Synthesis and Analysis of Minimalist Control Strategies for Swarm Robotic Systems

Anil Ozdemir

Supervisors: Dr Roderich Groß
Dr Andreas Kolling

Department of Automatic Control and Systems Engineering
The University of Sheffield

A Thesis Submitted for the Degree of Doctor of Philosophy

April 2020

Acknowledgements

I would like to start by thanking my supervisor Dr. Roderich Groß for his encouragement and guidance throughout my PhD studies. This research would not be possible without his constant support. His scientific knowledge and expertise were essential in my transition from student to researcher, and I am extremely grateful for all the knowledge I have gained from working with him. I would also like to thank to my second supervisor Dr. Andreas Kolling, for our fruitful scientific discussions.

Additionally I am grateful to all the members of Natural Robotics Lab—Chris, Fernando, Yuri, Gabriel, Stefan, Matt Doyle, João, Yue, Isaac, and Matt Hall. I would also like to thank F01 folks—my office-mates and colleagues—Felipe, Manal, Omar, Laura, Eduardo, Dan, John, Blayze, and Zeke.

I would like to give special thanks to my collaborator Dr. Melvin Gauci for his constant support during endless nights of submissions. Also, I would like to thank to my other collaborators Salomé Bonnet and Matt Hall, for all their help with the e-pucks. I am particularly grateful to Dr Daniela Rus and John Romanishin from MIT, for enabling the development of gathering control strategy and testing them in the M-Block robotic platform. Lastly, I would like to thank to João Marques for everything he taught be about modular robots, and keeping me sane during writing this thesis.

I also want to acknowledge my family for supporting and helping me to move abroad for my research—I am eternally grateful for you always being there for me! A final special thanks goes to Huseyin the Hafiz, for encouraging me throughout my studies and pushing me to pursue a PhD. Finally, I close with my significant other, Hope. Thanks for supporting and encouraging me to overcome any obstacle. Nay nay nu...

Abstract

The field of swarm robotics studies bio-inspired cooperative control strategies for large groups of relatively simple robots. The robots are limited in their individual capabilities, however, by inducing cooperation amongst them, the limitations can be overcome. Local sensing and interactions within the robotic swarm promote scalable, robust, and flexible behaviours. This thesis focuses on synthesising and analysing minimalist control strategies for swarm robotic systems. Using a computation-free swarming framework, multiple decentralised control strategies are synthesised and analysed. The control strategies enable the robots—equipped with only discrete-valued sensors—to reactively respond to their environment. We present the simplest control solutions to date to four multi-agent problems: finding consensus, gathering on a grid, shepherding, and spatial coverage. The control solutions—obtained by employing an offline evolutionary robotics approach—are tested, either in computer simulation or by physical experiment. They are shown to be—up to a certain extent—scalable, robust against sensor noise, and flexible to the changes in their environment. The investigated gathering problem is proven to be unsolvable using the deterministic framework. The extended framework, using stochastic reactive controllers, is applied to obtain provably correct solutions. Using no run-time memory and only limited sensing make it possible to realise implementations that are arguably free of arithmetic computation. Due to the low computational demands, the control solutions may enable or inspire novel applications, for example, in nanomedicine.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

In nature, social animal groups have been observed displaying self-organised behaviours (Bak, 1996; Camazine et al., 2001; Krause and Ruxton, 2002). The collective behaviour of the group is not dictated by a central entity, but rather emerges from the individuals' interactions with each other and the environment. Researchers observed that the cooperative behaviour of social animals is a result of the decentralised organisation of their group structure (Bonabeau et al., 1999). For example, ant colonies self-organise themselves for finding rich food sources and accordingly, selecting a nest site. The self-organisation phenomenon has led researchers to emulate such concepts in *artificial swarm systems* (Beni, 2005; Dorigo and Şahin, 2004). Taking inspiration from social animal behaviours, such as swarming of insect colonies, schooling of fish, and flocking of birds, *swarm robotics* offers design and coordination solutions for a large number of relatively simple robots (Şahin et al., 2008; Trianni, 2008).

Robotic systems have been at the core of automation for the past several decades. Their contribution to modern societies is not only to perform repetitive tasks instead of humans—as they do not get bored or tired—but also to offer enhanced capabilities. Some robots perform tasks with astonishing precision, speed, and consistency. Swarm robotic systems excel in certain tasks that require spatial distribution. As the system is composed of many units, the robots can solve tasks in parallel, hence faster, and have high redundancy (Parker, 2008). Moreover, as the robots in the swarm can collaborate, they potentially can exhibit more advanced capabilities than a group of non-interacting robots.

Collective behaviour of robots can emerge from local interactions within the group, enabling it to demonstrate scalability, robustness, and flexibility (Şahin, 2005). All three features are inter-dependent on each other. For instance, a swarm of robots tends to be scalable in the sense that adding more robots to the group would ideally improve performance until they reach the system's full capacity. Likewise, the failure of a few individual units would not affect the system significantly, allowing it to be robust against individual failures.

Another important aspect is that a swarm of robots is composed of simple units, meaning that the individual capabilities are limited. The simplicity of the units hold the potential of scaling them down to the micro- or nano-scale. This is particularly relevant to nanomedical applications, in which a single complex robot cannot be deployed due to space and energy constraints (Requicha, 2013). As such, any robot for these applications would inherently have limited capabilities. Recent advances in Micro/Nano Electro-Mechanical Systems (MEMS/NEMS) have enabled researchers to develop initial robotic prototypes of micro/nano sizes (Jalili, 2013). However at present, these robotic systems cannot utilise sophisticated control architectures, hence, their information acquisition capacities are strictly limited (Mavroidis and Ferreira, 2013; Requicha, 2003). These limitations are in line with the swarm robotic design paradigm, as restricted capabilities of individuals can be elaborated by local interactions within the group. As a result, there has been a growing interest in researching 'minimalist' control strategies for swarms of robots (Brown et al., 2018; Gauci, 2014; Mitrano et al., 2019; Wareham and Vardy, 2018).

1.1 Motivation

One potential application domain of minimalist swarm robots is in nanomedicine (Requicha, 2013). Nanotechnology is an emerging engineering discipline, enabling the production of 'nanorobots' of size near a nanometre, that is, $80000 - 100000$ times smaller than the diameter of a human hair. In this context, a minimal approach for designing swarm robotic systems could be beneficial for two reasons. Firstly, minimal design criteria require less physical features, hence, a robot can be scaled down in size (Requicha, 2013). There are, however, limitations when to scaling-down in size. The nanorobots will not be able to use standard robotic hardware, such as, conventional micro-controllers or sensors (Mavroidis and Ferreira, 2013). Furthermore, energy is particularly crucial at the micro- or nano-scale. Either the robot cannot be equipped with certain devices due to the 'limited' energy availability, or the operation time of the robot would be immensely short.

Secondly, minimal design allows the production of 'affordable', hence 'disposable', units in large numbers (Şahin, 2005). This is important when the nature of the task the robots are performing necessitates compromising a portion of the group. For example, during an extraterrestrial land exploration, it is likely for some of the robots to cease and not return to the operation centre. In addition, if the robotic units can cooperate with each other, without needing any additional means of communication, this would further enhance the attainability of the robotic system. Thus, there is a need to develop control strategies for robotic systems with crucially restricted capabilities in order for them to be implemented at micro- or nano-scales.

In his doctoral thesis, Gauci (2014) proposed *computation-free swarming*[1], a swarm robotic design paradigm to control robots of extreme simplicity. Gauci took a minimalist approach to further simplify the sensor and control architecture needed in a swarm of robots to exhibit collective behaviour. The major reason for such limitations is derived from the motivation of implementing 'minimalist robotic machines' which can operate in, for example, a blood vessel. The tasks robotic swarms solve might be trivial for von Neumann style computers; a central processing unit with dedicated monitoring device might be sufficient to orchestrate the group of robots (Mitchell, 2009). However, such implementation is often impractical in real-world applications, particularly in the human body, where the environment is incredibly stochastic and unpredictable. These constraints make it arduous to utilise a conventional central approach, such as, motion planning or guided locomotion. Thus, a reactive decentralised design could be vital to enable robustness and flexibility in the robotic system for such applications.

The computation-free swarming framework has so far been only tested on a small set of tasks, where the robots interact either only with each other (Gauci et al., 2014c) or with static objects (Gauci et al., 2014b) in a homogeneous environment. It is yet to be seen if this framework is applicable to scenarios where the robots perceive a wider range of features in their environment. Additionally, given the simplicity of the controllers obtained by computation-free swarming, scenarios in which the environment of the robots changes dynamically over time presents a challenge. Furthermore, there is a need to study the limitations of the framework, ideally identifying some problems that cannot be addressed by the present formulation.

[1]The term computation-free here refers to the utilised swarm controllers which are free of arithmetic computation. The controller design phase, on the contrary, is computationally intensive.

1.2 Problem Definition

The main focus of this thesis is to synthesise and analyse effective solutions to challenging swarm robotics problems with a minimalist approach termed *computation-free swarming* (Gauci, 2014). The robotic systems we consider in this thesis utilise reactive and decentralised control architectures; in addition, the swarming agents (robots) lack arithmetic computation units, communication capabilities, or sophisticated sensor units. Using deterministic control solutions obtained through computation-free swarming, Gauci et al. (2014c) demonstrated that swarms of robots can accomplish self-organised aggregation, and can cluster initially dispersed passive objects (Gauci et al., 2014b). However, the previous work is only one of the first steps towards easing the transition from conventional swarm robotics to nanorobotics applications in terms of information processing capabilities, leading to explore, investigate, and develop further.

Extremely minimalist solutions do not yet exist for various scenarios, such as, *spatial coverage*, *finding consensus*, and *shepherding*. In contrast to aggregation and clustering tasks, in spatial coverage, the swarm is tasked to distribute themselves throughout the environment by first diverging then remaining in their positions. In the case of finding consensus, the swarm collectively choose one of multiple options in their environment by congregating at the chosen option. In shepherding task, the robotic swarm—shepherds—interact with a group of dynamic sheep-like agents in order to herd them to a predefined goal location. Additionally, potential shortcomings of the deterministic control approach, within the concept of binary sensing and actuation, are to be discovered by studying the multi-agent *gathering* problem.

There is, nevertheless, a compromise to realise the 'reductionist' approach in order to minimalise the robotic system. A significant effort has to be made to reduce the complexity of the system, in both physical components and the control architecture (Wolpert and Macready, 1995). In this thesis, we employ an evolutionary robotics approach (Nolfi and Floreano, 2000; Trianni, 2008) to synthesise and analyse control strategies to four problems: spatial coverage, finding consensus, shepherding, and gathering. Each of the problems has pertinent importance and applications. Starting from the minimum amount of information required for robots to operate, we investigate the above four problems with necessary operating conditions. When the swarm's performance is insufficient, we relax the strict conditions by upgrading the information processing requirements and compare the performance trade-offs.

1.3 Aim and Objectives

Given the aforementioned problem statement, the aim of this thesis is twofold. First, to advance state-of-the-art swarm robotic control algorithms by utilising the minimal information processing framework—computation-free swarming. Second, to extend the computation-free swarming framework in multiple directions, including the addition of stochasticity to the robot's control system.

The spebegincific objectives to fulfil the aim are:

- To conduct a current state-of-the-art literature review in swarm robotics, as well as swarm intelligence. Additionally, to identify and compare existing swarm robotics control strategies.
- To synthesise control strategies for further challenging swarm robotics tasks using the computation-free swarming framework. The synthesised strategies shall be followed up by investigating the emerged behaviours.
- To computationally analyse performance and capabilities of the control strategies through multiple numerical studies. Moreover, to fully or partially, prove that the developed control strategies are guaranteed to lead to the desired outcome.
- To evaluate the feasibility of the obtained control strategies in practice by demonstrating them in a physical autonomous differential-wheeled robotic swarm.
- To investigate how the framework can be applied to self-reconfigurable modular robotic systems.
- To improve the capabilities of the framework by combining multiple sensor units.
- To extend the deterministic structure of the framework by allowing the robotic units to choose randomly amongst a limited number of predefined actions.

1.4 Preview of Contributions

The following state the contributions of this thesis:

- Design and implementation of a control strategy for groups of memoryless embodied agents (robots) with single-bit line-of-sight sensors to achieve spatial coverage in a two-dimensional bounded environment. Systematic comparison[2] of the proposed strategy is compared against multiple existing strategies, showing that the strategy outperforms a random walk strategy by 10% more coverage. This study is followed by systematic investigations in different environment models including obstacles—the swarm copes well with the restrictions, although, on average more robots ended up near the boundary resulting in reducing the efficiency in scenarios without robot redundancy. Demonstration[2], for the first time, of a swarm of such extremely simple agents navigating an *a priori* unknown maze. Experiments with a swarm of 25 physical fully autonomous robots, indicating the feasibility of the computationally obtained control solution in a real-world scenario.

- Design and implementation of a novel control strategy to a multi-agent collective choice problem—the first to be free of arithmetic computation, and the simplest solution to date. The agents extract only 1-trit (i.e. 1 ternary digit) of information from their environment using a single line-of-sight sensor. The majority of the agents choose between equal alternatives in 97.3% of the trials. Systematic examination[2] of the problem of choosing between multiple equal alternatives showed that the swarm is able to choose an option from up to four options present, and can cope with unequal alternatives. This is followed by systematic investigations of a large group using up to 100 agents yielding that the swarms' performance is only affected on a minor level by the increasing number of agents. The control strategy is ported onto a swarm of 20 physical autonomous robots, and the obtained results are compared with the simulation studies resulting in a fairly similar performance.

- Despite the complexity of the shepherding task, it is shown that neither arithmetic computation nor run-time memory is fundamentally essential to accomplish the task. The developed control strategy provides the simplest solution to the problem to date, requiring the shepherd agents to extract only 2-bits of information. Comprehensive analyses[2] of the control parameters and sensor noise sensitivity demonstrating the robustness of the shepherding control strategy. Further extension of the control strategy allowing the shepherd agents to extract 2-trits of information from their environment, yielding better scalability with the increased number of agents.

- Extension of the "computation-free" swarming concept (Gauci, 2014):

[2]This is done through a computer simulation that employs physical constraints.

  - by allowing the agents to retrieve multiple sensory information, control policies can take instantaneous spatial configuration into account. The agents can avoid deadlock situations by using stochastic control policy,
  - by allowing the agents to execute an action from a set of eligible actions with a certain probability.

  The extended computation-free swarming concept is then applied to multi-robot gathering problem on 2D grid.

- Mathematical proof demonstrating that there exists no deterministic control solutions for a multi-robot gathering problem on 2D grid.
- Development of two stochastic control policies for the multi-robot gathering problem:
  - *naïve stochastic control policy* chooses an eligible action based on uniform random selection,
  - *optimised stochastic control policy* chooses an eligible action from an optimised set of probability parameters that also takes the specific context of an agent into account.
- For the multi-robot gathering problem mathematical proof that the agents, when using a stochastic control policy (either naïve or optimised), almost surely reach a Pareto optimal spatial alignment in finite time, irrespective of initial positions. Demonstration and systematic comparison[2] of the naïve and optimised control policies for flexibility, scalability and robustness.

1.5 Publications

The work presented in this thesis has led to the following peer-reviewed publications:

1. **A. Özdemir**, J. W. Romanishin, R. Groß, and D. Rus "Decentralized Gathering of Stochastic, Oblivious Agents on a Grid: A Case Study with 3D M-Blocks," *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, IEEE, 2019, pp. 245–251.
2. **A. Özdemir**, M. Gauci, A. Kolling, M. D. Hall, and R. Groß, "Spatial Coverage Without Computation," *2019 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, pp. 9674–9680.

3. **A. Özdemir**, M. Gauci, S. Bonnet, and R. Groß, “Finding Consensus Without Computation,” *IEEE Robotics and Automation Letters*, 3(3), pp. 1346–1353. 2018.

4. **A. Özdemir**, M. Gauci, and R. Groß, “Shepherding with Robots That Do Not Compute,” in *Proceedings of the 14th European Conference on Artificial Life (ECAL)*. MIT Press, 2017, pp. 332—339.

The author orally presented Publication 3 in Brisbane, Australia at 2018 IEEE International Conference on Robotics and Automation (ICRA 2018) and Publication 4 in Lyon, France at the corresponding conference. Additionally, the author contributed to another project that is not featured in this thesis. This work has led to the following publication:

1. J. V. A. Marques, **A. Özdemir**, M. J. Doyle, D. Rus, and R. Groß, “Decentralized Pose Control of Modular Reconfigurable Robots Operating in Liquid Environments,” *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 4855–4861.

1.6 Thesis Overview

The structure of this thesis is organised as follows.

- Chapter 2 provides background and related works for this thesis. Section 2.1 presents the concepts of emergence and emergent behaviour. Section 2.2 presents the design and control principles for swarm robotics, with a detailed emphasis on computation-free swarming. Section 2.3 presents related work to the studies conducted in this thesis in alphabetical order: finding consensus, gathering, shepherding, and spatial coverage.

- Chapter 3 presents spatial coverage study using a swarm of robots. Section 3.1 introduces the coverage problem. Section 3.2 describes the problem objective in detail, including the environment and robot model, and simulation setup. Section 3.3 explains how the control solution is obtained using an evolutionary algorithm. In particular, the evaluation of candidate solutions, the evolutionary algorithm setup, controller selection, and mathematical analysis of the selected controller. Section 3.4 reports simulation studies, including sensory noise analysis, scalability analysis, and navigating a maze. Section 3.5 describes the physical robotic platform and the experiment setup, and reports the results. Section 3.6 concludes the chapter. This chapter is based on the author's original work given in Publication 2.

- Chapter 4 presents a control strategy for a swarm of robots to find consensus in choosing an option. Section 4.1 introduces the consensus problem. Section 4.2 describes the objective, the environment and the robot model, and the simulation setup. Section 4.3 explains the evolutionary algorithm setup that has been used to synthesise the control solution and the evaluation criteria. Section 4.4 analyses the behaviour of the swarm using the synthesised control solution. This includes analysis of sensory noise, scalability of the swarm, choosing between more than two options and between unequal options. Section 4.5 describes the experimental setup, the robotic platform used, and reports the results. Section 4.6 concludes the chapter. This chapter is based on the author's original work given in Publication 3.

- Chapter 5 presents a control strategy for the multi-robot shepherding problem. Section 5.1 introduces the shepherding problem. Section 5.2 describes the objective, the sheep and shepherd model, and the simulation setup. Section 5.3 explains the evolutionary algorithm setup to synthesise the controllers for shepherds, the evaluation criteria, and mathematical analysis of the selected controller. Section 5.4 analyses sensory noise, sensitivity, and scalability of the system. In addition, this section presents an advanced control strategy with enhanced sensing model. Section 5.5 concludes the chapter. This chapter is based on the author's original work given in Publication 4.

- Chapter 6 presents decentralised multi-robot gathering strategies and extends the computation-free swarming framework. Section 6.1 introduces the gathering problem. Section 6.2 describes the objective, the environment and the robot model, and mathematical analysis of the objective. Section 6.3 presents a deterministic control policy, with controller design criteria, as well as, theoretical analysis of impossibility of finding a deterministic controller for the given problem definition. Section 6.4 proposes a naïve stochastic control policy, that alleviates the impossibility of gathering. In addition, a mathematical analysis of the proof of convergence is given. Section 6.5 proposes an enhanced stochastic control policy by optimising probability distributions. Additionally, the representation of candidate solutions, the evolutionary algorithm setup, and selection of the controllers are presented. Furthermore, this sections presents mathematical analysis, an extended version of the proof given in Section 6.4. Section 6.6 reports the simulation studies for scalability and sensory noise analyses. Section 6.7 concludes the chapter. This chapter is based on the author's original work given in Publication 1.

- Chapter 7 summarises this thesis and discusses the contributions. Additionally, Section 7.1 presents potential future directions of the works presented in this thesis.

Chapter 2

Background and Related Work

This chapter presents background context and related work for this thesis. Section 2.1 presents the concept of emergence and emergent behaviour in biological groups, and insights regarding natural swarms and their cooperative behaviours. This is followed by artificial and engineered swarm systems. In Section 2.2, a brief introduction to swarm robotics is given alongside common design methods. Particular attention is given to minimal information processing and computation-free swarming concepts—the primary design method used throughout this thesis. In Section 2.3, swarm robotics related tasks are investigated, particularly those which are the main interests of this thesis, such as finding consensus, gathering, shepherding and spatial coverage.

2.1 Emergence and Emergent Behaviour

The role of emergence is significant in many areas of science, including physics, chemistry, biology, and systems theory (Bak, 1996). Emergence is a concept of a collective entity exhibiting a property which its individual parts do not have. For example, the formation of fractal patterns in snowflakes, ripple patterns in a sand dune, or a mound built by termites. While all these natural phenomena exhibit emergence, the causes behind all may be different. For example, snowflakes form unique fractal patterns based on thermodynamic changes in their own molecules which are triggered by external effects (e.g. air temperature, humidity in the local proxy). These changes are so sensitive such that a small difference in the initial conditions can result in a completely different pattern. Similarly, a sand dune is formed by an external wind effect. On the other hand, there are biological examples such as a mound built by a colony of self-organised

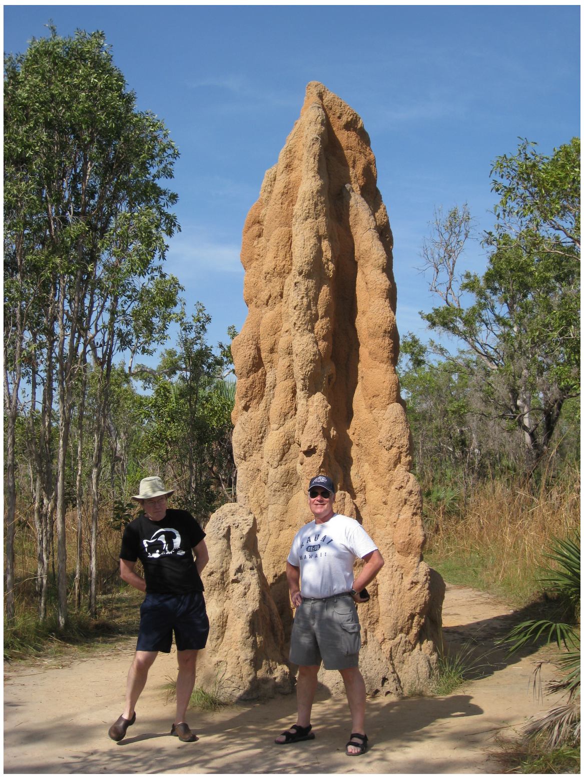


Fig. 2.1 An average termite length is up to 1.5 cm, whilst a termite colony can build a nest mound up to 600 cm in height. That is 400 times the size of a termite. A ratio that is comparable to the tallest modern-day skyscrapers, for example, Burj Khalifa—the tallest human-made building—is 487 times taller than the average human height (considering an average of 1.7 m) and for Shanghai Tower—the second tallest building—the ratio is 371 times. Image is reprinted from Brewbooks (2009).

termites. Only relying on local information sharing, without needing a leader, a colony of termites can excel at cooperatively carrying and allocating nest-building materials (Howse, 1970). Grassé et al. (1984) reported that this simple cooperative behaviour can emerge into a sophisticated nest mound which can be up to 600 cm in height (see Figure 2.1).

Emergent behaviour can be defined as a global behaviour that emerges through local interactions between the members of the whole. It is possible to observe emergent behaviour in nature, in the social life of humans, or in a group of machines (Camazine et al., 2001). Flocking birds, pedestrian crowd movement, or cooperative box pushing by multiple robots are amongst some familiar examples. A common feature between these examples is that they all are self-organised [1]. Self-organisation is a process of

[1]In the literature, it is common that the terms emergent behaviour and self-organisation are used interchangibly (Bonabeau et al., 1999)

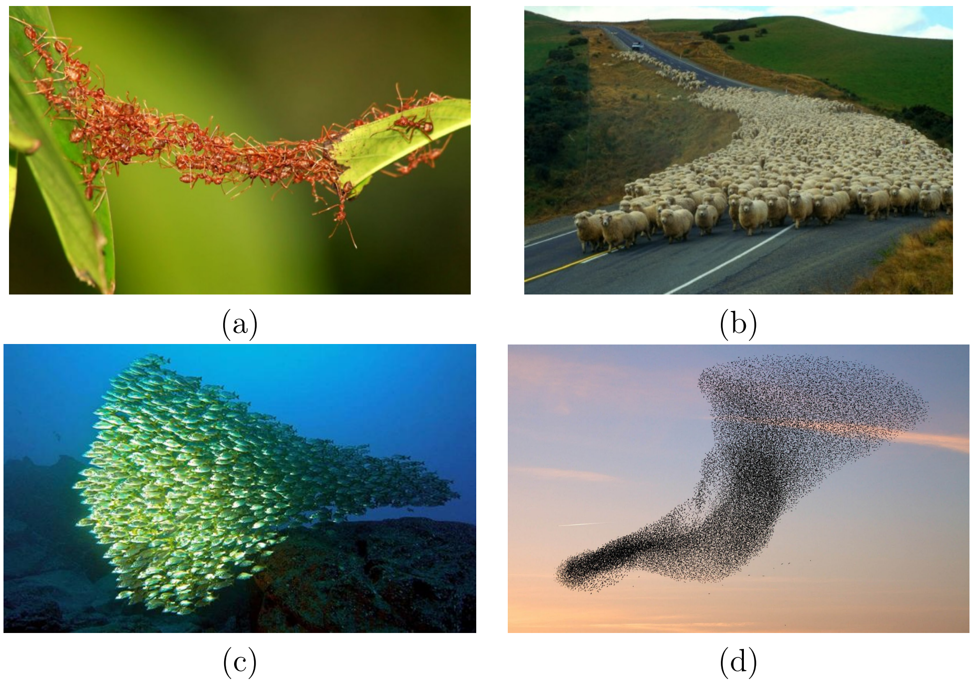


Fig. 2.2 Examples of biological emergent behaviour in biological swarms: (a) swarm of ants, (b) flock of sheep, (c) school of fish, (d) flock of birds. Images are reprinted from (a) Gordon (2013), (b) Carnemolla (2016), (c) 3atoms (2012), (d) Stass (2015).

forming an ordered structure by initially unordered entities through local interactions. More on self-organisation will be explained in Section 2.1.1.

2.1.1 Swarms in Nature

It is fascinating how a swarm of animals cooperate with each other to complete specific tasks, such as gathering food, avoiding predators, or building nest sites (Krause and Ruxton, 2002). Natural swarm behaviours are great sources of inspiration. The observed collective behaviours have led to many applications, especially in computer science and engineering (Beni, 2005; Bonabeau et al., 1999; Bonabeau and Meyer, 2001; Şahin, 2005). For example, numerous bio-inspired optimisation methods have been developed to minimise logistic costs (Dorigo and Blum, 2005), for cooperative transportation of warehouse goods (Chaimowicz et al., 2002; Wurman et al., 2008), or crowd simulation in media (Lin and Chen, 2007).

