacement d_B distinguishes behaviors that lead to the pushing of blocks from behaviors with limited or no block manipulation.

11.2.2 Results

As representative examples for all obtained fitness curves, Fig. 11.4 visualizes the increase of best fitness F (Eq. 3.1) over evaluations per block density $D_B \in \{0.04, 0.14\}$ in the standard arena. For the best evolved individuals for both block densities, the median best fitness in the last generation is 0.99 in the standard arena and 0.97 in the gradient arena. For the randomly generated ANN pairs, we find a median best fitness of maximally 0.6, that is, evolution successfully improves fitness over generations.



(a) low block density ($D_B \approx 0.04$)

(b) high block density ($D_B \approx 0.14$)

Figure 11.4: Best fitness F over evaluations e for the simulated standard arena per block density D_B of 20 independent evolutionary runs each in our object manipulation scenario. Only every 20th evaluation is printed for clearer illustration. Medians are indicated by red bars [374].

Emergent Behaviors

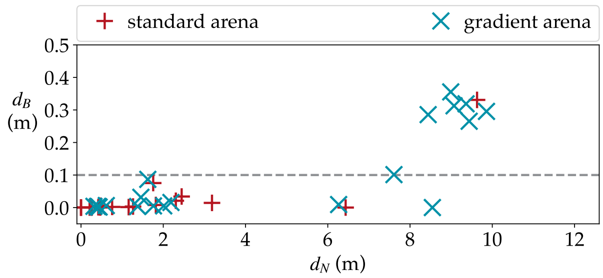
We analyze the behavior diversity of the best evolved individuals and the randomly generated individuals. A video of the emergent behaviors is online.[3] Fig. 11.5 quantifies the resulting behaviors based on distance d_N covered by all robots (Eq. 11.1) and block displacement d_B (Eq. 11.2). We distinguish three classes of behaviors: (i) circling, (ii) reverse driving, and (iii) behaviors that lead to the pushing of blocks.

3: https://doi.org/10.5281/zenodo.4293487 [343]

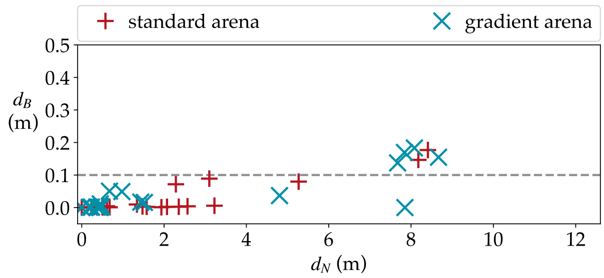
Circling The majority of the resulting behaviors of the best evolved individuals and the randomly generated ANN pairs has robots turn on the spot or go in small circles, see Fig. 11.6a. This results in short covered distances d_N and low block displacement (i.e., $d_B < 0.1$ m). In the evolutionary runs, we also find few cases where robots follow each other driving in a circle ('circle dance') leading to larger covered distances d_N, see Fig. 11.6b. By using hardware protection's escape behavior or, in rare cases, an intrinsic obstacle avoidance behavior, robots distribute themselves in the arena, which may lead to the pushing of few blocks. In most cases, the robots seem to execute a dispersion behavior for a limited duration at the beginning of the run until robots do not detect each other anymore.

Reverse Driving Some of the behaviors lead to robots driving backward until they are stopped by hardware protection, see Fig. 11.6c. Blocks are rarely pushed ($d_B < 0.1$ m) and the robots cover only short distances d_N. When all robots are stopped by hardware protection, the environment is static and thus easily predictable.

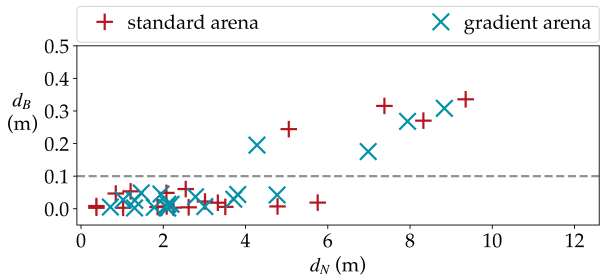
Behaviors Leading to the Pushing of Blocks We find behaviors leading to the pushing of blocks (i.e., $d_B > 0.1$ m) both in the evolutionary runs and for the randomly generated ANN



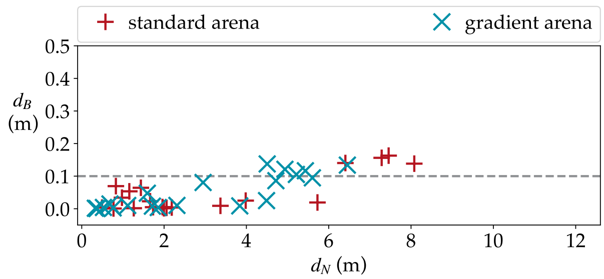
(a) best evolved individuals, low block density ($D_B \approx 0.04$)



(b) best evolved individuals, high block density ($D_B \approx 0.14$)



(c) randomly generated ANN pairs, low block density ($D_B \approx 0.04$)

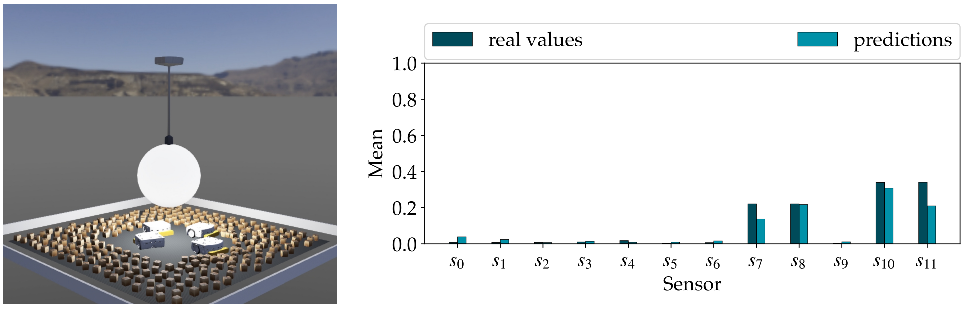


(d) randomly generated ANN pairs, high block density ($D_B \approx 0.14$)

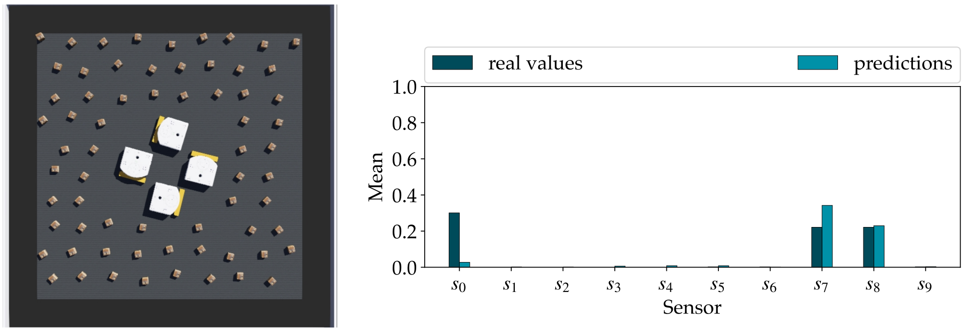
Figure 11.5: Distance d_N covered by all robots (Eq. 11.1) and block displacement d_B (Eq. 11.2) of the 20 best evolved individuals and 20 randomly generated individuals per arena and block density $D_B \in \{0.04, 0.14\}$ in our scenario aiming for object manipulation behaviors in realistic simulations. The dashed gray line marks a threshold between behaviors leading to the pushing of blocks (above line) and other behaviors [42, 48].

pairs. The best evolved individuals implement a random walk leading to the pushing of blocks by exploiting the hardware protection's boundary avoidance behavior. This also results in high covered distances d_N. Differences in the resulting block formations for the two block densities are probably caused by hardware protection. For the low block density, robots clear the arena from blocks or, in one run, push blocks around in the arena. By contrast, robots push blocks into small clusters in the high block density runs, see Fig. 11.6d. With increased block density, robots exceed the threshold of maximum pushed blocks faster and turn away from a potentially formed block cluster. In evolution, we find such behaviors more often in the gradient arena than in the standard arena for both block densities. The non-uniform light conditions in the gradient arena increase prediction difficulty, since light values fluctuate even in the simple circling behavior, thus not allowing for trivial predictions as in the standard arena. We assume that the overall increased task difficulty enables the emergence of more block pushing behaviors in the gradient arena. For the randomly generated ANN pairs, behaviors leading to the pushing of blocks are equally likely found for both arenas and block densities. We find random walk behaviors leading to the pushing of blocks as for the best evolved individuals in the standard arena. But in the gradient arena, the majority of block pushing behaviors of the randomly generated ANN pairs leads to robots pushing blocks while driving in large circles.

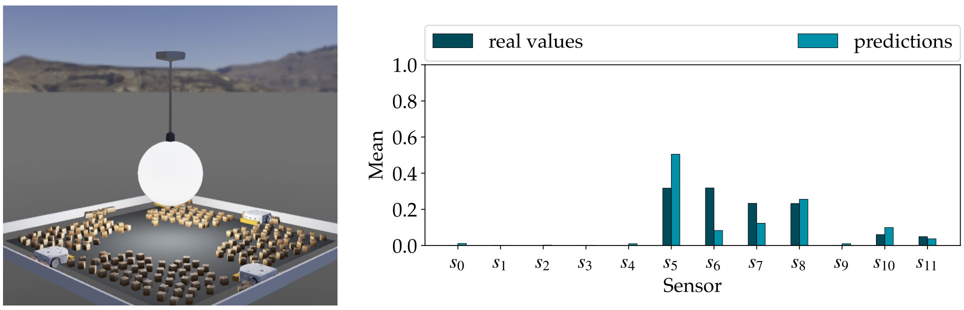
While we find the same behavior classes for the best evolved



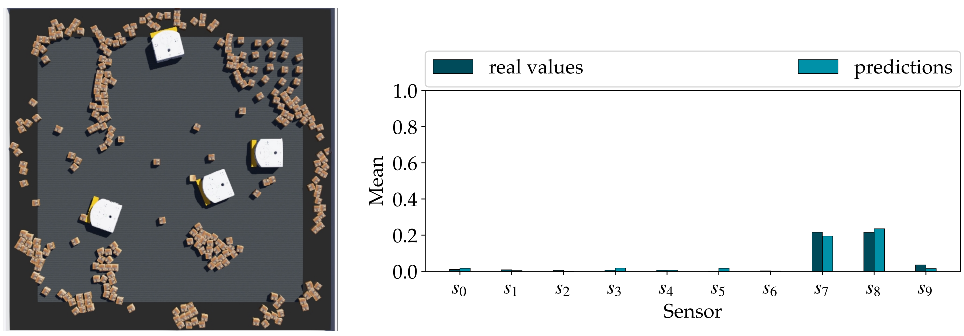
(a) circling (gradient arena, $D_B \approx 0.14$)



(b) circle dance (standard arena, $D_B \approx 0.04$)



(c) reverse driving (gradient arena, $D_B \approx 0.14$)



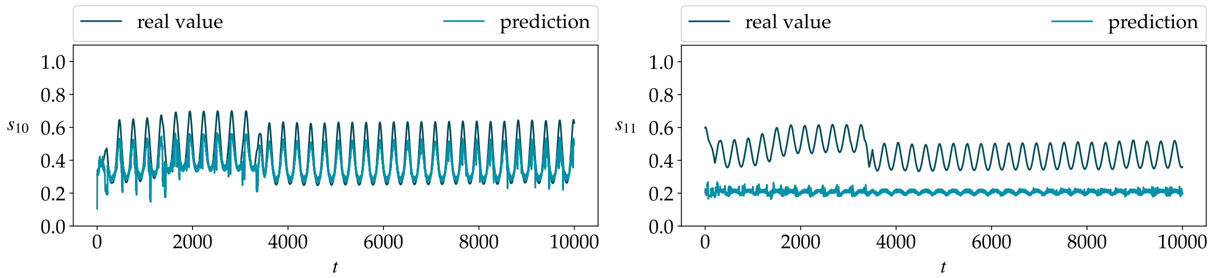
(d) pushing of blocks (standard arena, $D_B \approx 0.14$)

Figure 11.6: Robot and block positions at the end of the post-evaluation run and the swarm's mean sensor values and predictions over the post-evaluation run of four best evolved individuals in our scenario aiming for object manipulation behaviors in realistic simulations. $s_0, \ldots, s_4$ give the frontal horizontal proximity sensors, s_5 and s_6 the back horizontal proximity sensors, s_7 and s_8 the ground IR sensors, s_9 the pressure sensor, and s_{10} and s_{11} the light sensors (only used in the gradient arena).

individuals and the randomly generated ANN pairs, Fig. 11.5 shows differences between the behavior distributions based on distances d_N covered by all robots and block displacement d_B. As in our previous scenario (see Sec. 10.3), we find behavioral clusters for the best evolved individuals, while the randomly generated ANN pairs lead to a broader distribution. The best evolved individuals either lead to low block displacement d_B and low distance d_N covered by robots (i.e., circling and reverse driving), or high block displacement and high covered distance by robots (i.e., random walk leading to pushing of blocks). Intermediate covered distances by robots d_N are rarely found for the best evolved individuals and lead in several cases to the sophisticated 'circle dance' behavior (i.e., low d_B and high d_N). For the randomly generated ANN pairs, we find several behaviors leading to intermediate distances d_N covered by robots. In these cases, robots drive in larger circles, occasionally pushing blocks or avoiding other robots in some runs. Consequently, we find that both actors and predictors are optimized by evolution leading to more distinctive behaviors and better fitness, that is, prediction accuracy.

Predictions

A good interplay between actor and predictor (see Sec. 3.2) allows for high fitness and prediction accuracy. Fig. 11.6 exemplifies the mean sensor values and predictions of the different emergent behaviors. For circling (Fig. 11.6b) and behaviors leading to the pushing of blocks (Fig. 11.6d), we find that robots do not predict or sense other robots (i.e., $s_0, \dots, s_6 \approx 0$). The 'circle dance' behavior is an exception, since it leads to robots collectively driving in a circle detecting the robot in front with their front left IR sensor s_0, see Fig. 11.6b. In reverse driving (Fig. 11.6c), robots detect and predict the outer arena walls with their back IR sensors (s_5, s_6). For all behaviors, the real and predicted ground IR sensor values (s_7, s_8) match the light reflected from the arena's carpet. Pressure sensor values (s_9) and predictions are low as hardware protection forces robots to turn when pushing more than 10 blocks, that is, $s_9 > 0.25$. Thus, all sensors used in the standard arena allow for trivial predictions. But in the gradient arena, light intensity fluctuates based on the distance from the arena's center and thus, the light sensors (s_{10}, s_{11}) do not generally allow for trivial predictions. Reverse driving behaviors lead to constant, low light intensity as robots are usually stopped by hardware protection at the arena's boundaries where light intensity is low, see Fig. 11.6c. By contrast, the detected light intensity fluctuates in circling (Fig. 11.6b) and behaviors leading to the pushing of blocks. Fig. 11.7 illustrates oscillations of the actual light intensity detected by the light sensors s_{10} and s_{11} that are triggered by a repetitive circling behavior. For most behaviors, we find that predictions roughly follow these oscillations but with an offset, see Fig. 11.7b. In few cases,



(a) real and predicted values of s_{10}

(b) real and predicted values of s_{11}

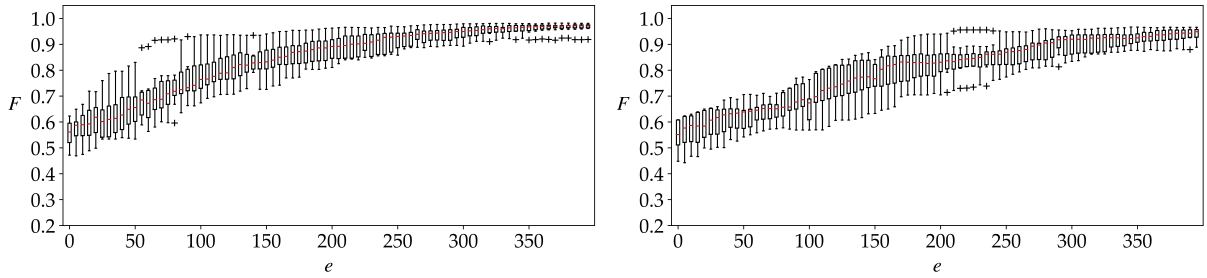
Figure 11.7: Real and predicted sensor values for light sensors s_{10} and s_{11} over post-evaluation runtime for one robot executing the circling behavior shown in Fig. 11.6a [374].

we find rather complex and adapted sensor predictions as depicted in Fig. 11.7a. Consequently, our approach, including the chosen ANN structure, generally allows for sophisticated predictor outputs.

A potential reason for the poor adaptation of the light sensor predictions to the actual values is that the two light sensors only account for one sixth of the total fitness. Thus, we want to put more pressure on their prediction accuracy by restricting the predictor outputs to light and pressure sensors only (i.e., no prediction of IR sensors $s_0, \ldots, s_8$). We do 20 independent evolutionary runs in the gradient arena with high block density. As before, we find predictions following the fluctuations in light intensity with an offset. This could also be caused by resetting the arena for the post-evaluation runs: robots may be at different positions in the arena at the end of the evolutionary run than at the beginning and have optimized their predictions for different light intensities as a result. We study this in more detail in the next section.

Robot-Environment Feedback Loop

Next to the poorly adapted light sensor predictions, we find that fitness of the post-evaluation runs is statistically significantly lower than the best fitness at the end of the evolutionary runs for all scenarios (MW-U, $p < 0.05$). Both aspects indicate a feedback loop during evolution: individuals change the environment and the altered environment leads to adapted individuals. As we reset the arena for post-evaluation, the individuals may not be adapted to this 'new' environment anymore. We test our hypothesis by doing 20 evolutionary runs predicting only light and pressure sensors in the gradient arena with high block density. By contrast to our previous runs, we do not reset the arena for post-evaluation. Thus, the best evolved individual is post-evaluated in the same environment as at the end of the evolutionary run. In this case, we no longer find statistically significantly different fitness values. Robots frequently move to the arena's edges



(a) low block density ($D_B \approx 0.04$)

(b) high block density ($D_B \approx 0.14$)

Figure 11.8: Best fitness F over evaluations e for both block densities D_B of eight independent evolutionary runs each in real robot experiments aiming for object manipulation behaviors. Only every 5th evaluation is printed for clearer illustration. Medians are indicated by red bars.

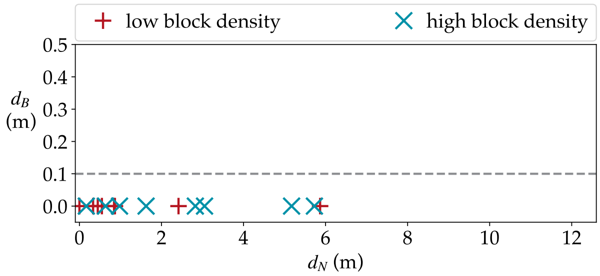


Figure 11.9: Approximated distance d_N covered by all robots and block displacement d_B of the eight best evolved individuals per block density $D_B \in \{0.04, 0.14\}$ in our real robot experiments aiming for object manipulation behaviors. The dashed gray line marks a threshold between behaviors leading to the pushing of blocks (above line) and other behaviors.

during evolution and thus sense only low light intensities. Consequently, adaptation to fluctuations in light intensity is not rewarding as those are only small at the arena's boundaries. Resetting the arena leads to higher light intensities and fluctuations, and thus lower prediction accuracy.

11.3 Real Robot Experiments

Next, we evolve object manipulation behaviors in real robot experiments with our minimize surprise approach. Each of the eight independent evolutionary runs per block density $D_B \in \{0.04, 0.14\}$ (see Sec. 11.1) took approximately 78 min.

Fig. 11.8 visualizes the increase of best fitness over evaluations for both block densities. We reach a median best fitness of 0.97 for the low block density and of 0.94 for the high block density.

Unlike for the simulations runs, we cannot directly log either robot or block trajectories in our real robot experiments. However, we can estimate the distance covered by all robots based on their wheel speeds[4] and track block movement based on the video footage. Since we do not find any block pushing during the post-evaluation phase when manually analyzing the videos, we refrain from using elaborate video analysis approaches and set block displacement d_B for all runs to 0.0 m. Fig. 11.9 visualizes the resulting behaviors based on distance d_N covered by all robots and the manually determined block displacement d_B. We find qualitatively the

4: Robot displacement between two time steps t and $t+1$ ($\Delta t = 100$ ms) is $\Delta P(t, t+1) = v(t)\Delta t$ with $v(t) = \frac{v_r(t)+v_l(t)}{2}$ being the linear velocity of the robot set at time step t.

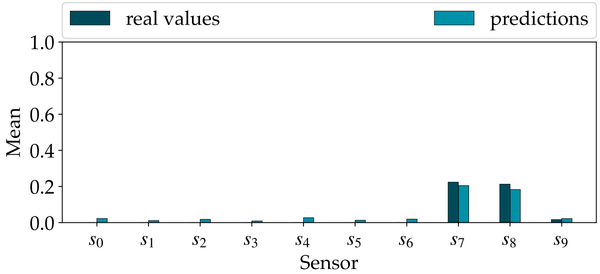
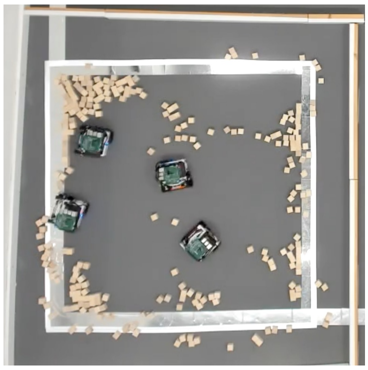


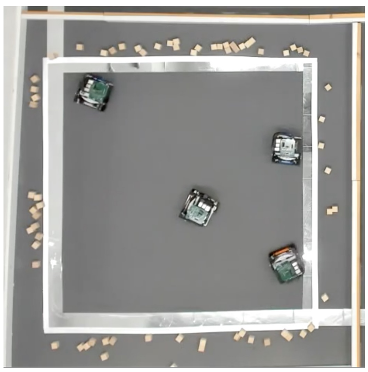
Figure 11.10: Mean sensor values and predictions over the post-evaluation run of one best evolved individual in our scenario aiming for object manipulation behaviors in real robot experiments. $s_0, \ldots, s_4$ give the frontal horizontal proximity sensors, s_5 and s_6 the back horizontal proximity sensors, s_7 and s_8 the ground IR sensors, and s_9 the pressure sensor.

same behaviors as in simulation: circling (Fig. 11.11a), reverse driving (Fig. 11.11b), and random walk behaviors potentially leading to the pushing of blocks (Fig. 11.11c). However, there are some differences in the observed behaviors, since we do not reset the arena for post-evaluation. We do not find pushing of blocks: robots circle in areas of the arena that have already been cleared of blocks, reverse driving robots are stopped by hardware protection quickly, and we find random walk behaviors only in arenas that have already been cleared of blocks completely. As in the simulation runs, we mainly find circling behaviors for both block densities. We find reverse driving only in three runs in the low block density. Random walk emerges once in the low block density and twice in the high block density (i.e., individuals with $d_N > 5$ m in Fig. 11.9). Sensor value predictions are also similar to the results in simulation, see Fig. 11.10 and Sec. 11.2.2. All horizontal proximity sensor values and predictions ($s_0, \ldots, s_6$) are most of the time approximately zero. The ground IR sensor values and predictions match the reflected light from the arena's carpet (i.e., $s7, s8 \approx 0.25$). As robots do not push blocks in the emergent behaviors, the pressure sensor (s_9) has low values and predictions.



(a) circling ($D_B \approx 0.14$)

(b) reverse driving ($D_B \approx 0.04$)



(c) random walk ($D_B \approx 0.04$)

Figure 11.11: Robot and block positions at the end of the post-evaluation run of three best evolved individuals in our scenario aiming for object manipulation behaviors in real robot experiments.

11.4 Discussion and Conclusion

In this second scenario using realistic simulations and real robot experiments, we have shown again that our minimize surprise approach effectively leads to the emergence of diverse behaviors. Compared to our previous scenario aiming for basic swarm behaviors, we increased the complexity by adding manipulable blocks to the robots' environment. Robots show an innate motivation to clear areas from blocks in order to simplify their predictions here. Structure formation of blocks is unlikely because robots tend to push blocks out of the arena instead of forming block clusters. This is probably caused by the bulldozer blade that only allows object pushing in 2D – a more sophisticate approach would require at least block-pulling capabilities or even a gripper [375]. Another limiting factor is our environment with fixed boundaries. In our previous collective construction scenario (see Ch. 9), agents lived on a torus grid, which simplified the positioning and allowed for more structures to emerge.

In our study, we have particularly shown that robots adapt to a changing environment at runtime as it is manipulated by themselves, that is, we have a robot-environment feedback loop during evolution. We argue that our minimize surprise approach increases reliability of the system in an open-ended process of adaptation because the robots autonomously adapt to any, possibly non-anticipated, situation. This happens due to the doctrine of accurate predictions and we speculate that predictable robot behaviors are considered generally to be safe and reliable behaviors as also argued by Friston et al. [340] for the biological case.

A big focus of our two scenarios was to show the feasibility of our approach in real-world settings. We have shown that our minimize surprise approach implements a viable option to deal with dynamic environments in real-world multi-robot systems as it can be executed online and onboard. However, we only presented the first step to adapt our approach for real-world applications, since our currently used evolutionary setup still relies on a central master robot. In future work, we plan to implement a fullscale distributed online onboard evolutionary approach to further increase the robustness of the system. Furthermore, we want to do more robot experiments in different scenarios to continue showing the diversity of emerging behaviors and the real-world capability of minimize surprise.

Conclusion and Outlook | 12

Chapter Contents

In this chapter, we ...

- *Sec. 12.1:* summarize our results and
- *Sec. 12.2:* present ideas for future work.

In this thesis, we have studied our minimize surprise approach in detail. We conclude by summarizing our results to present an overall picture of the potential of our minimize surprise approach and outline future work.

12.1 Summary and Discussion

We structure the summary and discussion of our results by answering the six research questions we have posed in the introduction (see Ch. 1).

Q1 How robust, scalable, and diverse are collective behaviors evolved with minimize surprise?

Based on a simple self-assembly scenario on a 2D torus grid, we have shown that our minimize surprise approach leads to robust, scalable, and diverse collective behaviors in Chs. 4 and 5. We have demonstrated the robustness of our minimize surprise approach and of the emergent behaviors in Sec. 5.1. When introducing sensor noise in our otherwise fully deterministic setup, our minimize surprise approach adapts by selecting behaviors that are robust to sensor errors (e.g., aggregation and dispersion in our self-assembly scenario). In addition, the resulting behaviors are robust to external disturbances, such as the damage of the self-assembled structure. The swarm reassembles into the same structure type after having removed or repositioned agents from the initially formed structure. In Sec. 5.2, we have proven that behaviors evolved with minimize surprise are scalable with swarm density. Actor-predictor pairs that are specialized for a specific swarm density can be reused in other swarm densities while still leading to the same behavior. Last but not least, we have shown that minimize surprise leads to behavioral diversity across several independent evolutionary runs. In Sec. 4.4, we have found that the emergent behavioral diversity depends on swarm density. High swarm densities lead to mainly grouping behaviors while dispersion prevails in low densities and the highest behavioral diversity emerges in intermediate densities. In Sec. 5.3, we have also demonstrated that our standard minimize surprise approach leads to competitive behavioral diversity compared to task-independent novelty search and MAP-Elites variants (see also research question Q3).

Q2 How can we engineer our minimize surprise approach so that desired behaviors emerge?

We can engineer self-organization towards desired behaviors by partially or completely predefining sensor predictions, see Ch. 6. Thus, our minimize surprise approach can be used in three variants: (i) fully task-independent giving evolution the freedom to come up with creative solutions, (ii) biasing evolution towards certain behaviors by partially predefining predictions, and (iii) fully task-specific by predefining all predictions to evolve desired behaviors. We have exemplified all three variants using our self-assembly scenario (see Sec. 4.1), since the grid world and the binary sensors used in this setup enable easy predefining of sensor predictions. Predefining predictions in real-world settings and for desired behaviors with time-variant sensor input is probably harder. Continuous sensor values, noise, and other factors may complicate determining sensor values associated with a specific behavior in real-world settings. While predefining time-variant sensor predictions is generally possible in our approach, it is challenging to specify the timing of changes in sensor input. Still, our approach to engineer self-organization towards desired behaviors by predefining predictions is promising and we are confident that we can also apply it in real-world settings and for evolving behaviors with time-variant sensor input.

Q3 How does minimize surprise compare to other approaches?

We have found that minimize surprise is effective and outperforms random search in Sec. 4.5. But minimize surprise is also competitive to state-of-the-art approaches. As mentioned before (see research question Q1), we have compared our minimize surprise approach to novelty search and MAP-Elites in Sec. 5.3. All three approaches output diverse behaviors: minimize surprise results in behavioral diversity across several independent evolutionary runs while novelty search and MAP-Elites generate behavioral diversity within one evolutionary run. When comparing the same number of evaluations per approach, our standard minimize surprise approach is competitive to novelty search and MAP-Elites in solution quality, fitness, and behavioral diversity. MAP-Elites and minimize surprise output high-quality solution sets at the end of the evolutionary run in contrast to novelty search that requires high post-evaluation effort to select high-quality solutions. MAP-Elites generates several diverse and high-performing solutions within one run but requires more evaluations in a single run to be able to fill the behavior-performance map with high-quality solutions. Thus, standard minimize surprise is potentially more suitable for the use on real robots. Furthermore, we have compared minimize surprise with predefined predictions to an evolutionary algorithm with a standard task-specific fitness function in Sec. 6.2. Both approaches lead to the emergence of the desired behavior, but minimize surprise with predefined predictions offers a more intuitive way of defining the fitness function.

Q4 How can we evolve dynamic behaviors that adapt their behavior according to varying sensor input using our minimize surprise approach?

In previous work and in our self-assembly scenario (see Ch. 4), we used static environments, except for the swarm behavior itself, in combination with our task-independent reward for prediction accuracy. This led to repetitive behaviors that are easy to predict. In Ch. 7, we have shown that modifications to the fitness function (e.g., including additional rewards for curiosity or homing), environment (e.g., changing obstacle positions), and agents (e.g., energy sharing between agents) have the potential to push evolution towards more dynamic behaviors that adapt to changes in their sensor input (i.e., behavioral plasticity [364]). Although none of the tested modifications guarantees the emergence of dynamic behaviors, using complex and dynamic environments seems to be the key to evolve such behaviors.

Q5 Can we evolve collective behaviors for scenarios with different environmental complexities and agent capabilities with minimize surprise?

In this thesis, we have successfully evolved collective behaviors for spatial organization, navigation, and decision-making with minimize surprise in six different scenarios. In the simplest case, the swarm itself was the only dynamic element in the environment (Chs. 4, 8, and 10). In other cases, the swarm could change the environment by pushing manipulable objects (Chs. 9 and 11) or the obstacle positions dynamically changed independent from the collective behavior (Ch. 7). While minimize surprise successfully optimized the actor-predictor ANN pairs in all scenarios, we have found that a careful configuration of environment and robots is essential to evolve collective behaviors that are useful for the user. In particular our collective perception scenario (see Ch. 8) has shown that a strong coupling between sensors and (desired) collective behavior is required for the evolution of useful behaviors. To elaborate, evolution will adapt the actors so that the prediction of the sensor values that can be influenced by the collective behavior will be simplified. Behaviors that are as simple as possible to allow for easy sensor predictions will potentially emerge. More complex behaviors, that depend on sensors whose values cannot be influenced by the collective behavior due to the experimental setup, are unlikely to evolve – especially if they do not simplify the prediction task. Moreover, since each additional predicted sensor reduces the impact on fitness of all other sensors proportionally, the set of predicted sensors should only include sensors relevant to the scenario and a potentially desired behavior. Although our first findings can provide some guidance, we still lack clear guidelines for the configuration of environment and robots to ensure the emergence of behaviors that are useful to the user.

Q6 Which adaptations are necessary to apply minimize surprise in real-world settings?

In Sec. 10.1, we have presented a centralized online and onboard evolutionary architecture for minimize surprise that has proven to be suitable for real-world settings. In realistic simulations and real-world experiments, we have evolved basic collective behaviors (see Ch. 10) and object manipulation behaviors (see Ch. 11) using this approach. As in our simpler simulation environments, we have found behavioral diversity across several independent evolutionary runs in these experiments. In our object manipulation scenario, we have found a robot-environment feedback-loop during evolution, that is, robots change the environment and in turn adapt to the changed environment. That means that minimize surprise allows robot swarms to continuously adapt their behavior to a changing environment, which is an important aspect for open-ended evolution [13, 376].

12.2 Outlook

Our studies in this thesis have answered many open questions on minimize surprise, but, as inherent to research, have raised several new ones. At the end of each chapter, we have already presented open aspects for future work for the specific topic. We conclude this thesis with a more general discussion of potential future extensions of our work.

In all studies on minimize surprise, we have used simple evolutionary algorithms to evolve the weights of neural networks with a fixed topology to generate diverse collective behaviors across several independent evolutionary runs. In future work, we want to study the effect on the resulting behaviors when using more sophisticated evolutionary algorithms with our minimize surprise fitness function. Not only the network weights could be evolved but also the network topology, for example, using NEAT [270], or even the robot morphology, since embodiment plays an essential role for intelligence [237]. Because our combination of minimize surprise with standard MAP-Elites led to promising results (see Sec. 5.3), combining minimize surprise with more sophisticated quality-diversity algorithms [44, 305] has likely potential.

All our experiments were run completely either in simulation or on the real robots, since the reality gap (i.e., differences between simulation and real world) makes a transfer of results from simulation to real robots challenging. However, using simulations to evolve controllers for real robots can bring advantages in optimization speed and prevention of wear and tear of our robot hardware. A promising option to avoid the reality gap is to start the evolutionary process in simulation to optimize our actor-predictor pairs and to continue the evolutionary process for a few more generations on the real robots to allow adaptation to the differences between simulation and real world.

In Sec. 5.2, we have shown that minimize surprise leads to behaviors that are often scalable with swarm density. But the performance of the actor or predictor may decrease when an actor-predictor pair is rerun in a much higher or lower swarm density than the one used during evolution. Scalability could be taken into account in the optimization process by evaluating the actor-predictor pairs in different swarm densities. This would probably have two effects: the resulting actor-predictor pairs would scale well with swarm density and the behavioral diversity would be decreased to behaviors that perform well with all densities.

The essence of our minimize surprise approach is that we evolve actor-predictor ANN pairs by rewarding only the predictor. The evolutionary operators are applied to the ANN pair as a whole, that is, actor and predictor are mutated. An alternative can be to use an approach of cooperative coevolution [377]. Actors and predictors are evolved alternately, that is, the actor stays fixed while the predictor is evolved and vice versa while always rewarding prediction accuracy. In this way, the actor has time to adapt to the predictor and the predictor has time to adapt to the actor without also having to adjust to changes to the other ANN. This can potentially allow for the emergence of more complex behaviors.

In our scenarios, swarms of ground mobile robots lived most of the times in static environments, except for the swarm behavior itself, and the swarm members predicted the values of exteroceptive sensors, which resulted in interesting but repetitive collective behaviors. Since the prediction of proprioceptive and interoceptive sensors can lead to a sense of agency and a sense of self in living organisms [345], it may also enable the evolution of interesting and potentially intelligent behaviors in robot swarms. Our experiments on the evolution of more dynamic behaviors in complex environments in Ch. 7 have shown that especially dynamic environments have the potential to push evolution towards dynamic behaviors that react to varying sensor input. Environments that are even more dynamic and unstructured (e.g., with continuously changing environmental conditions or other autonomous agents that are not part of the swarm) are also likely to lead to the evolution of more dynamic behaviors. Interesting is also to evolve behaviors on other robot platforms than the ground mobile robots used so far. Different robot platforms come with different challenges, for example, Uncrewed Aerial Vehicles (UAV) move in 3D space while our ground mobile robots move in 2D space, which may lead to completely different collective behaviors.

Minimize surprise is generally suitable for the application in real-world scenarios and allows for continuous adaptation to changing environments as we have shown in our experiments in Chs. 10 and 11, which is particularly crucial in dangerous and inaccessible environments [378]. But our currently used centralized online onboard evolutionary approach has a single point of failure, namely the master robot coordinating the evolutionary process. The system can be made more robust for long-term operation in the real world by using embodied evolution (i.e., online distributed evolution) [291, 292]. Embodied evolution is also considered to have the potential for realizing open-ended evolution. The latter aims for the ongoing adaptation to changes in the environment, the continuous generation of new behavioral patterns, and ongoing growth of complexity as found in nature [13, 376]. Our minimize surprise approach has already proven its capability of ongoing adaptation to environmental changes and of generating behavioral diversity over several evolutionary runs due to the task-independent reward for prediction accuracy. Combining minimize surprise with embodied evolution and placing the swarm in highly dynamic and unstructured environments, where novel and unforeseen situations can occur, may enable ongoing behavioral innovation and adaptation in one evolutionary run over long execution times. However, the operating time of real robots is limited by two factors, namely battery runtime and the robustness of the hardware. These aspects can be addressed in the setup by including a charging station and by repairing or replacing defective robots.

In all our experiments, the predictor has only been used as means to guide the evolution of collective behaviors. But it could also be used as an indicator for changes in the environment. That is, prediction accuracy of an optimized predictor will likely decrease severely when the environment changes considerably. For example, this could be used as warning signal for the user when the actor-predictor pair was optimized for an environment that is structured and stationary except for the swarm behavior itself. Or it could be used to influence the evolutionary process. Evolution could pause as soon as high prediction accuracy is reached and the current best evolved actor executed for a longer time period. As soon as a decrease in prediction accuracy is detected, evolution could continue to allow adaptation to the changes in the environment.

In this thesis, we have proven the great potential of our minimize surprise approach for evolving collective behaviors for robot swarms in simulation and on real robots. Future enhancements will make minimize surprise an even more versatile approach that is easy and intuitive to use for a variety of purposes and application scenarios. Users can apply minimize surprise to evolve desired behaviors by predefining predictions or even evolve a variety of interesting and potentially increasingly complex behaviors – maybe even to the emergence of intelligent behavior.

Appendix

A Minimize Surprise Based on Simple Machine Learning Methods

In this thesis, we use methods of evolutionary computation for the automatic generation of swarm robotic controllers. An alternative may be the use of standard machine learning methods, such as gradient descent and backpropagation [242]. Here, we show the potential and the challenges of using machine learning methods for minimize surprise in a simple experiment with a single robot.

A.1 Experimental Setup

We introduce the arena and robot platform first and present our machine learning-based minimize surprise approach afterwards.

A.1.1 Arena and Robot

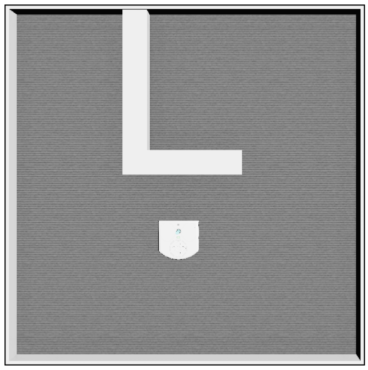


Figure A.1: Arena for our experiments on machine learning-based minimize surprise in the Webots [46] simulator. The arena is 1 m × 1 m and contains an L-shaped obstacle. A single Thymio II robot is initially placed approximately in the arena's center.

In our experiment, we use a single Thymio II [77] ground mobile robot (cf. Sec. 2.1.2) in the Webots [46] simulator that is initially placed approximately in the arena's center. We only use the robot's five horizontal front IR sensors $s_0, \dots, s_4$ (cf. Fig. 10.5), since we restrict robot movement to moving straight forward and turning on the spot. The wheel speeds for the Thymio II's differential drive are specified by a_0 and a_1. We place the robot in an arena of size $1\text{ m} \times 1\text{ m}$ that is bounded by white walls, see Fig. A.1. Compared to our previously studied swarm scenarios, the robot is in a completely static environment here, since not even identical swarm members introduce any dynamics. Thus, we increase the complexity of the environment by placing an additional L-shaped obstacle in the arena. To prevent collisions with the walls or the obstacle, we include a simple hardware protection layer. If the robot gets too close to a wall or the obstacle, the robot will turn away from the wall for a random time period.

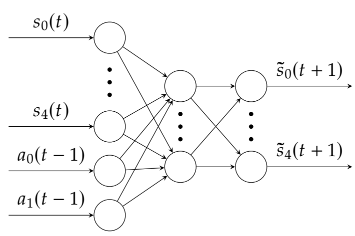


Figure A.2: Predictor ANN in our experiments on machine learning-based minimize surprise. The feedforward ANN receives the robot's current front IR sensor values $s_0(t), \dots, s_4(t)$ and its current wheel speeds $a_0(t)$ and $a_1(t)$ as input, and outputs sensor predictions $\tilde{s}_0(t+1), \dots, \tilde{s}_4(t+1)$ for time step $t+1$.