Biological swarms exhibit impressive flexibility and robustness (Camazine et al., 2001). In these systems, intelligent group behaviours have been observed. Although each individual has limited capabilities, through local communication and information transmission, they are able to complete sophisticated tasks. Social insects are able to accomplish tasks beyond the capabilities of an individual. For example, ants can overcome gaps in their path by attaching to one another, hence forming a 'bridge' [see Figure 2.2(a)]. The studies conducted by Graham et al. (2017) showed that the ants operate in a 'decentralised' manner to form a bridge, meaning that they do not rely on a leader coordinating their actions. Figure 2.2(b) shows an example of sheep flocking behaviour. A sheep flock can stay and act as a whole even though they are not coordinated by a shepherd. An example of underwater navigation by a school of fish is shown in Figure 2.2(c). Schooling is beneficial for fish as it is an effective strategy for defending themselves against predators. Aerial flocking is also an efficient behaviour for birds by allowing them to migrate to warmer locations as a group. Figure 2.2(d) shows an example image of a flock of birds manoeuvring.

Self-organisation

Self-organisation is a group-level property that refers to emerging a behaviour without external cues. It emerges through interaction within the group members. In natural swarms, these interactions can occur by means of information transfer in two distinct ways; signal-based or cue-based. Signal-based information transfer occurs intentionally by one individual conveying information. Different types of signal-based information-flow have evolved in nature. For example, bees wiggle dance to indicate the position of a nearby source of food to other fellow bees. Cue-based information transfer, on the other hand, is incidental. In cue-based information transfer, the environment is frequently used as a medium. For example, ants communicate through stigmergy to choose a nest site. The mechanisms behind the self-organisation can be interdependent and significantly complex. Two mechanisms can explain how self-organisation transpires; positive and negative feedback.

The positive feedback mechanism is responsible for amplifying quantities within the system. It is also known as self-enhancement, facilitation, and autocatalysis (Camazine et al., 2001). For example, the birth of new individuals increases the number of members in a community. This can be seen as positive feedback in population dynamics. While positive feedback is useful for increasing quantities, the nature of amplification can also be destructive. For example, a high amount of individuals within a group can result in a shortage of resources, hence, can cause a famine (Camazine et al., 2001). Contrarily, the negative feedback mechanism is responsible for reducing the quantities in a system. It stabilises the system by compensating the perturbations and fluctuations. An example of negative feedback is the death of individuals, resulting in the number of community members to be stabilised. In cellular biology, self-organised mechanisms often use negative feedback to achieve and maintain homeostasis (Camazine et al., 2001), that is, physical and chemical steady-state in a biological system.

Bonabeau et al. (1999) list two additional self-organising behaviours. The first is the amplification of randomness by means of fluctuations, such as, random walk, making error, or switching to another task randomly. In a sense, randomness is crucial for new solutions to emerge and fluctuations can lead to differences in growing structures. For example, an ant losing the trails of the colony—due to an error—can lead to finding unexploited food sources. The second additional self-organising behaviour is relying on multiple interactions. For example, a group of agents can self-organise by collectively interacting with each others trails in the environment. An extreme example is that, an individual agent, by interacting only with its own trail, can also self-organise itself. Thus, multiplicity of in-direct interactions can also emerge into a self-organised collective behaviour.

2.1.2 Collective Behaviour

Collective behaviours emerge in biological groups of animals as a result of social interactions between group members (Krause and Ruxton, 2002). These types of social behaviours are often coordinated, meaning that the group realises a common task by self-organisation. Collective behaviours are not only limited to achieving a particular task in the group's environment. Group-level locomotion, synchronicity, information transfer, decision-making, or avoiding predators are just a few examples of collective behaviours.

Swarming is a self-organised collective behaviour that is realised by a group of animals.[2] It refers to 'living' as a group and coordinating together. Besides preserving a group structure, it gives certain advantages to the individual members. It is observed that flocks of birds make use of spatial alignment to exploit aerodynamic properties (Ballerini et al., 2008), such as Canadian geese flying in a V-formation as shown in Figure 2.3(a).

Swarming has other important benefits for species. For example, being in a large group increases the chance of survival, as the probability of being attacked by a predator

[2]The term *swarming* is most commonly refers to the collective behaviour of social insects. Synonymous terms are *flocking* for birds, *herding* for hoofed mammal, and *schooling* for fish.

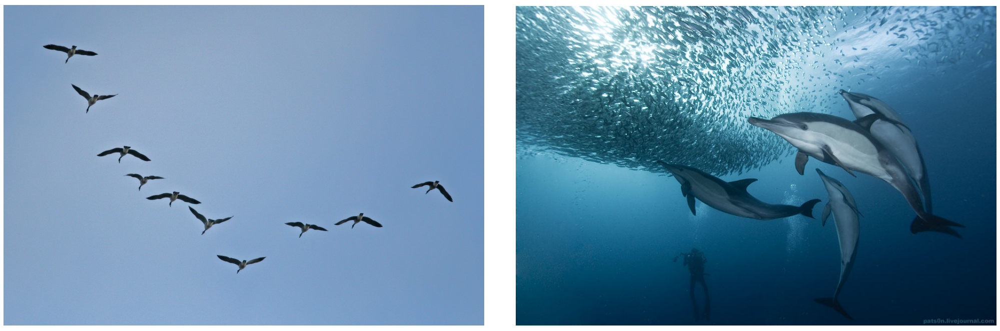


Fig. 2.3 Cooperative behaviours in nature: (a) v-formation in flock of birds and (b) bait ball by a school of fish. Images are reprinted from (a) Benson (2007) and (b) Safonov (2008).

is statistically lower than for that of an alone individual (Partridge, 1982). Moreover, a large group can act as a single intimidating entity which can confuse a predator (Cech and Moyle, 2000). An interesting example of this is a bait ball formed by a school fish as shown in Figure 2.3(b). Foraging and gathering food is another advantage of staying as a flock, as the individuals can disperse to search and share information to locate food sources (Pitcher et al., 1982).

2.1.3 Artificial and Engineered Swarms

Reynolds' seminal simulation algorithm started the artificial swarming paradigm. Reynolds (1987) developed a distributed model emulating flocking behaviour of birds. Each agent, or *boid*, executes a simple set of behavioural rules as given below:

- *Cohesion*—steer towards the centre of the flock
- *Alignment*—match velocity with neighbours in close proxy
- *Separation*—avoid collision with nearby flockmates

The three components of the model are combined linearly to create an equation of motion. The boids model paves a way to artificially animate swarming, flocking, herding, or schooling behaviour. Figure 2.4 shows a basic visual expression of the three rules. The rules are at the microscopic (individual) level and only require local sensing. The agents neither utilise global information nor are they directed by a leader, resulting in the collective motion to be generated solely by local interactions between the agents.

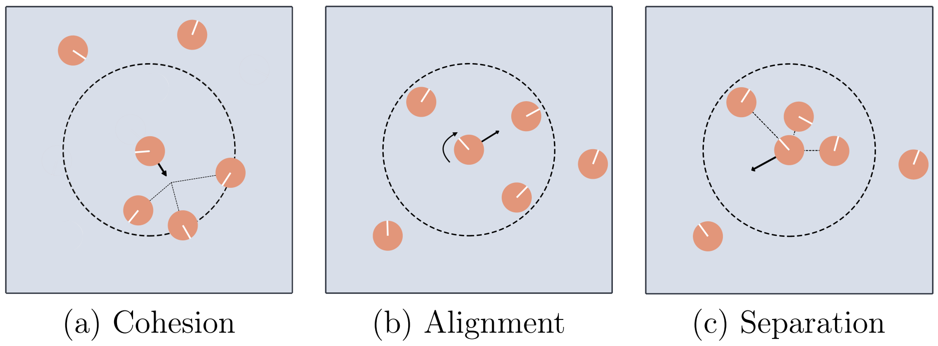


Fig. 2.4 An illustration of Reynold's flocking model with circular agents. Arrows indicate the direction the focal agent should move towards. Black dotted circle represent the region of influence.

Ballerini et al. (2008) observed that the collective behaviour in groups of animals is achieved more accurately under topological interactions between the neighbours in a close proxy. Thus, the boids model captures the biological swarming characteristics and transmits them onto artificial domains.

Once an observed swarming behaviour is expressed in formal mathematical terms, it becomes reproducible and adaptable. Artificial swarms are not only useful for computer simulations, but they can also be effective problem solvers. Numerous optimisation algorithms are inspired by natural self-organising swarming phenomena (Krause et al., 2013). These optimisation algorithms can be deployed to solve complex real-world tasks, especially combinatorial optimisation problems such as scheduling problems, assignment problems, and vehicle routing problems.

Artificial Swarm Intelligence

Beni and Wang (1993) introduced the term *swarm intelligence* in reference to a class of cellular robotic systems. Its use spread to cover a range of studies, from biological societies to robotics. Swarm intelligence simulates the social structures and interactions in a swarm, similarly to how artificial intelligence simulates the structure of an individual cognition. A swarm intelligence system usually involves individuals with limited intelligence forming a group that is distributively controlled by local interactions and simple rules. The system capitalises on swarm behaviour without requiring centralised control or a global model, to enable it to solve complex and large scale problems. The collective behaviour emerges through the communication, sensing and/or information flow among individuals (Fax and Murray, 2004).

The main application domain of swarm intelligence is optimisation problems. Dorigo (1992) proposed the *ant system* in his doctoral thesis. Five years later, Dorigo and Gambardella (1997) published Ant Colony System, an optimisation method inspired by natural swarms. Ant Colony System, more commonly known as, Ant Colony Optimization (ACO) is a metaheuristic solver that relies on finding optimal paths in graph structures. ACO makes use of the concept of stigmergy, as each artificial ant leaves and retrieves virtual pheromones in the environment. An optimal solution is calculated based on the pheromone concentration in the environment. Particle Swarm Optimization (PSO) is another metaheuristic optimisation method, that was introduced by Kennedy and Eberhart (1995). In PSO, each agent, or *particle*, has a position and velocity, similar to Reynold's (1987) boids model. However, in contrast, the particles can access global information. The particles influence each other like a social network. After a sufficient number of iterations, the particle swarm converges to an optimal position (i.e. solution) within the search space.

There are certain intrinsic properties that makes swarm intelligent systems preferable. These are namely, robustness, scalability and flexibility.

- *Scalability.* Scalability is beneficial in real-world applications, as it allows the system to be adaptable. For example, without major modifications, the system can cope with increased agent populations. It can also be a time-saving strategy, as in a large environment, the agents can distribute themselves effectively. Furthermore, agents can join or exit from the swarm without interrupting other individuals or the whole system.

- *Robustness.* Coping with failures is a crucial part of designing swarm intelligent systems. The swarm should be able to operate under disturbances without the system becoming vulnerable. The robustness relates to the scalability of the system. A system is considered robust to failures if it can cope with the absence of a portion of the agents. The performance of the system may decrease, however as long as the goal can be accomplished, the system remains robust.

- *Flexibility.* Several tasks can be achieved utilising the same swarm intelligent system with small modifications. Flexibility can also allow the system is to switch between strategies during an operation. This allows the system to be more adaptable to dynamically changing environments. Moreover, it is possible for systems to accomplish further tasks without necessarily ending the operation and starting again.

Swarm intelligence is not only limited to optimisation problems, it can also be used as a data mining technique. Ant Colony Optimization and Particle Swarm Optimization are widely used algorithms to solve optimisation problems. They are also widely used in the field of data mining. Parpinelli et al. (2001) proposed AntMiner, the first data mining algorithm that utilises ACO. The authors used ACO as a supervised classification method within their algorithm. For more detailed review of swarm intelligence algorithms used in data mining, reader can refer to Martens et al. (2011).

2.2 Swarm Robotics

Swarm robotics can be interpreted as a research domain of swarm intelligence applied to an embodied multi-robotics platform. Şahin (2005, p. 12) gives the following definition of swarm robotics:

> "Swarm robotics is the study of how large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among agents and between the agents and the environment."

Nevertheless, the author emphasises that the given quote does not provide a sufficient definition for swarm robotics. There are multiple research domains in multi-robotics, namely, *collective robotics*, *distributed robotics* and *robot colonies*. Swarm robotics distinguishes itself by a few differences. The major differences are that the swarm robots tend to be simple, low cost, and have minimal capabilities.

Şahin (2005) listed a few distinct properties that differentiate the field of swarm robotics from other multi-robot systems. The first property is that a swarm robotic platform consists of mobile and autonomous robots. The number of robots in the swarm should be large, ideally more than 10^2. There should be few groups within the swarm. That means the swarm should not be diverse, but rather homogeneous. For instance, a team of football-playing robots would not be considered as swarm robots as the role of each member is different. One of the core ideas of swarm robotics is to make use of a large number of members. Thus, the robots should be relatively simple and individually inefficient. In order to keep the system complexity low, local sensing and communication capabilities are preferable.

2.2.1 Design Principles

Swarm robotics systems are made up of relatively simple, economically efficient physical robots that can cooperate with each other (Brambilla et al., 2013; Şahin, 2005). Swarm robotic systems also feature the three intrinsic properties in swarm intelligence; flexibility, robustness, and scalability. Ideally, a swarm robotic system should be *flexible* in a way that the system responds well to changes in the environment. This is particularly desirable if the environment is precarious. The system should be *robust* so that the performance is not heavily affected by certain disturbances in the environment or within the system. *Scalability* is the third intrinsic property. Preferably, the system's performance, or ability, should be improved with the increasing number of group members. Certain design principles reveal these three intrinsic properties.

- *Decentralisation.* Decentralised control, or distributed control, has been observed in biological swarms (Deneubourg et al., 1983). Through competent cooperation rules, the system can solve large-scale problems. An advantage of decentralised control is that each robot can self-regulate, rather than rely on a central leader. As no single robot is responsible for the whole system, malfunctioning of one robot does not lead the task to fail. There is also less communication complexity than required for central control, allowing the swarm to maintain its reaction speed. However, global coherency becomes a major challenge, as each individual requires task specific information (Tan and Zheng, 2013).

- *Local Interaction.* Sensing and communication are two modes of interaction that an individual robot can perform. Local interaction is superior to global interaction with respect to scalability and flexibility (Ijspeert et al., 2001; Mataric, 1998). Due to impracticality and cost associated, global interaction is not suitable for a large group of robots. As the number of individuals increases in the group, the sensing and communication complexity increase the information acquisition exponentially. Additionally, system-wide interactions require more advanced sensing and communication capabilities, which also increases hardware complexity. Hence, local interaction strategies are beneficial for a swarm robotic system.

- *Homogeneity.* Homogeneity is an important design principle for swarm robotics systems. Ideally, there should be the least amount of labour division (i.e. task allocation) and the maximum number of individuals within each division (Şahin, 2005; Tan and Zheng, 2013). This is so that the inter-robot interactions can be modelled and analysed in a predictable way. Additionally, if the robotic swarm has certain probabilistic characteristics, heterogeneity may make it impossible to obtain similar behaviours at each run (Balch, 2000).

- *Economical.* Producing a robotic unit with a relatively low cost allows the mass-production of a robotic swarm. At the same time, the swarm robotic systems are energy efficient. In general, a swarm robot is less energy demanding, allowing it to require a relatively smaller battery. There are also swarm robotic systems that can use solar energy to recharge their battery during operation (Seyfried et al., 2004) or even share energy between one another (Escalera et al., 2018).

2.2.2 Design Approaches

In this section, we explain widely used approaches and methods for designing and controlling a swarm robotic system. There are numerous approaches reported in the literature, yet there is no standard categorisation. Dudek et al. (1993) proposed a taxonomy to classify the existing literature for swarm robotics systems based on technical features such as swarm size, reconfigurability, communication range and topology, or processing unit abilities. A different taxonomy was introduced by Gazi and Fidan (2007) categorising the studies as swarm coordination and control, design approaches, and mathematical modelling. Recently, Brambilla et al. (2013) proposed a taxonomy for swarm robotics design approaches and analysis methods. We believe that this thesis benefits the most by using categorisation presented in Brambilla et al. (2013), thus we adapt it for the rest of this section.

To the best of our knowledge, there is no 'one specific formal design method' that can guarantee to achieve any desired system-level behaviour by designing low-level individual behaviours under given design specifications. We speculate that this is due to the 'richness' of the behaviours that can be produced by swarms, as well as, the complexity that arises due to inter-agent and agent-environment interactions. Thus, swarm robotics system researchers prefer to design a 'tailored' system for a given task and requirements on robotic platforms. As given in Brambilla et al. (2013), two main-stream design approaches are behaviour-based design and automated design.[3]

A behaviour-based approach is where a robot's individual behaviour is manually designed in an iterative way that is commonly based on trial and error. One of the main inspirations for this approach comes from observing the behaviour of animals

[3]From a different perspective, these approaches can also be named as bottom-up and top-down design approaches, respectively (Crespi et al., 2008).

in social situations. The design process starts with a swarm of robots executing an initial behaviour algorithm while being examined by the designer. If the performance of the robotic swarm is not satisfactory, meaning that it does not produce the expected collective behaviour, the designer makes changes to the algorithm. The design process continues by tuning the algorithm until the swarm satisfies the given performance criteria.

Utilising a Finite State Machine is a common set of methods for designing individual behaviours. More specifically, Probabilistic Finite State Machines (PFSM) (Minsky, 1967) are frequently used techniques as they function in a similar way to social insects which have been observed to respond to sensory stimuli in a probabilistic way (Bonabeau et al., 1997). Behaviours of a robot are encoded as distinct states that can change depending on the sensory inputs and certain probability thresholds. PFSM are utilised for several canonical swarm robotic tasks, such as gathering (Soysal and Sahin, 2005), and collective decision making (Valentini et al., 2016). Artificial Potential Field is another common behaviour-based design method, specifically for coordinated motion (Khatib, 1986), exploration (Howard et al., 2002), and pattern formation (Spears et al., 2004). In basic terms, each robot computes virtual force vectors and acts upon them; for example, in a coordinated movement task, robots are attracted to a predefined goal and repelled from obstacles with certain strength based on the calculated virtual force magnitude.

A drawback of a behaviour-based design approach is that the desired behaviour may not emerge 'naturally' from the low-level individual behaviour. Thus, the set of behaviour-based methods may lead researchers to design ad-hoc systems that are not flexible or reusable. In contrast to the aforementioned approach, an automated design approach first describes the swarm behaviour at a high-level, then 'generates' a low-level individual behaviour. The individual agent behaviour is generated through optimisation of a utility function associated with a given task. It is an iterative process, yet the iterations are performed by a computer, unlike the behaviour-based approach. As the name suggests, this approach makes use of computer-aided design. An example of an automated design approach is reinforcement learning (Kaelbling et al., 1996; Tan, 1993).

Reinforcement learning techniques allow agents to 'learn' a behaviour by positive and negative feedback provided by a central system. The central system evaluates agents' actions based on the provided utility function and gives the agents rewards or penalties. The agents naturally tend to maximise their rewards, resulting in them adapting optimal behaviours. In more technical terms, the agents automatically learn an optimal strategy for a given problem by creating internal state-to-action mapping mechanisms. There are numerous studies using this technique, however, the method may suffer from high complexity and space-state dimension problems. For these reasons, reinforcement learning has a limited scope in swarm robotics community (Panait and Luke, 2005). A more frequently used approach is known as evolutionary robotics (Nolfi and Floreano, 2000).

In this thesis, we utilise an evolutionary robotics approach to design control algorithms for swarms of robots. The following section describes evolutionary robotics in detail, with particular attention paid to motivation and its roots in Darwinian evolutionary theories.

2.2.3 Evolutionary Algorithms and Evolutionary Robotics

This section starts with a peculiar introduction to evolutionary design paradigm to emphasise its importance in engineering design. Humans—by their nature—desire symmetry and they often design using symmetric rules (Norrman, 1999). This, however, may cause properties of asymmetric structures to be overlooked. In some cases, an asymmetric design might be superior to symmetric design. For example, Hornby et al. (2006) discovered this phenomenon while they were designing new antennas for various aerospace applications in NASA. They deemed to automate the antenna design process, and proceeded with employing an evolutionary algorithm (Hornby et al., 2006, p. 1):

> "Whereas the current practice of designing antennas by hand is severely limited because it is both time and labor intensive and requires a significant amount of domain knowledge, evolutionary algorithms can be used to search the design space and automatically find novel antenna designs that are more effective than would otherwise be developed.[...]"

The resulting antenna designs appear unusual as can be seen Figure 2.5, yet they far excel standard design patterns.

Evolutionary robotics is an automated design approach for robotic systems using evolutionary algorithms. The term design here can refer to both hardware design and the control architecture of the robotic system. Evolutionary robotics approach aims to create robust and adaptive systems with the aid of Darwinian evolutionary theories. The morphology of a robotic system—sensors and actuators—and its control

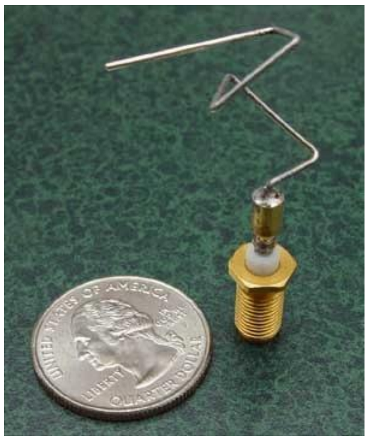


Fig. 2.5 An evolved antenna based on a specific NASA aerospace application. The image reprinted from Hornby et al. (2006).

architecture are considered simultaneously. This is a potential advantage of utilising an evolutionary robotics approach, as the robotic system is contemplated as a whole to reduce the inconsistencies between its individual parts. Evolutionary robotics design is a sophisticated procedure that heavily relies on appropriately utilising an evolutionary algorithm.

Evolutionary Algorithms

Evolutionary algorithms are a family of population-based metaheuristic black-box optimisation algorithms that artificially implement biological evolutionary theories (Eiben and Smith, 2008). The main principle is to minimise, or maximise, a fitness function[4], over substantial iterations.

During the 20^{th} century, four main streams of evolutionary algorithms were introduced. Fogel et al. (1966) introduced *evolutionary programming*, the first major evolutionary algorithm paradigm. Meanwhile, Holland et al. (1992) created the second major approach called *genetic algorithm*, that increased in popularity in 1975. In the early 1970s, Rechenberg and Schwefel proposed *evolution strategy* optimisation technique (Rechenberg, 1978). A variation of 'evolution strategy' is also widely used in the studies presented in this thesis. The fourth major algorithm, *genetic programming*, was proposed by Koza (1992).

The key concepts of an evolutionary algorithm are natural selection, mutation, and recombination. A group of candidate solutions (population) compete against each

[4]A fitness function is a cost, or an objective function in evolutionary computation terms.

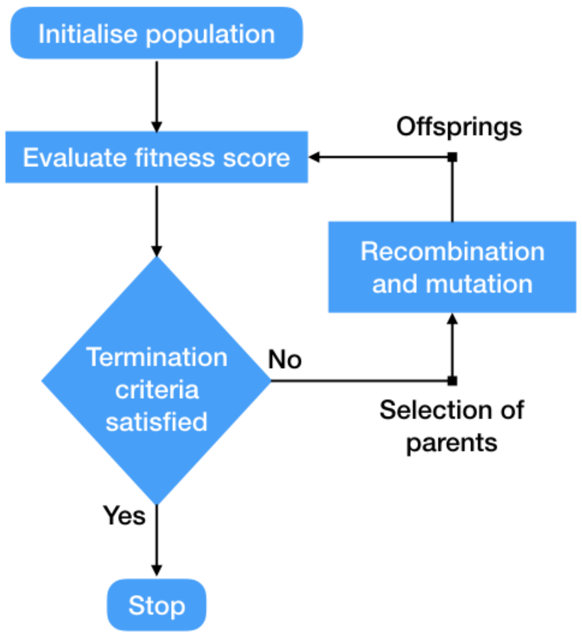


Fig. 2.6 A basic evolutionary algorithm mechanism.

other to ‘survive’ for the next iteration. Using an appropriate selection mechanism, a subset of the population can mutate and/or recombine by a predefined computational operator to create offspring that will compete in the next iteration. The choice of the selection mechanism, mutation and recombination operators strictly depend on the algorithm used. The user can get a desired outcome of the optimisation problem by varying certain numerical parameters related to these three concepts.

A basic working mechanism of an evolutionary algorithm is illustrated in Figure 2.6. The algorithm starts with an initial population of λ candidates. In general, the initial population is randomly generated by the algorithm. Each population member, a candidate solution, performs the task independently and is assigned a fitness score. In the next step, all fitness scores are evaluated and the termination criteria checked. In general, the termination criteria is either a fixed number of iterations or a desired fitness value with a given tolerance. If the criteria is not satisfied, then the best performing n individuals are selected as parents, and they reproduce new offspring based on predefined recombination and/or mutation operators in the algorithm. The members of the new population perform the task, and are evaluated in the next iteration. If the criteria is satisfied the algorithm terminates accordingly.

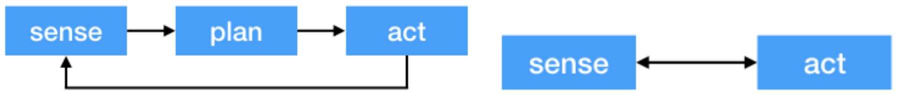
Evolutionary Robotics

Evolutionary robotics implements a design solution found by an evolutionary algorithm to a robotic system. In particular, evolutionary robotics approaches are commonly used for designing a control solution by the swarm robotics community. As it is an automatic design method, it does not require detailed *a priori* knowledge about the problem. The researcher is required to formulate the problem in a high-level, by designing an appropriate fitness function and certain optimisation parameters. A typical evolutionary robotics approach consists of three phases; setup, optimisation process, and analysis.

In the setup phase, a suitable fitness function needs to be designed to evaluate the performance of the system. It plays a crucial role in the setup, as it determines if the given candidate solution is a good fit or not. Deploying the control solution requires genotype-to-phenotype mapping. Genotype refers to the solution that is found by the evolutionary algorithm. Typically, a genotype refers to an array of numbers that encodes the control parameters. Phenotype, on the other hand, is the robotic platform that uses the genotype. Genotype-to-phenotype mapping is important as the control parameters are fine-tuned for a specific phenotype used in the *evolution* procedure.

In the optimisation phase, the evolutionary algorithm operates in order to create and select candidate solutions. This process follows the same procedure as illustrated in Figure 2.6. Depending on the parameter selection, the optimisation process can be time consuming. Most researchers prefer to conduct an offline evolution, which uses a simulated robotic system. Offline evolution has certain advantages compared to online. Floreano et al. (2008) noted three main challenges for performing an online evolution; (i) the time it requires, (ii) the robot can damage itself and the process may terminate, (iii) a human may need to be present to check the activity and the process. A drawback of offline evolution, however, is that the physical platform is simulated by a computer. Thus, there may be inconsistencies between the simulated platform and the physical one.

The final phase is the analysis of the optimisation results. In general, this phase has two possible outcomes; either the evolved solution is good enough and the design process is concluded, or it is not good enough, requiring the design process to be conducted again. If the latter is the case, then the design process restarts with the setup phase. Potential errors of the design problem can be investigated and a new



(a) deliberative control (b) reactive control

Fig. 2.7 Illustrations of two main control architectures.

set of design parameters can be implemented. The process then continues with the optimisation followed by the analysis process once again.