A.1.2 Machine Learning-Based Minimize Surprise Approach

As in our minimize surprise approach based on methods of evolutionary computation (cf. Ch. 3), we equip the robot with an actor-predictor pair. In this experiment, the predictor is a three-layer feedforward[1] ANN, see Fig. A.2. The predictor has seven input, seven hidden, and five output neurons. The ANN receives a robot's current front IR proximity sensor values $s_0(t), \dots, s_4(t)$ and its current wheel

1: We do not use a recurrent ANN as in previous experiments to simplify training using gradient descent and backpropagation.

Algorithm 5: Template for the reactive behavior serving as the actor for our machine learning-based minimize surprise approach.

parameters: sensor indices $i, j \in \{0, \dots, 4\}$,
sensor thresholds $\Gamma_0, \Gamma_1 \in [0, 1]$,
turning directions $\mathcal{T}_0, \mathcal{T}_1 \in \{0, 1\}$

if $s_i(t) > \Gamma_0$ **then**
    turn in direction $\mathcal{T}_0$
else if $s_j(t) > \Gamma_1$ **then**
    turn in direction $\mathcal{T}_1$
else
    drive straight forward (i.e., $a_0(t) = a_1(t)$)

speeds $a_0(t)$ and $a_1(t)$ as input and outputs IR sensor predictions $\tilde{s}_0(t+1), \dots, \tilde{s}_4(t+1)$ for time step $t+1$. Since the actual sensor values $s_0(t+1), \dots, s_4(t+1)$ of the next time step $t+1$ are the target values for the predictor outputs (i.e., sensor predictions $\tilde{s}_0(t+1), \dots, \tilde{s}_4(t+1)$) of the current time step t, a labeled set of training data can be automatically generated allowing to train the predictor in a self-supervised [274, 346] way. We implement and train the ANN with backpropagation and gradient descent using the `MLPRegressor`[2] class of the Python machine learning library scikit-learn [379] with standard parameters except for the number of neurons in the hidden layer that we set to seven neurons here. By default, the `MLPRegressor` class uses Adam [380] as the solver, the rectified linear unit function[3] as activation function, and a constant learning rate of 0.001. Each training sample maps the sensor values $s_0(t), \dots, s_4(t)$ and wheel speeds $a_0(t)$ and $a_1(t)$ of time step t to the sensor values $s_0(t+1), \dots, s_4(t+1)$ of the next time step $t+1$. Over a learning run, we automatically create a labeled data set adding one new training sample per time step. Once the data set contains 15,000 training samples, the oldest training sample is removed whenever a new training sample is added. That is, we limit the data set size to maximally 15,000 training samples. Every 750 time steps, we first test the predictor with the new 750 training samples (i.e., interleaved test-then-train evaluation or prequential evaluation) and train it incrementally [381] afterwards, that is, we use the existing model as a starting point and retrain it with the updated data set.

2: `https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html`

3: $f(x) = \max(0, x)$

As already explained in Sec. 3.2, we cannot manually or automatically generate labeled data to train the actor using (self-)supervised learning, since our minimize surprise approach is task-independent. Consequently, the deviation between the current and a targeted robot behavior cannot be determined and used for training. Therefore, we pair the predictor with a reactive behavior serving as the actor that is adapted during the learning run. Alg. 5 specifies the standard structure of the reactive behavior. The reactive behavior has six adaptable parameters: sensor indices i and j in $\{0, \dots, 4\}$ determining the used sensor values $s_i(t)$ and $s_j(t)$, sensor thresholds Γ_0 and Γ_1 in range $[0, 1]$, and turning directions $\mathcal{T}_0$ and $\mathcal{T}_1$ being 0 for counterclockwise (i.e., $a_0(t) = -a_1(t)$) and 1 for clockwise (i.e., $-a_0(t) = a_1(t)$) turning. These six parameters are uniformly randomly initialized within their specified range at the start of a learning run. The reactive behavior is adapted, similar to mutations in evolutionary computation, when one of the four following conditions[4] is met:

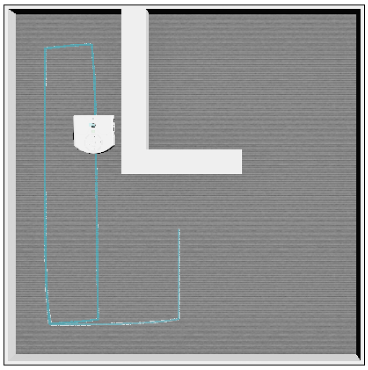
4: All four conditions are based on results from preliminary investigations.

1. The predictor performs poorly on the test set, that is, a score of less than −25.0 is reached.[5] In this case, the robot usually got stuck in a corner of the arena.
2. The hardware protection layer of the robot interferes to prevent collisions. This condition applies only after the first training of the predictor, since the previously executed reactive behavior may have led to a disadvantageous robot position in the arena (e.g., the robot got stuck in an arena corner). Thus, the robot can free itself from a potentially disadvantageous position using both the currently executed reactive behavior and the hardware protection in the first 750 time steps of the execution of a reactive behavior.
3. The last five scores have a variance of less than 0.00016 and a mean value of less then 0.9. This is an indicator that the learning process may be stuck in a local minimum.
4. The predictor has 15 negative scores in a row.

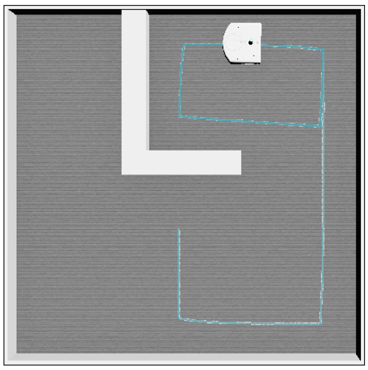
5: The `MLPRegressor` class uses the coefficient of determination R^2 of the prediction as testing score. The coefficient of determination is defined as $R^2 = 1 - \frac{\text{RSS}}{\text{TSS}}$ with residual sum of squares $\text{RSS} = \sum_r \sum_t (s_r(t) - \tilde{s}_r(t))^2$ and total sum of squares $\text{TSS} = \sum_r \sum_t (s_r(t) - \bar{s}_r)^2$ over all time steps t included in the test set with actual value $s_r(t)$ of sensor r at time step t, predicted value $\tilde{s}_r(t)$ for sensor r at time step t, and mean actual value $\bar{s}_r$ of sensor r. The best possible R^2 value, and thus score, is 1.0. Scores can be negative indicating that the model fits poorly.

When at least one of these four conditions is met, we reset the predictor network (i.e., weights are randomly initialized) and change one randomly chosen parameter of the reactive behavior. When adapted, sensor indices i or j are increased or decreased by one, sensor thresholds Γ_0 and Γ_1 are replaced by a uniformly randomly drawn value from range $[0, 1]$, and turning directions $\mathcal{T}_0$ and $\mathcal{T}_1$ are inverted.

We do eight independent learning runs in simulation and stop training if a score of 0.95 or higher is reached.



(a) clockwise wall following



(b) counterclockwise wall following

Figure A.3: Resulting wall following behaviors as representative examples for all runs using our machine learning-based minimize surprise approach. The blue line gives the robot's trajectory.

A.2 Results

The eight independent learning runs led to a score of at least 0.95 after a mean time of approximately 16 min (simulated time). Half of the runs resulted in clockwise wall following behaviors, see Fig. A.3a, and the other half in counterclockwise wall following behaviors, see Fig. A.3b. Due to the simple reactive behavior structure, the robot cannot turn around corners that exceed 180°. Consequently, the resulting wall following behaviors are simple. The robot drives on a rectangular path on an area of the arena that is separated by the L-shaped obstacle.

Alg. 6 gives the reactive behavior shown in Fig. A.3b as a representative example. In this case, both the outer right IR

Algorithm 6: Reactive behavior shown in Fig. A.3b generated using machine learning-based minimize surprise.

if $s_4(t) > 0.009$ **then**
  turn in direction 0 (i.e., counterclockwise)
else if $s_1(t) > 0.706$ **then**
  turn in direction 0 (i.e., counterclockwise)
else
  drive straight forward

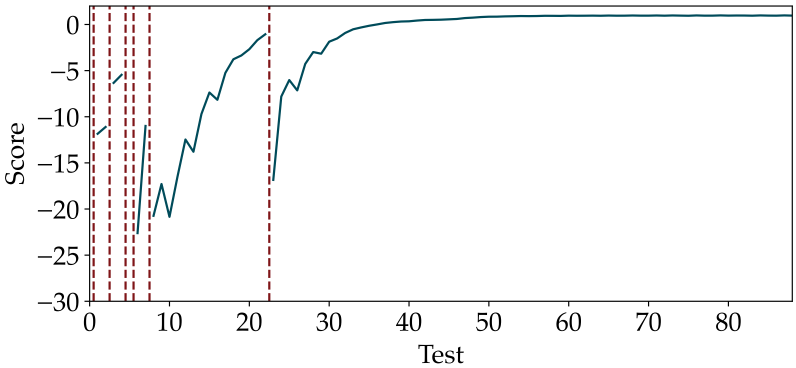


Figure A.4: Score of the predictor ANN over the learning run using machine learning-based minimize surprise resulting in the reactive behavior visualized in Fig. A.3b. The x-axis gives the test number with 750 time steps lying between two tests. The blue line gives the score, dashed red lines mark adaptations of the reactive behavior. Discontinuities in the curve indicate the reset of the predictor ANN (i.e., weights are randomly re-initialized). When there is no curve visible between two dashed red lines, the predictor performed poorly (i.e., score less than −25.0) causing adaptation (adaptation condition 1).

sensor s_4 and the second sensor from left s_1 (cf. Fig. 10.5) trigger a counterclockwise turn when detecting the arena walls or the obstacle.

As example, Fig. A.4 shows the course of the score over the learning run resulting in the wall following behavior visualized in Fig. A.3b. The reactive behavior is adapted six times until a score above 0.95 is reached and none of the adaptation conditions are met. In the shown run, adaptation is triggered three times by the interference of the hardware protection (condition 2), two times by test scores below −25.0 (condition 1), and one time by 15 negative scores in a row (condition 4). This is representative for all runs: conditions 1 and 2 are frequently causing adaptations while conditions three and four are only rarely met.

A.3 Discussion and Conclusion

Our minimize surprise approach using simple machine learning methods allows to generate robot controllers (i.e., the actor) as a by-product while training a predictor ANN using self-supervised learning. But the simplicity of our presented setup does not guarantee that it can be easily applied to other scenarios. On the contrary, preliminary experiments in a maze and a line following scenario did not lead to promising results yet. This is probably due to the simple structure of the reactive behavior, which serves as the actor here, limiting the possible behaviors. A potential solution is to increase the complexity of the reactive behavior or even to switch to an ANN as before. But this also increases the number of parameters that need to be adapted and thus may complicate the learning process or make it infeasible. Another critical point are the conditions causing the adaptation of the actor. We defined these conditions based on preliminary experiments and thus cannot guarantee that they are equally suitable for other scenarios.

However, the biggest challenge remains the application of a machine learning-based minimize surprise approach to multi-robot setups, since multi-agent learning is known to be difficult (cf. also Sec. 2.3.1). Training actor-predictor pairs in a homogeneous swarm may require a central coordinator gathering the training data and handling training of the predictor ANN and the adaptation of the actor. Also, the adaptation conditions have to be modified for the multi-robot setting. Evaluating the conditions for each robot individually may cause adaptations even when only a single robot performs poorly while determining suitable thresholds for conditions taking multiple robots into account may be challenging. Furthermore, the environment gets more complex when adding several robots, which may in turn require longer training phases. Thus, a machine learning-based minimize surprise approach requires a very careful configuration of the learning strategy. On the contrary, evolutionary algorithms enable us to optimize the actor-predictor pairs out of the box while being competitive to standard machine learning methods [17, 18].

Publications B

During my doctorate, I published the following peer-reviewed journal articles, conference papers, and workshop papers. The results of these papers, excluding the paper marked with '*', and other findings are presented in this thesis. References to the articles and papers are included in the relevant sections.

Journal Articles

- **Tanja Katharina Kaiser** and Heiko Hamann. 'Innate Motivation for Robot Swarms by Minimizing Surprise: From Simple Simulations to Real-world Experiments'. In: *IEEE Transactions on Robotics*, 38(6), pp. 3582-3601, 2022. DOI: 10.1109/TRO.2022.3181004
- **Tanja Katharina Kaiser** and Heiko Hamann. 'Engineered Self-organization for Resilient Robot Self-assembly with Minimal Surprise'. In: *Robotics and Autonomous Systems* 122, 2019, p. 103293. DOI: 10.1016/j.robot.2019.103293
- Mary Katherine Heinrich, Mohammad Divband Soorati, **Tanja Katharina Kaiser**, Mostafa Wahby, and Heiko Hamann. 'Swarm Robotics: Robustness, Scalability, and Self-X Features in Industrial Applications'. In: *it - Information Technology*, 61(4), 2019, pp. 159-167, DOI: 10.1515/itit-2019-0003

Conference Papers

- **Tanja Katharina Kaiser** and Heiko Hamann. 'Minimize Surprise MAP-Elites: A Task-Independent MAP-Elites Variant for Swarms'. In: *Genetic and Evolutionary Computation Conference Companion*. GECCO'22 Companion. [Poster Paper]. Boston, MA, USA: ACM, 2022. DOI: 10.1145/3520304.3528773
- * **Tanja Katharina Kaiser**, Marian Johannes Begemann, Tavia Plattenteich, Lars Schilling, Georg Schildbach, and Heiko Hamann. 'ROS2SWARM - A ROS 2 Package for Swarm Robot Behaviors'. In: *2022 International Conference on Robotics and Automation (ICRA)*, Philadelphia, PA, USA, 2022, pp. 6875-6881. DOI: 10.1109/ICRA46639.2022.9812417
- **Tanja Katharina Kaiser**, Christine Lang, Florian Andreas Marwitz, Christian Charles, Sven Dreier, Julian Petzold, Max Ferdinand Hannwald, Marian Johannes Begemann, and Heiko Hamann. 'An Innate Motivation to Tidy Your Room: Online Onboard Evolution of Manipulation Behaviors in a Robot Swarm'. In: *Distributed Autonomous Robotic Systems*. Ed. by Fumitoshi Matsuno,

Shun-ichi Azuma, and Masahito Yamamoto. Cham: Springer International Publishing, 2022, pp.190-201. DOI: 10.1007/978-3-030-92790-5_15

- **Tanja Katharina Kaiser** and Heiko Hamann. 'Evolution of Diverse Swarm Behaviors with Minimal Surprise'. In: *Artificial Life Conference Proceedings* 32, 2020, pp. 384-392. DOI: 10.1162/isal_a_00266
- **Tanja Katharina Kaiser** and Heiko Hamann. 'Diversity in Swarm Robotics with Task-independent Behavior Characterization'. In: *Genetic and Evolutionary Computation Conference Companion*. GECCO'20 Companion. [Extended Abstract]. Cancun, Mexico: ACM, 2020. DOI: 10.1145/3377929.3389949
- **Tanja Katharina Kaiser** and Heiko Hamann. 'Self-assembly in Patterns with Minimal Surprise: Engineered Self-organization and Adaptation to the Environment'. In: *Distributed Autonomous Robotic Systems*. Ed. by Nikolaus Corell, Mac Schwager, and Michael Otte. Cham: Springer International Publishing, 2019, pp. 183-195. DOI: 10.1007/978-3-030-05816-6_13

Workshop Papers

- **Tanja Katharina Kaiser** and Heiko Hamann. 'Self-organized Construction by Minimal Surprise'. In: *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. 2019, pp. 213-218. DOI: 10.1109/FAS-W.2019.00057

Supervised Bachelor and Master Theses and Student Projects C

During my doctorate, I co-supervised several bachelor and master theses as well as student projects, some of which contributed to the results presented in this thesis. References to chapters indicate to which parts of the thesis the respective student project has contributed.

Bachelor Theses

- Tavia Plattenteich. *Extending the ROS 2 Swarm Behavior Package to the TurtleBot3 Burger and the Jackal UAV*. November 2021.
- Max Hannawald. *Swarm Construction with Minimize Surprise*. December 2020. Ch. 11.
- Marian Begemann. *Development of a Swarm Behavior ROS 2 Package for the TurtleBot3 Waffle Pi*. December 2020.
- Larisa Brauna. *Parameter Tuning for Minimize Surprise*. October 2020. Ch. 4
- Christian Charles. *Implementation of Blind Bulldozing with Minimize Surprise*. June 2020. Ch. 11.
- Christopher Kluth. *Adaptation of the Minimal Surprise Approach for Real Robots*. November 2018. App. A.

Master Theses

- Nils Bischoff (Technische Universität Berlin). *An Embodied Evolution Approach to Minimize Surprise*. July 2022.
- Tristan Potten. *Evolving Robot Swarms in Collective Perception with Minimize Surprise*. December 2021. Ch. 8.
- Christopher Kluth. *Evolution of Complex Behaviors in Dynamic Environments by Minimizing Surprise*. June 2021. Ch. 7.

Student Projects

- Christian Charles. *Proof of Concept: Extension of ROS2swarm to Drones*. Master project. 2022.
- Cassie Yuan (University of Texas at Austin, USA). *Conducting Minimize Surprise Experiments with a Swarm of Real Thymio II Robots*. DAAD RISE Germany. May - August 2021. Chs. 10 and 11.
- Vincent Jansen and Daniel Tidde. *Implementation of Minimalist Flocking and the Majority Rule for the ROS 2 Swarm Package*. Bachelor project. April 2021.
- Christopher Kluth and Moritz Hoffmann. *Implementation of Schmidhuber's World Models on a Real Thymio II Robot*. Master project. December 2020.

- Isabella Grossart. *Automated Sewing Pattern Drawing with Thymio II Robots*. Bachelor project. October 2020.
- Steffen Fleischmann. *Testing ROS 2 on TurtleBot3 Waffle Pi*. Bachelor project. October 2020.
- Jonas Schroerschwarz and Fabian Domberg. *Set-up of Simulations for the Mobile Robots Tutorials*. Bachelor project. October 2020.
- Florian Marwitz and Christine Lang. *(1+1)-Evolution with Minimize Surprise on Thymio II Robots*. Bachelor project. February 2020. Chs. 10 and 11.
- Till Aust, Christian Charles, and Max Hannawald. *Implementation of a Communication Infrastructure for Evolution on a Swarm of Real Thymio II Robots*. Bachelor project. October 2019. Chs. 10 and 11.
- Juhee Park (Ohio State University, USA). *Evolution of Robot Behaviors with Machine Learning-based Minimize Surprise in a Realistic Simulator*. DAAD RISE Germany. May - August 2019. App. A.
- Till Aust, Christian Charles, and Max Hannawald. *Comparison of Variants of Minimize Surprise to Q-Learning*. Bachelor project. April 2019.

Acknowledgements

Five years of joyful research, unforeseen obstacles, hard work, and commitment culminated in this dissertation. I could not have accomplished it without the help and support of many amazing people.

First and foremost, I thank Heiko Hamann who gave me the opportunity to work with him and to become part of his research group. Without limitations, he offered and still offers great support and mentoring, helping me to become a better researcher and scientist.

I also owe special thanks to Sanaz Mostaghim and Philipp Rostalski who without hesitation agreed to become my second referee and the head of my doctoral committee, respectively.

My colleagues at the Institute of Computer Engineering, many of whom also became close friends, have earned my sincere gratitude. Christopher, Eddy, Julian K., Julian P., Kristian, Mary Katherine, Mohammad, Mostafa, Sven, Till and many others helped me overcome the challenges that arose along the way and have made the last five years even more enjoyable. Thank you.

During my time at the Institute of Computer Engineering, I had the chance to co-supervise many student projects and theses. I thank Cassie, Christian, Christine, Christopher, Daniel, Fabian, Florian, Isabella, Jonas, Juhee, Larisa, Marian, Max, Moritz, Nils, Steffen, Tavia, Tristan, and Vincent not only for contributing to my research but also for growing together through our challenges.

Participation in international conferences is one of the great perks of being a researcher. Thank you to all the anonymous reviewers who provided valuable feedback, all the senior scientists who gave me precious advice, and all the junior scientists with whom I not only discussed research but also explored new cities.

There are three people to whom I owe it in large part that I found my way into swarm robotics research. First, my master thesis supervisor, Christopher, had the right intuition about which research topic would fascinate me the most and supported me in finding my way into academia. Second and third, my fellow students and friends Michael and Dennis greatly supported me during our bachelor studies and inspired me to explore many topics in endless discussions at our kitchen table. Many thanks!

Last but not least, thanks to my family and friends who supported me in manifold ways during this journey. Thank you, Priscilla, Martin, Tristan, Susanna, Konstantin, Moritz, Joscha, Petra, Richard, Lucas, and many more for everything.