In swarm robotics community, there are many examples of evolutionary robotics studies. Dorigo et al. (2004) conducted an evolutionary robotics approach to synthesise control strategies for a gathering and coordinated motion task with a swarm of self-assembling simulated *s-bots* (Mondada et al., 2003). Trianni and Nolfi (2009) evolved control solutions for self-organising synchronisation problem using a swarm of s-bots both in a simulation environment and with physical robots.

2.2.4 Control Architectures

In the previous sections, we covered design principles and common approaches to design a swarm robotic system. The design approaches we gave in Section 2.2.2 consider the system-level behaviour for a swarm to achieve a task, thus, acting on a high-level perspective. Control architectures, on the other hand, operate at the individual level and are responsible for coupling sensor information and actuation using the given set of *a priori* designed instructions. In this section, we explain two control architectures that are frequently utilised, namely, *deliberative control* and *reactive control*.

In general, deliberative control architecture is sophisticated and requires the accompanying robot to be computationally competent. It utilises an "sense-plan-act" paradigm [see Figure 2.7(a)]. Creating a detailed environment model is at the core of deliberative control. At each control cycle, the robot gathers sensor data and plans its next action, while continuing building the environment model. Information flows between multiple channels of sensing and planning modules before the robot takes action. Thus, inter-robot and robot-environment interactions are solely planned actions. The deliberative control architecture is rigorous and powerful, however, it comes at the expense of computationally powerful hardware. Martinoli and Mondada (1997, p. 1) summarised the deliberative control scheme as follows:

> "[...]First, sensing the environment, then detecting features, then constructing and modifying a world model, reasoning for the task and the world model in order to find some sequence of actions which might lead to success, then executing the action sequence one step at the time while updating the world model and replanning it if necessary at any stage. This is a very time consuming operation and requires a remarkable computational power and basic knowledge."

Considering the given constraints, deliberative control is not suitable for *simple* robotic platforms, such as the ones that are used in a swarm robotic system. A reactive control architecture, on the other hand, does not require an internal representation of the environment. It is based on acting with regards to the sensory input, without needing to generate a sophisticated model. Thus, reactive control directly links the sensory inputs to actuation outputs, following a generic "sense-and-act" paradigm [see Figure 2.7(b)]. The low-computation demanding nature of the reactive control architecture renders it more suitable to utilise in swarm robotic systems.

These two architectures hold major differences. A deliberative control architecture allows making predictions for future actions, based on its past knowledge, however, a prediction is entirely impossible for a reactive control. Thus, a system designed with a reactive control scheme has to consider the environment it operates in advance, as well as the potential interactions it may encounter. This is one of the main reasons the behaviour-based design approach given in Section 2.2.2 requires a trial and error process. On the other hand, creating a detailed world-model slows the robotic system down, as planning requires time and the robot becomes unable to respond to the changes instantaneously. This is a major advantage of a reactive system, as the name suggests, it can deal with changes immediately.

Given the above points, a natural question arises; "how much intelligence and reasoning should a robotic system hold?". For a sophisticated robot such as one that performs walking in unknown terrain, finding objects in a room, and/or speaking to humans, a detailed architecture is required. In contrast, a swarm robotic system is simple and computationally less demanding, thus the intelligence required can be at a minimum.

Braitenberg (1986) introduced a number of thought experiments employing extremely simple "vehicles" and demonstrated that complex behaviours can emerge from atomic behaviours. The vehicles incorporated a reactive 'sense-and-act' paradigm through their primitive sensors. Sensor readings of a vehicle, e.g. light intensity or proximity to an object, are directly coupled with its wheels. Depending on the connection of the sensors and actuators, vehicles were able to exhibit interesting behaviours such as, "fear", "aggression", "love", or "hate".

It is worth mentioning one specific reactive control architecture, namely the *subsumption architecture* proposed by Brooks (1986). The architecture allows the performance of real-time tasks, without creating symbolic representations of the world. It decomposes the overall behaviour into smaller sub-behaviours, paving a way to parallelise sensory inputs while combining them in commonly-shared outputs. In simpler terms, the architecture tightly couples sensory-motor schemes. Brooks' approach was significantly different than widely used traditional AI in the 80s and 90s. Despite several criticisms made in the Robotics and AI communities, in general, the subsumption architecture was well-received, especially in the multi-robot community. Several demonstrations showed the practicality of the architecture. In the light of the practical benefits of system simplification, the work presented in this thesis is strongly inspired by a 'minimal information processing' framework, so-called *computation-free swarming* (Gauci, 2014).

The computation-free swarming framework, a novel swarm-robotics design and control framework, was introduced by Gauci et al. (2014a,c). The framework employs a simplified control architecture, which may be thought of as a basic version of the subsumption architecture, utilising an evolutionary robotics approach to design the system. To make it clearer, hereby the term "computation-free" only refers to the reactive control architecture, and not to the design method by any means. As the name suggests, minimal information processing is at the core of this framework. The reactive nature of the controller and the extreme simplicity of the discrete sensor mechanism lead the controller to be 'memoryless' and 'free of arithmetic computation'.[5] The next section explains this framework in more detail, discussing its strengths and weaknesses, as well as presenting additional studies that have adopted it.

2.2.5 Computation-free Swarming

Gauci et al. (2014a,c) proposed the *computation-free swarming* concept, a swarm robotic design and control framework. The controller to be designed using the framework is memoryless and computation-free, thus allowing a group of robots with minimal processing capabilities to achieve sophisticated tasks. Inspired from a limited sensing

[5]The obtained controller solution itself does not require a run-time memory or arithmetic computation.

mechanism—windshield (Yu et al., 2012), a limited angular field of view—Gauci et al. (2014a) further reduced the limited sensing mechanism to an extreme by introducing the *line-of-sight* sensor.

In the simplest form, a line-of-sight sensor provides discrete readings, such as the type of object present in the focal robot's direct line of sight. The sensor does *not* provide any qualitative information about the perceived object, such as bearing angle, distance, or height. The windshield sensor mechanism operates similarly, however, one major difference is the angle of the field-of-view. When the angle of the view is wide, multiple objects are likely to be detected within the sensing range. This multiplicity can then increase the complexity of the controller proportional to the number of distinct sensing states. On the other hand, a line-of-sight sensor—identical to the field-of-view sensor with an angle of 0-degree—can detect only one object at a time. This restriction in the sensing mechanism reduces the necessary processing capabilities of the controller, as the information collected is minimal.

The controller architecture proposed by Gauci et al. (2014c) is based on a simple "sense-and-act" paradigm. From one perspective, the controller is a direct mapping between the sensor reading input to actuation output. More specifically, the information extracted from the environment, the type of object, is translated onto the motor system. Utilising only limited sensing, lack of information sharing, fully reactive response, in addition to not requiring run-time memory, result into the controller to be free of arithmetic computation.

Self-organised emergent behaviours, by their nature, are complex phenomena Camazine et al. (2001). Simplification of such complex systems results in better understanding of the dynamics. Swarms of simple robots, in the future, can also be useful in application domains where it is not possible to utilise complex hardware. Considering this, it is important to design a system in a parsimonious way. Gauci (2014, p. v) explains the motivation behind the framework:

> "[...]The motivation for this work is to contribute in paving the way for the implementation of swarm robotic systems at physically small scales, which will open up new application domains for their operation. At these scales, the space and energy available for the integration of sensors and computational hardware within the individual robots is at a premium.[...]"

The framework was first applied to the problem of multi-robot aggregation (Gauci et al., 2014a,c). The robots use a single line-of-sight sensor that returns discrete readings about the environment; namely, what the robot is instantaneously pointing towards. An offline evolutionary robotics approach was conducted to obtain the control solution in the simulation environment with physically constrained differential-wheel robots. The proposed controller is the simplest solution to the multi-robot aggregation problem to date. The authors demonstrated the performance trade-offs using sensors that varied in an angle of view and sensing range. The controller is also scalable and succeeded to aggregate up to 1000 robots in a simulation environment.

The framework was then applied to a more complex scenario in the form of object clustering (Gauci et al., 2014b). This time, the robots had to interact with static items in the environment to bring them together into a single cluster. A major difference in this study, however, is using the line-of-sight sensor to differentiate between two types of objects: robots and static items. This has led to recent developments in computation-free swarming paradigm.

Johnson and Brown (2015) adopted the framework to explore some additional problems, such as perimeter formation and foraging, however, only in a simulation environment. In the follow up work, using novelty search, Brown et al. (2018) discovered that the framework can also be used to produce wall following, dispersal, and milling behaviours. Although, the authors did not further investigate the capabilities of the obtained control strategies. Recently, Wareham and Vardy (2018) formally examined the computational-free swarming concept for grid-based environments, under certain assumptions such as discontinuous movements. The authors showed that the design problem, given an arbitrary task, cannot be solved in polynomial time.[6] Even though the obtained theoretical results are not directly applicable to the previously designed computation-free controllers, the authors demonstrated that efficient solutions exist for a restricted class of problems.

2.3 Swarm Robotics Tasks

In this section, we present the swarm robotics tasks we studied in this thesis. Note that the order is alphabetical, that is, a different order from the presentation in the thesis. These studies are finding consensus, gathering, shepherding, and spatial coverage.

[6]The authors acknowledged that the results are only valid if widely believed conjectures (e.g. $P \neq NP$) are true.

2.3.1 Finding Consensus

Finding consensus is a well-studied problem in the swarm robotics community and beyond. It is a canonical study for many disciplines, including computer science, economics, physics, and neuroscience, however, the problem might be referred to differently. In swarm robotics context, the problem is commonly referred to as "collective choice" or "collective decision making". In broad terms, the problem refers to finding a *consensus*, or general agreement, by the agents in a collective manner. More often, in swarm robotics, the problem is solved by a swarm of self-organised agents that do not have *a priori* covenant information.

A number of solutions have been proposed for collective choice problems. In some studies, environmental cues play an important role for choosing an option. While other studies allow the robots to communicate with each other explicitly, generally, to share an opinion regarding an option. The designed control strategies vary from artificial neural networks, to an algorithm that is based on a random walk with various non-constant waiting times and purely phototaxis behaviours. In this thesis, we are particularly interested in "best-of-n" and "symmetry breaking". One can find a detailed review of the best-of-n problem in a swarm robotics context in Valentini et al. (2017).

Halloy et al. (2007) used robots to explore the collective choice problem in cockroaches. Naturally, cockroaches prefer darker shelters over lighter ones. The researchers introduced a group of robots coated with pheromone such that they were accepted by a group of cockroaches as conspecifics. The robots were programmed to "[...] explore their environment autonomously [and] tune their resting time [in the shelters] in relation to the presence of cockroaches, as cockroaches do" (Halloy et al., 2007). The robots, being programmed to prefer the lighter shelter, were able to socially influence the cockroaches so that they, on average, also made this "unnatural" choice. Francesca et al. (2012) also investigated shelter selection process by cockroaches. The authors evolved a neural network controller and verified their algorithm with real e-puck robots. These two works are also interesting in the perspective of animal-robot interaction.

Hsieh et al. (2008) proposed bio-inspired quorum-based stochastic control policies for assignment of swarm of robots to multiple nest sites. The swarm of robots are modelled as population fractions for each nest site. Each robot switches between maximum and constant transition rates in a probabilistic way and is assumed to move from one nest site to another, as well as, able to estimate the population size of a nest site. Furthermore, each robot has full *a priori* knowledge of the interconnection topology graph, that is equivalent to equipping the robots with an environment map, and control parameters with every other robot. The control policy enabled the robots to successfully redistribute themselves to available nest sites.

Parker and Zhang (2009) studied a scenario in which a group of robots is expected to choose the best out of a number of unequal options. The robots behaviour is inspired by the decision-making processes in insects. They employ an active recruitment strategy that relies on inter-robot communication. The robots start by looking for options and advocating them to each other, always switching selection to the best-known option. Once a robot's selection becomes sufficiently popular (reaching a *quorum*), the robot becomes committed to it. This enables the group to reach consensus.

Hamann et al. (2010) studied how a homogeneous group of robots can collectively choose between two global maxima in a light-intensity field. Each robot moves in a straight line until it encounters another robot. Then, it stops and counts the total number of robots in its neighbourhood. If this is above a certain threshold, the robot measures the light intensity and waits for a time proportional to this intensity. This creates a positive feedback effect which enables symmetry breaking between the two options.

Valentini et al. (2016) studied a swarm of robots that collectively choose among two unequal options. At any moment in time, each robot has an opinion about which option is best. The robot either explores the option, or exchanges information with its neighbours. In the latter case, the robot locally broadcasts its opinion for a duration that is proportional to the perceived quality of the preferred option. Moreover, for a fixed time period it monitors incoming messages and then updates its opinion using the majority rule. The robot then switches to exploring the potentially new option, and the process repeats indefinitely.

In all of the above examples, the agents perform arithmetic computations to determine where or when to move [e.g. artificial potential fields (Halloy et al., 2007), artificial neural networks (Francesca et al., 2012), timeouts (Halloy et al., 2007; Hamann et al., 2010; Valentini et al., 2016) or pseudo-random numbers (Hsieh et al., 2008)] or to update internal representations [e.g. preferences (Hsieh et al., 2008; Parker and Zhang, 2009; Valentini et al., 2016)]. Moreover, the agents need to store information during run time (e.g. quality estimates, behavioural states, or counters). Their sensors typically provide rich information, such as a count of the number of nearby agents.

2.3.2 Gathering

Gathering is another canonical problem that is studied actively by the swarm robotics community. Based on the context, the problem is referred to as robot aggregation (Correll and Martinoli, 2011), gathering (Gordon et al., 2004), or rendezvous (Alpern, 1995). Gathering is a collective behaviour in which the agents self-organise themselves in order to congregate at a point in their environment. Gathering is commonly observed in living organisms such as bacteria, fish and mammals (Camazine et al., 2001; Krause and Ruxton, 2002). It is beneficial for the living groups in multiple ways, for example, to build a nest, to share thermal energy, or to repel predators (Parrish and Edelstein-Keshet, 1999).

In nature, it is common that gathering is guided by heterogeneities in the environment, for example, a warmer nest site would be a better place to shelter. However, gathering in homogeneous environments has also been observed in nature. Deneubourg et al. (1990) reported that a group of uniformly distributed bark beetle larvae can aggregate rapidly in the centre of an homogeneous Petri dish. In an homogeneous environment, there is no cue to hinder where to meet, it rather dynamically emerges through the local interactions within the group.

A gathering strategy could be either stochastic or deterministic. A number of biological models include a certain degree of stochasticity (Deneubourg et al., 1990; Parrish and Edelstein-Keshet, 1999). Stochasticity in the models is often injected by random walk, or by response to a gradient such as density of a cluster. In this thesis, we are interested in both stochastic and deterministic gathering strategies. Both approaches present advantages and disadvantages. In general, deterministic control strategies require the initial placement of the robots to form some sort of a connectivity, or visibility, graph. This requirement renders it difficult for robots to operate in a large environment due to the constraints in the sensing and communication range. On the other hand, a deterministic strategy can lead to convergence in the system faster as it does not constitute randomness.

In the rest of this section, we focus on gathering of multi-agent systems in continuous and discrete domains. First, we present deterministic and stochastic gathering control strategies in continuous space. Then, we present gathering strategies for robots operating in a discrete space.

For gathering in continuous space, a number of solutions require that each robot determines the relative position of all other robots in its local neighbourhood. For example, Ji and Egerstedt (2007) presented a solution that is guaranteed to solve the gathering problem, provided that the visibility graph corresponding to the robots' initial spatial distribution is connected. Other solutions required that each robot determines the bearing of all other robots in its local neighbourhood (Gordon et al., 2004), again assuming initial connectivity. For robots using a line-of-sight sensor, it was shown that a single bit of information—whether another robot is detected or not—could be sufficient to solve the gathering problem, though only if the sensing range is unlimited (Gauci et al., 2014c).

Barel et al. (2017) proposed a probabilistic algorithm for gathering agents with 1-bit, unlimited range sensors. At every time step, each agent assumed a random orientation, and then moves forward if no other agent is present in the half-plane behind it, and rests, otherwise. The correctness of the algorithm was proven under the assumptions that the agents act synchronously, could jump instantaneously from one pose to another, and did not have physical bodies. Moreover, to avoid deadlock situations, the binary sensor was shown to require a half-disk blind region.

Ozsoyeller et al. (2019) presented a solution guaranteeing that a pair of robots, operating in an environment with polygonal obstacles, was guaranteed to meet almost surely. The solution involved repetitively tossing a coin to decide whether to rest in place, or move in a way that covers the environment. The strategy was extended to more than two robots, provided they could communicate.

For gathering in discrete space, Fatès (2010) presented a gathering algorithm for agents that could communicate by modifying their environment. The environment was considered to be a lattice structure and the agents are modelled as virtual amoebas. Each agent could move randomly into a neighbouring cell based on reaction-diffusion concentration in their local proxy. The proposed algorithm was robust to sensory noise and copes well with the presence of obstacles in the environment.

Cord-Landwehr et al. (2016) and Fischer et al. (2017) presented solutions for robots with constant memory and no memory (oblivious), respectively. The solutions were guaranteed to converge in linear and quadratic time. They required each robot to determine the relative position of all other robots in its local neighbourhood, comprising more than 100 cells, and a visibility graph that is initially connected. The robots were not embodied; where multiple robots occupy the same cell, all but one were removed.

Walter (2018) proposed two algorithms to solve the gathering problem with hexagonal modules. The modules were equipped with light and contact sensors. The algorithms were distributed and decentralised and did not require the modules to communicate with each other. The authors considered a set of observations to show the convergence of the algorithms.

While the aforementioned examples proposed decentralised control solutions to the discrete gathering problem with constraints on the agents, they assume unrealistic prerequisites. Either the setups lack physical constraints, such as the agents are not embodied (Cord-Landwehr et al., 2016; Fischer et al., 2017) and requiring to modify their environment as a means of communication (Fatès, 2010), or they are required to access global information such as their locations and orientations (Walter, 2018). Unlike previous solutions to the gathering problem with such restricted agents, the policies we propose in this thesis are not limited to specific initial positions, take the agent's embodiment into account, and require only four trits (i.e. ternary digits) of sensory information, though the latter comes at the expense of unlimited-range sensing[7].

2.3.3 Shepherding

The *shepherding* problem involves guiding the motion of multiple dynamic sheep agents by one or more shepherd agents towards a prespecified goal location. Aside from the shepherding of actual sheep by dogs, this problem, in a more general setting, has other manifestations—one example is human crowd control in large-scale events by trained officials. Potential robotic applications involving this task include: containing oil spillages (oil on the surface of water behaves as a dynamic entity), manipulation of micro-bacteria, and other uses in nanomedicine (Lien et al., 2005; Requicha, 2003).

Vaughan et al. (2000) conducted one of the earliest studies on the shepherding problem. They developed a controller strategy to herd (real) ducks using a single robot. The controller required an external camera system for tracking the robot's position and orientation. This, along with information on the centre of mass and size of the flock of ducks, was computed to provide quasi-instantaneous path planning for the robot to herd the ducks towards a goal location.

Lien et al. (2005) proposed and analysed several formations that the shepherds can assume. The shepherds are required to count the number of other shepherds, estimate the flock size, and construct a graph structure and ideally solve an optimisation problem on-board. Their work suggested that multiple robotic shepherds are superior to a single one in solving the problem.

[7]The sensing range needs to be sufficiently long to cover the environment.

Strömbom et al. (2014) developed an algorithm based on empirical data from sheep/dog interactions. The authors conducted a real-life experiment with 46 merino sheep in South Australia in order to collect movement date. A trained sheepdog is verbally guided towards the goal location. The proposed algorithm contains two steps: firstly, the shepherds gather a dispersed flock of sheep; secondly, they herd them to a goal location. Based on the distribution of the sheep flock, the shepherd agents can switch behaviour between step one and step two.

Razali et al. (2010) was inspired by the biological immune system model to solve shepherding problem. The developed algorithm uses immune network theory that makes the agents adaptable to changes in their environment. Although, the agents can execute different strategies, and are required to communicate with each other to estimate whether two agents use a similar strategy or not. Based on three threshold parameters, the agents can calculate their strategies via a simple action-selection process.

In contrast to the traditional shepherding problem, Çelikkanat and Şahin (2010) investigate the navigation of a swarm of robots using informed individuals. Some of the individuals are externally guided and their task is to steer the rest of the flock in the direction of the goal. The informed individuals know the goal direction as well as share information with the other flockmates to have a consensus. Therefore their action is an outcome of the weighted summation. After the simulations and experiments, the authors mention that steerability of a swarm of robots depends on the ratio of the informed members in the swarm and the weight of the choice of direction.

Pierson and Schwager (2015) proposed a control strategy for a multi-agent system herding non-cooperative sheep agents. The shepherd agents assume unicycle motion model and capable of performing arithmetic computations to calculate the controller, ideal heading and velocity. Using Lyapunov theory, the authors proved that this controller is always guaranteed to herd the flock to the goal location. The work was further extended to three dimensions in (Pierson and Schwager, 2017) using an appropriate nonholonomic motion model.

Lee and Kim (2017) proposed a control strategy based on artificial potential fields, requiring the agents to operate in a bounded and obstacle-free environment. The shepherd agents are equipped with sensors allowing the agents to measure the relative distance and bearing angle and velocity (both magnitude and direction) of neighbour agents. The control strategy requires the shepherds to switch behaviours based on various decision criteria. Additionally, the control strategy comprised of 4 control inputs that requires 4 scaling parameters.

In all of these works, the controller strategies required the shepherd agents to have memory, perform arithmetic computations, and exchange information with each other. In addition, they required the sensors to provide sophisticated information, such as, relative distance and estimating group size.

2.3.4 Spatial Coverage

Spatial coverage is a task performed by a group of agents distributing themselves to cooperatively *cover* an environment, or specific regions of interest within. This task has a number of applications. For instance, agents may be required to monitor a given area, to log data, or to detect abnormal events and relay an alarm to a central station. The robots may also be required to service the environment, such as watering or applying chemicals to a field of crops.

We are interested in self-organised dispersion of a group of agents that cover an environment simultaneously. This differs from the problem of visiting every location at once (Choset, 2001) or repeatedly (Rutishauser et al., 2009). Usually this requires the agents to maintain connectivity as a network or planning their path in advance. These problems are interesting for certain applications, in the former, when one agent makes a local observation, it can be shared throughout the whole group. In the latter, over time the robot pass over every location in the environment, as required in applications such as lawn mowing or vacuum cleaning.

Spatial coverage may also be used for searching as complete coverage constitutes a systematic strategy. Consider a situation where a group of robots start at a 'nest' location and must forage for a food source at an unknown location (provided it is within a bounded environment). By spreading out in the environment, the robots can minimise the expected time for finding the food source; once this is found by a robot, the information can be shared with the swarm, which would then transition to a different strategy to begin foraging.

One of the earliest studies of multi-robot coverage was reported by Howard et al. (2002). The authors showed that a swarm of robots, by emulating the movements of charged particles in a potential field, could disperse within an office-like environment. Each robot used relative position information about nearby robots and obstacles. While the attained formations may not be uniform, they are guaranteed to be stable.

McLurkin and Smith (2007) studied a strategy to deploy a swarm of robots in a bounded environment, where each robot moved away from its k nearest neighbours. For $k = 2$, the robots obtained an almost uniform distribution, whereas for $k \gg 2$, they ended up at the boundary. They could also disperse in open space while maintaining connectivity. The robots used an infra-red communication system to obtain relative positions.

Correll and Martinoli (2006) investigated the performance of swarms of robots when inspecting parts of a turbine engine. Each robot executes a schema-based decentralised controller which uses local potential fields to generate its action. The authors performed simulation and physical robot experiments. The results suggest that detailed geometric structures should be considered, as environment morphology affects the spatial coverage performance of a swarm of robots.

Schwager et al. (2006) studied a swarm of robots that, when put in a bounded environment, assume positions that optimise an *a priori* unknown utility function. This could allow regions of importance to be monitored more densely. The robots used local sensing to sample and approximate the utility function. Their controller achieves near-optimal coverage, and was tested on the same platform as in McLurkin and Smith (2007).

Ramaithitima et al. (2015) proposed a solution to the coverage problem that relies on only touch and bearing sensors. As the environment is not known in advance, the robots get sequentially deployed, until complete coverage is achieved. The control strategy is guaranteed to provide a complete coverage for a sufficient number of robots.

Prorok et al. (2011) proposed a combined spatial and non-spatial probabilistic modelling method. The models focus on spatial distribution and result in accurate predictions of the performance of the system. The authors apply the Fokker-Planck diffusion model to an agents spatial distribution over time. The models are validated through simulation and real robot experiments. However, the latter is only applied to a single robot scenario.

The aforementioned related work proposed alternative approaches that either require computation, localisation, or more elaborate coordination strategies. While some of the works listed may perform better, they are also computationally more demanding. This renders it difficult to implement extremely simple robotic systems, such as micro- and nano-scale mobile machines that lack fully-fledged CPUs or wireless radios.

2.3.5 Concluding Remarks

Recently, there has been a surging interest in exploring what is possible with robots that operate at a level of extreme simplicity; a question that had hitherto received little attention (Becker et al., 2013; Groß and Dorigo, 2008; Jones and Mataric, 2003). A major motivation for such robotic systems is their potential feasibility for implementation at small scales, where more conventional robotic systems are not feasible to implement due to the acute constraints on the space available for hardware (Requicha, 2003). While the proposed strategies can solve the discussed problems and some of them can guarantee a solution, they also require complex and sophisticated robot models. The common feature in the related work presented is that the agents are required obtain rich (high amount) information from their environment and often store the information in memory. In addition, they are capable of performing complex calculations, and are assumed to utilise flawless communication channels. These hardware requirements render it difficult for large quantities of these robots (e.g. $\gg 10^3$) to be produced. Moreover, they render it difficult for the platforms to be scaled down in size to the submillimetre level. In the next chapter, we will present a minimalist control strategy for multi-agent spatial coverage problem.

Chapter 3

Spatial Coverage

3.1 Introduction

In the previous chapter, we provided background knowledge for swarm robotics, mainly focusing on evolutionary robotics and computation-free swarming. Additionally, we reviewed existing literature for the relevant swarm robotic tasks, specifically, finding consensus, gathering, shepherding and spatial coverage.

Previously, Gauci et al. (2014b,c) demonstrated computation-free swarming for multi-robot aggregation and clustering tasks. The common aspect for both tasks is that the robotic swarms are expected to converge towards a point in the space. Whilst the tasks are not straight-forward, it is yet to be seen if the robotic swarm could accomplish an opposite scenario in which they need to *disperse* themselves throughout the environment. We hypothesise that, using the computation-free swarming framework, a divergent, self-organised behaviour is achievable in a controllable way such that once the robots disperse, they can maintain their positions. This behaviour then can lead to a number of practical applications, including covering an area and navigating a maze.

Following the hypothesis stated above, in this chapter we study the multi-robot spatial coverage problem for which a group of robots is required to cooperatively cover an environment or specific regions of interest within. The problem is relevant for a number of applications. For instance, robots may be required to monitor a given area, perhaps to log data, or to detect abnormal events and relay an alarm to a central station. The robots may also be required to service the environment, such as watering or applying fertiliser to a field of crops. Finally, complete coverage constitutes a systematic strategy for search.

In this work, we assume that the environment is bounded, and that all parts are to be covered continuously. This differs from the problem of visiting every location once (Choset, 2001), or repeatedly (Portugal and Rocha, 2011; Rutishauser et al., 2009). We present the first solution to the spatial coverage problem that is applicable to anonymous robots that lack the ability to compute, store run-time information, or communicate. Hence, our control strategy is the first to be free of arithmetic computation.

This chapter is organised as follows. Section 3.2 presents the problem formulation, including the computation-free control architecture and performance measures. Section 3.3 describes the evolutionary process to synthesise a controller, and presents mathematical analysis of the obtain the controller. Section 3.4 presents simulation studies that evaluate the performance with respect to simple benchmarks, and examine the effect of sensory noise, swarm size, and environment shape, and abilities of navigating a maze. Section 3.5 presents experiments with $n = 25$ physical e-pucks and reports the results. Section 3.6 concludes the chapter.

3.2 Problem Definition

3.2.1 Environment and Robot Model

Consider a two-dimensional (2D) bounded environment, $\mathcal{E} \subset \mathbb{R}^2$, with n autonomous mobile robots. The robots are anonymous, that is, indistinguishable from each other and execute an identical controller. They lack the capability of performing arithmetic computation, and have no run-time memory. Moreover, they are unable to communicate with each other, cannot localise, and have no knowledge of the environment, $\mathcal{E}$, nor of the number of robots, n.

At time t, robot i's position and orientation is written as $x_i(t) \in \mathcal{E}$ and $\theta_i(t) \in [0, 2\pi)$, respectively. Robots are modelled as non-overlapping open disks of radius r which are fully contained in $\mathcal{E}$.

Each robot moves using a differential-drive. The robot's two wheels are symmetrically placed at an inter-wheel distance of d_{wheel}. Its linear (v) and angular (ω) velocities—in its local reference frame—are

$$v = \frac{v_\ell(t) + v_r(t)}{2} v_{\max} \quad \text{and} \quad \omega = \frac{v_r(t) - v_\ell(t)}{d_{\text{wheel}}} v_{\max}, \tag{3.1}$$

where $v_\ell(t), v_r(t) \in [-1, 1]$ are the normalised left and right wheel velocities along the ground, $v_{\max}$ is the maximum velocity, and $d_{\text{wheel}} \leq 2r$.

Each robot is equipped with an unlimited-range[1] binary line-of-sight sensor that detects whether another robot is present in the line of sight directly ahead of the robot. The sensor "scans" only up to the first encountered object (which might be the environment boundary). Thus, the sensor is unable to detect a robot if it is behind obstacles or walls. Note that, the sensor provides neither distance information nor multiplicity of encountered objects. Formally, the sensor reports

$$s(t) = \begin{cases} 1 & \text{if another agent is detected;} \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

The robot executes a deterministic controller,

$$c\colon \{0, 1\} \longrightarrow [-1, 1] \times [-1, 1]. \tag{3.3}$$

At time t, c assigns sensor reading $s(t)$ to a pair of wheel velocities. Formally,

$$c(s(t)) = (v_\ell(t), v_r(t)) = \begin{cases} (v_{\ell,0}, v_{r,0}) & \text{if } s(t) = 0, \\ (v_{\ell,1}, v_{r,1}) & \text{otherwise,} \end{cases} \tag{3.4}$$

where $(v_{\ell,k}, v_{r,k})$ is the pair of wheel velocities for $s(t) = k \in \{0, 1\}$. Using $\mathbf{v} = (v_{\ell,0}, v_{r,0}, v_{\ell,1}, v_{r,1}) \in [-1, 1]^4$, any reactive control strategy can be expressed.

3.2.2 Objective

The coverage literature has used a number of performance criteria for coverage that either relate to the area a robot covers, special positions robots should occupy, or special measures of importance of parts of the space. In our work, we consider the following two performance criteria. The first criterion, *cell coverage*, determines coverage quality by relating robot positions to a given partitioning of the environment into cells. The goal is to occupy every cell with at least one robot. Such a partitioning may be provided by a user, derived from a utility function that represents the importance of the space, or

[1]In practice, the robots require sufficiently long sensor range to cover the environment.

simply be uniform (as in our scenarios). The second criterion, *area coverage*, measures the joint area that is close to some robot.

The *cell coverage* at time t is defined by

$$\mathcal{P}_{\text{cell}}(t) = \frac{m_{\text{occupied}}(t)}{m}, \tag{3.5}$$

where m is the total number of cells and $m_{\text{occupied}}(t)$ is the number of cells that contain at least one robot at time t. It follows that

$$1/m \leq \mathcal{P}_{\text{cell}} \leq \min(n/m, 1). \tag{3.6}$$

The *area coverage* at time t is defined by

$$\mathcal{P}_{\text{area}}(t) = \frac{A\big(\bigcup_{i=1}^{n} \mathcal{N}_i\big)}{A(\mathcal{E})}, \tag{3.7}$$

where $\mathcal{N}_i = \{p \in \mathcal{E} : \|p - x_i(t)\| \leq r_{\text{cover}}\}$, $r_{\text{cover}} > r$ denotes the distance up to which the robot covers the environment, and $A(S)$ is the area of S. It follows that

$$\frac{\pi r_{\text{cover}}^2}{A(\mathcal{E})} < \mathcal{P}_{\text{area}} \leq \min\Big(\frac{n\pi r_{\text{cover}}^2}{A(\mathcal{E})}, 1\Big). \tag{3.8}$$

Note that the cell partitions and coverage radius have no bearing on the robot's behaviour. They are merely used to measure performance. Figure 3.1 illustrates both performance measures in a square environment with $n = 25$ robots. The environment is partitioned into 25 equally-sized square cells.

3.2.3 Simulation Setup

We use an open-source physics-engine Enki (Magnenat et al., 2009), which simulates the dynamics and kinematics of rigid bodies in 2D. Space is represented continuously (with floating point precision). The control cycle of the robots is activated every 0.1 s and the physics are updated at 0.01 s intervals.

The robot platform is the e-puck (Mondada et al., 2009). In Enki, an e-puck is modelled as a disk of radius $r = 3.7\,\text{cm}$ and mass 152 g. Its maximum velocity is $v_{max} = 12.8\,\text{cm/s}$. The inter-wheel distance is $d_{\text{wheel}} = 5.1\,\text{cm}$. Throughout all simulation runs, 5% uniform noise is affecting the velocity of each wheel—the default

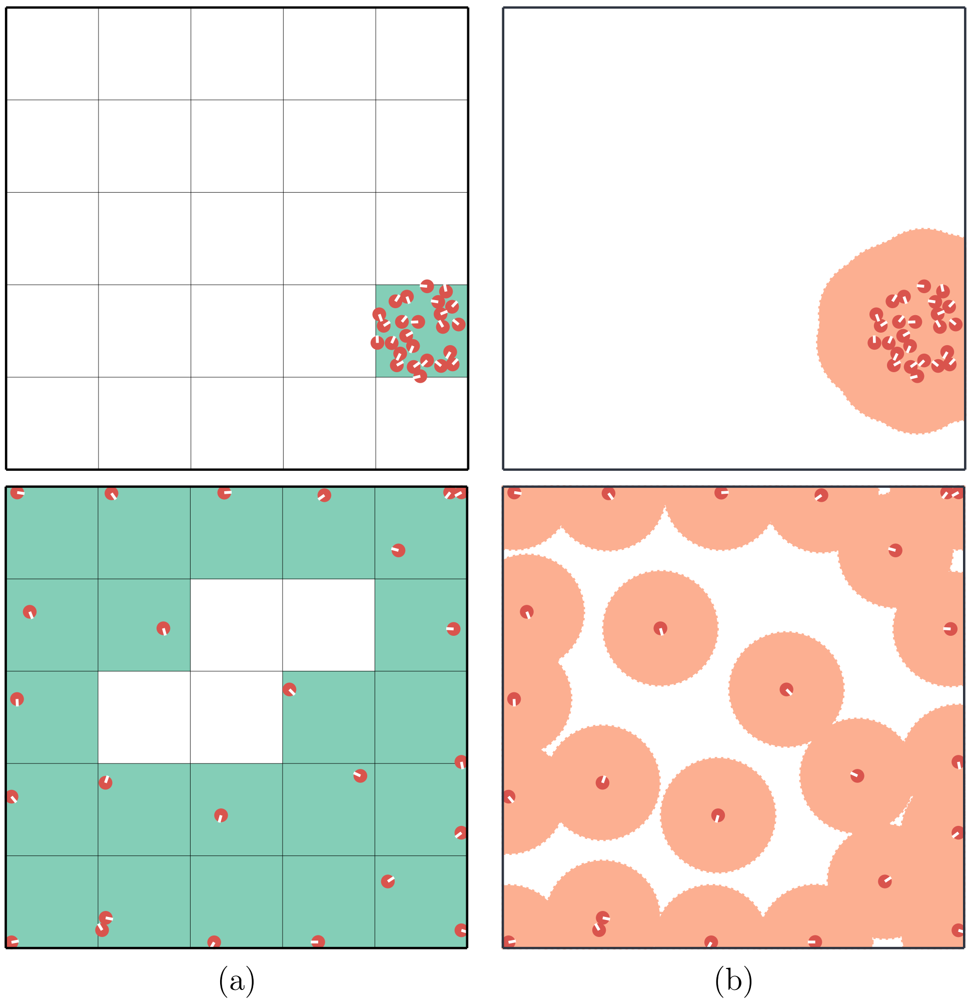


Fig. 3.1 A group of 25 robots performing coverage with two performance measures illustrated at the beginning (top) and end (bottom) of a trial: (a) Cell coverage uses a decomposition of the environment (square cells), and reports the fraction of cells with at least 1 robot; (b) area coverage assumes that each robot covers all points within a certain range, and reports the fraction of the environment's area that the robots collectively cover. The corresponding coverage (percentage) for (a) is 4.0% (top) and 84.0% (bottom). For (b) it is 11.6% (top) and 69.8% (bottom).

setup in Enki. The velocity parameters [see Equation (3.4)] are represented as floating arithetmic digits with 6 figures.

3.3 Controller Synthesis

In this section, we present the controller synthesis. We employ an evolutionary robotics approach (Gauci et al., 2014c; Nolfi and Floreano, 2000; Trianni, 2008) for designing the deterministic reactive controller. We first describe how a candidate control solution is evaluated in Section 3.3.1. Second, we describe the evolutionary algorithm setup that is used to evaluate the candidate solutions. Third, we explain the controller selection procedure. Finally, we present the selected controller, and mathematical analysis on the movement it produces.

3.3.1 Evaluation of Candidate Solutions

Candidate solutions are controllers that are considered by the evolutionary process. They are represented in continuous space, by quadruples $\mathbf{v} = (v_{\ell,0}, v_{r,0}, v_{\ell,1}, v_{r,1}) \in [-1, 1]^4$.

For each candidate solution, 20 simulation trials are performed using $\mathcal{E} = [0, 300] \times [0, 300]$ (cm) and $n = 25$ robots. Each trial lasts for $T = 120\,\text{s}$, corresponding to 1200 updates of the robot's control cycle. The robots are initially placed with random position and orientation within the same random cell, all chosen from uniform distributions.

The environment is decomposed into a 5×5 grid of cells, as shown in Figure 3.1(a). At the beginning of the trial, all robots are placed with uniformly random orientation and position at a distance of up to 30 cm from the centre of a uniformly randomly chosen cell.[2] All candidate solutions in a generation are evaluated on the same set of initial configurations.

[2]Initialising the robots in a circular region prevents orientation bias.

The performance of the swarm in a trial is measured using the cell coverage measure.[3] Formally the fitness function is

$$F = \frac{2}{T(T+1)} \sum_{t=1}^{T} t\mathcal{P}_{\text{cell}}(t). \tag{3.9}$$

The performance at time $t < T$ is taken into account, weighted by t, to reward solutions that reach good coverage faster. The constant factor normalises F to $(0, 1]$.[4] The overall fitness of the candidate solution is the mean performance across 20 trials. The fitness function given in Equation (3.9) characterises the desired goal for the robots, that is, to cover the entire bounded environment with a uniform density.

3.3.2 Evolutionary Algorithm

Candidate solutions are synthesised using the Covariance Matrix Adaptation-Evolution Strategy (CMA-ES), a black-box optimisation method that is quasi parameter-free (Hansen and Ostermeier, 2001). In its original version, CMA-ES operates in unconstrained real space $\mathbb{R}^{2d}$, where $d = 2$ in our study. Therefore, we need to perform a transformation from the candidate solutions provided by CMA-ES onto valid controllers. We achieve this by applying the following sigmoid-based function to each value in the candidate solution:

$$\text{sig}(x) = \frac{1 - e^{-x}}{1 + e^{-x}}, \quad \forall x \in \mathbb{R}. \tag{3.10}$$

The only external parameters that are required by the CMA-ES algorithm are the following: a population size λ, an initial guess of a solution, $\mathbf{m}^{(0)}$, and an initial step size $\sigma^{(0)}$. We set $\mathbf{m}^{(0)} = \mathbf{0}$ and $\sigma^{(0)} = 0.72$. Using Monte Carlo simulations, Gauci et al. (2014b) reported that these settings provide an approximately uniform distribution over $[-1, 1]^{2d}$ in the initial generation.

The evolutionary algorithm maximises the fitness function given in Equation (3.9). In CMA-ES, a population size, λ needs to be chosen. There are no strict guidelines; as discussed in Hansen and Ostermeier (2001). The authors discuss that whilst a smaller population size usually converges faster, a larger population size avoids local optima. To choose λ, we follow the advise given by the authors, that is, $\lambda \leq 2n + 10$, where

[3]The area coverage measure was not used during the evolutionary process because it is computationally more demanding.

[4]In principle, a 0-value is not possible, as at least 1 cell has to be covered.

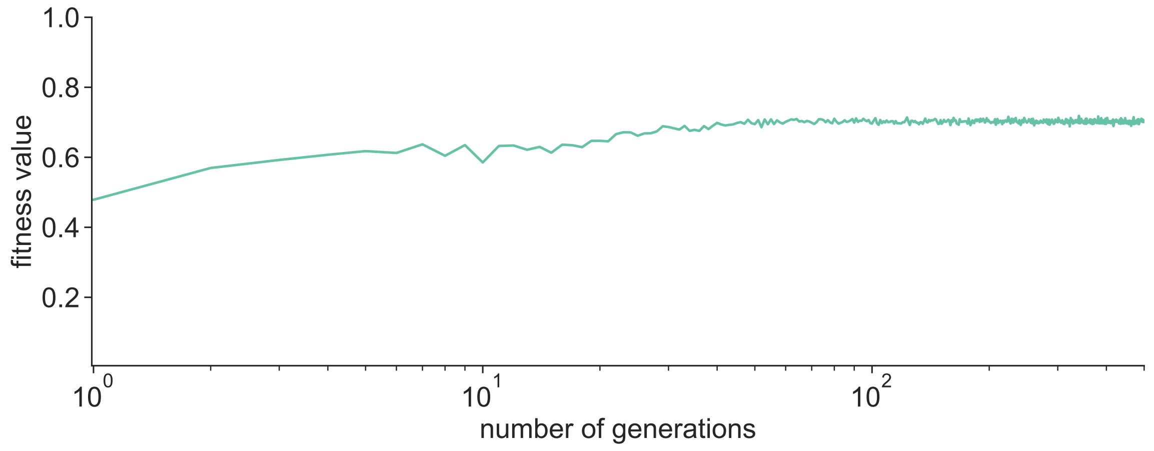


Fig. 3.2 Fitness dynamics of 50 evolutionary runs for 500 generations. The line plot shows the mean fitness value and the envelope shows the minimum and the maximum value of each generation. The number of generations in x-axis is given in logarithmic scale.

n is the number of control parameters. We use a population of $\lambda = 12$ candidate solutions, of which $\mu = \lambda/2 = 6$ are selected for reproduction. Initially, the candidate solutions are generated randomly using uniform distributions. In every generation, each candidate solution is evaluated via simulation trials as described in Section 3.3.1. The evolutionary run terminates after 500 generations.

3.3.3 Controller Selection

The source code for interaction between the Enki physics library and CMA-ES was adapted from the source code used in Gauci et al. (2014c). Although the implementation of CMA-ES remained the same, substantial changes were made to the remaining source code, including improvements on the simulation setup, environment model, random initialisation, maze-wall generation, fitness function calculation, random walk model and file input/output. The batch-job submission, simulation compiler, visualiser (renderer), and data analysis source code were written by the author.

In total, 50 evolutionary runs were conducted. The fitness dynamics are shown in Figure 3.2. In 4 runs, the evolution prematurely converged towards solutions of lower quality than in the other 46 runs. The mean fitness value for each generation stalls after approximately 100 generations, indicating, there is only a minor improvement in the performance.

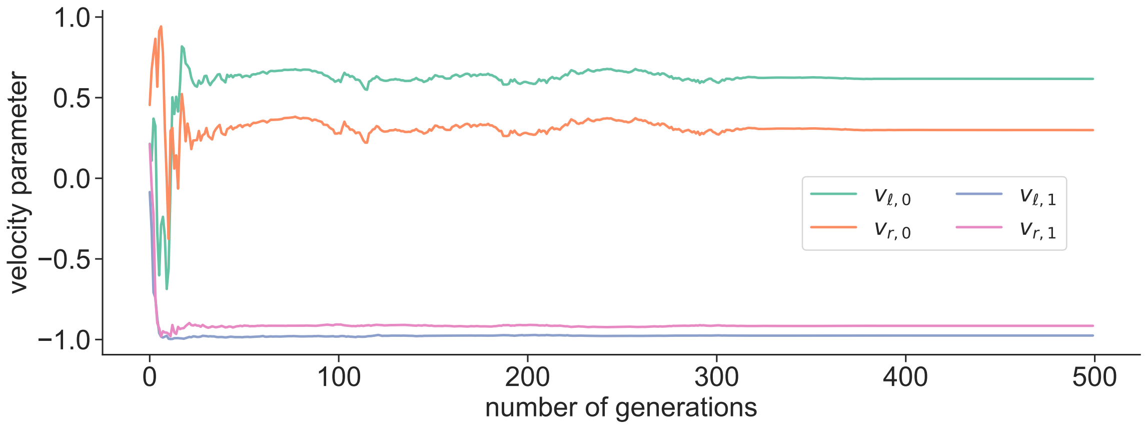


Fig. 3.3 Control parameter evolution for the *best* controller. Each line plot represents a velocity parameter.

Table 3.1 The parameters of the best controller.

nothing	robot (agent)
$v_{\ell,0}$	$v_{\ell,1}$
0.719558	-0.998071
$v_{r,0}$	$v_{r,1}$
0.412543	-0.911843

To choose the *best* controller out of the 50 evolutionary runs, we post-evaluated the highest-ranked candidate solutions from the last generation of each run using 200 additional simulations, and chose the one with the highest mean performance.

Figure 3.3 shows the evolution of the selected controller parameters. After approximately 50 generations, there is almost no change in the parameters $(v_{\ell,1}, v_{r,1})$. On the other hand, for $(v_{\ell,0}, v_{r,0})$ it took 300 generations to converge to the final values. This is inline with the fitness dynamics, where there is no significant performance improvement once the parameters stall.

The following section presents the best controller and mathematical analysis on the movement it produces.

3.3.4 Mathematical Analysis

The parameters of the best controller are given in Table 3.1. Figure 3.4 shows a pictorial representation of the controller. The red and blue arcs show the directions

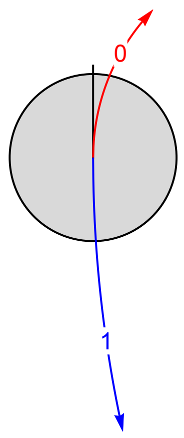


Fig. 3.4 A pictorial representation of the best controller. The red arc shows the primitive behaviour of the robot when its sensor reads $s = 0$ (nothing), and the blue arc shows the behaviour when $s = 1$ (robot/agent).

the robot would head depending on its sensor reading. The angle and length of the arcs are proportional to the angular and linear velocities of the controller.

As long as the sensor reading does not change, the robot follows a circular trajectory of radius R, with an angular velocity ω (Dudek and Jenkin, 2010). We obtain $R_0 = 9.40\,\text{cm}$ and $\omega_0 = -0.77\,\text{rad/s}$ for $s = 0$, and $R_1 = 56.48\,\text{cm}$ and $\omega_1 = 0.216\,\text{rad/s}$ for $s = 1$. When the robot detects another robot in its line of sight, $s = 1$, it moves rapidly backward (with 95.5% of the maximum linear speed) along a circular trajectory, in a counter-clockwise fashion. Otherwise, it moves forward (with 56.6% of the maximum linear speed), along a circular trajectory, in a clockwise fashion. If the two radii, R_0 and R_1 were the same, the robot would remain on its orbit indefinitely (assuming no collisions), and hence would be unable to spatially disperse. We hypothesise that for any $R_0 \ll R_1$, spatial separation can be achieved.

We analyse the movement of one individual robot executing the *best* controller. For the analysis, we assume that the robots response to changes in the sensor reading $s(t)$ instantaneously, unlike the simulation setup (see Section 3.2.3—the robot control cycle is activated every 0.1 s).

Lemma 1. *Using wheel speeds* $(v_{\ell,1}, v_{r,1})$ *a robot repel from a static robot.*

Proof. Recall that we have $(v_{\ell,1}, v_{r,1}) = (-0.998071, -0.911843)$. Using the kinematic motion model for differential drive robots given in Equation (3.1), the linear and the angular speeds of the focal robot is

$$\begin{aligned} v_1 &= \frac{-0.998071 + (-0.911843)}{2} 12.8 &= -12.223\ cm/s, \\ \omega_1 &= \frac{-0.911843 - (-0.99807)}{5.1} 12.8 &= 0.216\ rad/s, \end{aligned} \tag{3.11}$$

Let's assume that two robots are aligned in a linear formation from centre-to-centre, with the focal robot pointing at the centre of the static robot. Then the focal robot moves backwards with linear speed of 12.223 cm/s and angular speed of 0.216 rad/s, resulting in a curvilinear trajectory with large curvature.

Whenever the robot stops detecting the static robot due to the curvilinear motion, it then detects $s = 0$, causing to drive with the following velocities

$$\begin{aligned} v_0 &= \frac{0.719558 + 0.412543}{2} 12.8 &= 7.245\ cm/s, \\ \omega_0 &= \frac{0.412543 - 0.719558}{5.1} 12.8 &= -0.77\ rad/s. \end{aligned} \tag{3.12}$$

The linear and angular velocities given in Equation (3.12) leading the robot to move forward with an opposite direction of rotation, cause the robot to detect the static robot. The linear distance travelled by the robot can be given as

$$d_v = (\alpha v_0 + \beta v_1)\Delta t, s \tag{3.13}$$

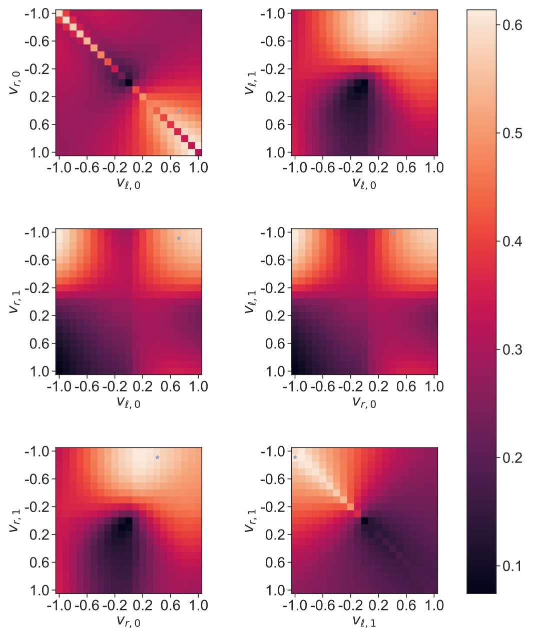
where α and β are the number of time-steps (or portions) the robot utilised consecutively, until it periodically goes back. We have $\omega_0 > \omega_1$, implying that $\alpha < \beta$. The distance travelled,

$$d_v = \big(7.245\alpha - 12.223\beta\big)\Delta t < 0, \tag{3.14}$$

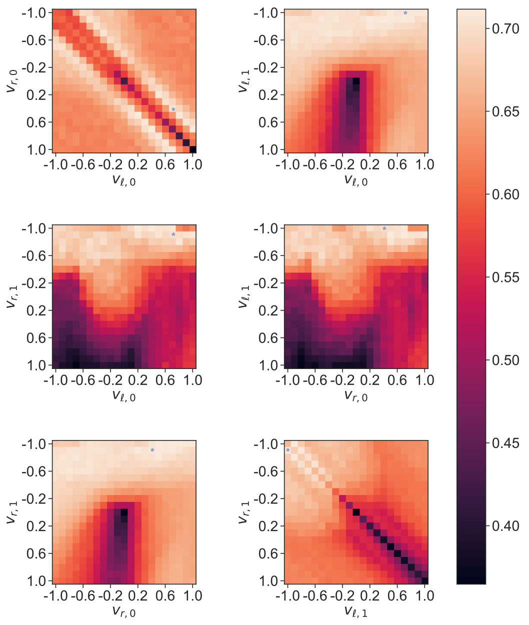
is negative, hence, results in the focal robot to repel from the other robot. ■

Theorem 1. *Using wheel speeds $(v_{\ell,1}, v_{r,1})$ a dynamic robot repels from another dynamic robot.*

Proof. From Lemma 1, we know that a dynamic robot can repel from a static robot. Let's assume that two robots are in a linear formation with their sensors pointing towards on one another's centre. When both of the robots are dynamic, the time period that they detect the other robot ($s = 1$) is shorter, leading to detecting nothing ($s = 0$) earlier. However, as both robots are rotating when $s = 0$, the time period that



(a)



(b)

Fig. 3.5 Exhaustive search over 4 control parameters with grid cells representing the (a) mean, (b) maximum fitness performance. The blue asterisks on the heatmaps show the optimised controller parameter combination.

they detect $s = 0$ is also shorter. As a result, the variations from the initial linear formation would be larger, however, the distance they travelled away from each other would also be larger than if of one the robots was static. ■

3.3.5 Grid Search

In this section, we perform a grid search over the parameter space, in order to estimate the (approximate) global optima of the search space and discuss the significance of the best 'evolved' controller. We divide the search space for each of the 4 velocity parameters, $[-1, 1]$, into 21 data points, and simulate each combination of parameters $K = 20$ times. This computationally expensive procedure requires a total of $21^4 \times 20 = 3889620$ simulation trials. As in the default setup, we simulated $n = 25$ robots for $T = 120\,\text{s}$.

The results of the grid search is 5-dimensional; four of them are the independent velocity parameters (between -1 and 1), and the fifth is the fitness value (between 0 and 1). To visualise the 5-dimensional data, we plotted 2-dimensional heatmaps with (x, y) coordinates represent the velocity parameters, and the cell colour represents the fitness value. Since there are 4 parameters, and we need to choose 2 at a time, there are $\binom{4}{2} = 6$ plots. On the other hand, when one visualises 5D space on 3D subspace, the other 2 independent parameters become obsolete. In order to highlight the characteristics of the results, we opt to plot the fitness values in two settings.