Bibliography

[1] Sangbae Kim, Cecilia Laschi, and Barry Trimmer. 'Soft Robotics: A Bioinspired Evolution in Robotics'. In: *Trends in Biotechnology* 31.5 (2013), pp. 287–294. DOI: 10.1016/j.tibtech.2013.03.002.

[2] Ronald C. Arkin. *Behavior-based Robotics*. MIT Press, 1998.

[3] Duncan E. Jackson, Mike Holcombe, and Francis L. W. Ratnieks. 'Trail Geometry Gives Polarity to Ant Foraging Networks'. In: *Nature* 432.7019 (2004), pp. 907–909. DOI: 10.1038/nature03105.

[4] Nigel R. Franks, Andy Wilby, Bernard Walter Silverman, and Chris Tofts. 'Self-Organizing Nest Construction in Ants: Sophisticated Building by Blind Bulldozing'. In: *Animal Behaviour* 44 (1992), pp. 357–375. DOI: 10.1016/0003-3472(92)90041-7.

[5] John T. Emlen. 'Flocking Behavior in Birds'. In: *The Auk* 69.2 (Apr. 1952), pp. 160–170. DOI: 10.2307/4081266.

[6] Heiko Hamann. *Swarm Robotics: A Formal Approach*. Cham: Springer International Publishing, 2018. DOI: 10.1007/978-3-319-74528-2.

[7] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. 'Swarm Robotics: A Review from the Swarm Engineering Perspective'. In: *Swarm Intelligence* 7.1 (2013), pp. 1–41. DOI: 10.1007/s11721-012-0075-2.

[8] Vito Trianni. *Evolutionary Swarm Robotics - Evolving Self-Organising Behaviours in Groups of Autonomous Robots*. Vol. 108. Studies in Computational Intelligence. Berlin, Germany: Springer, 2008. DOI: 10.1007/978-3-540-77612-3.

[9] Eric Bonabeau, Marco Dorigo, and Guy Théraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[10] Gianpiero Francesca and Mauro Birattari. 'Automatic Design of Robot Swarms: Achievements and Challenges'. In: *Frontiers in Robotics and AI* 3 (2016), p. 29. DOI: 10.3389/frobt.2016.00029.

[11] David Silver et al. 'A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go Through Self-Play'. In: *Science* 362.6419 (2018), pp. 1140–1144. DOI: 10.1126/science.aar6404.

[12] Jayesh K. Gupta, Maxim Egorov, and Mykel Kochenderfer. 'Cooperative Multi-Agent Control Using Deep Reinforcement Learning'. In: *International Conference on Autonomous Agents and Multiagent Systems*. Springer. 2017, pp. 66–83. DOI: 10.1007/978-3-319-71682-4_5.

[13] Stephane Doncieux, Nicolas Bredeche, Jean-Baptiste Mouret, and Agoston E. Eiben. 'Evolutionary Robotics: What, Why, and Where to'. In: *Frontiers in Robotics and AI* 2 (2015), p. 4. DOI: 10.3389/frobt.2015.00004.

[14] Stefano Nolfi and Dario Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, 2000.

[15] Josh C. Bongard. 'Evolutionary Robotics'. In: *Communications of the ACM* 56.8 (2013), pp. 74–83. DOI: 10.1145/2493883.

[16] Agoston E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Berlin, Heidelberg: Springer, 2015. DOI: 10.1007/978-3-662-44874-8.

[17] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. *Evolution Strategies as a Scalable Alternative to Reinforcement Learning*. 2017. arXiv: 1703.03864 [stat.ML].

[18] Sebastian Risi and Kenneth O. Stanley. 'Deep Neuroevolution of Recurrent and Discrete World Models'. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '19. Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 456–462. DOI: 10.1145/3321707.3321817.

[19] Stephane Doncieux and Jean-Baptiste Mouret. 'Beyond Black-Box Optimization: A Review of Selective Pressures for Evolutionary Robotics'. In: *Evolutionary Intelligence* 7.2 (2014), pp. 71–93. DOI: 10.1007/s12065-014-0110-x.

[20] Andrew L. Nelson, Gregory J. Barlow, and Lefteris Doitsidis. 'Fitness Functions in Evolutionary Robotics: A Survey and Analysis'. In: *Robotics and Autonomous Systems* 57.4 (2009), pp. 345–370. DOI: 10.1016/j.robot.2008.09.009.

[21] Fernando Silva, Miguel Duarte, Luís Correia, Sancho Moura Oliveira, and Anders Lyhne Christensen. 'Open Issues in Evolutionary Robotics'. In: *Evolutionary Computation* 24.2 (June 2016), pp. 205–236. DOI: 10.1162/EVCO_a_00172.

[22] Wendy Aguilar, Guillermo Santamaría-Bonfil, Tom Froese, and Carlos Gershenson. 'The Past, Present, and Future of Artificial Life'. In: *Frontiers in Robotics and AI* 1 (2014). DOI: 10.3389/frobt.2014.00008.

[23] Stephane Doncieux and Jean-Baptiste Mouret. 'Behavioral Diversity Measures for Evolutionary Robotics'. In: *IEEE Congress on Evolutionary Computation*. 2010, pp. 1–8. DOI: 10.1109/CEC.2010.5586100.

[24] Joel Lehman and Kenneth O. Stanley. 'Exploiting Open-endedness to Solve Problems Through the Search for Novelty'. In: *Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems*. Ed. by S. Bullock, J. Noble, R. Watson, and M. A. Bedau. MIT Press, 2008, pp. 329–336.

[25] Valerio Sperati, Vito Trianni, and Stefano Nolfi. 'Evolving Coordinated Group Behaviours Through Maximisation of Mean Mutual Information'. In: *Swarm Intelligence* 2.2 (2008), pp. 73–95. DOI: 10.1007/s11721-008-0017-1.

[26] Magnus Egerstedt, Jonathan N. Pauli, Gennaro Notomista, and Seth Hutchinson. 'Robot Ecology: Constraint-based Control Design for Long Duration Autonomy'. In: *Annual Reviews in Control* 46 (2018), pp. 1–7. DOI: 10.1016/j.arcontrol.2018.09.006.

[27] Nicolas Bredeche, Jean-Marc Montanier, Wenguo Liu, and Alan F. T. Winfield. 'Environment-Driven Distributed Evolutionary Adaptation in a Population of Autonomous Robotic Agents'. In: *Mathematical and Computer Modelling of Dynamical Systems* 18.1 (2012), pp. 101–129. DOI: 10.1080/13873954.2011.601425.

[28] Heiko Hamann. 'Evolution of Collective Behaviors by Minimizing Surprise'. In: *14th International Conference on the Synthesis and Simulation of Living Systems (ALIFE 2014)*. Ed. by Hiroki Sayama, John Rieffel, Sebastian Risi, René Doursat, and Hod Lipson. MIT Press, 2014, pp. 344–351.

[29] Andy Clark. *Surfing Uncertainty: Prediction, Action, and the Embodied Mind*. Oxford University Press, 2015.

[30] Peter Dayan, Geoffrey E. Hinton, Radford M. Neal, and Richard S. Zemel. 'The Helmholtz Machine'. In: *Neural Computation* 7.5 (1995), pp. 889–904.

[31] Karl Friston. 'The Free-Energy Principle: A Unified Brain Theory?' In: *Nature Reviews Neuroscience* 11.2 (2010), pp. 127–138. DOI: 10.1038/nrn2787.

[32] Léo Pio-Lopez, Ange Nizard, Karl Friston, and Giovanni Pezzulo. 'Active Inference and Robot Control: A Case Study'. In: *Journal of The Royal Society Interface* 13.122 (2016), p. 20160616. DOI: 10.1098/rsif.2016.0616.

[33] Toon Van de Maele, Tim Verbelen, Ozan Çatal, Cedric De Boom, and Bart Dhoedt. 'Active Vision for Robot Manipulators Using the Free Energy Principle'. In: *Frontiers in Neurorobotics* 15 (2021), p. 14. DOI: 10.3389/fnbot.2021.642780.

[34] Guillermo Oliver, Pablo Lanillos, and Gordon Cheng. 'An Empirical Study of Active Inference on a Humanoid Robot'. In: *IEEE Transactions on Cognitive and Developmental Systems* (2021), p. 1. DOI: 10.1109/TCDS.2021.3049907.

[35] Daniel Ari Friedman, Alec Tschantz, Maxwell J. D. Ramstead, Karl Friston, and Axel Constant. 'Active Inferants: An Active Inference Framework for Ant Colony Behavior'. In: *Frontiers in Behavioral Neuroscience* 15 (2021), p. 126. DOI: 10.3389/fnbeh.2021.647732.

[36] Rafael Kaufmann, Pranav Gupta, and Jacob Taylor. 'An Active Inference Model of Collective Intelligence'. In: *Entropy* 23.7 (2021). DOI: 10.3390/e23070830.

[37] Richard Borkowski and Heiko Hamann. 'Evolving Robot Swarm Behaviors by Minimizing Surprise: Results of Simulations in 2-d on a Torus'. In: *Evolving Collective Behaviors in Robotics Workshop, GECCO 2017*. ACM, 2017, pp. 1679–1680. DOI: 10.1145/3067695.3082548.

[38] Tanja Katharina Kaiser and Heiko Hamann. 'Self-Assembly in Patterns with Minimal Surprise: Engineered Self-Organization and Adaptation to the Environment'. In: *Distributed Autonomous Robotic Systems*. Ed. by Nikolaus Correll, Mac Schwager, and Michael Otte. Cham: Springer International Publishing, 2019, pp. 183–195. DOI: 10.1007/978-3-030-05816-6_13. ©Springer Nature Switzerland AG 2019.

[39] Tanja Katharina Kaiser and Heiko Hamann. 'Engineered Self-Organization for Resilient Robot Self-Assembly with Minimal Surprise'. In: *Robotics and Autonomous Systems* 122 (2019), p. 103293. DOI: 10.1016/j.robot.2019.103293. ©2019 Elsevier B.V.

[40] Tanja Katharina Kaiser and Heiko Hamann. 'Diversity in Swarm Robotics with Task-Independent Behavior Characterization'. In: *Genetic and Evolutionary Computation Conference Companion*. GECCO'20 Companion. [Extended Abstract]. Cancun, Mexico: ACM, 2020. DOI: 10.1145/3377929.3389949.

[41] Tanja Katharina Kaiser and Heiko Hamann. 'Evolution of Diverse Swarm Behaviors with Minimal Surprise'. In: *Artificial Life Conference Proceedings* 32 (2020), pp. 384–392. DOI: 10.1162/isal_a_00266. ©2020 Massachusetts Institute of Technology, CC BY 4.0.

[42] Tanja Katharina Kaiser and Heiko Hamann. *Innate Motivation for Robot Swarms by Minimizing Surprise: From Simple Simulations to Real-World Experiments*. 2022. DOI: 10.1109/TRO.2022.3181004. ©2022 IEEE.

[43] Tanja Katharina Kaiser and Heiko Hamann. *Minimize Surprise MAP-Elites: A Task-Independent MAP-Elites Variant for Swarms*. [Poster Paper]. Boston, MA, USA, 2022. DOI: 10.1145/3520304.3528773.

[44] Jean-Baptiste Mouret and Jeff Clune. *Illuminating Search Spaces by Mapping Elites*. 2015. arXiv: 1504.04909 [cs.AI].

[45] Sean Lim, Shiyi Wang, Barry Lennox, and Farshad Arvin. 'BeeGround - An Open-Source Simulation Platform for Large-Scale Swarm Robotics Applications'. In: *2021 7th International Conference on Automation, Robotics and Applications (ICARA)*. 2021, pp. 75–79. DOI: 10.1109/ICARA51699.2021.9376494.

[46] Olivier Michel. 'Cyberbotics Ltd. Webots™: Professional Mobile Robot Simulation'. In: *International Journal of Advanced Robotic Systems* 1.1 (2004), p. 5. DOI: 10.5772/5618.

[47] Tanja Katharina Kaiser and Heiko Hamann. 'Self-Organized Construction by Minimal Surprise'. In: *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. 2019, pp. 213–218. DOI: 10.1109/FAS-W.2019.00057. ©2019 IEEE.

[48] Tanja Katharina Kaiser, Christine Lang, Florian Andreas Marwitz, Christian Charles, Sven Dreier, Julian Petzold, Max Ferdinand Hannawald, Marian Johannes Begemann, and Heiko Hamann. 'An Innate Motivation to Tidy Your Room: Online Onboard Evolution of Manipulation Behaviors in a Robot Swarm'. In: *Distributed Autonomous Robotic Systems*. Ed. by Fumitoshi Matsuno, Shun-ichi Azuma, and Masahito Yamamoto. Cham: Springer International Publishing, 2022, pp. 190–201. DOI: 10.1007/978-3-030-92790-5_15. Reproduced with permission from Springer Nature.

[49] Oxford University Press. *Definition of Robot Noun from the Oxford Advanced Learner's Dictionary*. 2021. URL: https://www.oxfordlearnersdictionaries.com/definition/english/robot (visited on 05/21/2021).

[50] Adrienne Mayor. *Gods and Robots: Myths, Machines, and Ancient Dreams of Technology*. Princeton University Press, 2018.

[51] Bruno Siciliano and Oussama Khatib, eds. *Springer Handbook of Robotics*. Cham: Springer International Publishing, 2016. DOI: 10.1007/978-3-319-32552-1.

[52] Pinar Urhal, Andrew Weightman, Carl Diver, and Paulo Bartolo. 'Robot Assisted Additive Manufacturing: A Review'. In: *Robotics and Computer-Integrated Manufacturing* 59 (2019), pp. 335–345. DOI: 10.1016/j.rcim.2019.05.005.

[53] Matteo Lavit Nicora, Roberto Ambrosetti, Gloria J. Wiens, and Irene Fassi. 'Human-Robot Collaboration in Smart Manufacturing: Robot Reactive Behavior Intelligence'. In: *Journal of Manufacturing Science and Engineering* 143.3 (Dec. 2020). 031009. DOI: 10.1115/1.4048950.

[54] Mikkel Rath Pedersen, Lazaros Nalpantidis, Rasmus Skovgaard Andersen, Casper Schou, Simon Bøgh, Volker Krüger, and Ole Madsen. 'Robot Skills for Manufacturing: From Concept to Industrial Deployment'. In: *Robotics and Computer-Integrated Manufacturing* 37 (2016), pp. 282–291. DOI: 10.1016/j.rcim.2015.04.002.

[55] Bo Preising, T. C. Hsia, and Brent Mittelstadt. 'A Literature Review: Robots in Medicine'. In: *IEEE Engineering in Medicine and Biology Magazine* 10.2 (1991), pp. 13–22. DOI: 10.1109/51.82001.

[56] Brian Davies. 'A Review of Robotics in Surgery'. In: *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine* 214.1 (2000), pp. 129–140. DOI: 10.1243/0954411001535309.

[57] Fernando Soto, Jie Wang, Rajib Ahmed, and Utkan Demirci. 'Medical Micro/Nanorobots in Precision Medicine'. In: *Advanced Science* 7.21 (2020), p. 2002203. DOI: 10.1002/advs.202002203.

[58] Klaus Schilling and Christoph Jungius. 'Mobile Robots for Planetary Exploration'. In: *Control Engineering Practice* 4.4 (1996), pp. 513–524. DOI: 10.1016/0967-0661(96)00034-2.

[59] Angel Flores-Abad, Ou Ma, Khanh Pham, and Steve Ulrich. 'A Review of Space Robotics Technologies for On-Orbit Servicing'. In: *Progress in Aerospace Sciences* 68 (2014), pp. 1–26. DOI: 10.1016/j.paerosci.2014.03.002.

[60] Jerzy Sąsiadek. 'Space Robotics and its Challenges'. In: *Aerospace Robotics: Selected Papers from I Conference on Robotics in Aeronautics and Astronautics*. Ed. by Jerzy Sąsiadek. Berlin, Heidelberg: Springer, 2013, pp. 1–8. DOI: 10.1007/978-3-642-34020-8_1.

[61] Fred Sistler. 'Robotics and Intelligent Machines in Agriculture'. In: *IEEE Journal on Robotics and Automation* 3.1 (1987), pp. 3–6. DOI: 10.1109/JRA.1987.1087074.

[62] David Ball, Patrick Ross, Andrew English, Tim Patten, Ben Upcroft, Robert Fitch, Salah Sukkarieh, Gordon Wyeth, and Peter Corke. 'Robotics for Sustainable Broad-Acre Agriculture'. In: *Field and Service Robotics: Results of the 9th International Conference*. Ed. by Luis Mejias, Peter Corke, and Jonathan Roberts. Cham: Springer International Publishing, 2015, pp. 439–453. DOI: 10.1007/978-3-319-07488-7_30.

[63] Gert Kootstra, Xin Wang, Pieter M. Blok, Jochen Hemming, and Eldert Van Henten. 'Selective Harvesting Robotics: Current Research, Trends, and Future Directions'. In: *Current Robotics Reports* (2021), pp. 1–10. DOI: 10.1007/s43154-020-00034-1.

[64] Rudolph Triebel et al. 'SPENCER: A Socially Aware Service Robot for Passenger Guidance and Help in Busy Airports'. In: *Field and Service Robotics: Results of the 10th International Conference*. Ed. by David S. Wettergreen and Timothy D. Barfoot. Cham: Springer International Publishing, 2016, pp. 607–622. DOI: 10.1007/978-3-319-27702-8_40.

[65] Stefanie Paluch, Jochen Wirtz, and Werner H. Kunz. 'Service Robots and the Future of Services'. In: *Marketing Weiterdenken: Zukunftspfade für eine marktorientierte Unternehmensführung*. Ed. by Manfred Bruhn, Christoph Burmann, and Manfred Kirchgeorg. Wiesbaden: Springer Fachmedien, 2020, pp. 423–435. DOI: 10.1007/978-3-658-31563-4_21.

[66] Katharina Blöcher and Rainer Alt. 'AI and Robotics in the European Restaurant Sector: Assessing Potentials for Process Innovation in a High-Contact Service Industry'. In: *Electronic Markets* (2020), pp. 1–23. DOI: 10.1007/s12525-020-00443-2.

[67] Mordechai Ben-Ari and Francesco Mondada. *Elements of Robotics*. Cham: Springer International Publishing, 2018, p. 308. DOI: 10.1007/978-3-319-62533-1.

[68] Minopu Asada. 'Robotics'. In: *Encyclopedia of Information Systems*. Ed. by Hossein Bidgoli. New York: Elsevier, 2003, pp. 707–722. DOI: 10.1016/B0-12-227240-4/00150-7.

[69] Nikolaus Correll, Bradley Hayes, Christoffer Heckman, and Alessandro Roncone. *Introduction to Autonomous Robots: Mechanisms, Sensors, Actuators, and Algorithms*. MIT Press, Cambridge, MA, 2022.

[70] Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. 'A Review of Mobile Robots: Concepts, Methods, Theoretical Framework, and Applications'. In: *International Journal of Advanced Robotic Systems* 16.2 (2019), p. 1729881419839596. DOI: 10.1177/1729881419839596.

[71] Justin E. Manley. 'Unmanned Surface Vehicles, 15 Years of Development'. In: *OCEANS 2008*. 2008, pp. 1–4. DOI: 10.1109/OCEANS.2008.5152052.

[72] Wei Wang, Banti Gheneti, Luis A. Mateos, Fabio Duarte, Carlo Ratti, and Daniela Rus. 'Roboat: An Autonomous Surface Vehicle for Urban Waterways'. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019, pp. 6340–6347. DOI: 10.1109/IROS40897.2019.8968131.

[73] Christoph Osterloh, Thilo Pionteck, and Erik Maehle. 'MONSUN II: A Small and Inexpensive AUV for Underwater Swarms'. In: *ROBOTIK 2012; 7th German Conference on Robotics*. 2012, pp. 1–6.

[74] Florian Berlinger, Melvin Gauci, and Radhika Nagpal. 'Implicit Coordination for 3D Underwater Collective Behaviors in a Fish-Inspired Robot Swarm'. In: *Science Robotics* 6.50 (2021). DOI: 10.1126/scirobotics.abd8668.

[75] Giacomo Picardi, Mrudul Chellapurath, Saverio Iacoponi, Sergio Stefanni, Cecilia Laschi, and Marcello Calisti. 'Bioinspired Underwater Legged Robot for Seabed Exploration with Low Environmental Disturbance'. In: *Science Robotics* 5.42 (2020). DOI: 10.1126/scirobotics.aaz1012.

[76] Chuang Li et al. 'Fast and Programmable Locomotion of Hydrogel-Metal Hybrids Under Light and Magnetic fields'. In: *Science Robotics* 5.49 (2020). DOI: 10.1126/scirobotics.abb9822.

[77] Fanny Riedo, Morgane Chevalier, Stéphane Magnenat, and Francesco Mondada. 'Thymio II, a Robot That Grows Wiser with Children'. In: *2013 IEEE Workshop on Advanced Robotics and its Social Impacts*. IEEE. 2013, pp. 187–193. DOI: 10.1109/ARSO.2013.6705527.