In Figure 3.5(a), the grid cell colours represent the *mean* performance of any combination of control parameters with the given two velocity parameters. In other words, the fitness value for $(v_{\ell,0}, v_{r,0})$ is calculated as the following:

$$F^*(v_{\ell,0}, v_{r,0}) = \text{mean}_{(v_{\ell,1},v_{r,1})} F(v_{\ell,0}, v_{r,0}, v_{\ell,1}, v_{r,1}) \tag{3.15}$$

In a similar fashion, Figure 3.5(b) shows the *maximum* performance over the 2D subspace. Formally,

$$F^*(v_{\ell,0}, v_{r,0}) = \max_{(v_{\ell,1},v_{r,1})} F(v_{\ell,0}, v_{r,0}, v_{\ell,1}, v_{r,1}) \tag{3.16}$$

From Figure 3.5, it is clear that the evolved best controller lies in a global optimum region (the lighter-coloured regions) in each of the sub-plots. As the robotic model is differential-wheeled, setting the velocity parameters to their opposite values would lead to a behaviour that is symmetric. The heatmaps reflect this; for instance, the combinations of $(v_{\ell,0}, v_{r,1})$ and $(v_{r,0}, v_{\ell,1})$ in Figure 3.5(a) and (b) are the same.

The grid search study also highlights the significance of the floating digits used. As one can see, in some of the regions on the heatmaps, the colour transitions between two grid cells are sharp, meaning that the changes in performance are significant. This indicates that using 2 floating digits would lead to a coarse representation, and would not capture the characteristics of the system as desired. Further simulation results show that there is a 6.68% and 4.60% increment in performance when using 6 floating digits compared to 2 and 3, respectively.

Additionally, we performed a sensitivity analysis via means of standard deviation of the fitness performance. For each of the control parameter pairs (as in Figure 3.5), we calculated the standard deviation of the average performance (i.e. fitness value). The results yielded that the $(v_{\ell,0}, v_{r,0})$ pair has a standard deviation of 0.0748 and the pair $(v_{\ell,1}, v_{r,1})$ has a standard deviation of 0.1213 using Equation (3.15). When using Equation (3.16), the standard deviation values are 0.0386 and 0.0605, respectively. These results indicate that the swarm's performance is more sensitive to the changes for the control parameters $(v_{\ell,1}, v_{r,1})$ than to $(v_{\ell,0}, v_{r,0})$.

3.4 Simulation Studies

In this section, we evaluate the performance of the controller (Section 3.3.3) using a series of simulation studies. Unless otherwise stated, we use the same experimental setup as during the optimisation process (see Section 3.3.1). We report the coverage performance observed at the end of the simulation trials using both cell coverage and area coverage metrics as given in Equation (3.5)-(3.7), respectively.

For area coverage, we calculate a lower bound for the smallest radius that can obtain complete coverage. Coverage radius of r_{cover} can be obtained by using the following equation derived by Kershner (1939):

$$n\pi r_{\text{cover}}^2 = \frac{2\pi\sqrt{3}}{9} A(\mathcal{E}). \tag{3.17}$$

By solving Equation (3.17) for $n = 25$ robots and $A(\mathcal{E}) = 300 \times 300\,\text{cm}^2$, we obtain $r_{\text{cover}} = 37.22\,\text{cm}$.

3.4.1 Performance Comparison with Different Strategies

We compare the computation-free controller against three other controllers:

- *Open-loop*: The robot moves backward with maximum speed; $(v_{\ell,0}, v_{r,0}) = (v_{\ell,1}, v_{r,1}) = (-1, -1)$. This controller characterises the robot's response to any sensor reading as repelling. Due to symmetry, this strategy is identical to $\mathbf{v} = (1, 1, 1, 1)$.

- *Greedy*: The robot moves backward with maximum speed if another robot is detected, $(v_{\ell,1}, v_{r,1}) = (-1, -1)$, and otherwise turns clockwise on the spot with maximum angular velocity; $(v_{\ell,0}, v_{r,0}) = (1, -1)$. This controller allows the robot to 'search' for the presence of any other robot by rotating on the spot, and repelling from it once it encounters one.

- *Random walk*: We use the random walk framework studied in (Dimidov et al., 2016). The random walk consists of alternating straight-line segments of random length and on-the-spot rotations by random angles. The nature of the random walk is characterised by a triple, (ρ, α, β). The first parameter, $\rho \in (0, 1)$ controls the correlation between angles of subsequent segments, while the other two parameters, $\alpha \in (0, 2]$ and $\beta \in (0, \infty)$, control, respectively, the shape and scale of the distribution from which the lengths of the segments are drawn (for details, see Dimidov et al. (2016)). To optimise (ρ, α, β), we follow the same process that was used for optimizing our computation-free controller. As in Section 3.3.3, 50 evolutionary runs were conducted, and the best controller of each run was post-evaluated to select the overall best random walk for the present environment.

For each strategy, 1000 trials were performed. Figure 3.6 shows the results. Regarding cell coverage (left), the mean performance is 76.0% for the proposed controller, 22.4% for the deterministic *open-loop* controller, 60.8% for the deterministic, *greedy* controller, and 60.8% for the optimised *random walk* controller, respectively. Regarding area coverage (right), the mean performance is 71.2% for the proposed controller, 16.0% for the deterministic *open-loop* controller, 51.8% for the deterministic, *greedy* controller, and 61.3% for the optimised *random walk* controller, respectively. The proposed controller achieves a 25.6–38.8% reduction in the uncovered area when compared to the *random walk* controller. In addition, its performance is more consistent; the variances for cell and area coverage are 4.99% and 6.36% for the proposed controller and *random walk* controller, respectively. To evaluate the significance of the results, we conducted a one-way ANOVA test. The F-values for cell coverage is 18543 and for area coverage is 47395, whilst the p-values are both ≈ 0.0, indicating the swarms' performance is significantly different when using one of the four controllers.

3.4.2 Sensory Noise Analysis

We examine the situation that noise is affecting the reading values of the binary line-of-sight sensor. This noise is in addition to the noise affecting the wheel velocities.

We consider three types of noise:

- *False-negative*: For $s(t) = 1$, with probability $p \in [0, 1]$ the reading value is replaced by a random value, $X(t) \in \{0, 1\}$, which is uniformly chosen at time t. For $s(t) = 0$, the original value is retained.
- *False-positive*: For $s(t) = 0$, with probability $p \in [0, 1]$ the reading value is replaced by a random value, $X(t) \in \{0, 1\}$, which is uniformly chosen at time t. For $s(t) = 1$, the original value is retained.
- *Combined*: With probability $p \in [0, 1]$ the reading value is replaced by a random value, $X(t) \in \{0, 1\}$, which is uniformly chosen at time t.

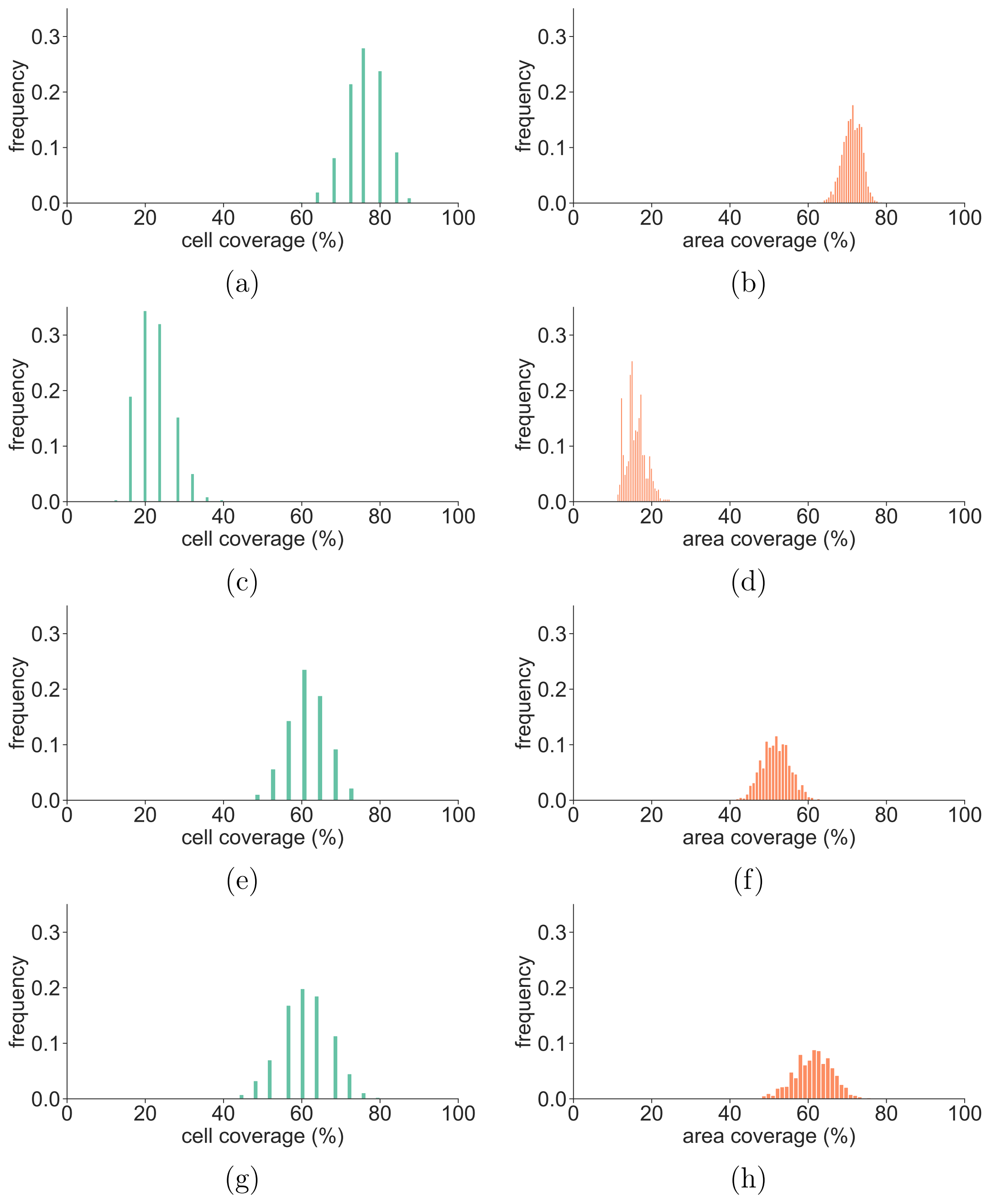


Fig. 3.6 Histogram showing the frequency of cell coverage (left) and area coverage (right) percentages, as observed in 1000 simulation trials with $n = 25$ robots, controlled by (a)–(b) our optimised controller, (c)–(d) an *open-loop* controller, and (e)–(f) the *greedy* controller, all of which are deterministic and computation-free, as well as (g)–(h) the optimised *random walk* controller, which uses computation to count control cycles while moving forward and turning, and to generate pseudo-random numbers from tailored distributions for the step length and turning angle. The environment has dimensions 300 cm × 300 cm.

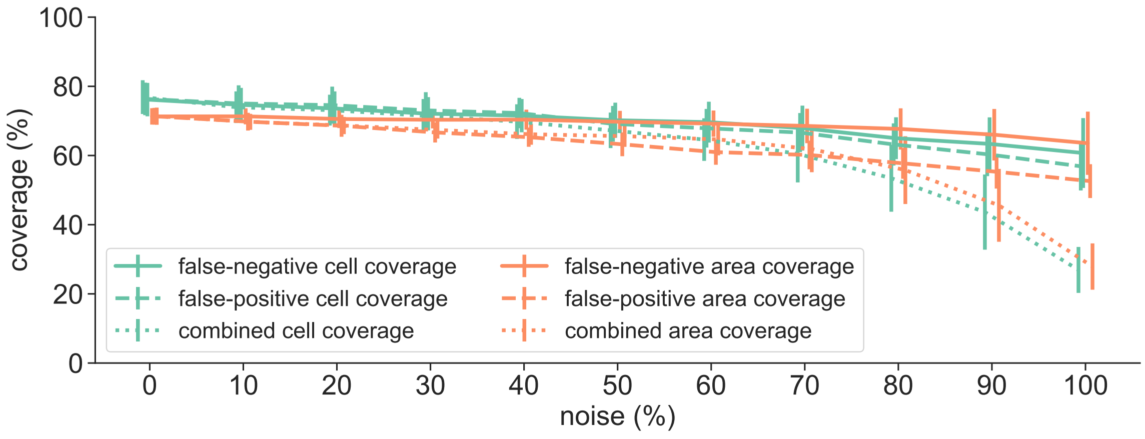


Fig. 3.7 Cell coverage (blue) and area coverage (orange) of $n = 25$ robots with noisy sensors and the computation-free controller. Three types of noise: (i) false-negative (solid), (ii) false-positive (dashed), and (iii) combined (dotted). Mean values and standard deviations based on 100 simulation trials per level and type of noise. The environment has dimensions 300 cm × 300 cm.

We run 100 simulation trials for each $p \in \{0, 0.1, 0.2, \ldots, 1\}$.

Figure 3.7 shows the results. The controller is highly robust to noise on only one sensor reading (i.e. either false-negative or false-positive), with both performance measures reporting over 60% coverage with 100% noise. The performance degrades faster if noise is present on both sensor readings (i.e. combined), down to around 20% at 100% noise. Note, however, that in this case 100% noise represents a purely random sensor reading.

3.4.3 Scalability Analysis

The default setup contains no redundancy. If a single robot fails, complete coverage can no longer be achieved. We investigate the performance of swarms of $n \in \{5, 10, 15, \ldots, 100\}$ robots in an environment of constant size (the default environment). Irrespective of the swarm size, the robots are initialised in a circular region of radius 60 cm, which is uniformly randomly placed within the boundaries of the environment. We run 100 simulation trials for $T = 120$ s per swarm size.

Figure 3.8 shows the results. The average performance rapidly improves for up to around 40 robots, then plateaus, and finally improves again for 80 and more robots. The presence of the plateau, where the performance may even slightly decrease, is a counter-intuitive result, to be further investigated in the future.

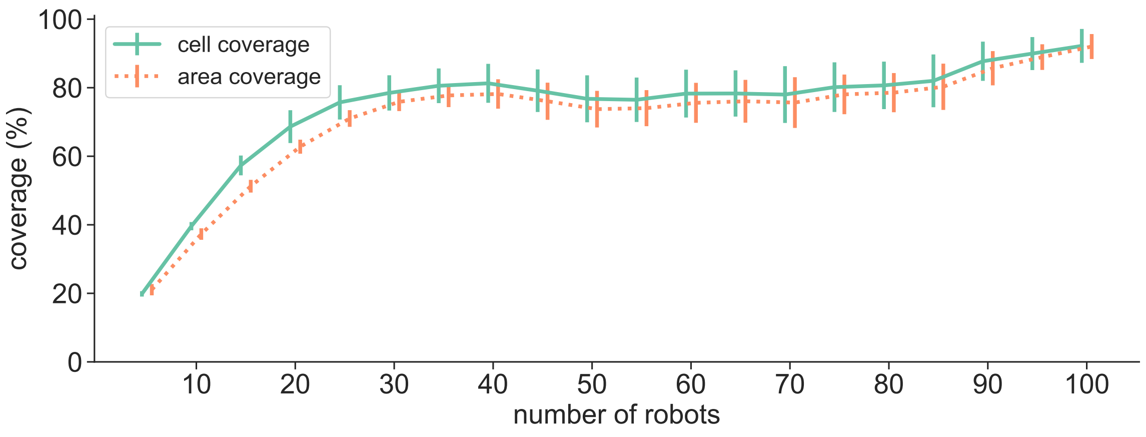


Fig. 3.8 Cell coverage (blue) and area coverage (orange) of $n = 5, 10, 15, \ldots, 100$ robots in a 300 cm × 300 cm environment using the computation-free controller. Mean values and standard deviations based on 100 simulation trials.

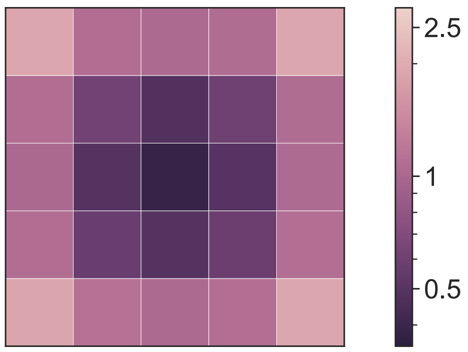


Fig. 3.9 Spatial distribution of $n = 25$ robots in a 300 cm × 300 cm environment. Each cell indicates the average number of robots present at the end of 1000 simulation trials.

3.4.4 Effect of the Environment Shape

We investigate the spatial distribution of robots in more detail, and consider the effect of the environment shape.

Figure 3.9 presents a heatmap of the average number of robots that ended up in the 25 cells at the end of 1000 simulation trials. On average, the robots are more likely to be present at the environmental boundary, and in particular, in the four corners. Note that the robots are unable to detect the boundary. They repel from each other in an attempt to cover as much area as possible.

To investigate the swarms' ability to spread through an elongated, narrow corridor, a further 1000 simulation runs are performed in a 25×1 cell environment, of 1500 cm × 60 cm dimensions. Figure 3.10 shows a heatmap of the spatial distribution. The results

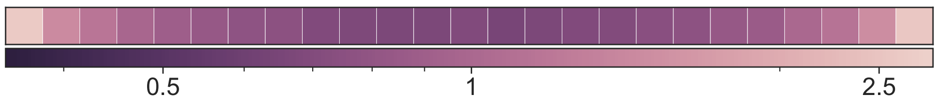


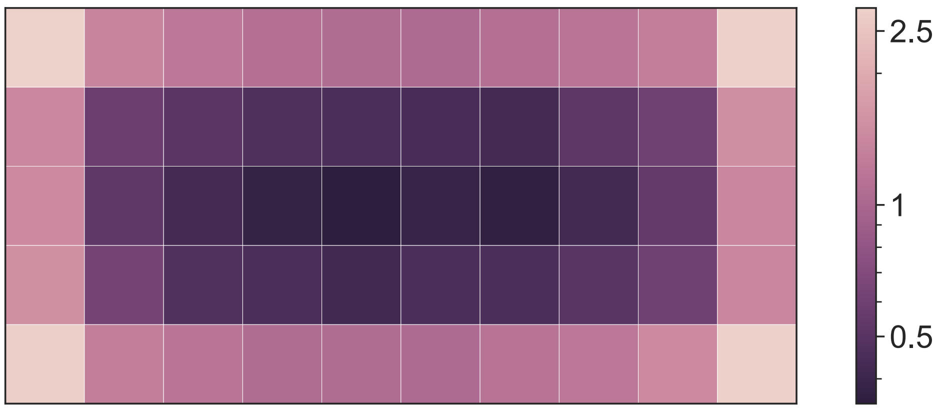
Fig. 3.10 Spatial distribution of $n = 25$ robots in an elongated, narrow corridor environment, of dimensions 1500 cm × 60 cm. All robots start from the cell in the centre and execute the computation-free controller for $T = 600$ s. Each cell indicates the average number of robots present at the end of 1000 simulation trials.

are consistent with those obtained in the square environment—the robots are more likely to end up near the corner than in the centre. However, every cell is covered, on average, by 0.65 robots or more.

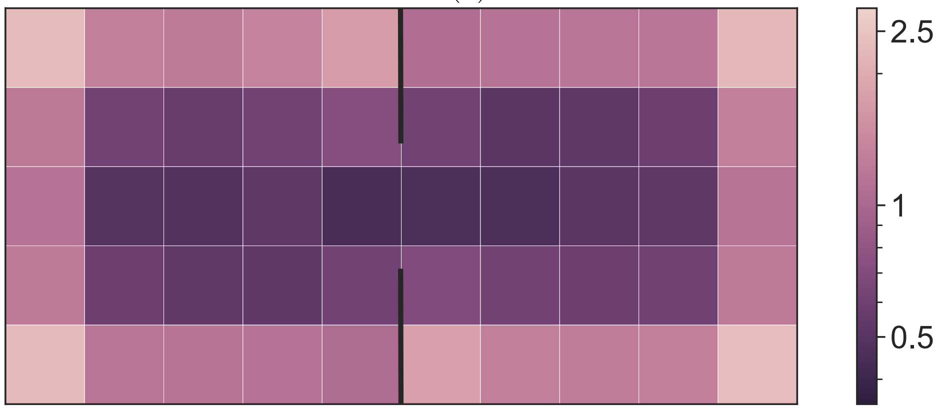
Figure 3.11 shows heatmaps for three 600 cm × 300 cm environments. We assume a 10 × 5 cell composition. The first environment is free of obstacles. The second environment is split into two by a thin vertical wall in the centre. The wall contains an orifice in the middle. The third environment contains two vertical walls, creating a z-shaped parkour. As the environment is twice the size of the original environment, we used $n = 50$ and $T = 240$ s. In general, the swarm copes well with the restrictions. It can be noted that the distributions are not fully symmetric in Figure 3.11(b).

3.4.5 Navigating a Maze

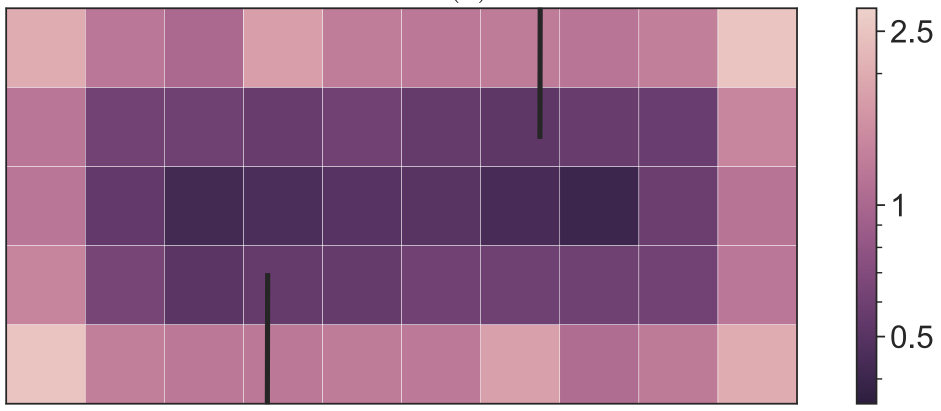
In the following, we test the ability of the computation-free controller to make a swarm navigate a simple but unknown maze. The maze, shown in Figure 3.12a, contains a number of challenges, including two dead-ends. The robots enter the maze on the left-hand chamber, at a rate of one per 10 s, 12 s, 15 s, or 30 s. They execute the same controller as used in the coverage experiments. Robots are removed as soon as they are fully contained within the right-hand chamber. Figure 3.12b shows the number of robots within the maze over time. As one can see, the swarm can navigate the maze with a throughput of 1 robot per 15 s. However, as the input rate of new robots increases, the maze becomes increasingly crowded, up to the point that no new robots can be placed.



(a)

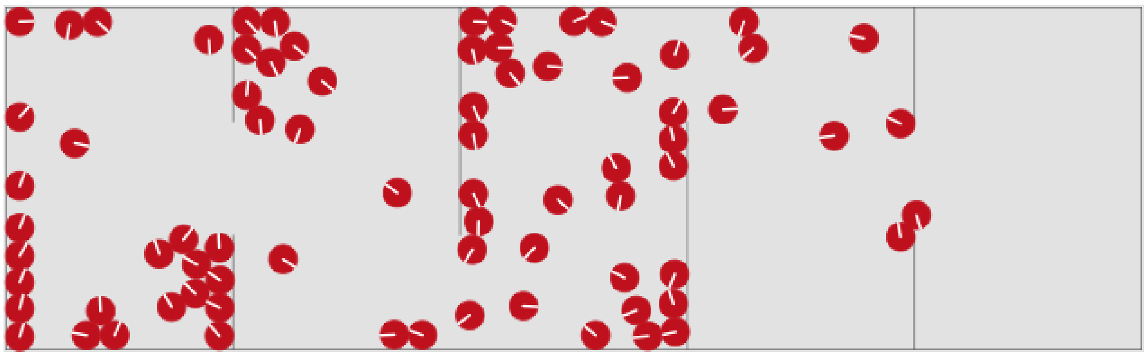


(b)

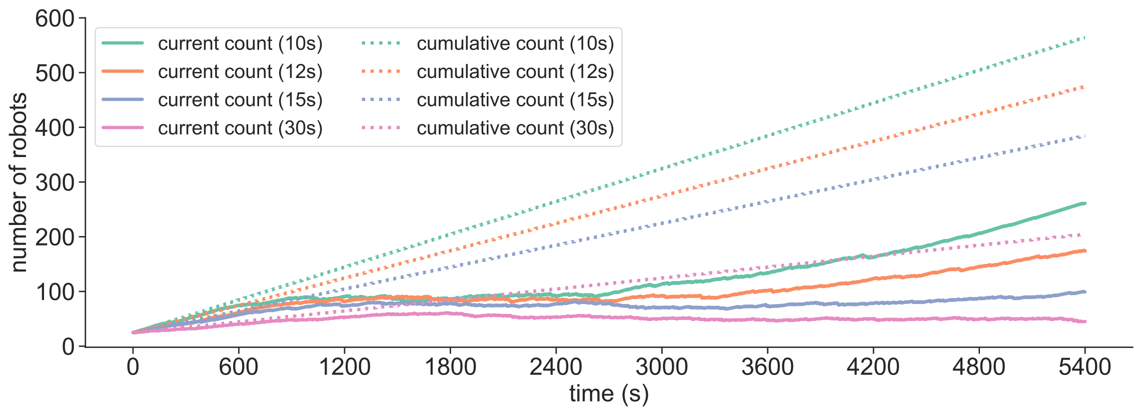


(c)

Fig. 3.11 Spatial distribution of $n = 50$ robots in a 600 cm × 300 cm environment, with (a) no obstacles, (b) a pair of internal walls, creating an orifice, and (c) a pair of internal walls, creating a z-shaped parkour. Each cell indicates the average number of robots present at the end of 1000 simulation trials.



(a)



(b)

Fig. 3.12 A swarm of robots using the computation-free controller to navigate a maze. New robots enter the left chamber of the maze environment at a constant rate of one per 10 s, 12 s, 15 s, 30 s. Robots are removed as soon as they are fully within the right chamber. (a) Snapshot taken after 3600 s with a rate of one robot per 15 s. (b) Number of robots that are, or have been, within the maze over time for each of the rates.

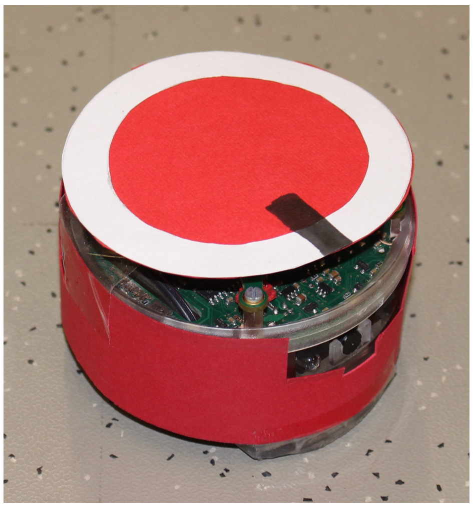


Fig. 3.13 Close view of the robotic platform used in the experiments. The e-puck is coated in *red* to be identifiable by other e-pucks. The marker on its top is used by the tracking system for the post-analysis.