[78] Francesco Mondada et al. 'The e-puck, a Robot Designed for Education in Engineering'. In: *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*. Vol. 1. IPCB: Instituto Politécnico de Castelo Branco. 2009, pp. 59–65.

[79] Widodo Budiharto. 'Design of Tracked Robot with Remote Control for Surveillance'. In: *Proceedings of the 2014 International Conference on Advanced Mechatronic Systems*. 2014, pp. 342–346. DOI: 10.1109/ICAMechS.2014.6911568.

[80] Christian Ducros, Gérard Hauser, Najib Mahjoubi, Philippe Girones, Laurence Boisset, Antoine Sorin, Eric Jonquet, Jean Michel Falciola, and Albert Benhamou. 'RICA: A Tracked Robot for Sampling and Radiological Characterization in the Nuclear Field'. In: *Journal of Field Robotics* 34.3 (2017), pp. 583–599. DOI: 10.1002/rob.21650.

[81] Philippe Sardain, Mostafa Rostami, and Guy Bessonnet. 'An Anthropomorphic Biped Robot: Dynamic Concepts and Technological Design'. In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 28.6 (1998), pp. 823–838. DOI: 10.1109/3468.725353.

[82] Fumiya Iida and Rolf Pfeifer. 'Cheap Rapid Locomotion of a Quadruped Robot: Self-Stabilization of Bounding Gait'. In: *Intelligent autonomous systems*. Vol. 8. IOS Press Amsterdam, The Netherlands. 2004, pp. 642–649.

[83] Uluc Saranli, Martin Buehler, and Daniel E. Koditschek. 'RHex: A Simple and Highly Mobile Hexapod Robot'. In: *The International Journal of Robotics Research* 20.7 (2001), pp. 616–631. DOI: 10.1177/02783640122067570.

[84] Hironori Adachi, Noriho Koyachi, Tatsuo Arai, A. Shimiza, and Yashiroh Nogami. 'Mechanism and Control of a Leg-Wheel Hybrid Mobile Robot'. In: *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No.99CH36289)*. Vol. 3. 1999, pp. 1792–1797. DOI: 10.1109/IROS.1999.811738.

[85] Zirong Luo, Jianzhong Shang, Guowu Wei, and Lei Ren. 'A Reconfigurable Hybrid Wheel-track Mobile Robot Based on Watt II Six-bar Linkage'. In: *Mechanism and Machine Theory* 128 (2018), pp. 16–32. DOI: 10.1016/j.mechmachtheory.2018.04.020.

[86] Isaiah Brand, Josh Roy, Aaron Ray, John Oberlin, and Stefanie Oberlix. 'PiDrone: An Autonomous Educational Drone Using Raspberry Pi and Python'. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 1–7. DOI: 10.1109/IROS.2018.8593943.

[87] Alberto Elfes, S. Siqueira Bueno, Marcel Bergerman, and Josue Guimaraes Ramos. 'A Semi-Autonomous Robotic Airship for Environmental Monitoring Missions'. In: *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*. Vol. 4. 1998, pp. 3449–3455. DOI: 10.1109/ROBOT.1998.680971.

[88] Junzhi Yu, Rui Ding, Qinghai Yang, Min Tan, Weibing Wang, and Jianwei Zhang. 'On a Bio-Inspired Amphibious Robot Capable of Multimodal Motion'. In: *IEEE/ASME Transactions on Mechatronics* 17.5 (2012), pp. 847–856. DOI: 10.1109/TMECH.2011.2132732.

[89] Alessandro Crespi, Konstantinos Karakasiliotis, André Guignard, and Auke Jan Ijspeert. 'Salamandra Robotica II: An Amphibious Robot to Study Salamander-like Swimming and Walking Gaits'. In: *IEEE Transactions on Robotics* 29.2 (2013), pp. 308–320. DOI: 10.1109/TRO.2012.2234311.

[90] E. Potemkin, P. Astafurov, A. Osipov, M. Malenkov, V. Mishkinyuk, and P. Sologub. 'Remote-Controlled Robots for Repair and Recovery in the Zones of High Radiation Levels'. In: *Proceedings 1992 IEEE International Conference on Robotics and Automation*. Los Alamitos, CA, USA: IEEE Computer Society, 1992, pp. 80–82. DOI: 10.1109/ROBOT.1992.220331.

[91] Evan Ackerman. *Everything You Need to Know About NASA's Perseverance Rover Landing on Mars*. 2021. URL: https://spectrum.ieee.org/automaton/aerospace/robotic-exploration/nasa-perseverance-rover-landing-on-mars-overview (visited on 06/11/2021).

[92] Robin R. Murphy. *Introduction to AI Robotics*. 1st. Cambridge, MA, USA: MIT Press, 2000.

[93] Hugh Durrant-Whyte and Tim Bailey. 'Simultaneous Localization and Mapping: Part I'. In: *IEEE Robotics Automation Magazine* 13.2 (2006), pp. 99–110. DOI: 10.1109/MRA.2006.1638022.

[94] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. 'Markov Localization for Mobile Robots in Dynamic Environments'. In: *Journal of Artificial Intelligence Research* 11 (1999), pp. 391–427.

[95] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. PEARSON, 2016.

[96] Rodney A. Brooks. 'Intelligence Without Representation'. In: *Artificial Intelligence* 47.1 (1991), pp. 139–159. DOI: 10.1016/0004-3702(91)90053-M.

[97] Maja J. Matarić. 'Behavior-Based Control: Main Properties and Implications'. In: *Proceedings, IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems*. 1992, pp. 46–54.

[98] Apoorva Kumar, Rahul Gautam, and Rahul Kala. 'Motion Planning for a Chain of Mobile Robots Using A* and Potential Field'. In: *Robotics* 7.2 (2018). DOI: 10.3390/robotics7020020.

[99] Marco Dorigo and Mauro Birattari. 'Swarm Intelligence'. In: *Scholarpedia* 2.9 (2007). revision #138640, p. 1462. DOI: 10.4249/scholarpedia.1462.

[100] Marco Dorigo, Guy Theraulaz, and Vito Trianni. 'Swarm Robotics: Past, Present, and Future [Point of View]'. In: *Proceedings of the IEEE* 109.7 (2021), pp. 1152–1165. DOI: 10.1109/JPROC.2021.3072740.

[101] Guang-Zhong Yang et al. 'The Grand Challenges of Science Robotics'. In: *Science Robotics* 3.14 (2018). DOI: 10.1126/scirobotics.aar7650.

[102] Erol Şahin. 'Swarm Robotics: From Sources of Inspiration to Domains of Application'. In: *Swarm Robotics*. Ed. by Erol Şahin and William M. Spears. Berlin, Heidelberg: Springer, 2005, pp. 10–20.

[103] Gerardo Beni. 'From Swarm Intelligence to Swarm Robotics'. In: *Swarm Robotics*. Ed. by Erol Şahin and William M. Spears. Berlin, Heidelberg: Springer, 2005, pp. 1–9.

[104] Iñaki Navarro and Fernando Matía. 'An Introduction to Swarm Robotics'. In: *International Scholarly Research Notices* 2013 (2013). DOI: 10.5402/2013/608164.

[105] Erol Şahin and Alan F. T. Winfield. 'Special Issue on Swarm Robotics.' In: *Swarm Intelligence* 2.2-4 (2008), pp. 69–72.

[106] Marco Dorigo and Erol Şahin. 'Guest Editorial'. In: *Autonomous Robots* 17.2–3 (Sept. 2004), pp. 111–113. DOI: 10.1023/B:AURO.0000034008.48988.2b.

[107] Scott Camazine, Jean-Louis Deneubourg, Nigel R. Franks, James Sneyd, Guy Theraulaz, and Eric Bonabeau. *Self-Organization in Biological Systems*. Princeton University Press, 2020. DOI: 10.1515/9780691212920.

[108] Tom De Wolf and Tom Holvoet. 'Emergence Versus Self-Organisation: Different Concepts but Promising When Combined'. In: *Engineering Self-Organising Systems*. Ed. by Sven A. Brueckner, Giovanna Di Marzo Serugendo, Anthony Karageorgos, and Radhika Nagpal. Berlin, Heidelberg: Springer, 2005, pp. 1–15. DOI: 10.1007/11494676_1.

[109] Heiko Hamann, Gabriele Valentini, and Marco Dorigo. 'Population Coding: A New Design Paradigm for Embodied Distributed Systems'. In: *Swarm Intelligence*. Ed. by Marco Dorigo, Mauro Birattari, Xiaodong Li, Manuel López-Ibáñez, Kazuhiro Ohkura, Carlo Pinciroli, and Thomas Stützle. Cham: Springer International Publishing, 2016, pp. 173–184. DOI: 10.1007/978-3-319-44427-7_15.

[110] Heiko Hamann, Gabriele Valentini, Yara Khaluf, and Marco Dorigo. 'Derivation of a Micro-Macro Link for Collective Decision-Making Systems'. In: *Parallel Problem Solving from Nature – PPSN XIII*. Ed. by Thomas Bartz-Beielstein, Jürgen Branke, Bogdan Filipič, and Jim Smith. Cham: Springer International Publishing, 2014, pp. 181–190. DOI: 10.1007/978-3-319-10762-2_18.

[111] Andreagiovanni Reina, Roman Miletitch, Marco Dorigo, and Vito Trianni. 'A Quantitative Micro-Macro Link for Collective Decisions: The Shortest Path Discovery/Selection Example'. In: *Swarm Intelligence* 9.2 (2015), pp. 75–102. DOI: 10.1007/s11721-015-0105-y.

[112] Michael Rubenstein, Christian Ahler, Nick Hoff, Adrian Cabrera, and Radhika Nagpal. 'Kilobot: A Low Cost Robot with Scalable Operations Designed for Collective Behaviors'. In: *Robotics and Autonomous Systems* 62.7 (2014). Reconfigurable Modular Robotics, pp. 966–975. DOI: 10.1016/j.robot.2013.08.006.

[113] Francesco Mondada, Giovanni Cosimo Pettinaro, Ivo W. Kwee, André Guignard, Luca Maria Gambardella, Dario Floreano, Stefano Nolfi, Jean-Louis Deneubourg, and Marco Dorigo. *SWARM-BOT: A Swarm of Autonomous Mobile Robots with Self-Assembling Capabilities*. 2002.

[114] Francesco Mondada, Giovanni Cosimo Pettinaro, Andre Guignard, Ivo W. Kwee, Dario Floreano, Jean-Louis Deneubourg, Stefano Nolfi, Luca Maria Gambardella, and Marco Dorigo. 'SWARM-BOT: A New Distributed Robotic Concept'. In: *Autonomous Robots* 17.2 (2004), pp. 193–221. DOI: 10.1023/B:AURO.0000033972.50769.1c.

[115] Farshad Arvin, John Murray, Chun Zhang, and Shigang Yue. 'Colias: An Autonomous Micro Robot for Swarm Robotic Applications'. In: *International Journal of Advanced Robotic Systems* 11.7 (2014), p. 113. DOI: 10.5772/58730.

[116] Carlo Pinciroli et al. 'ARGoS: A Modular, Multi-engine Simulator for Heterogeneous Swarm Robotics'. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, pp. 5027–5034. DOI: 10.1109/IROS.2011.6094829.

[117] Stéphane Magnenat, Philippe Rétornaz, Michael Bonani, Valentin Longchamp, and Francesco Mondada. 'ASEBA: A Modular Architecture for Event-Based Control of Complex Robots'. In: *IEEE/ASME Transactions on Mechatronics* 16.2 (2011), pp. 321–329. DOI: 10.1109/TMECH.2010.2042722.

[118] Stéphane Magnenat and Francesco Mondada. 'Aseba Meets D-Bus: From the Depths of a Low-Level Event-Based Architecture into the Middleware Realm'. In: *IEEE TC-Soft Workshop on Event-based Systems in Robotics (EBS-RO)*. 2009.

[119] Melanie Schranz, Martina Umlauft, Micha Sende, and Wilfried Elmenreich. 'Swarm Robotic Behaviors and Current Applications'. In: *Frontiers in Robotics and AI* 7 (2020), p. 36. DOI: 10.3389/frobt.2020.00036.

[120] Julia K. Parrish and Leah Edelstein-Keshet. 'Complexity, Pattern, and Evolutionary Trade-Offs in Animal Aggregation'. In: *Science* 284.5411 (1999), pp. 99–101. DOI: 10.1126/science.284.5411.99.

[121] Elena O. Budrene and Howard C. Berg. 'Dynamics of Formation of Symmetrical Patterns by Chemotactic Bacteria'. In: *Nature* 376.6535 (1995), pp. 49–53.

[122] Martina Szopek, Thomas Schmickl, Ronald Thenius, Gerald Radspieler, and Karl Crailsheim. 'Dynamics of Collective Decision Making of Honeybees in Complex Temperature Fields'. In: *PLOS ONE* 8.10 (Oct. 2013), pp. 1–11. DOI: 10.1371/journal.pone.0076250.

[123] Raphael Jeanson, Colette Rivault, Jean-Louis Deneubourg, Stephane Blanco, Richard Fournier, Christian Jost, and Guy Theraulaz. 'Self-Organized Aggregation in Cockroaches'. In: *Animal Behaviour* 69.1 (2005), pp. 169–180. DOI: 10.1016/j.anbehav.2004.02.009.

[124] Harrington Wells, Patrick H. Wells, and Peyton Cook. 'The Importance of Overwinter Aggregation for Reproductive Success of Monarch Butterflies (Danaus plexippus L.)' In: *Journal of Theoretical Biology* 147.1 (1990), pp. 115–131. DOI: 10.1016/S0022-5193(05)80255-3.

[125] Jarosław Kobak, Małgorzata Poznańska, and Tomasz Kakareko. 'Effect of Attachment Status and Aggregation on the Behaviour of the Zebra Mussel Dreissena Polymorpha'. In: *Journal of Molluscan Studies* 75.2 (Jan. 2009), pp. 119–126. DOI: 10.1093/mollus/eyn046.

[126] Caitlin Black, Ben Collen, Daniel Johnston, and Tom Hart. 'Why Huddle? Ecological Drivers of Chick Aggregations in Gentoo Penguins, Pygoscelis papua, across Latitudes'. In: *PLOS ONE* 11.2 (Feb. 2016), pp. 1–12. DOI: 10.1371/journal.pone.0145676.

[127] N. E. Shlyakhov, I. V. Vatamaniuk, and A. L. Ronzhin. 'Survey of Methods and Algorithms of Robot Swarm Aggregation'. In: *Journal of Physics: Conference Series* 803 (2017), p. 012146. DOI: 10.1088/1742-6596/803/1/012146.

[128] Serge Kernbach, Ronald Thenius, Olga Kernbach, and Thomas Schmickl. 'Re-embodiment of Honeybee Aggregation Behavior in an Artificial Micro-Robotic System'. In: *Adaptive Behavior* 17.3 (2009), pp. 237–259. DOI: 10.1177/1059712309104966.

[129] Thomas Schmickl and Heiko Hamann. 'BEECLUST: A Swarm Algorithm Derived from Honeybees'. In: *Bio-Inspired Computing and Communication Networks*. Ed. by Yang Xiao. 1st. CRC Press, 2011. DOI: 10.1201/b10781.

[130] Simon Garnier, Jacques Gautrais, Masoud Asadpour, Christian Jost, and Guy Theraulaz. 'Self-Organized Aggregation Triggers Collective Decision Making in a Group of Cockroach-like Robots'. In: *Adaptive Behavior* 17.2 (2009), pp. 109–133. DOI: 10.1177/1059712309103430.

[131] Nikolaus Correll and Alcherio Martinoli. 'Modeling and Designing Self-Organized Aggregation in a Swarm of Miniature Robots'. In: *The International Journal of Robotics Research* 30.5 (2011), pp. 615–626. DOI: 10.1177/0278364911403017.

[132] Onur Soysal and Erol Şahin. 'Probabilistic Aggregation Strategies in Swarm Robotic Systems'. In: *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.* 2005, pp. 325–332. DOI: 10.1109/SIS.2005.1501639.

[133] Vito Trianni, Roderich Groß, Thomas H. Labella, Erol Şahin, and Marco Dorigo. 'Evolving Aggregation Behaviors in a Swarm of Robots'. In: *Advances in Artificial Life*. Ed. by Wolfgang Banzhaf, Jens Ziegler, Thomas Christaller, Peter Dittrich, and Jan T. Kim. Berlin, Heidelberg: Springer, 2003, pp. 865–874. DOI: 10.1007/978-3-540-39432-7_93.

[134] Jorge Gomes, Paulo Urbano, and Anders Lyhne Christensen. 'Evolution of Swarm Robotics Systems with Novelty Search'. In: *Swarm Intelligence* 7.2-3 (2013), pp. 115–144. DOI: 10.1007/s11721-013-0081-z.

[135] Belkacem Khaldi, Fouzi Harrou, Foudil Cherif, and Ying Sun. 'Flexible and Efficient Topological Approaches for a Reliable Robots Swarm Aggregation'. In: *IEEE Access* 7 (2019), pp. 96372–96383. DOI: 10.1109/ACCESS.2019.2930677.

[136] Heiko Hamann. 'Szenarien'. In: *Schwarmintelligenz*. Berlin, Heidelberg: Springer, 2019, pp. 45–67. DOI: 10.1007/978-3-662-58961-8_3.

[137] Sean E. Walker. 'Density and Dispersion'. In: *Nature Education Knowledge* 3.10 (2011), p. 3.

[138] Maja J. Matarić. 'Designing and Understanding Adaptive Group Behavior'. In: *Adaptive Behavior* 4.1 (1995), pp. 51–80. DOI: 10.1177/105971239500400104.

[139] Emre Ugur, Ali Emre Turgut, and Erol Şahin. 'Dispersion of a Swarm of Robots Based on Realistic Wireless Intensity Signals'. In: *2007 22nd International Symposium on Computer and Information Sciences*. 2007, pp. 1–6. DOI: 10.1109/ISCIS.2007.4456899.

[140] Andrew Howard, Maja J. Matarić, and Gaurav S. Sukhatme. 'Mobile Sensor Network Deployment using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem'. In: *Distributed Autonomous Robotic Systems 5*. Ed. by Hajime Asama, Tamio Arai, Toshio Fukuda, and Tsutomu Hasegawa. Tokyo: Springer Japan, 2002, pp. 299–308.

[141] David Payton, Mike Daily, Regina Estowski, Mike Howard, and Craig Lee. 'Pheromone Robotics'. In: *Autonomous Robots* 11.3 (2001), pp. 319–324. DOI: 10.1023/A:1012411712038.

[142] Miguel Duarte, Sancho Oliveira, and Anders Christensen. 'Hybrid Control for Large Swarms of Aquatic Drones'. In: vol. ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems. ALIFE 2020: The 2020 Conference on Artificial Life. July 2014, pp. 785–792. DOI: 10.1162/978-0-262-32621-6-ch127.

[143] Miguel Duarte, Vasco Costa, Jorge Gomes, Tiago Rodrigues, Fernando Silva, Sancho Moura Oliveira, and Anders Lyhne Christensen. 'Evolution of Collective Behaviors for a Real Swarm of Aquatic Surface Robots'. In: *PLOS ONE* 11.3 (Mar. 2016), pp. 1–25. DOI: 10.1371/journal.pone.0151834.

[144] Hans Meinhardt. *Models of Biological Pattern Formation*. Academic Press Inc., 1982.

[145] James S. Langer. 'Instabilities and Pattern Formation in Crystal Growth'. In: *Rev. Mod. Phys.* 52 (1 Jan. 1980), pp. 1–28. DOI: 10.1103/RevModPhys.52.1.

[146] Daniel Walgraef. *Spatio-Temporal Pattern Formation: With Examples from Physics, Chemistry, and Materials Science*. Springer Science & Business Media, 2012.

[147] Mehdi Moussaid, Simon Garnier, Guy Theraulaz, and Dirk Helbing. 'Collective Information Processing and Pattern Formation in Swarms, Flocks, and Crowds'. In: *Topics in Cognitive Science* 1.3 (2009), pp. 469–497. DOI: 10.1111/j.1756-8765.2009.01028.x.

[148] Philip Ball. 'Forging Patterns and Making Waves from Biology to Geology: a Commentary on Turing (1952) 'The Chemical Basis of Morphogenesis''. In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 370.1666 (2015), p. 20140218. DOI: 10.1098/rstb.2014.0218.

[149] William M. Spears, Diana F. Spears, Jerry C. Hamann, and Rodney Heil. 'Distributed, Physics-Based Control of Swarms of Vehicles'. In: *Autonomous Robots* 17.2 (2004), pp. 137–162. DOI: 10.1023/B:AURO.0000033970.96785.f2.

[150] Yan Meng, Hongliang Guo, and Yaochu Jin. 'A Morphogenetic Approach to Flexible and Robust Shape Formation for Swarm Robotic Systems'. In: *Robotics and Autonomous Systems* 61.1 (2013), pp. 25–38. DOI: 10.1016/j.robot.2012.09.009.

[151] Jean-Louis Deneubourg, Serge Aron, Simon Goss, and Jacques M. Pasteels. 'The Self-Organizing Exploratory Pattern of the Argentine Ant'. In: *Journal of Insect Behavior* 3.2 (1990), pp. 159–168.

[152] Shervin Nouyan, Alexandre Campo, and Marco Dorigo. 'Path Formation in a Robot Swarm'. In: *Swarm Intelligence* 2.1 (2008), pp. 1–23. DOI: 10.1007/s11721-007-0009-6.

[153] Valerio Sperati, Vito Trianni, and Stefano Nolfi. 'Self-Organised Path Formation in a Swarm of Robots'. In: *Swarm Intelligence* 5.2 (2011), pp. 97–119. DOI: 10.1007/s11721-011-0055-y.

[154] Heiko Hamann, Mare Szymanski, and Heinz Worn. 'Orientation in a Trail Network by Exploiting its Geometry for Swarm Robotics'. In: *2007 IEEE Swarm Intelligence Symposium*. 2007, pp. 310–315. DOI: 10.1109/SIS.2007.367953.

[155] George M. Whitesides and Bartosz Grzybowski. 'Self-Assembly at All Scales'. In: *Science* 295.5564 (2002), pp. 2418–2421. DOI: 10.1126/science.1070821.

[156] Carl Anderson, Guy Theraulaz, and Jean-Louis Deneubourg. 'Self-Assemblages in Insect Societies'. In: *Insectes Sociaux* 49.2 (2002), pp. 99–110. DOI: 10.1007/s00040-002-8286-y.

[157] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. 'Programmable Self-Assembly in a Thousand-Robot Swarm'. In: *Science* 345.6198 (2014), pp. 795–799. DOI: 10.1126/science.1254295.

[158] Mohammad Divband Soorati, Mary Katherine Heinrich, Javad Ghofrani, Payam Zahadat, and Heiko Hamann. 'Photomorphogenesis for Robot Self-Assembly: Adaptivity, Collective Decision-Making, and Self-Repair'. In: *Bioinspiration & biomimetics* 14.5 (2019), p. 056006. DOI: 10.1088/1748-3190/ab2958.

[159] Francesco Mondada, Luca Maria Gambardella, Dario Floreano, Stefano Nolfi, Jean-Louis Deneuborg, and Marco Dorigo. 'The Cooperation of Swarm-bots: Physical Interactions in Collective Robotics'. In: *IEEE Robotics Automation Magazine* 12.2 (2005), pp. 21–28. DOI: 10.1109/MRA.2005.1458313.

[160] Rehan O'Grady, Roderich Groß, Anders Lyhne Christensen, and Marco Dorigo. 'Self-assembly Strategies in a Group of Autonomous Mobile Robots'. In: *Autonomous Robots* 28.4 (2010), pp. 439–455. DOI: 10.1007/s10514-010-9177-0.

[161] Roderich Groß and Marco Dorigo. 'Evolution of Solitary and Group Transport Behaviors for Autonomous Robots Capable of Self-Assembling'. In: *Adaptive Behavior* 16.5 (2008), pp. 285–305. DOI: 10.1177/1059712308090537.

[162] Petras Swissler and Michael Rubenstein. 'FireAnt3D: a 3D Self-Climbing Robot Towards Non-Latticed Robotic Self-Assembly'. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 3340–3347. DOI: 10.1109/IROS45743.2020.9341116.

[163] Petras Swissler and Michael Rubenstein. 'ReactiveBuild: Environment-Adaptive Self-Assembly of Amorphous Structures'. In: *Distributed Autonomous Robotic Systems*. Ed. by Fumitoshi Matsuno, Shun-ichi Azuma, and Masahito Yamamoto. Springer International Publishing. Cham, 2022, pp. 363–375. DOI: 10.1007/978-3-030-92790-5_28.

[164] Eric Klavins. 'Programmable Self-Assembly'. In: *IEEE Control Systems Magazine* 27.4 (2007), pp. 43–56. DOI: 10.1109/MCS.2007.384126.

[165] Hossein Ahmadzadeh, Ellips Masehian, and Masoud Asadpour. 'Modular Robotic Systems: Characteristics and Applications'. In: *Journal of Intelligent & Robotic Systems* 81.3-4 (2016), pp. 317–357. DOI: 10.1007/s10846-015-0237-8.

[166] Stephen Wolfram. 'Cellular Automata as Models of Complexity'. In: *Nature* 311.5985 (1984), pp. 419–424. DOI: 10.1038/311419a0.

[167] Martin Gardner. 'The Fantastic Combinations of John Conway's New Solitaire Game Life'. In: *Scientific American* 223 (1970), pp. 120–123.

[168] Andrew Adamatzky. *Game of Life Cellular Automata*. London: Springer London, 2010. DOI: 10.1007/978-1-84996-217-9.

[169] Guy A. Hoelzer, Rich Drewes, Jeffrey Meier, and René Doursat. 'Isolation-by-Distance and Outbreeding Depression are Sufficient to Drive Parapatric Speciation in the Absence of Environmental Influences'. In: *PLoS Computational Biology* 4.7 (2008), e1000126. DOI: 10.1371/journal.pcbi.1000126.

[170] David Medernach, Simon Carrignon, René Doursat, Taras Kowaliw, Jeannie Fitzgerald, and Conor Ryan. 'Evolution of Heterogeneous Cellular Automata in Fluctuating Environments'. In: vol. ALIFE 2016, the Fifteenth International Conference on the Synthesis and Simulation of Living Systems. ALIFE 2021: The 2021 Conference on Artificial Life. July 2016, pp. 216–223. DOI: 10.1162/978-0-262-33936-0-ch041.

[171] Wilfried Elmenreich and István Fehérvári. 'Evolving Self-Organizing Cellular Automata Based on Neural Network Genotypes'. In: *Self-Organizing Systems*. Ed. by Christian Bettstetter and Carlos Gershenson. Berlin, Heidelberg: Springer, 2011, pp. 16–25.

[172] Rolf Hoffmann. 'How Agents Can Form a Specific Pattern'. In: *Cellular Automata*. Ed. by Jarosław Wąs, Georgios Ch. Sirakoulis, and Stefania Bandini. Cham: Springer International Publishing, 2014, pp. 660–669. DOI: 10.1007/978-3-319-11520-7_70.

[173] Rolf Hoffmann and Dominique Désérable. 'Line Patterns Formed by Cellular Automata Agents'. In: *Cellular Automata*. Ed. by Samira El Yacoubi, Jaroslaw Wąs, and Stefania Bandini. Cham: Springer International Publishing, 2016, pp. 424–434. DOI: 10.1007/978-3-319-44365-2_42.

[174] Rolf Hoffmann and Dominique Désérable. 'Domino Pattern Formation by Cellular Automata Agents'. In: *The Journal of Supercomputing* (May 2018), pp. 1–15. DOI: 10.1007/s11227-018-2429-z.

[175] Rolf Hoffmann. 'Checkerboard Pattern Formed by Cellular Automata Agents'. In: *Shortest Path Solvers. From Software to Wetware*. Ed. by Andrew Adamatzky. Cham: Springer International Publishing, 2018, pp. 239–264. DOI: 10.1007/978-3-319-77510-4_9.

[176] Can Öztürkeri and Colin Johnson. 'Evolution of Self-Assembling Patterns in Cellular Automata Using Development'. In: *J. Cellular Automata* 6 (Jan. 2011), pp. 257–300.

[177] Nigel R. Franks and Ana B. Sendova-Franks. 'Brood Sorting by Ants: Distributing the Workload Over the Work-Surface'. In: *Behavioral Ecology and Sociobiology* 30.2 (1992), pp. 109–123. DOI: 10.1007/BF00173947.

[178] Marc Martin, Bastien Chopard, and Paul Albuquerque. 'Formation of an Ant Cemetery: Swarm Intelligence or Statistical Accident?' In: *Future Generation Computer Systems* 18.7 (2002). Selected papers from CA2000 (6th International Workshop on Cellular Automata of IFIP working group 1.5, Osaka, Japan, Aug. 21-22, 2000) and ACRI2000 (4th International Conference on Cellular Automata in Research and Industry, Karlsruhe, Germany, Oct. 4-6, 2000), pp. 951–959. DOI: 10.1016/S0167-739X(02)00074-2.

[179] Mitchel Resnick. *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. Mit Press, 1997.

[180] Ralph Beckers, Owen E. Holland, Jean-Louis Deneubourg, and Rodney A. Brooks. 'From Local Actions to Global Tasks: Stigmergy and Collective Robotics'. In: *Proceedings of ALIFE IV*. MIT Press. 1994.

[181] Alexander Scheidler, Daniel Merkle, and Martin Middendorf. 'Swarm Controlled Emergence for Ant Clustering'. In: *International Journal of Intelligent Computing and Cybernetics* (2013). DOI: 10.1108/17563781311301526.

[182] Owen Holland and Chris Melhuish. 'Stigmergy, Self-Organization, and Sorting in Collective Robotics'. In: *Artificial life* 5.2 (1999), pp. 173–202.

[183] Matt Wilson, Chris Melhuish, Ana B. Sendova-Franks, and Samuel Scholes. 'Algorithms for Building Annular Structures with Minimalist Robots Inspired by Brood Sorting in Ant Colonies'. In: *Autonomous Robots* 17.2 (2004), pp. 115–136. DOI: 10.1023/B:AURO.0000033969.52486.3d.

[184] Vegard Hartmann. 'Evolving Agent Swarms for Clustering and Sorting'. In: *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*. GECCO '05. Washington DC, USA: Association for Computing Machinery, 2005, pp. 217–224. DOI: 10.1145/1068009.1068042.

[185] André Heie Vik. 'Evolving Annular Sorting in Ant-Like Agents'. In: *Advances in Artificial Life*. Ed. by Mathieu S. Capcarrère, Alex A. Freitas, Peter J. Bentley, Colin G. Johnson, and Jon Timmis. Berlin, Heidelberg: Springer, 2005, pp. 594–603. DOI: 10.1007/11553090_60.

[186] Melvin Gauci, Jianing Chen, Wei Li, Tony J. Dodd, and Roderich Gross. 'Clustering Objects with Robots That Do Not Compute'. In: *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems*. AAMAS '14. Paris, France: International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 421–428.

[187] Gregory Vorobyev, Andrew Vardy, and Wolfgang Banzhaf. 'Supervised Learning in Robotic Swarms: From Training Samples to Emergent Behavior'. In: *Distributed Autonomous Robotic Systems*. Ed. by M. Ani Hsieh and Gregory Chirikjian. Berlin, Heidelberg: Springer, 2014, pp. 435–448.

[188] Guy Theraulaz, Eric Bonabeau, and Jean-Louis Deneubourg. 'The Origin of Nest Complexity in Social Insects'. In: *Complexity* 3.6 (1998), pp. 15–25. DOI: 10.1002/(SICI)1099-0526(199807/08)3:6<15::AID-CPLX3>3.0.CO;2-V.

[189] GA Rösch. 'Über die Bautätigkeit im Bienenvolk und das Alter der Baubienen'. In: *Zeitschrift für vergleichende Physiologie* 6.2 (1927), pp. 264–298.

[190] Wirgiliusz Żurowski. 'Building Activity of Beavers'. In: *Acta Theriol* 37.4 (1992), pp. 403–411.

[191] Esie C. Collias and Nicholas E. Collias. 'Nest Building and Nesting Behavior of the Sociable Weaver Philetairus Socius'. In: *Ibis* 120.1 (1978), pp. 1–15. DOI: 10.1111/j.1474-919X.1978.tb04994.x.

[192] Kirstin H. Petersen, Nils Napp, Robert Stuart-Smith, Daniela Rus, and Mirko Kovac. 'A Review of Collective Robotic Construction'. In: *Science Robotics* 4.28 (2019). doi: 10.1126/scirobotics.aau8479.

[193] Chris A. C. Parker, Hong Zhang, and C. Ronald Kube. 'Blind Bulldozing: Multiple Robot Nest Construction'. In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*. Vol. 2. 2003, pp. 2010–2015. doi: 10.1109/IROS.2003.1248950.

[194] Christian Charles. 'Implementation of Blind Bulldozing with Minimize Surprise'. Lübeck, Germany: Universität zu Lübeck, Germany, June 2020.

[195] Justin Werfel, Kirstin Petersen, and Radhika Nagpal. 'Designing Collective Behavior in a Termite-Inspired Robot Construction Team'. In: *Science* 343.6172 (2014), pp. 754–758. doi: 10.1126/science.1245842.

[196] Mary Katherine Heinrich, Mostafa Wahby, Mohammad Divband Soorati, Daniel Nicolas Hofstadler, Payam Zahadat, Phil Ayres, Kasper Støy, and Heiko Hamann. 'Self-Organized Construction with Continuous Building Material: Higher Flexibility Based on Braided Structures'. In: *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. 2016, pp. 154–159. doi: 10.1109/FAS-W.2016.43.

[197] Nils Napp and Radhika Nagpal. 'Distributed Amorphous Ramp Construction in Unstructured Environments'. In: *Distributed Autonomous Robotic Systems*. Ed. by M. Ani Hsieh and Gregory Chirikjian. Berlin, Heidelberg: Springer, 2014, pp. 105–119. doi: 10.1007/978-3-642-55146-8_8.

[198] Frederico Augugliaro et al. 'The Flight Assembled Architecture Installation: Cooperative Construction with Flying Machines'. In: *IEEE Control Systems Magazine* 34.4 (2014), pp. 46–64. doi: 10.1109/MCS.2014.2320359.

[199] Geoff S. Nitschke, Martijn C. Schut, and Agoston E. Eiben. 'Evolving Behavioral Specialization in Robot Teams to Solve a Collective Construction Task'. In: *Swarm and Evolutionary Computation* 2 (2012), pp. 25–38. doi: 10.1016/j.swevo.2011.08.002.

[200] Caroline Strickland, David Churchill, and Andrew Vardy. 'A Reinforcement Learning Approach to Multi-Robot Planar Construction'. In: *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. 2019, pp. 238–244. doi: 10.1109/MRS.2019.8901087.

[201] Andrew Vardy. 'Orbital Construction: Swarms of Simple Robots Building Enclosures'. In: *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. 2018, pp. 147–153. doi: 10.1109/FAS-W.2018.00040.

[202] Andrew Vardy and Dalia S. Ibrahim. *A Swarm of Simple Robots Constructing Planar Shapes*. 2020. doi: 10.48550/arXiv.2004.13888. arXiv: 2004.13888 [cs.RO].

[203] Andrew Vardy. 'Robot Distancing: Planar Construction with Lanes'. In: *ANTS 2020: Swarm Intelligence*. Ed. by Marco Dorigo, Thomas Stützle, Maria J. Blesa, Christian Blum, Heiko Hamann, Mary Katherine Heinrich, and Volker Strobel. Vol. 12421. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 229–242. doi: 10.1007/978-3-030-60376-2_18.

[204] Dmitrii Viktorovich Radakov. *Schooling in the Ecology of Fish*. J. Wiley, 1973.

[205] John Stodart Kennedy and Vincent Brian Wigglesworth. 'The Migration of the Desert Locust (Schistocerca gregaria Forsk.) I. The Behaviour of Swarms. II. A Theory of Long-range Migrations'. In: *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences* 235.625 (1951), pp. 163–290. doi: 10.1098/rstb.1951.0003.

[206] Anthony Ronald Entrican Sinclair. *The African Buffalo: A Study of Resource Limitation of Populations*. Chicago, Ill.(USA) Univ. of Chicago Press, 1977.

[207] Craig W. Reynolds. 'Flocks, Herds and Schools: A Distributed Behavioral Model'. In: *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. 1987, pp. 25–34.

[208] Michele Ballerini et al. 'Interaction Ruling Animal Collective Behavior Depends on Topological Rather than Metric Distance: Evidence from a Field Study'. In: *Proceedings of the National Academy of Sciences* 105.4 (2008), pp. 1232–1237. doi: 10.1073/pnas.0711437105.

[209] Ali Emre Turgut, Hande Çelikkanat, Fatih Gökçe, and Erol Şahin. 'Self-organized Flocking in Mobile Robot Swarms'. In: *Swarm Intelligence* 2.2 (2008), pp. 97–120. DOI: 10.1007/s11721-008-0016-2.

[210] Sabine Hauert, Severin Leven, Maja Varga, Fabio Ruini, Angelo Cangelosi, Jean-Christophe Zufferey, and Dario Floreano. 'Reynolds Flocking in Reality with Fixed-Wing Robots: Communication Range vs. Maximum Turning Rate'. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, pp. 5015–5020. DOI: 10.1109/IROS.2011.6095129.

[211] Christoph Moeslinger, Thomas Schmickl, and Karl Crailsheim. 'Emergent Flocking with Low-End Swarm Robots'. In: *ANTS 2010: Swarm Intelligence*. Ed. by Marco Dorigo et al. Vol. 6234. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 424–431. DOI: 10.1007/978-3-642-15461-4_40.

[212] Christoph Moeslinger, Thomas Schmickl, and Karl Crailsheim. 'A Minimalist Flocking Algorithm for Swarm Robots'. In: *Advances in Artificial Life. Darwin Meets von Neumann*. Ed. by George Kampis, István Karsai, and Eörs Szathmáry. Berlin, Heidelberg: Springer, 2011, pp. 375–382. DOI: 10.1007/978-3-642-21314-4_47.

[213] Eliseo Ferrante, Ali Emre Turgut, Cristián Huepe, Alessandro Stranieri, Carlo Pinciroli, and Marco Dorigo. 'Self-Organized Flocking with a Mobile Robot Swarm: A Novel Motion Control Method'. In: *Adaptive Behavior* 20.6 (2012), pp. 460–477. DOI: 10.1177/1059712312462248.

[214] Tucker Balch and Maria Hybinette. 'Social Potentials for Scalable Multi-Robot Formations'. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 1. 2000, pp. 73–80. DOI: 10.1109/ROBOT.2000.844042.

[215] Gianluca Baldassarre, Stefano Nolfi, and Domenico Parisi. 'Evolving Mobile Robots Able to Display Collective Behaviors'. In: *Artificial Life* 9.3 (2003), pp. 255–267. DOI: 10.1162/106454603322392460.

[216] Rita Parada Ramos, Sancho Moura Oliveira, Susana Margarida Vieira, and Anders Lyhne Christensen. 'Evolving Flocking in Embodied Agents Based on Local and Global Application of Reynolds' Rules'. In: *PLOS ONE* 14.10 (Oct. 2019), pp. 1–16. DOI: 10.1371/journal.pone.0224376.

[217] Bert Hölldobler and Edward O. Wilson. *Journey to the Ants: A Story of Scientific Exploration*. Harvard University Press, 1998.

[218] Simon Goss, Serge Aron, Jean-Louis Deneubourg, and Jacques Marie Pasteels. 'Self-Organized Shortcuts in the Argentine Ant'. In: *Naturwissenschaften* 76.12 (1989), pp. 579–581. DOI: 10.1007/BF00462870.

[219] Stephen C. Pratt. 'Optimal Timing of Comb Construction by Honeybee (Apis mellifera) Colonies: A Dynamic Programming Model and Experimental Tests'. In: *Behavioral Ecology and Sociobiology* 46.1 (1999), pp. 30–42.

[220] Gabriele Valentini, Davide Brambilla, Heiko Hamann, and Marco Dorigo. 'Collective Perception of Environmental Features in a Robot Swarm'. In: *ANTS 2016: Swarm Intelligence*. Ed. by Marco Dorigo, Mauro Birattari, Xiaodong Li, Manuel López-Ibáñez, Kazuhiro Ohkura, Carlo Pinciroli, and Thomas Stützle. Vol. 9882. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 65–76. DOI: 10.1007/978-3-319-44427-7_6.

[221] Gabriele Valentini, Eliseo Ferrante, and Marco Dorigo. 'The Best-of-n Problem in Robot Swarms: Formalization, State of the Art, and Novel Perspectives'. In: *Frontiers in Robotics and AI* 4 (2017), p. 9. DOI: 10.3389/frobt.2017.00009.

[222] Gabriele Valentini, Heiko Hamann, and Marco Dorigo. 'Efficient Decision-Making in a Self-Organizing Robot Swarm: On the Speed Versus Accuracy Trade-Off'. In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. AAMAS '15. Istanbul, Turkey: International Foundation for Autonomous Agents and Multiagent Systems, 2015, pp. 1305–1314. DOI: 10.5555/2772879.2773319.