3.5 Experiments

3.5.1 Experimental Setup

To validate the feasibility of computation-free coverage in a real environment, we conducted experiments using $n = 25$ physical e-puck robots in a bounded $300\,\text{cm} \times 300\,\text{cm}$ environment. The arena was logically split into a 5×5 grid of cells of $60\,\text{cm} \times 60\,\text{cm}$ dimensions.

For each experimental trial, the e-pucks are initialised in a different cell, as was done in the computer simulations. The cells are selected using a uniformly random distribution. Moreover, the robots' assume random orientations during initialization. The robots operate for $T = 120\,\text{s}$.

3.5.2 Porting of the Controller

We use the e-puck robots (Mondada et al., 2009) to validate the obtained controller on a physical platform. The e-puck has a CMOS-RGB camera that faces in the forward direction, and is used to emulate the line-of-sight sensor. Each e-puck is wrapped in a red 'coat' and operates in a well lit, white environment to improve the reliability of detection. The e-puck is also equipped with a red 'topper' to enable easier detection from the overhead camera for post-analysis. A close view of an e-puck robot is shown in Figure 3.13. Each e-puck is slightly physically different and, due to only having two wheels, the camera direction, along the pitch axis, may slightly change during a robot's movement. To account for these misalignments, the line-of-sight sensor probes not 1 pixel, but rather a vertical column of 7 pixels, taken symmetrically from the centre of the image.

The line-of-sight sensor returns a positive reading ($s = 1$) if the colour of any of the 7 pixels is not bright[5], and a negative reading ($s = 0$), otherwise.

Even though the experiments in this chapter focuses on a particular robotic system, e-pucks, in general the control strategy can be used in another ground robotic system with similar capabilities.[6] The strategy can be ported onto a different robotic system, though, the evolutionary algorithm may need to be re-employed, as the fitness values strictly depend on the simulation results considering a particular robotic swarm. For the robotic systems that are highly similar, such as mobile differential-wheeled robots, one can use Equation (3.1) to obtain the linear and angular velocities for the control parameters given in Table 3.1. Then, one can convert the linear and angular velocities back to the control parameters for the desired robotic system.

3.5.3 Results

We performed 20 experimental trials. All trials were recorded by the overhead camera, and are available in the online supplementary material (Özdemir et al., 2019a). From these video recordings, the positions of robots could be tracked using OpenCV tracking library (Bradski, 2000). Figure 3.14 shows an example sequence of snapshots from a typical trial.

Figure 3.15 summarises experimental results for all trials. On average, the swarm achieved a cell coverage and area coverage of 65.2% and 64.9%, respectively, which outperforms the previously obtained benchmarks (recall that the corresponding values for the random walk controller were 60.8% and 61.3%, in simulation). The reduction in performance may be attributed to the increased friction between the walls and robots,

[5]We test $(R, G, B) \prec (180, 180, 140)$, where (R, G, B) is the RGB triplet of the colour, and $\prec$ induces the partial order called product (or component-wise) order. Whereas a test if a value is smaller than another requires computation when implemented by digital circuits, an implementation in the analogue world is trivial (single high-gain differential amplifier). Note that irrespective of how the line-of-sight sensor is implemented on the e-puck, the control logic remains free from arithmetic computation.

[6]Technically, the robot needs at least 2 degrees of freedom. A robot operating on a water surface would also be suitable.

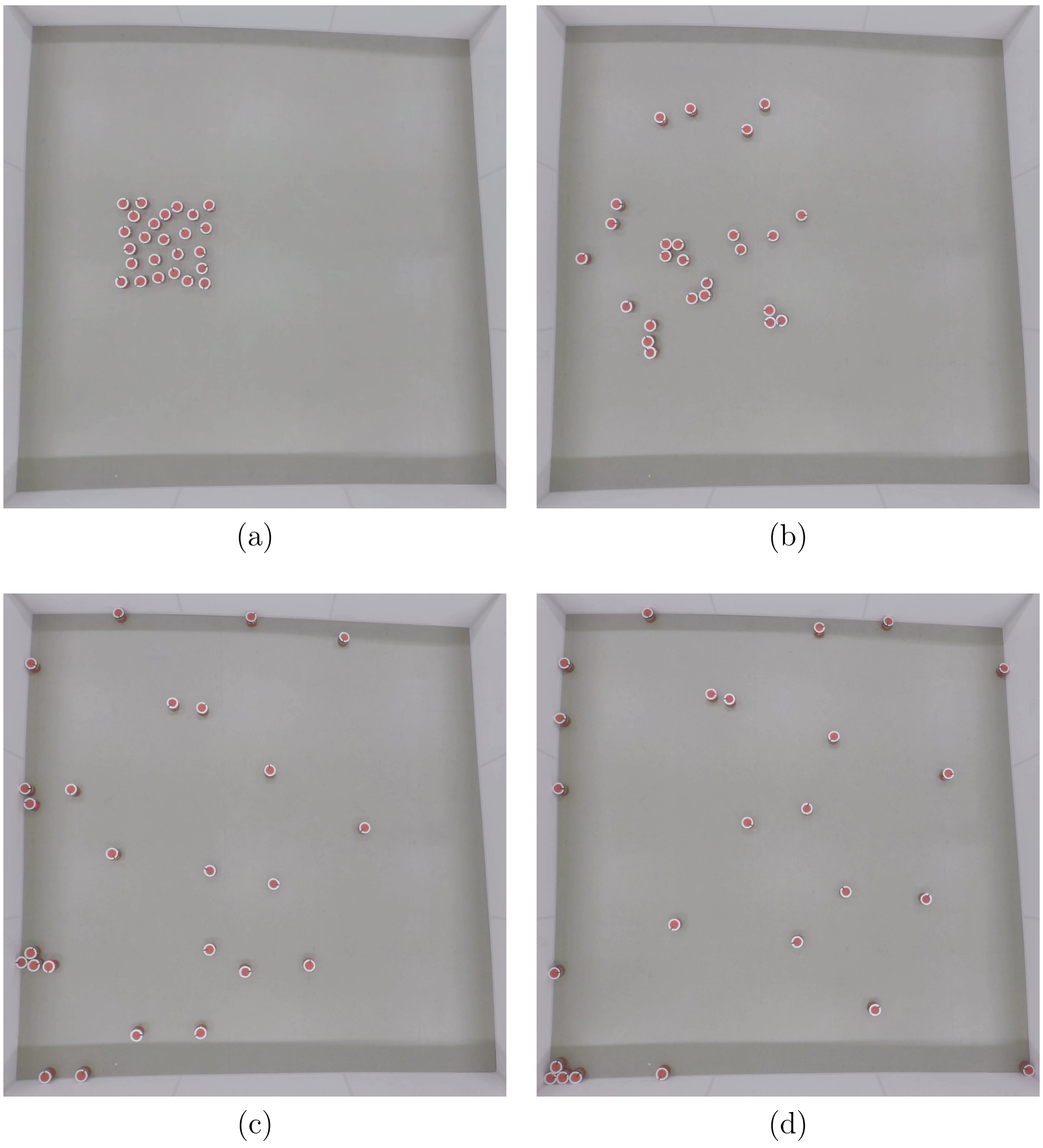
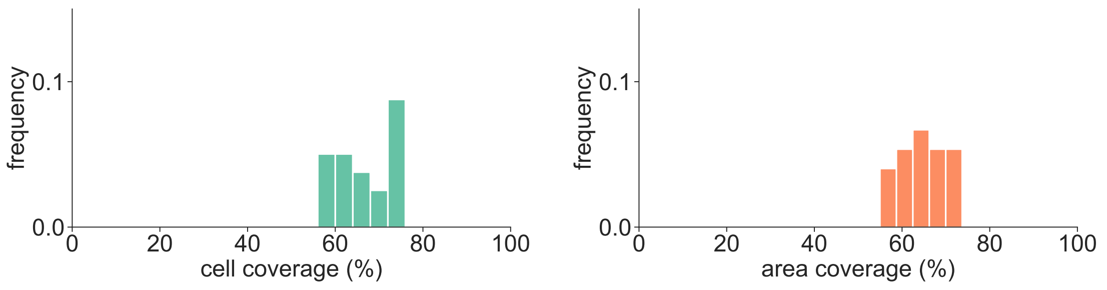


Fig. 3.14 Sequence of snapshots taken from a typical experimental trial with $n = 25$ physical e-pucks operating in a 300 cm × 300 cm environment. The snapshots were taken at (a) 0 s, (b) 10 s, (c) 30 s, and (d) 120 s.



(a) (b)

Fig. 3.15 Histogram showing the frequency of cell coverage (a) and area coverage (b) percentages, as observed in 20 experimental trials with $n = 25$ robots, controlled by our optimised controller, which is deterministic and computation-free. The environment has dimensions 300 cm $\times$ 300 cm.

which could prevent them from continuing to rotate upon collision. Other factors include sensory noise and unknown hardware failures. Out of the 20 trials, 3 robots powered off during a trial, possibly due to low battery, which was rectified for the following trials.

3.6 Discussions

In this chapter, we presented the simplest solution so far to the problem of cooperatively covering an unknown two-dimensional space with a swarm of anonymous mobile robots. The proposed controller is applicable to robots that lack run-time computation or storage. The solution requires only a single bit of information per robot—whether or not another robot is present in its direct line of sight.

A series of computer simulations showed that the controller outperformed a random walk (both solutions being optimised off-line). On average, in situations with no robot redundancy, it covered around 71–76% (depending on the coverage measure being used) of the space, whereas the random walk covered around 61% of the space. The swarm performance was found to degrade gracefully in the presence of noise. In the case of either false-negative or -positive noise, the swarm showed robust performance up to a noise level of 100%. Moreover, the performance was not particularly affected by the robot density in the environment. An analysis of the spatial distribution revealed that on average more robots ended up near the boundary, and in particularly any corners, reducing the efficiency in scenarios without robot redundancy. A further simulation experiment revealed that a constant-rate inflow of robots can navigate a maze up to a critical rate.

Despite the robots are unable to measure distances, experiments with swarms of 25 physical e-puck robots demonstrated the feasibility of computation-free coverage on a real physical platform. They revealed a moderate decrease in performance, 65.2% and 64.9% coverage, when compared to simulation trials.

In this chapter, we focused on the coverage performance and bounded regions that were to be uniformly covered. The simple nature of the control strategy allowed the robots to consume less energy and power. A potential application in the short-to-medium term could be deploying a swarm of simple robots for monitoring an area, logging data, and reporting anomalies. In the next chapter, we will focus on consensus finding strategies of swarm robots to collectively choose a direction to move.

Chapter 4

Finding Consensus

4.1 Introduction

In the previous chapter, we synthesised and analysed a computation-free control strategy for multi-agent spatial coverage problem. The agents extracted only a single-bit of information, whether there is another agent in their direct line of sight. The strategy allowed the agents to simultaneously cover the environment. The spatial coverage abilities of the swarm can also lead to a systematic exploration strategy. In addition, the robotic swarm was also tested in a maze navigation scenario.

We know that the computation-free swarm is able to aggregate at a point in the environment without any prior knowledge (Gauci et al., 2014c). Now, the question is, can computation-free swarm agents congregate at a point that is chosen from a discrete set of alternatives (e.g. two alternatives)? The underlying question requires the swarm to be able to "find consensus" on which of the alternatives they will 'choose'. We hypothesise that a swarm of agents utilising a deterministic computation-free controller can emerge a self-organised behaviour to choose between multiple equal—or unequal—alternatives in their environment. Hence, in this chapter, we will investigate another canonical multi-agent task that requires coordinated movement abilities; finding consensus.

The ability of groups of decision-making agents to reach consensus has been studied in a range of disciplines (Bose et al., 2017; Conradt and Roper, 2005; Tsitsiklis, 1984). In general, a group of agents, for example, humans, animals, robots, are operating in an environment that presents multiple options to choose from. The agents have some means of accessing information about these options, and of influencing each other. For example, some ant species use pheromone trails to select the most efficient path to a food source from multiple options (Beckers et al., 1990). In addition, some ant and bee species perform “house hunting”, in which they collectively select and move to a new nest site (Visscher, 2007). The objective for the agents is to reach an agreement on which option to choose. Hereafter, we refer to this problem as the *collective choice* problem.

In this chapter, we study the collective choice problem with a group of simplistic robotic agents, and are interested in the situation where:

- The group of agents is homogeneous, in other words, all agents are identical;
- There are two or more options to choose from;
- The options are of equal value[1], so the group of agents is no better off choosing one over the other;
- The agents’ environment does not contain any cues that could help or hinder the selection process.

This chapter is organised as follows. Section 4.2 defines the collective choice problem, the environment model, and the sensing, locomotion, and control capabilities of the agents. Section 4.3 presents the evolutionary algorithm for synthesising the control strategy. Section 4.4 presents the results obtained when testing the control strategy on swarms of simulated robots. Section 4.5 describes how the strategy was ported to a physical robot platform, and presents the experimental results obtained with swarms of 20 e-puck robots. Section 4.6 concludes the chapter.

4.2 Problem Definition

4.2.1 Objective

Consider a 2D, bounded environment with two identical, circular objects, A and B, referred to as *options*. The options are placed equidistant from the centre of the environment. The environment does not contain any other cues. The scenario thus corresponds to the symmetric option qualities and costs variant of the best-of-n

[1]This assumption is relaxed in Section 4.4.6, where options of different sizes are considered.

problem (Valentini et al., 2017). A group of N mobile agents is initially placed within a region in the centre.

The collective choice problem requires the group to commit to either of the two options within a fixed time period. An agent is considered to be committed to option $X \in \{A, B\}$, if it is within a certain range of X. Throughout this chapter, an agent can commit to at most one option. The group is considered to be committed to option X, if more than $N/2$ agents—the majority—are committed to X. Note that even if the group committed to an option, a minority of agents could still have committed to the other option, resulting in a split. Splitting is in general undesirable, however, for simplistic agents, not always avoidable (Franks et al., 2007).

4.2.2 Environment and Robot Model

Each agent is equipped with one line-of-sight sensor at its front, which is able to detect the type of object at which it is pointing. The sensor has a limited range. At each time step, t, an agent's sensor provides one of three possible readings:

$$s(t) = \begin{cases} 2 & \text{if an option is detected,} \\ 1 & \text{if another agent is detected,} \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

The agent moves using a differential-drive wheel configuration (Dudek and Jenkin, 2010). In other words, it can move forwards or backwards in arcs of arbitrary radius—including straight motion and on-the-spot rotation. The agent therefore has two degrees of freedom, namely the rotational velocities of the left and right wheels, which we denote by v_ℓ and v_r, respectively. Each wheel velocity can be normalised to the interval $[-1, 1]$, where -1 and 1 represent a wheel rotating with maximum speed backwards or forwards, respectively.

We require that the controller[2] shall neither perform any arithmetic computations, nor store any run-time information. From this constraint, it follows that the controller directly maps the sensor reading onto two parameters in $[-1, 1]$—one for each wheel

[2]In the following, we refer to the control strategy simply as the *controller*.

velocity. This takes place at each time step t. Formally,

$$c(s(t)) = \big(v_\ell(t), v_r(t)\big) = \begin{cases} (v_{\ell,0}, v_{r,0}) & \text{if } s(t) = 0, \\ (v_{\ell,1}, v_{r,1}) & \text{if } s(t) = 1, \\ (v_{\ell,2}, v_{r,2}) & \text{if } s(t) = 2, \end{cases} \tag{4.2}$$

where $v_{\ell,i} \in [-1, 1]$ represents the left wheel velocity corresponding to sensor reading $i \in \{0, 1, 2\}$, and similarly for $v_{r,i}$. Note that the controller is fully specified by the six parameters.

The experimental setup is shown in Figure 4.1. It defines a region in the centre for the robots to start from. At the beginning of a trial, $N = 20$ robots are placed at random positions and with random orientations within this region. The setup also defines two *commitment regions*, one for option A, the other for option B. If a robot resides within a commitment region, it is considered committed to the corresponding option.

4.2.3 Simulation Setup

Each agent is a simulated e-puck robot (Mondada et al., 2009), which is a miniature mobile robot with a differential-drive wheel configuration. The e-puck has a circular body with a radius of 3.7 cm and a mass of 152 g. The inter-wheel distance is 5.1 cm, and the maximum wheel velocity is 6.24 rad/s, corresponding to a maximum linear velocity of the robot of 12.8 cm/s. The line-of-sight sensor is implemented by casting a ray and checking whether it intersects with any other object. The ray has a length of 200 cm, limiting the range of the sensor.

The simulator is implemented using the built-in e-puck model of the Enki physics engine (Magnenat et al., 2009). Enki simulates the dynamics and interactions of rigid bodies in 2D. The simulation physics and the control cycle are updated at rates of 100 times per second and 10 times per second, respectively.

4.3 Controller Synthesis

The problem now reduces to finding the six controller parameters that produce the desired behaviour. This is an optimisation problem over the real subspace $[-1, 1]^6 \subset \mathbb{R}^6$.

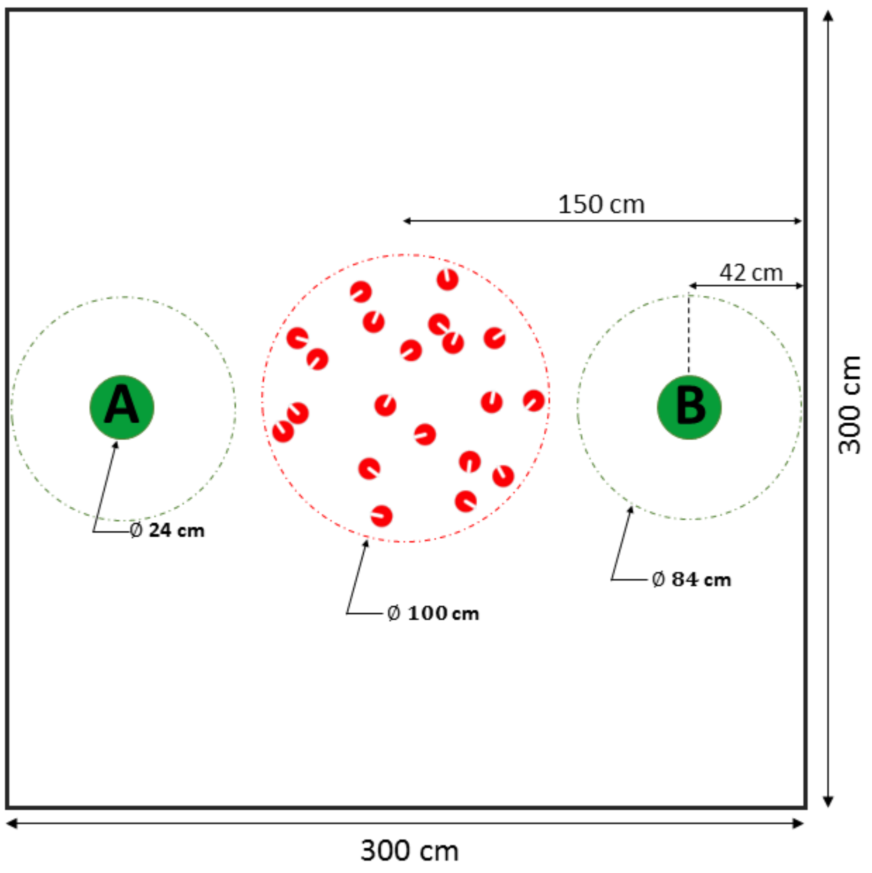


Fig. 4.1 The illustration of the environment. The black lines indicate the boundary of the environment. The robots start from random positions and random orientations within the central region. The green solid disks represent options *A* and *B*. The circular regions around them indicate the corresponding commitment regions.

We approach this problem via the evolutionary robotics approach, where each possible solution can be assigned a score via some evaluation method. An evolutionary algorithm is employed to find good solutions by iteratively generating candidate solutions and using their quality as feedback. Our evaluation method for a candidate solution (i.e. a controller) consists of running a computer simulation with all agents employing that controller. A suitable metric is used to determine the quality of the candidate solution based on the global behaviour that it produces on the agents. Note that while this process is computationally intensive, it is only run once, and it is run off-board of the robots. The obtained controller that goes on-board the robots is free of arithmetic computation. In the following, we describe the evaluation of candidate solutions, the evolutionary algorithm, and the controller selection.

4.3.1 Evaluation of Candidate Solutions

The evaluation uses a bounded square environment with sides 300 cm, containing a group of $N = 20$ robots.[3] The simulation is run for $T = 5000$ time steps (i.e. 500 s). We define a quality measure $Q(t)$, which characterises the distribution of robots at time step t during the trial:

$$Q(t) = \frac{1}{P} min \left\{ \sum_{i=1}^{N} ||x_i(t) - x_A||^2, \sum_{i=1}^{N} ||x_i(t) - x_B||^2 \right\}, \tag{4.3}$$

where $P = (2r)^2 N$ is a scaling factor, r is the radius of the robot's body, $x_i(t)$ is the position of robot i at time step t, and x_A and x_B are the positions of options A and B, respectively. $Q(t)$ is minimised if the robots collectively opt for either option. The fitness function, to be minimised by the evolutionary algorithm, is

$$F = \sum_{t=1}^{T} tQ(t). \tag{4.4}$$

The fitness function is the weighted sum of the quality measure $Q(t)$ and the time step t, over a simulation trial of duration T. By taking the time step into account, the fitness function rewards solutions for reaching consensus—the earlier, the better.

[3]Details about their initial placement are described in Section 4.2.2.

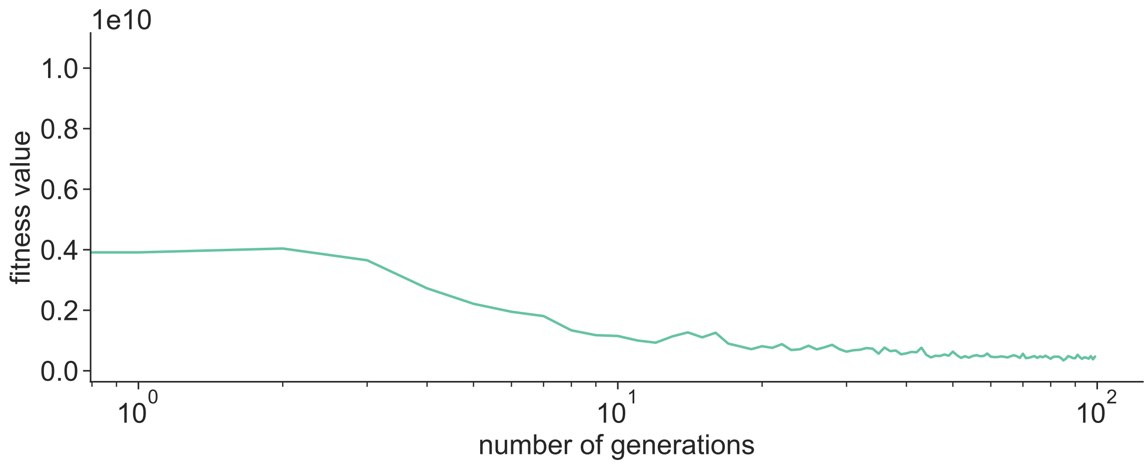


Fig. 4.2 Fitness dynamics of 40 evolutionary runs for 100 generations. The green line shows the mean fitness value, and the envelope represents the minimum and maximum for each generation. The number of generations in the x-axis is given in logarithmic scale.

4.3.2 Evolutionary Algorithm

We use the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen and Ostermeier, 2001). CMA-ES is a derivation-free, stochastic, black-box optimisation method. We use a population of $\lambda = 20$ candidate solutions, of which $\mu = 10$ are selected for reproduction.[4]

The algorithm is executed for 100 generations. Each candidate solution is evaluated 20 times per generation [using Equation (4.4)] and the average fitness value is used.

The evolutionary algorithm requires[5] a transformation for the candidate solutions from the real numbers space $\mathbb{R}^{2d}$ to $[-1, 1]^{2d}$. In order to transform the control parameters, we use a sigmoid-based function, the same as given in Equation (3.10). Additionally, CMA-ES requires an initial guess of a solution, $\mathbf{m}^{(0)}$, and an initial step size, $\sigma^{(0)}$. As in Section 3.3.2, we use $\mathbf{m}^{(0)} = \mathbf{0}$ and $\sigma^{(0)} = 0.72$. Note that the initial guess vector $\mathbf{m}^{(0)}$ is $2d = 6$ dimensional.

Figure 4.2 shows the fitness dynamics of 40 evolutionary runs. The fitness values are high numbers, and unlike the previous chapter, we are unable to bound them in the interval of $[0, 1]$. This is due to the continuous and cumulative nature of the fitness function.

[4]We use the same lower bound to determine the population size, λ, as discussed in Section 3.3.2. However, rather than the bound itself ($\lambda = 16$), we chose $\lambda = 20$ as the final value.

[5]Originally, CMA-ES operates in unconstrained real space.

Table 4.1 The parameters of the best controller [see Equation (4.2)] and the resulting motion primitive.

nothing	robot (agent)	option
$v_{\ell,0}$	$v_{\ell,1}$	$v_{\ell,2}$
0.989377	0.999426	-0.106746
$v_{r,0}$	$v_{r,1}$	$v_{r,2}$
-0.348408	0.992379	0.965466

4.3.3 Controller Selection

No changes were made to the CMAS-ES implementation. However, changes made to the source code were the environment model, random initialisation, fitness function calculation, and random walk model.

We performed 40 evolutionary runs using the aforementioned settings. For each run, we examined the controller of the final generation that exhibited on average the best performance according to Equation (4.4). We observed that six of these 40 controllers achieved a good performance level. We then conducted preliminary experimental trials using these six controllers, and opted for a controller that retained a good performance level in the physical setup. Note that overdesign—also known as overfitting—is a common issue in evolutionary robotics (Birattari et al., 2016), and may explain why some controllers perform differently in reality than in simulation (Jakobi et al., 1995). We refer to this controller as the *best* controller (see Table 4.1). Figure 4.3 shows a pictorial representation of the controller. The black straight line indicates the heading direction of the robot. The red, blue, and green arcs show the directions the robot would head depending on its sensor reading. The angle and length of the arcs are proportional to the angular and linear velocities of the controller.

4.4 Simulation Studies

In this section, the best controller is evaluated using simulation experiments. We evaluate the robots' commitments after 300 s (i.e. $T = 3000$ time steps).[6]

[6]During the evolutionary process, a larger trial duration of 500 s was used to support the incremental development of promising solutions. Post-analysis of the best controller, however, revealed that 300 s is sufficient for the swarm to reach consensus, and hence this trial duration is used throughout all simulation and physical experiments.

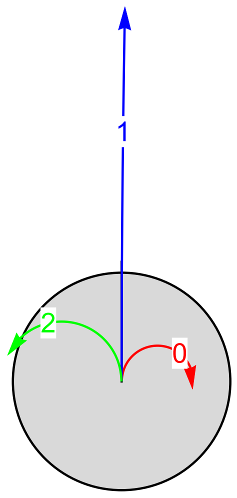


Fig. 4.3 A pictorial representation of the best controller. The black straight line is the heading direction of the robot. The red arc shows the primitive behaviour of the robot when its sensor reads $s = 0$ (nothing), the blue arc is when $s = 1$ (robot) and the green arc is when $s = 2$ (option).

4.4.1 Analysis of the Behaviours

We conducted 1000 trials using the setup described in Section 4.2.2.

Figure 4.4 shows the number of trials in which $N_A \in \{0, 1, 2, \ldots, N\}$ and $N_B \in \{0, 1, 2, \ldots, N\}$ robots committed, respectively, to options A and B. In 97.3% of the trials, the swarm committed to one of the options, either A (i.e. $N_A > N/2$) or B (i.e. $N_B > N/2$). In 12.8% of the trials, the swarm split across both options (i.e. $N_A > 0$ and $N_B > 0$). In Figure 4.4, the inner region of the triangle is virtually empty.[7] In other words, in almost all cases where the swarm split, there were no uncommitted robots left in the environment (i.e. $N_A + N_B = N$).

We now analyse the best controller (see Table 4.1) in more detail. Consider a robot at time step t. If the robot detects nothing ($s(t) = 0$), it turns to the right $[(v_{\ell,0}, v_{r,0}) = (0.989377, -0.34840)]$. If it detects another robot ($s(t) = 1$), it moves forward while slightly turning to the right $[(v_{\ell,1}, v_{r,1}) = (0.999426, 0.992379)]$. If it detects an option, it turns to the left $[(v_{\ell,2}, v_{r,2}) = (-0.106746, 0.965466)]$. Once the robot loses sight of an option, and detects nothing, it turns to the right, hence likely detecting the option again. The process of alternatively detecting an option and nothing results in the robot to approach the (left edge of the) option. If the option is occluded by other robots, however, these are detected instead, resulting in the robot

[7] To make outliers visible, a log scale had to be used for the colour bar.

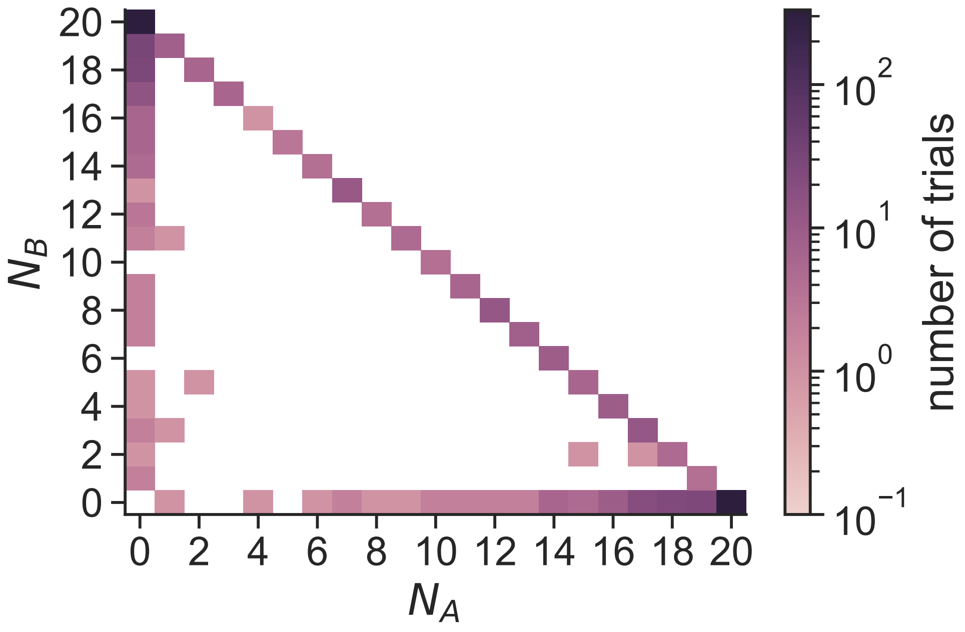


Fig. 4.4 Number of trials in which N_A and N_B robots committed, respectively, to options A and B. In total, 1000 simulation trials with $N = 20$ robots were conducted, each for a 300 s duration.

moving directly towards them (and the option). This facilitates reaching a consensus in the swarm. As more robots join, an orbiting behaviour is observed, where each robot follows the robot in front of it. If there is no such robot, a robot slides along the perimeter of the option in a clockwise fashion, while alternately detecting the option and nothing, and keeps doing so until detecting a robot.

Figure 4.5 shows a sequence of snapshots taken from a typical trial. A video recording of an example trial is available in the online supplementary material (Özdemir et al., 2018).

4.4.2 The Effects of the Robot Starting Positions

We investigate how the initial starting positions affect the performance of the swarm. We performed 1000 trials for each investigated scenario.

First, we initialised the robots randomly in a circular region four times larger than the one used before. The change in performance was not significant—it only dropped from 97.3% to 96.6%. We then initialised the robots randomly anywhere in the environment. The majority of the swarm committed in 83.4% of the trials. In

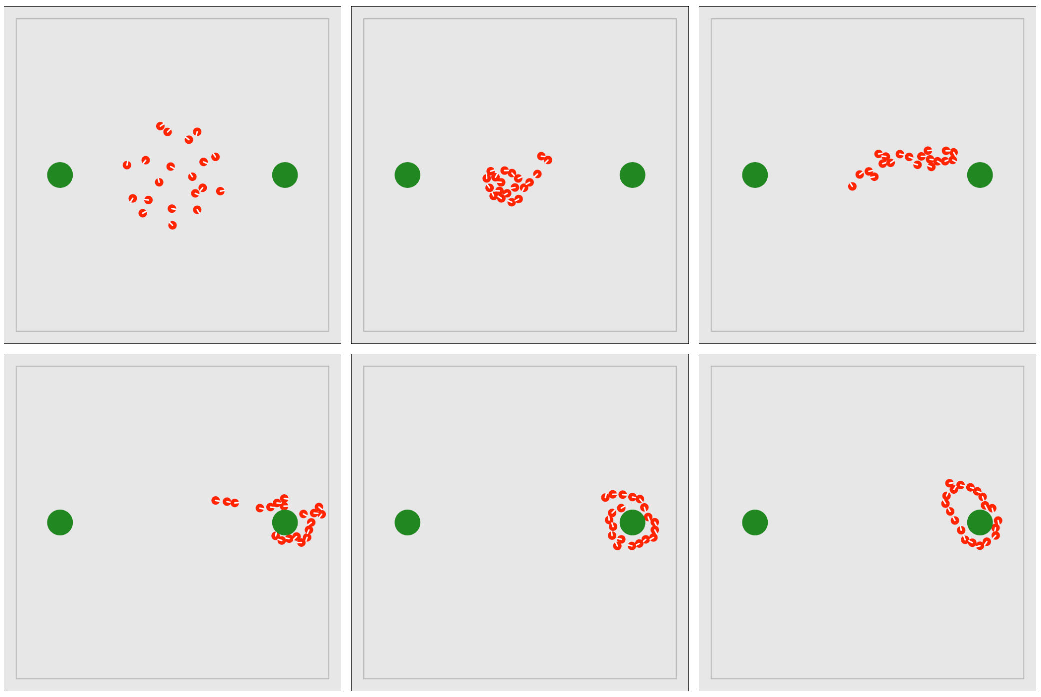


Fig. 4.5 Sequence of snapshots showing a swarm of 20 simulated robots choosing option B. They were taken (from the top left to the bottom right) once $0, 10, 20, 30, 40$, and 300 s had elapsed. Initially the swarm aggregates around the centre of the arena. The motion of the robots causes symmetry-breaking and as a result the swarm collectively approaches the option on the right. When $t = 40$ s the robots orbit around option B and remain committed to their choice.

addition, when we also changed the sensing range to unlimited (i.e. long enough to detect any point in the arena), the swarm committed in 94.6% of the trials. These results show that, as long as the robots have a long enough sensing range, their initial configuration has only a low impact on performance.

To explore the capabilities of the robots utilising a shorter sensing range in a sparse initial distribution, we reran the controller evolutionary process using the evolutionary setup described in Section 4.3 and select best controller for each new setup. The sensor range was limited to 200 cm (as before), 100 cm, and 50 cm, and in each case, the robots were initialised randomly anywhere in the environment. For the best controllers, 1000 simulation trials were performed.

The swarm committed in 98.6%, 90.3% and 48.4% of the trials to an option when the robots were equipped with a sensor range of 200 cm, 100 cm and 50 cm, respectively. These results indicate that our computation-free swarming framework tolerates some limitations in the sensor range, but is unable to cope with strictly short sensing range. This is in line with (Gauci et al., 2014c), which shows for the framework—albeit for a different task—that there exists no memory-less solution to maintain connectivity, unless the sensor range is sufficiently large.

Additionally, we explored the connectivity of the swarm. We considered that two robots are connected if a robot detects another robot using line-of-sight sensor. For simplicity, we assumed that the graph is undirected (directed graph was converted to undirected). We measured the number of clusters in the connectivity graph at the beginning of each of 1000 trials. On average, there were 11.16 clusters when the robots were initialised as in Figure 4.1. This means initially more than half of the robots were not connected in the terms of sensing. The number of clusters increased to 14.45 when the robots were initialised in 4 times larger initialisation region. Accordingly, when the robots were initialised anywhere in the environment, there were 16.86 clusters on average. These average number of connected robots indicated that the robots are not remarkably sensitive to the initialisation process.

4.4.3 Sensory Noise Analysis

We investigate how robust the controller is with respect to sensor noise. False negative noise was introduced in the robot's sensor as follows. If the robot had either another robot or an option in front of it, with probability p it would not detect it; in other

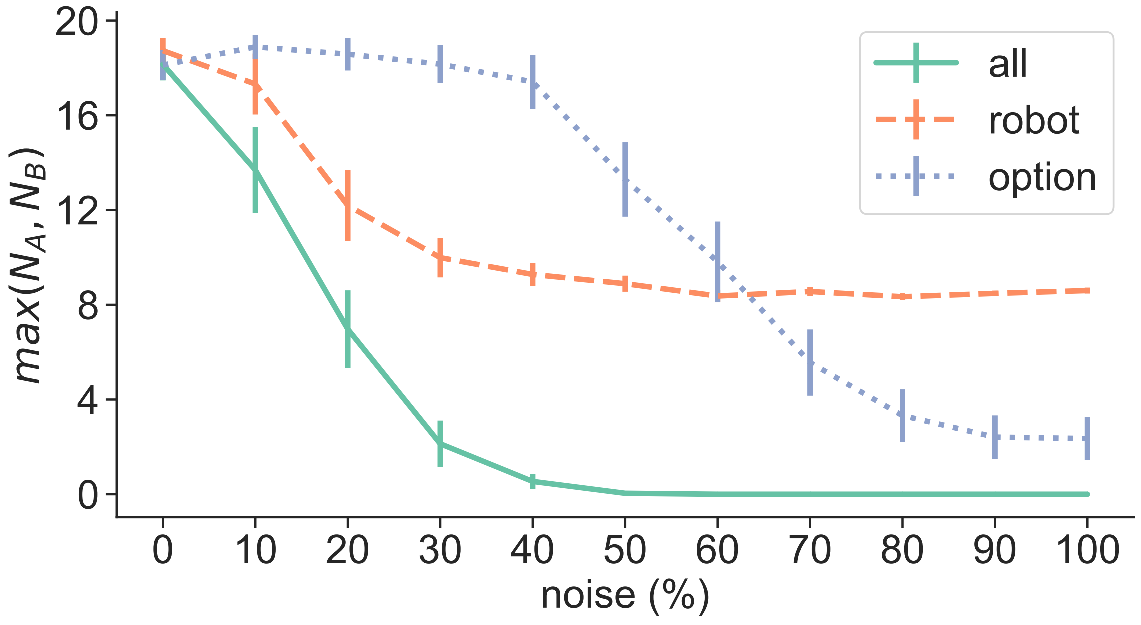


Fig. 4.6 Effects of sensor noise, p, on the swarm performance, $\max(N_A, N_B)$. For each setting, 100 simulation trials with 20 robots were conducted and averaged. The duration of trials was 300 s. The error bars represent the ± standard error.

words, it would obtain the incorrect sensor reading $s(t) = 0$ (nothing). The probability of misdetection p was varied from 0 to 1 in increments of 0.1.

Figure 4.6 shows the maximum number of robots having committed to the same option, that is, $\max(N_A, N_B)$. The lilac dotted line represents the performance of the swarm when the noise is restricted to the detection of options. The swarm copes well with this type of noise; its performance is affected only for noise levels of $\geq 50\%$. The orange dashed line represents the performance of the swarm when the noise is restricted to the detection of other robots. The performance of the swarm for noise levels of more than 50% remains at around 9 committed robots, that is, $\max(N_A, N_B) \approx N/2$. Once the robots can no longer detect each other, as expected, the swarm splits in two, about equally sized, sub-groups. The green solid line represents the performance of the swarm when the noise is affecting both the detection of other robots and the options. In this case, the performance drops more rapidly than in the other cases, suggesting that there is a compounding effect from the different types of noise. This limitation of the controller may make it unsuitable for applications in unstructured real-world environments. However, if a noise model is known for a particular environment, this could be incorporated into the controller evolutionary process, and this may yield a better controller.

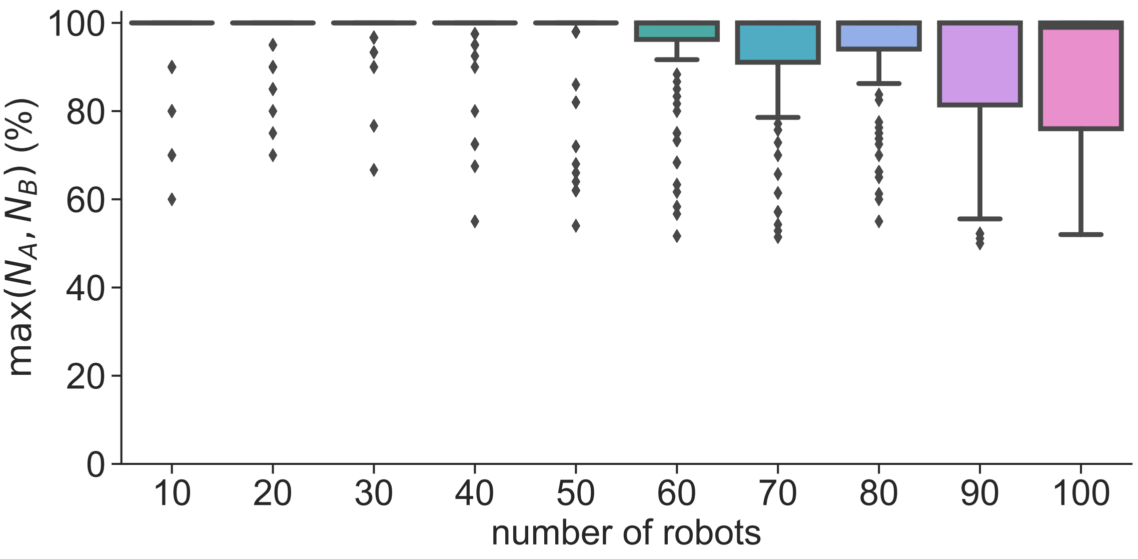


Fig. 4.7 Effects of the swarm size, N, on the swarm performance, $\max(N_A, N_B)$ (in percentage). For each setting, 100 simulation trials were conducted. The duration of trials was 300 s. The data is represented using box plots. Here a robot is considered committed to the option that is nearest.

4.4.4 Scalability Analysis

We investigate the scalability of the controller by measuring the performance for swarms of $N \in \{10, 20, \ldots, 100\}$ robots. Note that the more robots in the swarm, the harder it would be for all of them to fit inside the commitment region, as defined in Figure 4.1. To alleviate this problem, and thereby allowing a fair comparison between different swarm sizes, we removed the boundary of the environment and redefined the commitment regions to be the left and right half-planes, splitting the environment in its centre in half. In other words, each robot is committed at all times to its nearest option. We do not determine if the swarm (majority of robots) commit to the same option, but rather examine the percentage of robots committing to the options. At the beginning of each trial, about 50% of the robots are committed to either of the two options.

Figure 4.7 presents the results of 100 trials per setting. The performance scales reasonably well with the numbers of robots, despite the options being placed at a constant distance from each other. The average commitment to a same option is 97.9% for 20 robots and 88.5% for 100 robots. When near an option, the swarm orbits around it. The more robots, the bigger the radius of the orbit becomes. As a result the performance of large swarms drops with respect to the half-plane measure.

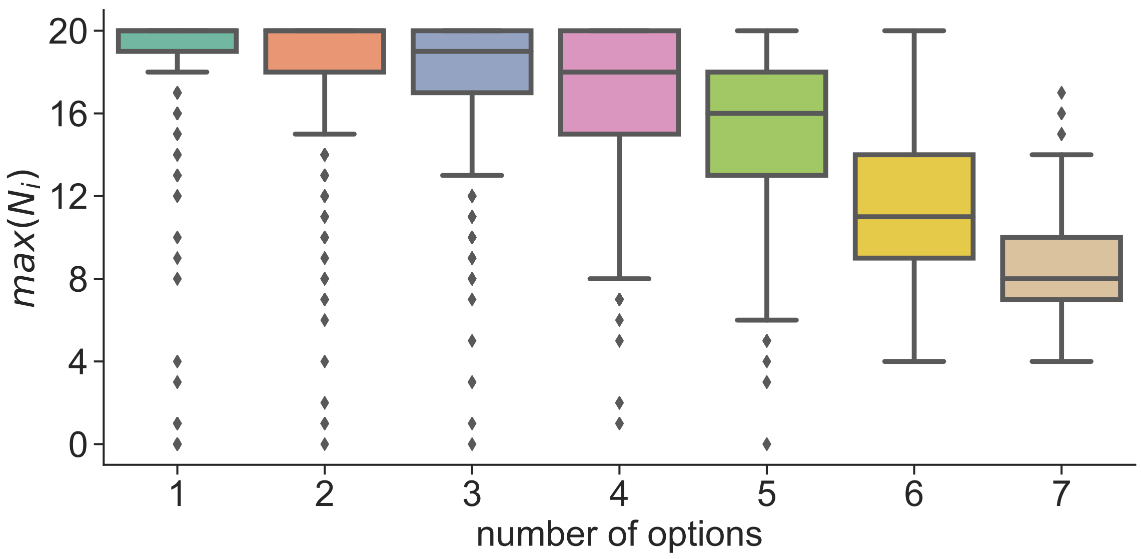


Fig. 4.8 Performance of a swarm choosing between n options, that is, the maximum number of robots committed to a same option. For each setting, 1000 simulation trials were conducted.

4.4.5 Choosing Between More Than Two Options

We explore the scenario with $n > 2$ options. Apart from the number and positions of options, the environment remains as shown in Figure 4.1. One option is placed as option B in Figure 4.1, whereas the remaining $n - 1$ options are equally spaced along the circle with the same centre as the environment. With this configuration, $n = 7$ is the maximum such that the commitment regions do not overlap. We therefore performed 1000 trials for each $n = 1, 2, \ldots, 7$.

The results are shown in Figure 4.8. The performance degrades gracefully as the number of options increases, even though the controller was optimised for $n = 2$ options. For $n = 7$ options, the swarm did not commit in the majority of the trials. We observed that the swarm could orbit around multiple options. As neighbouring options are in close proximity, robot were more likely to be attracted by them.

4.4.6 Choosing Between Unequal Alternatives

In this section, we investigate the ability of the swarm to choose between two *unequal* alternatives. This scenario corresponds to the asymmetric option qualities and symmetric option costs variant of the best-of-n problem (Valentini et al., 2017). Option A

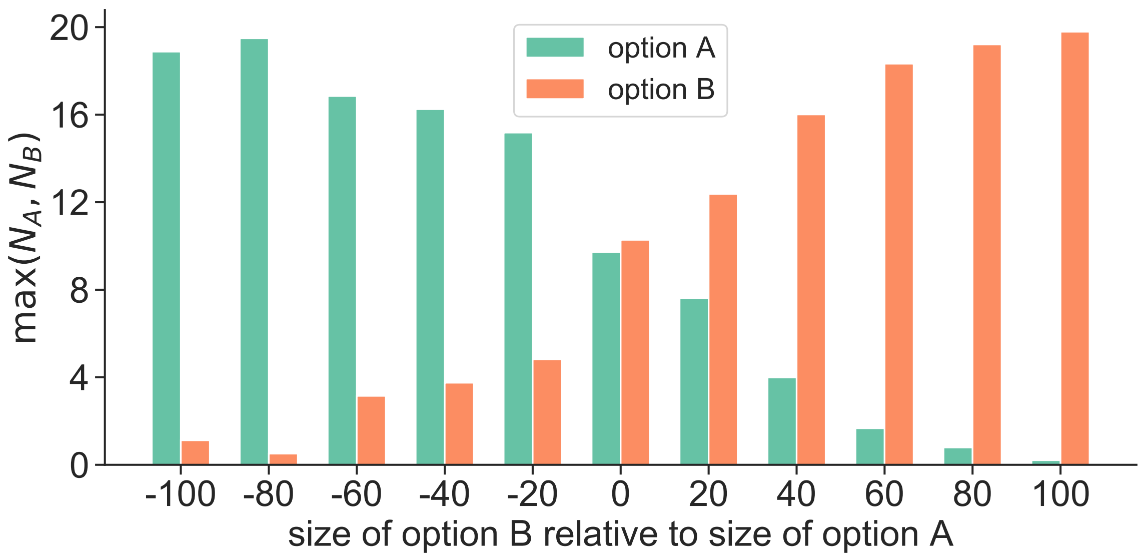


Fig. 4.9 Ability of the controller to let a swarm of robots choose between unequal alternatives. The bars shows the average percentage of robots committed to options A and B, respectively (100 simulation trials of 300 s duration). For details, see text.

was kept identical, as shown in Figure 4.1. However, option B was changed in radius from −100% (implying it is effectively removed) to +100%, by 20% increments. Our hypothesis was that the larger option, if any, will be preferred. As in Section 4.4.4, we removed the environment boundary and redefined the commitment regions using half-planes. This was done to prevent the situation that it is harder for the swarm to squeeze into a relatively small commitment region, as the physical dimension of the option increases.

Figure 4.9 shows the percentage of robots committed to options A and B at the end of 100 trials (per setting). As expected, in trials with equally sized options, the robots have no preference. As option B becomes smaller or larger, however, the robots increasingly succeeds in detecting such differences. When option B has twice the radius of option A, they almost exclusively opt for it.

The finding suggests that while the controller was designed and optimised for a particular problem—choosing among equal alternatives—it can also be used to choose the largest of unequal alternatives. The controller would be unable to consistently choose the smallest of equidistant, unequal alternatives. Moreover, it might favour smaller but closer options over larger but more distant ones.

4.5 Experiments

In this section, the best controller is evaluated using experiments with physical robots.

4.5.1 Porting of the Controller

To validate the controller on a physical platform, we use the e-puck robot (Mondada et al., 2009). The line-of-sight sensor was emulated using the on-board camera, which is a 640×480 active-pixel sensor. To determine the sensor value, a centred $a \times b$ pixel region is used. We chose $a = 2$ columns to ensure that the emulated sensor points exactly towards the front, and $b = 15$ rows to improve the sensing range—misaligned cameras (pitch axis) would otherwise cause false negatives. The sensor detects the colour of the object it is pointed at. The sensor reading $s(t) = 0$ if no object (i.e. effectively the white boundary) is detected, $s(t) = 1$ if a red object (robot) is detected, and $s(t) = 2$ if a green object (options A or B) is detected. The aforementioned detection procedure uses arithmetic computation. The controller, however, remains computation-free.

As described in Section 3.5.2, one can use the same procedure to deploy the control strategy presented in this chapter in another ground robotic system. It is sufficient to convert the control parameters in Table 4.1 to linear and angular velocities, then converting them back to the appropriate motor commands.

4.5.2 Experimental Setup

The robots operate in a $300\,\text{cm} \times 300\,\text{cm}$ environment, which is bounded by a white wall of height $50\,\text{cm}$. The options are represented as green cylinders with a diameter of $24\,\text{cm}$ and a height of $10\,\text{cm}$. They are placed as indicated in Figure 4.1.

We distributed the robots in a hexagonal grid pattern as shown in Figure 4.13(a). Random permutations were used to determine the order of placing the robots on the 20 grid locations. The orientation of each robot was uniformly chosen from $[0, \pi)$.

The trial was started by broadcasting an infrared signal to all robots using a remote control. No human intervention took place; where robots ceased motion during a trial, they were left in the environment. The trial duration was $300\,\text{s}$.

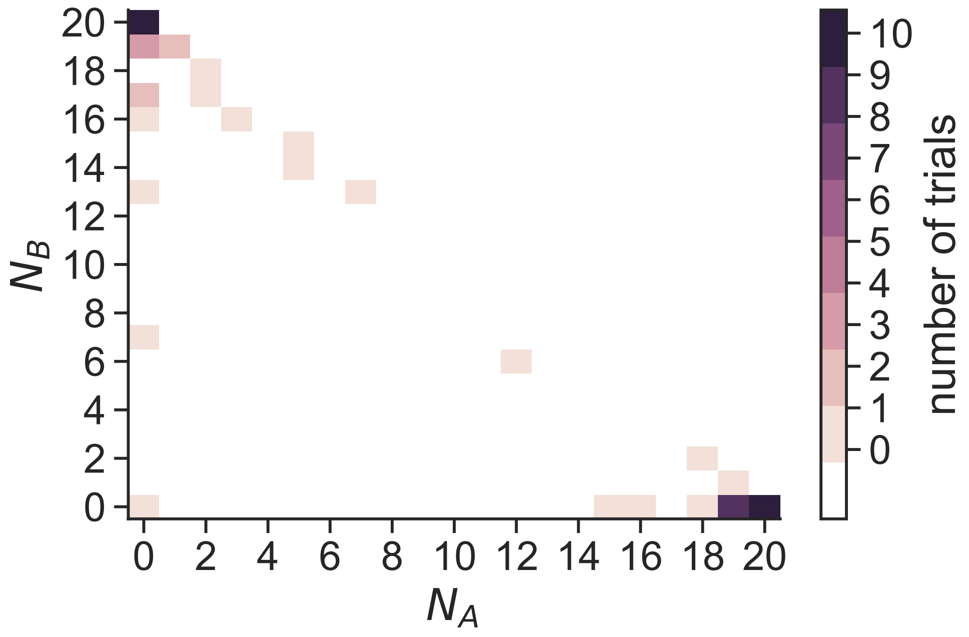


Fig. 4.10 Number of experimental trials in which N_A and N_B robots committed to options A and B, respectively. In total, 50 trials with 20 physical e-pucks were conducted, each for a 300 s duration.

All trials were recorded by an overhead camera at a rate of 25 fps. The recordings were analysed using the OpenCV (Bradski, 2000) computer vision library. Distortion effects in the images were removed and the positions of robots tracked automatically.

4.5.3 Results

A set of 50 experimental trials were conducted using $N = 20$ e-puck robots. Video recordings of all trials are available in the supplementary materials.

Figure 4.10 shows the number of robots committed to either option A or B (N_A and N_B, respectively). In 96% of the trials, the swarm committed to one of the options, A or B; in other words, the majority of the robots ended up choosing that option. In 25 trials, the swarm committed to A, whereas in 23 trials, it committed to B.