[223] Gabriele Valentini, Heiko Hamann, and Marco Dorigo. 'Self-Organized Collective Decision Making: The Weighted Voter Model'. In: *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems*. AAMAS '14. Paris, France: International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 45–52. DOI: 10.5555/2615731.2615742.

[224] Julia T. Ebert, Melvin Gauci, and Radhika Nagpal. 'Multi-Feature Collective Decision Making in Robot Swarms'. In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '18. Stockholm, Sweden: International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 1711–1719.

[225] Palina Bartashevich and Sanaz Mostaghim. 'Multi-Featured Collective Perception with Evidence Theory: Tackling Spatial Correlations'. In: *Swarm Intelligence* 15 (2021), pp. 83–110. DOI: 10.1007/s11721-021-00192-8.

[226] Volker Strobel, Eduardo Castelló Ferrer, and Marco Dorigo. 'Managing Byzantine Robots via Blockchain Technology in a Swarm Robotics Collective Decision Making Scenario'. In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '18. Stockholm, Sweden: International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 541–549.

[227] Palina Bartashevich and Sanaz Mostaghim. 'Positive Impact of Isomorphic Changes in the Environment on Collective Decision-Making'. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '19. Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 105–106. DOI: 10.1145/3319619.3321984.

[228] Julia T. Ebert, Melvin Gauci, Frederik Mallmann-Trenn, and Radhika Nagpal. 'Bayes Bots: Collective Bayesian Decision-Making in Decentralized Robot Swarms'. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 7186–7192. DOI: 10.1109/ICRA40945.2020.9196584.

[229] Palina Bartashevich and Sanaz Mostaghim. 'Ising Model as a Switch Voting Mechanism in Collective Perception'. In: *Progress in Artificial Intelligence*. Ed. by Paulo Moura Oliveira, Paulo Novais, and Luís Paulo Reis. Cham: Springer International Publishing, 2019, pp. 617–629. DOI: 10.1007/978-3-030-30244-3_51.

[230] Palina Bartashevich and Sanaz Mostaghim. 'Benchmarking Collective Perception: New Task Difficulty Metrics for Collective Decision-Making'. In: *EPIA 2019: Progress in Artificial Intelligence*. Ed. by Paulo Moura Oliveira, Paulo Novais, and Luís Paulo Reis. Cham: Springer International Publishing, 2019, pp. 699–711.

[231] Ahmed Almansoori, Muhanad Hayder Mohammed Mohammed, Jean-Noel Colin, and Elio Tuci. 'On the Evolution of Mechanisms for Collective Decision Making in a Swarm of Robots'. In: *Proceedings of the XV International Workshop on Artificial Life and Evolutionary Computation (WIVACE)*. 2021.

[232] Giuseppe Morlino, Vito Trianni, and Elio Tuci. 'Collective Perception in a Swarm of Autonomous Robots'. In: *Proceedings of the International Conference on Evolutionary Computation - Volume 1: ICEC, (IJCCI 2010)*. INSTICC. SciTePress, 2010, pp. 51–59. DOI: 10.5220/0003075400510059.

[233] Thomas Schmickl, Christoph Möslinger, and Karl Crailsheim. 'Collective Perception in a Robot Swarm'. In: *Swarm Robotics*. Ed. by Erol Şahin, William M. Spears, and Alan F. T. Winfield. Berlin, Heidelberg: Springer, 2007, pp. 144–157. DOI: 10.1007/978-3-540-71541-2_10.

[234] Sergey Kornienko, Olga Kornienko, C. Constantinescu, M. Pradier, and Paul Levi. 'Cognitive Micro-Agents: Individual and Collective Perception in Microrobotic Swarm'. In: *Proceedings of the IJCAI-05 Workshop on Agents in Real-Time and Dynamic Environments, Edinburgh, UK*. 2005, pp. 33–42.

[235] David King and Philip Breedon. 'Towards Cooperative Robotic Swarm Recognition: Object Classification and Validity'. In: *Advanced Design and Manufacture III*. Vol. 450. Key Engineering Materials. Trans Tech Publications Ltd, Jan. 2011, pp. 320–324. DOI: 10.4028/www.scientific.net/KEM.450.320.

[236] Alan M. Turing. 'I.—Computing Machinery and Intelligence'. In: *Mind* LIX.236 (Oct. 1950), pp. 433–460. DOI: 10.1093/mind/LIX.236.433.

[237] Rolf Pfeifer and Josh Bongard. *How the Body Shapes the Way We Think: A New View of Intelligence*. MIT Press, 2006.

[238] Gobinda G. Chowdhury. 'Natural Language Processing'. In: *Annual Review of Information Science and Technology* 37.1 (2003), pp. 51–89. DOI: 10.1002/aris.1440370103.

[239] Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter. *Handbook of Knowledge Representation*. San Diego, CA, USA: Elsevier Science, 2007.

[240] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis and Machine Vision*. Boston, MA: Springer US, 1993. DOI: 10.1007/978-1-4899-3216-7.

[241] Tom M. Mitchell. *Machine Learning*. McGraw-hill New York, 1997.

[242] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[243] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Berlin, Heidelberg: Springer, 2006.

[244] Geoffrey E Hinton, Terrence Joseph Sejnowski, et al. *Unsupervised Learning: Foundations of Neural Computation*. MIT Press, 1999.

[245] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[246] Ian H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. *Data Mining: Practical Machine Learning Tools and Techniques*. 4th. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016.

[247] Yann LeCun and Ishan Misra. *Self-Supervised Learning: The Dark Matter of Intelligence*. 2021. URL: https://ai.facebook.com/blog/self-supervised-learning-the-dark-matter-of-intelligence/ (visited on 06/29/2021).

[248] Stefan Leijnen and Fjodor van Veen. 'The Neural Network Zoo'. In: *Proceedings* 47.1 (2020). DOI: 10.3390/proceedings2020047009.

[249] Jan Peters, Daniel D. Lee, Jens Kober, Duy Nguyen-Tuong, J. Andrew Bagnell, and Stefan Schaal. 'Robot Learning'. In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Cham: Springer International Publishing, 2016, pp. 357–398. DOI: 10.1007/978-3-319-32552-1_15.

[250] Lorenzo Canese, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Re, and Sergio Spanò. 'Multi-Agent Reinforcement Learning: A Review of Challenges and Applications'. In: *Applied Sciences* 11.11 (2021). DOI: 10.3390/app11114948.

[251] Sven Gronauer and Klaus Diepold. 'Multi-Agent Deep Reinforcement Learning: A Survey'. In: *Artificial Intelligence Review* 55.2 (2022), pp. 895–943. DOI: 10.1007/s10462-021-09996-w.

[252] Liviu Panait and Sean Luke. 'Cooperative Multi-agent Learning: The State of the Art'. In: *Autonomous Agents and Multi-agent Systems* 11.3 (2005), pp. 387–434. DOI: 10.1007/s10458-005-2631-2.

[253] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. 'Multi-agent Reinforcement Learning: A Selective Overview of Theories and Algorithms'. In: *Handbook of Reinforcement Learning and Control*. Ed. by Kyriakos G. Vamvoudakis, Yan Wan, Frank L. Lewis, and Derya Cansever. Cham: Springer International Publishing, 2021, pp. 321–384. DOI: 10.1007/978-3-030-60990-0_12.

[254] Yixin Huang, Shufan Wu, Zhongcheng Mu, Xiangyu Long, Sunhao Chu, and Guohong Zhao. 'A Multi-Agent Reinforcement Learning Method for Swarm Robots in Space Collaborative Exploration'. In: *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*. 2020, pp. 139–144. DOI: 10.1109/ICCAR49639.2020.9107997.

[255] Toshiyuki Yasuda and Kazuhiro Ohkura. 'Collective Behavior Acquisition of Real Robotic Swarms Using Deep Reinforcement Learning'. In: *2018 Second IEEE International Conference on Robotic Computing (IRC)*. 2018, pp. 179–180. DOI: 10.1109/IRC.2018.00038.

[256] Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. 'Deep Reinforcement Learning for Swarm Systems'. In: *Journal of Machine Learning Research* 20.54 (2019), pp. 1–31.

[257] Charles Darwin. *The Origin of Species*. PF Collier & Son New York, 1909.

[258] Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. 'Artificial Intelligence through Simulated Evolution'. In: Wiley, Chichester, UK, 1966.

[259] John Henry Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, 1992.

[260] Ingo Rechenberg. *Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Hollboog Verlag, Stuttgart, 1973.

[261] Hans-Paul Paul Schwefel. *Evolution and Optimum Seeking: The Sixth Generation*. USA: John Wiley & Sons, Inc., 1993.

[262] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Vol. 1. MIT Press, 1992.

[263] J. Schmidhuber. 'Evolutionary Computation Versus Reinforcement Learning'. In: *2000 26th Annual Conference of the IEEE Industrial Electronics Society. IECON 2000. 2000 IEEE International Conference on Industrial Electronics, Control and Instrumentation. 21st Century Technologies*. Vol. 4. 2000, pp. 2992–2997. DOI: 10.1109/IECON.2000.972474.

[264] Madalina M. Drugan. 'Reinforcement Learning Versus Evolutionary Computation: A Survey on Hybrid Algorithms'. In: *Swarm and Evolutionary Computation* 44 (2019), pp. 228–246. DOI: 10.1016/j.swevo.2018.03.011.

[265] David E. Moriarty, Alan C. Schultz, and John J. Grefenstette. 'Evolutionary Algorithms for Reinforcement Learning'. In: *Journal of Artificial Intelligence Research* 11 (1999), pp. 241–276.

[266] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. *Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning*. 2017. DOI: 10.48550/arXiv.1712.06567.

[267] Jorge Gomes, Pedro Mariano, and Anders Lyhne Christensen. 'Cooperative Coevolution of Partially Heterogeneous Multiagent Systems'. In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. AAMAS '15. Istanbul, Turkey: International Foundation for Autonomous Agents and Multiagent Systems, 2015, pp. 297–305.

[268] Dario Floreano and Stefano Nolfi. 'God Save the Red Queen! Competition in Co-Evolutionary Robotics'. In: *Genetic Programming 1997: Proceedings of the Second Annual Conference*. San Francisco, CA, USA: Morgan Kaufmann, 1997, pp. 398–406.

[269] Dario Floreano, Peter Dürr, and Claudio Mattiussi. 'Neuroevolution: From Architectures to Learning'. In: *Evolutionary Intelligence* 1.1 (2008), pp. 47–62. DOI: 10.1007/s12065-007-0002-4.

[270] Kenneth O. Stanley and Risto Miikkulainen. 'Evolving Neural Networks Through Augmenting Topologies'. In: *Evolutionary Computation* 10.2 (June 2002), pp. 99–127. DOI: 10.1162/106365602320169811.

[271] Peter Dürr, Claudio Mattiussi, and Dario Floreano. 'Neuroevolution with Analog Genetic Encoding'. In: *Parallel Problem Solving from Nature - PPSN IX*. Ed. by Thomas Philip Runarsson, Hans-Georg Beyer, Edmund Burke, Juan J. Merelo-Guervós, L. Darrell Whitley, and Xin Yao. Berlin, Heidelberg: Springer, 2006, pp. 671–680. DOI: 10.1007/11844297_68.

[272] Stefano Nolfi, Domenico Parisi, and Jeffrey L. Elman. 'Learning and Evolution in Neural Networks'. en. In: *Adaptive Behavior* 3.1 (June 1994), pp. 5–28. DOI: 10.1177/105971239400300102.

[273] Stefano Nolfi and Anne Spalanzani. 'Learning and Evolution: On the Effects of Directional Learning'. en. In: *Artificial Life* (2000), pp. 1–16.

[274] David Ha and Jürgen Schmidhuber. 'Recurrent World Models Facilitate Policy Evolution'. In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. NIPS'18. Montréal, Canada: Curran Associates, Inc., 2018, pp. 2455–2467.

[275] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. *OpenAI Gym*. 2016. arXiv: 1606.01540 [cs.LG].

[276] Anders Lyhne Christensen and Marco Dorigo. 'Evolving an Integrated Phototaxis and Hole-Avoidance Behavior for a Swarm-bot'. In: *Proceedings of the 10th International Conference on the Simulation and Synthesis of Living Systems (Alife X)*. 2006, pp. 248–254.

[277] Simon Jones, Alan F.T. Winfield, Sabine Hauert, and Matthew Studley. 'Onboard Evolution of Understandable Swarm Behaviors'. In: *Advanced Intelligent Systems* 1.6 (2019), p. 1900031. DOI: 10.1002/aisy.201900031.

[278] Jacqueline Heinerman, Alessandro Zonta, Evert Haasdijk, and Agoston E. Eiben. 'On-line Evolution of Foraging Behaviour in a Population of Real Robots'. In: *Applications of Evolutionary Computation*. Ed. by Giovanni Squillero and Paolo Burelli. Cham: Springer, 2016, pp. 198–212. DOI: 10.1007/978-3-319-31153-1_14.

[279] Marco Dorigo et al. 'Evolving Self-Organizing Behaviors for a Swarm-bot'. In: *Autonomous Robots* 17.2 (2004), pp. 223–245. DOI: 10.1023/B:AURO.0000033973.24945.f3.

[280] Roderich Gross and Marco Dorigo. 'Towards Group Transport by Swarms of Robots'. In: *International Journal of Bio-Inspired Computation* 1.1-2 (2009), pp. 1–13. DOI: 10.1504/IJBIC.2009.022770.

[281] Nathanael Aubert-Kato et al. 'Evolutionary Optimization of Self-Assembly in a Swarm of Bio-Micro-Robots'. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '17. Berlin, Germany: Association for Computing Machinery, 2017, pp. 59–66. DOI: 10.1145/3071178.3071289.

[282] Joshua P. Hecker, Kenneth Letendre, Karl Stolleis, Daniel Washington, and Melanie E. Moses. 'Formica ex Machina: Ant Swarm Foraging from Physical to Virtual and Back Again'. In: *ANTS 2012: Swarm Intelligence*. Ed. by Marco Dorigo, Mauro Birattari, Christian Blum, Anders Lyhne Christensen, Andries P. Engelbrecht, Roderich Groß, and Thomas Stützle. Vol. 7461. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 252–259. DOI: 10.1007/978-3-642-32650-9_25.

[283] John Ericksen, Melanie Moses, and Stephanie Forrest. 'Automatically Evolving a General Controller for Robot Swarms'. In: *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2017, pp. 1–8. DOI: 10.1109/SSCI.2017.8285399.

[284] Karl Sims. 'Evolving 3d Morphology and Behavior by Competition'. In: *Artificial Life* 1.4 (July 1994), pp. 353–372. DOI: 10.1162/artl.1994.1.4.353.

[285] Joshua E. Auerbach and Joshua C. Bongard. 'On the Relationship Between Environmental and Morphological Complexity in Evolved Robots'. In: *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*. GECCO '12. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 2012, pp. 521–528. DOI: 10.1145/2330163.2330238.

[286] Hod Lipson and Jordan B. Pollack. 'Automatic Design and Manufacture of Robotic Lifeforms'. In: *Nature* 406.6799 (2000), pp. 974–978. DOI: 10.1038/35023115.

[287] Pablo Funes and Jordan Pollack. 'Evolutionary Body Building: Adaptive Physical Designs for Robots'. In: *Artificial Life* 4.4 (1998), pp. 337–357. DOI: 10.1162/106454698568639.

[288] Fernando Silva, Miguel Duarte, Sancho Oliveira, Luís Correia, and Anders Christensen. 'The Case for Engineering the Evolution of Robot Controllers'. In: *Artificial Life Conference Proceedings* 26 (2014), pp. 703–710. DOI: 10.1162/978-0-262-32621-6-ch113.

[289] Nick Jakobi, Phil Husbands, and Inman Harvey. 'Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics'. In: *Advances in Artificial Life*. Ed. by Federico Morán, Alvaro Moreno, Juan Julián Merelo, and Pablo Chacón. Berlin, Heidelberg: Springer, 1995, pp. 704–720.

[290] Nicolas Bredeche and Nicolas Fontbonne. 'Social Learning in Swarm Robotics'. In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 377.1843 (2022), p. 20200309. DOI: 10.1098/rstb.2020.0309.

[291] Nicolas Bredeche, Evert Haasdijk, and Abraham Prieto. 'Embodied Evolution in Collective Robotics: A Review'. In: *Frontiers in Robotics and AI* 5 (2018), p. 12. DOI: 10.3389/frobt.2018.00012.

[292] Agoston E. Eiben, Serge Kernbach, and Evert Haasdijk. 'Embodied Artificial Evolution'. In: *Evolutionary Intelligence* 5.4 (2012), pp. 261–272. DOI: 10.1007/s12065-012-0071-x.

[293] Jacqueline Heinerman, Massimiliano Rango, and Agoston E. Eiben. 'Evolution, Individual Learning, and Social Learning in a Swarm of Real Robots'. In: *2015 IEEE Symposium Series on Computational Intelligence*. 2015, pp. 1055–1062. DOI: 10.1109/SSCI.2015.152.

[294] Roderich Groß, Yue Gu, Wei Li, and Melvin Gauci. 'Generalizing GANs: A Turing Perspective'. In: *Advances in Neural Information Processing Systems (NIPS)*. 2017, pp. 6319–6329.

[295] Wei Li, Melvin Gauci, and Roderich Groß. 'Turing Learning: A Metric-Free Approach to Inferring Behavior and its Application to Swarms'. In: *Swarm Intelligence* 10.3 (Sept. 2016), pp. 211–243. DOI: 10.1007/s11721-016-0126-1.

[296] Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. 'Quality Diversity: A New Frontier for Evolutionary Computation'. In: *Frontiers in Robotics and AI* 3 (2016), p. 40. DOI: 10.3389/frobt.2016.00040.

[297] Mark A. Bedau, John S. McCaskill, Norman H. Packard, Steen Rasmussen, Chris Adami, David G. Green, Takashi Ikegami, Kunihiko Kaneko, and Thomas S. Ray. 'Open Problems in Artificial Life'. In: *Artificial Life* 6.4 (Sept. 2000), pp. 363–376. doi: 10.1162/106454600300103683.

[298] Joel Lehman and Kenneth O. Stanley. 'Abandoning Objectives: Evolution Through the Search for Novelty Alone'. In: *Evolutionary Computation* 19.2 (2011), pp. 189–223. doi: 10.1162/EVCO_a_00025.

[299] Daniele Gravina, Antonios Liapis, and Georgios Yannakakis. 'Surprise Search: Beyond Objectives and Novelty'. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. GECCO '16. Denver, Colorado, USA: Association for Computing Machinery, 2016, pp. 677–684. doi: 10.1145/2908812.2908817.

[300] Leo Cazenille, Nicolas Bredeche, and Nathanael Aubert-Kato. 'Exploring Self-assembling Behaviors in a Swarm of Bio-Micro-Robots using Surrogate-Assisted MAP-Elites'. In: *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2019, pp. 238–246. doi: 10.1109/SSCI44817.2019.9003047.

[301] Vassilis Vassiliades, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. 'Using Centroidal Voronoi Tessellations to Scale Up the Multidimensional Archive of Phenotypic Elites Algorithm'. In: *IEEE Transactions on Evolutionary Computation* 22.4 (2018), pp. 623–630. doi: 10.1109/TEVC.2017.2735550.

[302] Matthew C. Fontaine, Julian Togelius, Stefanos Nikolaidis, and Amy K. Hoover. 'Covariance Matrix Adaptation for the Rapid Illumination of Behavior Space'. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. GECCO '20. Cancún, Mexico: Association for Computing Machinery, 2020, pp. 94–102. doi: 10.1145/3377930.3390232.

[303] Tae Jong Choi and Julian Togelius. 'Self-referential Quality Diversity through Differential MAP-Elites'. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '21. Lille, France: Association for Computing Machinery, 2021, pp. 502–509. doi: 10.1145/3449639.3459383.

[304] Jean-Baptiste Mouret and Glenn Maguire. 'Quality Diversity for Multi-Task Optimization'. In: *Proc. of the 2020 Genetic and Evolutionary Comput. Conf.* GECCO '20. Cancún, Mexico: Assoc. for Comput. Machinery, 2020, pp. 121–129. doi: 10.1145/3377930.3390203.

[305] Daniele Gravina, Antonios Liapis, and Georgios N. Yannakakis. 'Blending Notions of Diversity for MAP-Elites'. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '19. Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 117–118. doi: 10.1145/3319619.3321987.

[306] Joel Lehman and Kenneth O. Stanley. 'Evolving a Diversity of Virtual Creatures through Novelty Search and Local Competition'. In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. GECCO '11. Dublin, Ireland: Association for Computing Machinery, 2011, pp. 211–218. doi: 10.1145/2001576.2001606.

[307] Daniele Gravina, Antonios Liapis, and Georgios N. Yannakakis. 'Quality Diversity Through Surprise'. In: *IEEE Transactions on Evolutionary Computation* 23.4 (Aug. 2019), pp. 603–616. doi: 10.1109/tevc.2018.2877215.

[308] Giuseppe Paolo, Alexandre Coninx, Stephane Doncieux, and Alban Laflaquière. 'Sparse Reward Exploration via Novelty Search and Emitters'. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '21. Lille, France: Association for Computing Machinery, 2021, pp. 154–162. doi: 10.1145/3449639.3459314.

[309] Angelo Cangelosi and Matthew Schlesinger. *Developmental Robotics: From Babies to Robots*. The MIT Press, 2014.

[310] Pierre-Yves Oudeyer and Frederic Kaplan. 'What is Intrinsic Motivation? A Typology of Computational Approaches'. In: *Frontiers in Neurorobotics* 1 (2009), p. 6. doi: 10.3389/neuro.12.006.2007.

[311] Marco Mirolli and Gianluca Baldassarre. 'Functions and Mechanisms of Intrinsic Motivations'. In: *Intrinsically Motivated Learning in Natural and Artificial Systems*. Ed. by Gianluca Baldassarre and Marco Mirolli. Berlin, Heidelberg: Springer, 2013, pp. 49–72. doi: 10.1007/978-3-642-32375-1_3.

[312] Kathryn Merrick. 'Value Systems for Developmental Cognitive Robotics: A Survey'. In: *Cognitive Systems Research* 41 (2017), pp. 38–55. doi: 10.1016/j.cogsys.2016.08.001.

[313] Richard M. Ryan and Edward L. Deci. 'Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions'. In: *Contemporary Educational Psychology* 25.1 (2000), pp. 54–67. DOI: 10.1006/ceps.1999.1020.

[314] Gianluca Baldassarre. 'What are Intrinsic Motivations? A Biological Perspective'. In: *2011 IEEE International Conference on Development and Learning (ICDL)*. Vol. 2. 2011, pp. 1–8. DOI: 10.1109/DEVLRN.2011.6037367.

[315] Christoph Salge, Cornelius Glackin, and Daniel Polani. 'Changing the Environment Based on Empowerment as Intrinsic Motivation'. In: *Entropy* 16.5 (2014), pp. 2789–2819. DOI: 10.3390/e16052789.

[316] Jürgen Schmidhuber. 'A Possibility for Implementing Curiosity and Boredom in Model-Building Neural Controllers'. In: *Proceedings of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats*. 1991, pp. 222–227.

[317] Ralf Der and Georg Martius. *The Playful Machine: Theoretical Foundation and Practical Realization of Self-organizing Robots*. Berlin, Heidelberg: Springer, 2012. DOI: 10.1007/978-3-642-20253-7.

[318] Alexander S. Klyubin, Daniel Polani, and Chrystopher L. Nehaniv. 'All Else Being Equal Be Empowered'. In: *Advances in Artificial Life*. Ed. by Mathieu S. Capcarrère, Alex A. Freitas, Peter J. Bentley, Colin G. Johnson, and Jon Timmis. Berlin, Heidelberg: Springer, 2005, pp. 744–753. DOI: 10.1007/11553090_75.

[319] Jürgen Schmidhuber. 'Curious Model-Building Control Systems'. In: *Proceedings of the 1991 IEEE International Joint Conference on Neural Networks*. Vol. 2. 1991, pp. 1458–1463. DOI: 10.1109/IJCNN.1991.170605.

[320] Pierre-Yves Oudeyer, Frederic Kaplan, and Verena V. Hafner. 'Intrinsic Motivation Systems for Autonomous Mental Development'. In: *IEEE Transactions on Evolutionary Computation* 11.2 (2007), pp. 265–286. DOI: 10.1109/TEVC.2006.890271.

[321] Jan Metzen and Frank Kirchner. 'Incremental Learning of Skill Collections Based on Intrinsic Motivation'. In: *Frontiers in Neurorobotics* 7 (2013), p. 11. DOI: 10.3389/fnbot.2013.00011.

[322] Xiao Huang and John Weng. *Novelty and Reinforcement Learning in the Value System of Developmental Robots*. Ed. by Christopher G. Prince, Yiannis Demiris, Yuval Marom, Hideki Kozima, and Christian Balkenius. 2002.

[323] Nihat Ay, Nils Bertschinger, Ralf Der, Frank Güttler, and Eckehard Olbrich. 'Predictive Information and Explorative Behavior of Autonomous Robots'. In: *The European Physical Journal B* 63.3 (2008), pp. 329–339. DOI: 10.1140/epjb/e2008-00175-0.

[324] Marcus M. Scheunemann, Christoph Salge, Daniel Polani, and Kerstin Dautenhahn. *Human Perception of Intrinsically Motivated Autonomy in Human-Robot Interaction*. 2020. arXiv: 2002.05936 [cs.RO].

[325] Martin Biehl, Christian Guckelsberger, Christoph Salge, Simón C. Smith, and Daniel Polani. 'Expanding the Active Inference Landscape: More Intrinsic Motivations in the Perception-Action Loop'. In: *Frontiers in Neurorobotics* 12 (2018). DOI: 10.3389/fnbot.2018.00045.

[326] Joshua Achiam and Shankar Sastry. *Surprise-Based Intrinsic Motivation for Deep Reinforcement Learning*. 2017.

[327] Thomas Parr, Giovanni Pezzulo, and Karl J. Friston. *Active Inference: The Free Energy Principle in Mind, Brain, and Behavior*. The MIT Press, Mar. 2022. DOI: 10.7551/mitpress/12441.001.0001.

[328] Glen Berseth, Daniel Geng, Coline Devin, Nicholas Rhinehart, Chelsea Finn, Dinesh Jayaraman, and Sergey Levine. *SMiRL: Surprise Minimizing Reinforcement Learning in Dynamic Environments*. 2020. arXiv: 1912.05510 [cs.LG].

[329] Md Mohiuddin Khan, Kathryn Kasmarik, and Michael Barlow. 'Toward Computational Motivation for Multi-Agent Systems and Swarms'. In: *Frontiers in Robotics and AI* 5 (2018), p. 134. DOI: 10.3389/frobt.2018.00134.

[330] Philippe Capdepuy, Daniel Polani, and Chrystopher L. Nehaniv. 'Maximization of Potential Information Flow as a Universal Utility for Collective Behaviour'. In: *2007 IEEE Symposium on Artificial Life*. 2007, pp. 207–213. DOI: 10.1109/ALIFE.2007.367798.

[331] Alexander S. Klyubin, Daniel Polani, and Chrystopher L. Nehaniv. 'Empowerment: A Universal Agent-centric Measure of Control'. In: *2005 IEEE Congress on Evolutionary Computation*. Vol. 1. 2005, pp. 128–135. DOI: 10.1109/CEC.2005.1554676.

[332] Mikhail Prokopenko, Vadim Gerasimov, and Ivan Tanev. 'Evolving Spatiotemporal Coordination in a Modular Robotic System'. In: *From Animals to Animats 9*. Ed. by Stefano Nolfi, Gianluca Baldassarre, Raffaele Calabretta, John C. T. Hallam, Davide Marocco, Jean-Arcady Meyer, Orazio Miglino, and Domenico Parisi. Berlin, Heidelberg: Springer, 2006, pp. 558–569. DOI: 10.1007/11840541_46.

[333] Peter Thestrup Waade, Nace Mikus, and Christoph Mathys. 'Inferring in Circles: Active Inference in Continuous State Space Using Hierarchical Gaussian Filtering of Sufficient Statistics'. In: *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. Ed. by Michael Kamp et al. Cham: Springer International Publishing, 2021, pp. 810–818. DOI: 10.1007/978-3-030-93736-2_57.

[334] Karl Friston. 'A Theory of Cortical Responses'. In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 360.1456 (2005), pp. 815–836. DOI: 10.1098/rstb.2005.1622.

[335] Karl Friston, James Kilner, and Lee Harrison. 'A Free Energy Principle for the Brain'. In: *Journal of Physiology-Paris* 100.1 (2006). Theoretical and Computational Neuroscience: Understanding Brain Functions, pp. 70–87. DOI: 10.1016/j.jphysparis.2006.10.001.

[336] Karl Friston. 'The History of the Future of the Bayesian Brain'. In: *NeuroImage* 62.2 (2012). 20 Years of fMRI, pp. 1230–1233. DOI: 10.1016/j.neuroimage.2011.10.004.

[337] Plato. *Republic*. Vol. VII. 375 BC.

[338] Hermann von Helmholtz. *Handbuch der physiologischen Optik*. Leipzig, Germany: Ludwig Voss, 1867.

[339] Karl J. Friston, Jean Daunizeau, James Kilner, and Stefan J. Kiebel. 'Action and Behavior: A Free-Energy Formulation'. In: *Biological Cybernetics* 102.3 (2010), pp. 227–260. DOI: 10.1007/s00422-010-0364-z.

[340] Karl Friston, Christopher Thornton, and Andy Clark. 'Free-Energy Minimization and the Dark-Room Problem'. In: *Frontiers in Psychology* 3 (2012), p. 130. DOI: 10.3389/fpsyg.2012.00130.

[341] Philipp Schwartenbeck, Thomas FitzGerald, Ray Dolan, and Karl Friston. 'Exploration, Novelty, Surprise, and Free Energy Minimization'. In: *Frontiers in Psychology* 4 (2013), p. 710. DOI: 10.3389/fpsyg.2013.00710.

[342] Vito Trianni, Elio Tuci, Kevin M. Passino, and James A. R. Marshall. 'Swarm Cognition: An Interdisciplinary Approach to the Study of Self-Organising Biological Collectives'. In: *Swarm Intelligence* 5.1 (2011), pp. 3–18. DOI: 10.1007/s11721-010-0050-8.

[343] Tanja Katharina Kaiser and Heiko Hamann. *Innate Motivation for Robot Swarms by Minimizing Surprise: From Simple Simulations to Real-World Experiments (plots, data and video)*. Apr. 2021. DOI: 10.5281/zenodo.4665989. CC BY 4.0.

[344] Zachary C. Lipton, John Berkowitz, and Charles Elkan. *A Critical Review of Recurrent Neural Networks for Sequence Learning*. arXiv. 2015. arXiv: 1506.00019 [cs.LG].

[345] Elizabeth A. Quattrocki and Karl Friston. 'Autism, Oxytocin and Interoception'. In: *Neuroscience & Biobehavioral Reviews* 47 (2014), pp. 410–430. DOI: 10.1016/j.neubiorev.2014.09.012.

[346] Mirko Nava, Jérôme Guzzi, R. Omar Chavez-Garcia, Luca M. Gambardella, and Alessandro Giusti. 'Learning Long-Range Perception Using Self-Supervision From Short-Range Sensors and Odometry'. In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1279–1286. DOI: 10.1109/LRA.2019.2894849.

[347] Edmund Ronald and Marc Schoenauer. 'Genetic Lander: An Experiment in Accurate Neuro-genetic Control'. In: *Parallel Problem Solving from Nature — PPSN III*. Ed. by Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer. Berlin, Heidelberg: Springer, 1994, pp. 452–461. DOI: 10.1007/3-540-58484-6_288.

[348] Payam Zahadat, Heiko Hamann, and Thomas Schmickl. 'Evolving Diverse Collective Behaviors Independent of Swarm Density'. In: *Workshop Evolving Collective Behaviors in Robotics (GECCO 2015)*. [Extended Abstract]. ACM, 2015, pp. 1245–1246. DOI: 10.1145/2739482.2768492.

[349] Payam Zahadat, Heiko Hamann, and Thomas Schmickl. 'Evolving Collective Behaviors With Diverse But Predictable Sensor States'. In: *13th European Conference on Artificial Life (ECAL 2015)*. [Extended Abstract]. MIT Press, 2015, p. 174.

[350] Richard P. Feynman, Robert B. Leighton, and Matthew Sands. *The Feynman Lectures on Physics Vol. 1*. Vol. 1. New York: Basic Books, 2010.

[351] Stéphane Doncieux, Jean-Baptiste Mouret, Nicolas Bredeche, and Vincent Padois. 'Evolutionary Robotics: Exploring New Horizons'. In: *New Horizons in Evolutionary Robotics*. Ed. by Stéphane Doncieux, Nicolas Bredeche, and Jean-Baptiste Mouret. Berlin, Heidelberg: Springer, 2011, pp. 3–25. DOI: 10.1007/978-3-642-18272-3_1.

[352] Salvador García, Daniel Molina, Manuel Lozano, and Francisco Herrera. 'A Study on the Use of Non-Parametric Tests for Analyzing the Evolutionary Algorithms' Behaviour: A Case Study on the CEC'2005 Special Session on Real Parameter Optimization'. In: *Journal of Heuristics* 15.6 (2008), p. 617. DOI: 10.1007/s10732-008-9080-4.

[353] Henry B. Mann and Donald R. Whitney. 'On a Test of Whether One of Two Random Variables is Stochastically Larger Than the Other'. In: *The Annals of Mathematical Statistics* (1947), pp. 50–60.

[354] William H. Kruskal and W. Allen Wallis. 'Use of Ranks in One-Criterion Variance Analysis'. In: *Journal of the American Statistical Association* 47.260 (1952), pp. 583–621.

[355] Carlo Bonferroni. 'Teoria Statistica Delle Classi e Calcolo Delle Probabilita'. In: *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commericiali di Firenze* 8 (1936), pp. 3–62.

[356] Paolo Pagliuca and Stefano Nolfi. 'Robust Optimization Through Neuroevolution'. In: *PLOS ONE* 14.3 (Mar. 2019), pp. 1–27. DOI: 10.1371/journal.pone.0213193.

[357] Iñaki Fernández Pérez, Amine Boumaza, and François Charpillet. 'Learning Collaborative Foraging in a Swarm of Robots Using Embodied Evolution'. In: *ECAL 2017, the Fourteenth European Conference on Artificial Life*. MIT Press. 2017, pp. 162–161. DOI: 10.7551/ecal_a_028.

[358] Dominik Fischer, Sanaz Mostaghim, and Larissa Albantakis. 'How Swarm Size During Evolution Impacts the Behavior, Generalizability, and Brain Complexity of Animats Performing a Spatial Navigation Task'. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '18. Kyoto, Japan: Association for Computing Machinery, 2018, pp. 77–84. DOI: 10.1145/3205455.3205646.

[359] Sondre A. Engebraaten, Jonas Moen, Oleg A. Yakimenko, and Kyrre Glette. 'A Framework for Automatic Behavior Generation in Multi-Function Swarms'. In: *Frontiers in Robotics and AI* 7 (2020), p. 175. DOI: 10.3389/frobt.2020.579403.

[360] Ronald Aylmer Fisher. 'Statistical Methods for Research Workers'. In: *Breakthroughs in Statistics*. Springer, 1992, pp. 66–70.

[361] Jorge Gomes, Pedro Mariano, and Anders Lyhne Christensen. 'Devising Effective Novelty Search Algorithms: A Comprehensive Empirical Study'. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. GECCO '15. Madrid, Spain: Association for Computing Machinery, 2015, pp. 943–950. DOI: 10.1145/2739480.2754736.

[362] Joel Lehman and Kenneth O. Stanley. 'Efficiently Evolving Programs Through the Search for Novelty'. In: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*. GECCO '10. Portland, Oregon, USA: Association for Computing Machinery, 2010, pp. 837–844. DOI: 10.1145/1830483.1830638.

[363] Linge Bai and David Breen. 'Calculating Center of Mass in an Unbounded 2D Environment'. In: *Journal of Graphics Tools* 13.4 (2008), pp. 53–60. DOI: 10.1080/2151237X.2008.10129266.

[364] Jônata Tyska Carvalho and Stefano Nolfi. 'Behavioural Plasticity in Evolving Robots'. In: *Theory in Biosciences* 135.4 (2016), pp. 201–216. DOI: 10.1007/s12064-016-0233-y.

[365] Karine Miras, Eliseo Ferrante, and Agoston E. Eiben. 'Environmental Influences on Evolvable Robots'. In: *PloS one* 15.5 (2020), e0233848. DOI: 10.1371/journal.pone.0233848.

[366] Merriam-Webster. *"Curiosity". Merriam-Webster.com Dictionary*. 2022. URL: https://www.merriam-webster.com/dictionary/curiosity (visited on 02/07/2022).

[367] Celeste Kidd and Benjamin Y. Hayden. 'The Psychology and Neuroscience of Curiosity'. In: *Neuron* 88.3 (2015), pp. 449–460. DOI: 10.1016/j.neuron.2015.09.010.

[368] Pierre-Yves Oudeyer, Frédéric Kaplan, Verena V. Hafner, and Andrew Whyte. 'The Playground Experiment: Task-Independent Development of a Curious Robot'. In: *Proceedings of the AAAI spring symposium on developmental robotics*. Stanford, California. 2005, pp. 42–47.

[369] Nigel R. Franks, Anna Dornhaus, Jon P. Fitzsimmons, and Martin Stevens. 'Speed Versus Accuracy in Collective Decision Making'. In: *Proceedings of the Royal Society of London. Series B: Biological Sciences* 270.1532 (2003), pp. 2457–2463. DOI: 10.1098/rspb.2003.2527.

[370] Kevin M. Passino and Thomas D. Seeley. 'Modeling and Analysis of Nest-Site Selection by Honeybee Swarms: The Speed and Accuracy Trade-off'. In: *Behavioral Ecology and Sociobiology* 59.3 (2006), pp. 427–442. DOI: 10.1007/s00265-005-0067-y.

[371] Mostafa Wahby and Heiko Hamann. 'On the Tradeoff Between Hardware Protection and Optimization Success: A Case Study in Onboard Evolutionary Robotics for Autonomous Parallel Parking'. In: *Applications of Evolutionary Computation. EvoApplications 2015*. Ed. by Antonio M. Mora and Giovanni Squillero. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 759–770. DOI: 10.1007/978-3-319-16549-3_61.

[372] Paul Levi and Serge Kernbach. *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution*. Berlin, Heidelberg: Springer, 2010. DOI: 10.1007/978-3-642-11692-6.

[373] Nicolas Bredeche, Evert Haasdijk, and Agoston E. Eiben. 'On-Line, On-Board Evolution of Robot Controllers'. In: *Artifical Evolution*. Ed. by Pierre Collet, Nicolas Monmarché, Pierrick Legrand, Marc Schoenauer, and Evelyne Lutton. Berlin, Heidelberg: Springer, 2010, pp. 110–121. DOI: 10.1007/978-3-642-14156-0_10.

[374] Tanja Katharina Kaiser, Christine Lang, Florian Andreas Marwitz, Christian Charles, Sven Dreier, Julian Petzold, Max Ferdinand Hannawald, Marian Johannes Begemann, and Heiko Hamann. *An Innate Motivation to Tidy Your Room: Online Onboard Evolution of Manipulation Behaviors in a Robot Swarm (plots and video)*. Nov. 2020. DOI: 10.5281/zenodo.4293487. CC BY 4.0.

[375] Michael Allwright, Weixu Zhu, and Marco Dorigo. 'An Open-Source Multi-robot Construction System'. In: *HardwareX* 5 (2019), e00050. DOI: 10.1016/j.ohx.2018.e00050.

[376] Norman Packard, Mark A. Bedau, Alastair Channon, Takashi Ikegami, Steen Rasmussen, Kenneth O. Stanley, and Tim Taylor. 'An Overview of Open-Ended Evolution: Editorial Introduction to the Open-Ended Evolution II Special Issue'. In: *Artificial Life* 25.2 (May 2019), pp. 93–103. DOI: 10.1162/artl_a_00291.

[377] Mitchell A. Potter and Kenneth A. De Jong. 'A cooperative coevolutionary approach to function optimization'. In: *Parallel Problem Solving from Nature — PPSN III*. Ed. by Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 249–257. DOI: 10.1007/3-540-58484-6_269.

[378] Agoston E. Eiben, Emma Hart, Jon Timmis, Andy M. Tyrrell, and Alan F. Winfield. 'Towards Autonomous Robot Evolution'. In: *Software Engineering for Robotics*. Ed. by Ana Cavalcanti, Brijesh Dongol, Rob Hierons, Jon Timmis, and Jim Woodcock. Cham: Springer International Publishing, 2021, pp. 29–51. DOI: 10.1007/978-3-030-66494-7_2.

[379] Fabian Pedregosa et al. 'Scikit-learn: Machine Learning in Python'. In: *Journal of Machine Learning Research* 12.85 (2011), pp. 2825–2830.

[380] Diederik P. Kingma and Jimmy Ba. 'Adam: A Method for Stochastic Optimization'. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[381] Alexander Gepperth and Barbara Hammer. 'Incremental Learning Algorithms and Applications'. In: *European symposium on artificial neural networks (ESANN)*. 2016.