Over the course of the experiments, the robots were set to operate for 300 s, a total of 1000 times (50 trials with 20 robots). In 2.7% of these cases, the robot ceased motion at some point during the trial. This may happen for a variety of reasons, including a lost contact with the battery or a low battery state. Figure 4.11 shows the number of trials for each combination of $\max(N_A, N_B)$. The colour of each trial indicates

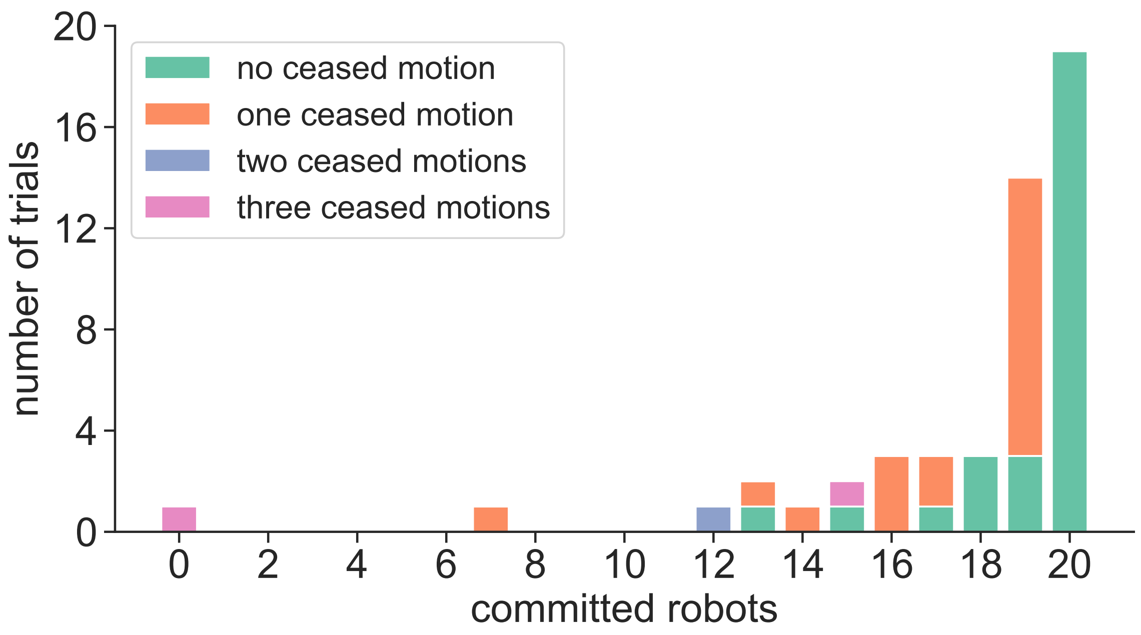


Fig. 4.11 Breakdown of the 50 experimental trials according to the maximum number of robots that committed to the same option, $\max(N_A, N_B)$. Each trial is shown with a colour indicating how many of the 20 physical robots ceased motion.

how many robots ceased motion; the latter was manually determined, through visual inspection of the overhead video recordings. The more robots with ceased motion, the more the performance was affected.

Figure 4.12 shows the maximum number of robots committed to the same option over time, $\max(N_A, N_B)$. The green line indicates the mean and the green envelope the $\pm$ standard error across the 50 trials.

Figure 4.13 shows the behaviour of the robots during a typical trial. In this trial, it takes approximately 35 s for the first robot to approach the option. The rest of the robots tend to follow, and the whole swarm is committed to the option after 150 s. The swarm then remains in the commitment area until the end of the trial.

4.6 Discussions

In this study, we showed that a group of embodied agents can collectively choose, without arithmetic computation, between multiple alternatives in an environment. The agents we considered used a single line-of-sight sensor, obtaining a ternary digit of information about the environment. The agents could not communicate, nor store

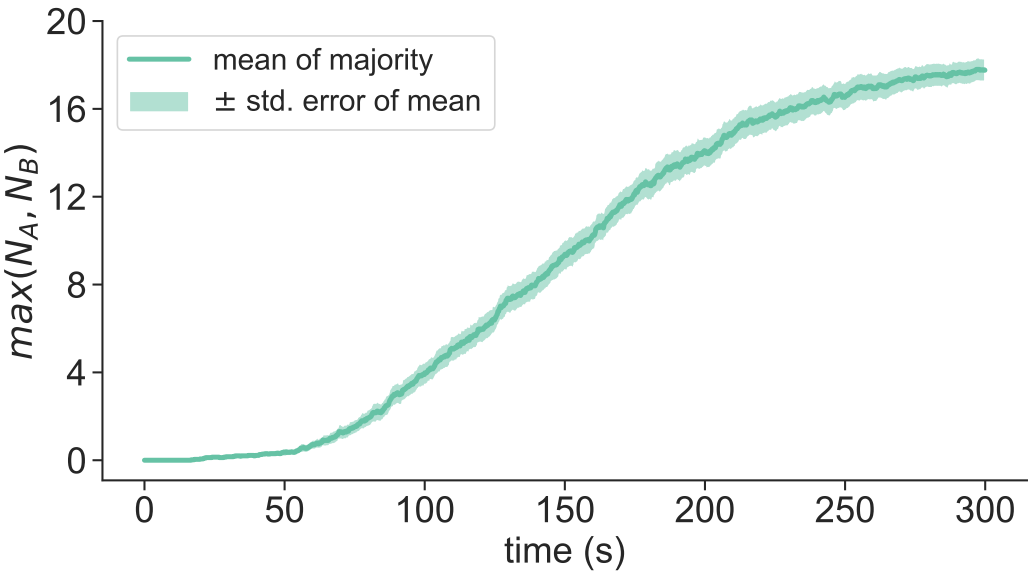


Fig. 4.12 Dynamics of $\max(N_A, N_B)$, averaged over the 50 experimental trials with 20 physical robots.

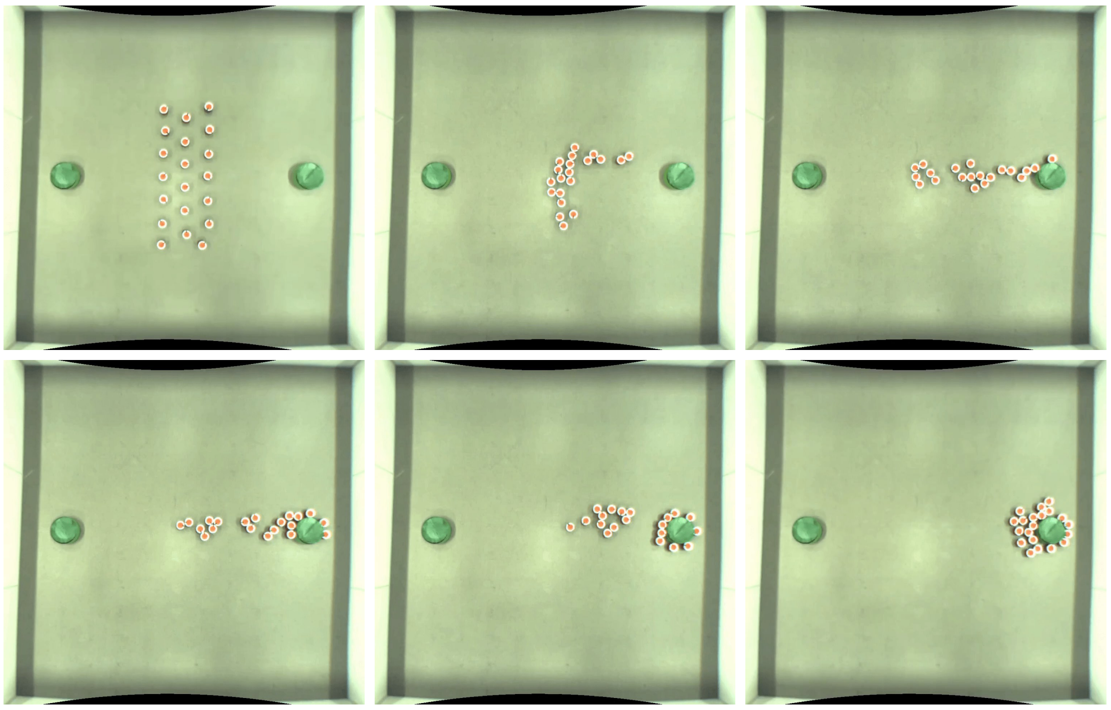


Fig. 4.13 A sequence of snapshots from a typical experimental trial with 20 physical robots. They were taken (from the top left to the bottom right) once $0, 20, 40, 80, 120,$ and 300 s had elapsed. Due to distortion removal, blank pixels occur at the top and bottom of the images.

any information during run time. They directly mapped the sensor reading onto constant-value motor commands. Compared to previous solutions to the collective choice problem, the proposed control strategy requires significantly lower information processing capabilities—at the expense of a longer sensing range—and could be implemented on platforms that lack an arithmetic logic unit.

Using computer simulations, we demonstrated that the control strategy was fairly robust with respect to sensory noise as well as changes in the number of robots or options. We also showed that the strategy works well for a range of different initial configurations, provided that the sensor's range is sufficiently long. We examined the problems of choosing between equal alternatives and between unequal alternatives. In the latter case, an option's quality was reflected by its size (the bigger, the better). To choose between options of the same size but unequal qualities, the robots would need to be equipped with sensors to detect such differences. Assuming that only a limited number of quality levels are possible, our framework could be adapted accordingly.

We ported the control strategy onto the e-puck platform, and performed 50 experimental trials with 20 physical robots. The swarm succeeded in choosing an option in 96% of the trials, despite some robots ceasing motion during the trials. Note that, the controller used in the proof-of-concept implementation was free of arithmetic computations, even though the sensor was not.

The extreme simplicity of our control strategy makes it potentially applicable to robotic systems operating at the submillimeter-scale. For example, nanorobots could be configured to collectively target one of multiple regions of interest at a time. In the next chapter, we will focus on collective behaviours of a swarm of robots that is moving towards a goal while coordinating another dynamic swarm.

Chapter 5

Shepherding

5.1 Introduction

In the previous two chapters, we proposed control strategies for the multi-agent coverage and collective choice problems. In the former, the agents retrieved 1-bit (i.e. a binary digit) of information from their environment, whether another agent is present in their direct line of sight. In the latter, they additionally needed to detect the presence of an option (i.e. goal) yielding them to retrieve 1-trit (i.e. a ternary digit) of information The agents were expected to find consensus by committing to one of the multiple goals in their environment. This study also allowed us to gain understanding of a collective behaviour, that is, coordinated movement.

One of the critical features of a swarm is its coordinated movement abilities. If a swarm of agents exhibit advanced coordination skills, this can be transferrable onto another task. For instance, coordination can lead the swarm to also be able to manipulate an object. Ants are a great example from nature; they can carry large prey to their nest through self-organised coordinated behaviours. Inspired by nature, we hypothesise that a swarm of computation-free agents can coordinate their movements to cooperate with each other in order to manipulate another group of objects. Thus, in this chapter, we further investigate the coordinated movement strategies for the control agents to coordinate another group of agents in a *shepherding* scenario.

The shepherding problem involves the *shepherd* agents to guide the motion of dynamic *sheep* agents towards a pre-specified goal location. In addition to the previous study, which required the agents to distinguish between two objects (i.e. an agent or a goal), the agents (shepherds) can now distinguish between three objects: another shepherd, the goal, or a sheep agent. Compared to the literature, to the best of our knowledge, we propose the simplest control solution to the multi-agent shepherding problem, requiring the agents to extract only 2-bits of information.

This chapter is organised as follows. Section 5.2 defines the problem, details the computation-free shepherd agent capabilities, the sheep flock model, and the simulation setup. Section 5.3 presents the evolutionary process for synthesising the shepherd controllers, detailing the objective function, and the controller selection procedure. Section 5.4 evaluates the obtained controller through multiple computer simulation experiments. Section 5.5 concludes the chapter.

5.2 Problem Definition

5.2.1 Objective

Consider an unbounded, continuous-space environment in 2D containing n *shepherd* agents, m *sheep* agents, and an object representing the *goal location*. The shepherd (or sheep) agents are anonymous (i.e. indistinguishable) and execute an identical controller. The shepherds are employing a computation-free controller—their controller lack the ability of performing arithmetic computations. Moreover, the shepherds do not have run-time memory and unable to communicate with each other.

The goal is a cylindrical static object, located in $g \in \mathbb{R}^2$. In the beginning, both sheep and shepherd agents are uniformly randomly placed in an initialisation region away from the goal location. Figure 5.1 illustrates the experimental setup. The shepherding problem is to control the shepherds such that they gather the sheep and herd them towards the goal location.

5.2.2 Shepherd Agents

A shepherd agent has a line-of-sight sensor pointing forwards. The range of the sensor is unlimited.[1] In a practical scenario, the environment could be bounded. The sensor would then need to span the entire environment. The sensor reading, $s(t)$, is defined

[1]This assumption is only to ensure any agent (or the goal) in the environment is detectable by a shepherd. Practically, it needs to be at least the length of the arena.

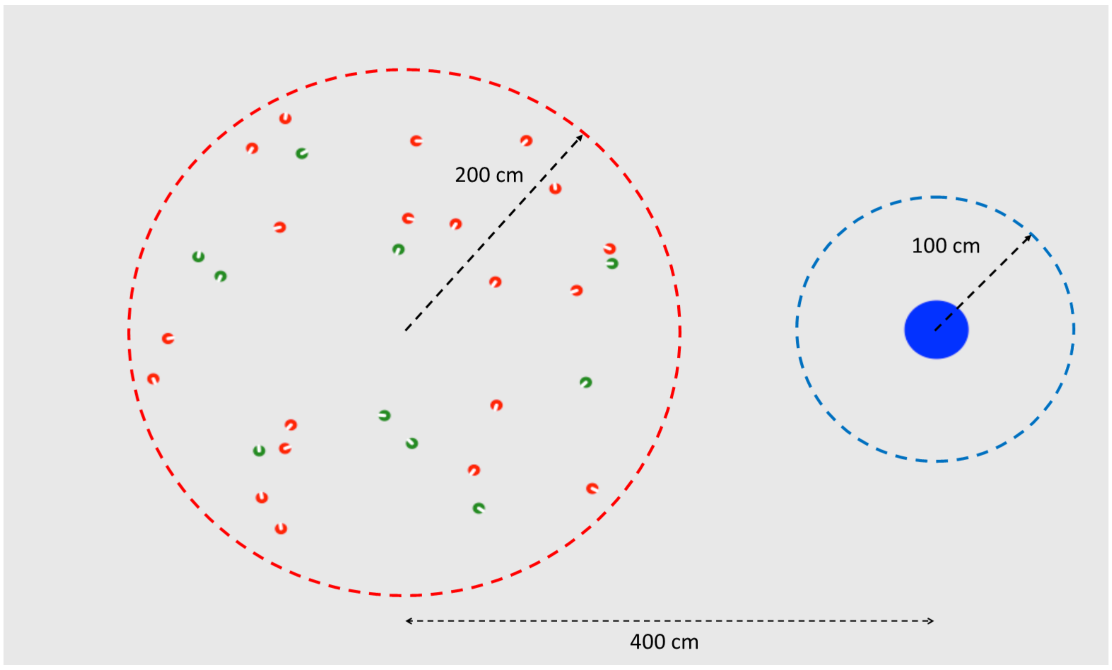


Fig. 5.1 Illustration of the experimental setup. The goal object is represented by the blue disk, the shepherds and the sheep are indicated by the green and the red disks, respectively. Initially, the robots are randomly distributed into a circular region of radius 200 cm and 400 cm away from the goal object. The goal region is the circular area indicated by the blue dashed line with radius 100 cm from the centre of the goal object.

by the first object that is detected in the line of sight, if any:

$$s(t) = \begin{cases} 0 & \text{if no object is detected,} \\ 1 & \text{if a sheep is detected,} \\ 2 & \text{if a shepherd is detected,} \\ 3 & \text{if the goal is detected.} \end{cases} \tag{5.1}$$

Note that sensor reading $s(t)$ does not contain any information about the distance to a perceived object, or about the number of objects in a given area.

All shepherds execute an identical controller. The controller is reactive: it maps the shepherd's input $s(t)$ onto its output—the pair of wheel velocities, $(v_\ell,\, v_r)$. Formally, the controller c is defined by

$$c\left(s(t)\right) = \left(v_\ell(t), v_r(t)\right) = \begin{cases} (v_{\ell,0}, v_{r,0}) & \text{if } s(t) = 0, \\ (v_{\ell,1}, v_{r,1}) & \text{if } s(t) = 1, \\ (v_{\ell,2}, v_{r,2}) & \text{if } s(t) = 2, \\ (v_{\ell,3}, v_{r,3}) & \text{if } s(t) = 3, \end{cases} \tag{5.2}$$

where $v_{\ell,k}$ and $v_{r,k}$ are the left and right wheel velocities for $k \in \{0, 1, 2, 3\}$. Note that the controller does not need to store information during run-time, and does not need to perform arithmetic computations. The velocity pairs in Equation (5.2) are the control parameters that need to be found for the swarm (i.e. shepherds) in order to produce the desired global behaviour. The synthesis of the controller c will be presented in Section 5.3.

5.2.3 Sheep Agents

The behaviour of the sheep agents is not subjected to an evolutionary process, unlike the shepherds. The sheep agents execute a fixed, manually-designed behaviour in which they react both to each other and to shepherd agents. This behaviour is based on the magnitude-dependent motion control model proposed by Ferrante et al. (2012). The sheep assumed to have omnidirectional vision[2], and can distinguish between shepherds and other sheep; however, they are unaware of the goal. Each sheep is repelled by

[2]It is typical for natural "prey" to have wide fields of view (Piggins and Phillips, 1996).

all shepherds and—to a lesser extent—by other sheep[3]. Let S be the set of (suitably relabelled) indices of all agents. Formally, the repulsion force is given by

$$\mathbf{F}_i = \sum_{j \in S \setminus \{i\}} \frac{c_j}{\|x_i - x_j\|^2} \hat{\mathbf{r}}_{ji}, \tag{5.3}$$

where x_i is the position of sheep i, x_j is the position of agent j (either a shepherd or a sheep, but excluding the focal sheep), $\hat{\mathbf{r}}_{ji}$ is the unit vector pointing from agent j towards sheep i, and c_j is 450 if agent j is a shepherd, and 100 otherwise. In other words, the sheep repel more strongly from the shepherds than from other sheep.

The motion of each sheep is the result of (i) the repulsion force and (ii) a natural tendency to move forward. The wheel velocities of the sheep model can be given as

$$\begin{pmatrix} v_\ell \\ v_r \end{pmatrix} = \begin{pmatrix} K_1 & K_2 \\ K_1 & -K_2 \end{pmatrix} \begin{pmatrix} f_x \\ f_y \end{pmatrix} + \begin{pmatrix} u \\ u \end{pmatrix}, \tag{5.4}$$

where f_x and f_y are the horizontal and vertical components of the repulsion force in the sheep's local coordinate frame, $K_1 = 2.0$ and $K_2 = 1.3$ are the linear and angular gain, and $u = 2.0\,\text{cm/s}$ is the constant forward speed. The maximum wheel speed for a sheep is 6.4 cm/s—half of a shepherd's maximum wheel speed. If the wheel velocities exceed their range, they get truncated.

5.2.4 Simulation Setup

We conduct simulation trials using Enki (Magnenat et al., 2009), an open-source 2D rigid bodies physics engine. The shepherds and sheep are modelled as e-puck robots (Mondada et al., 2009). The robots are represented as cylinders of radius 3.7 cm. In order to allow the line-of-sight sensor to make distinctions, the shepherds are coloured green and the sheep are coloured red. The goal is represented as a blue cylinder and has a radius of 22.2 cm, which is six times larger than the radius of the e-puck.

The robots have two wheels arranged in a differential drive configuration (axle length = 5.2 cm). The robot needs to set the desired velocities of its left and right wheels, $v_\ell \in [-1, 1]$ and $v_r \in [-1, 1]$, where -1 and 1 represent the normalised maximum

[3]Although it may be unrealistic to assume that a sheep could perceive all other agents in the environment, the magnitude of the repulsive force decreases as the square of distance, and hence its effect can be neglected when the other agent is far away.

angular velocity at which the wheel can turn backward and forward, respectively. The default uniform noise of 5 % is applied to each value. The corresponding maximum velocity of the robot ranges in $\pm 12.8\,\mathrm{cm/s}$. The control cycle of the robot is activated every 0.1 s and the simulation physics is updated every 0.01 s.

5.3 Controller Synthesis

We have devised the structure of a reactive controller, c, for the shepherds [Equation (5.2)]. The remaining problem is to optimise the eight control parameters that leads to the desired global behaviour, that is, the shepherding. We employ an evolutionary robotics approach to solve this problem (Nolfi and Floreano, 2000; Trianni et al., 2008), whereby an evolutionary algorithm searches for the 'best' controller parameters that minimise an objective function.

5.3.1 Evaluation of Candidate Solutions

The evolutionary algorithm requires that each candidate solution—controller—is assigned a value reflecting its quality (hereafter referred to as *fitness*) that reflects how well it addresses the problem. This is achieved by running a number of simulations using the controller and computing an objective function (hereafter referred to as *fitness function*) based on the performance of the agents during these simulations. Figure 5.1 shows the simulation setup. Initially, $n = 10$ shepherds and $m = 20$ sheep are uniformly randomly distributed within a circular region of radius 200 cm, whose centre is 400 cm away from the goal.

The fitness function for a single simulation—to be minimised—is

$$F = \sum_{t=1}^{T} t f(t), \tag{5.5}$$

where T is the duration of a simulation trial and

$$f(t) = \frac{1}{P} \sum_{i=1}^{m} \|\overline{x}(t) - x_i(t)\|^2 \|\overline{x}(t) - g\|^2, \tag{5.6}$$

where $P = (2r)^2 m$ is the scaling factor, r is the radius of the agents, $x_i(t)$ is the position of sheep i at time t, $\overline{x}(t)$ is the centroid of the sheep flock at time t, and g is the position of the goal.

Equation (5.6) takes into account how widely the sheep are scattered and how far away they are from the goal. The weighted summation over time in Equation (5.5) rewards solutions for accomplishing the task faster, while still giving prominence to a stable configuration later on in the simulation. The overall fitness of a controller is given by averaging F over a number N of simulations with different initial conditions.

5.3.2 Evolutionary Algorithm

We use the Covariance Matrix Adaptation-Evolution Strategy (CMA-ES) (Hansen and Ostermeier, 2001) as the evolutionary algorithm. CMA-ES is a stochastic black-box optimisation algorithm and operates on real-valued decision variables. It self-adapts the variance of each decision variable, as well as all the covariances between the decision variables.

In our problem, the decision variables are the set of all possible left and right wheel speeds corresponding to the sensor readings of the shepherd robots [Equation 5.2]. Considering normalised wheel speeds, this corresponds to the space $[-1, 1]^{2d}$, where $d = 4$ is the number of possible sensor states. We use a sigmoid-based function to map the real-valued variables in $\mathbb{R}$ to $[-1, 1]$:

$$\text{sig}(x) = \frac{1 - e^{-x}}{1 + e^{-x}}, \quad \forall x \in \mathbb{R}. \tag{5.7}$$

For this study, we set the population size to $\lambda = 20$, where $\mu = 10$ are selected for reproduction.[4] The other two additional settings for the optimiser, we set the initial guess parameter to $\mathbf{m}^{(0)} = \mathbf{0}$ and the initial step size to $\sigma^{(0)} = 0.72$.[5]

An evolution was run for 80 generations, and in each generation, each of the $\lambda = 20$ candidate solutions was evaluated using $N = 50$ trials with different initial configurations of agents. Each trial returned a fitness value according to Equation (5.5), and the average of these 50 values was used as the overall fitness of that candidate solution. Each trial lasted 1500 s [i.e. $T = 15000$ in Equation (5.5)].

[4] We use the same lower bound to determine the population size, λ, as discussed in Section 3.3.2. However, rather than the bound itself ($\lambda = 16$), we chose $\lambda = 20$ as the final value.

[5] These two parameters are the same as in Chapter 3.

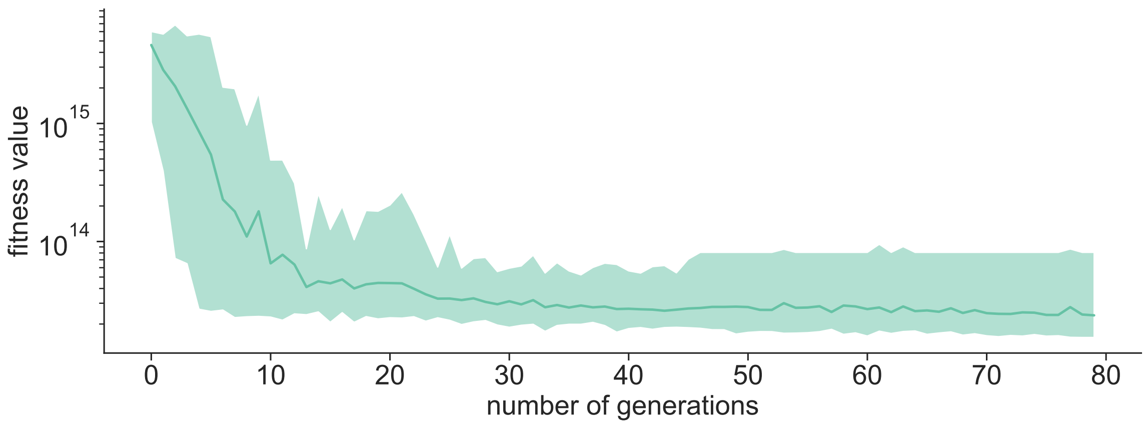


Fig. 5.2 Evolutionary dynamics of 30 runs for 80 generations. The average fitness of each generation is represented by the solid line. The envelope indicates the minimum and the maximum fitness values in each generation.

5.3.3 Controller Selection

No changes were made to the CMAS-ES implementation. However, changes made to the source code were the environment model, random initialisation, fitness function calculation, and sheep model (sensing and actuation).

A set of 30 evolutions were performed. Figure 5.2 shows the evolutionary dynamics for 80 generations. During the first approximately 20 generations, the fitness values of the best individuals improve rapidly. The fitness values continue to improve thereafter, but seem to approach a stable plateau.

The best (i.e. lowest) fitness amongst $\lambda = 20$ candidate solutions in the last generation was considered as the best controller for that evolution. As 30 evolutions were performed, there were 30 best controller candidates in total. In a post-evaluation session, each of these 30 controllers was re-evaluated 100 times in simulations trials with different initial distribution of agents. The controller with the best average fitness score was selected, and is considered as the *best controller* across the set of evolutions. The best controller parameters are provided in Table 5.1.

5.3.4 Mathematical Analysis

In this section, we present theoretical analysis for the behaviours of the shepherds.

Lemma 2. *Using control parameters given in Table 5.1, n shepherds can circle $m = 1$ sheep.*

Table 5.1 The *best* controller velocity pairs.

nothing	sheep	shepherd	goal
$v_{\ell,0}$	$v_{\ell,1}$	$v_{\ell,2}$	$v_{\ell,3}$
0.9998	0.0082	0.5471	0.9993
$v_{r,0}$	$v_{r,1}$	$v_{r,2}$	$v_{r,3}$
0.8520	0.9996	0.6098	0.9447

Proof. Using the kinematics equations given in Equation (3.1), we can calculate the linear and angular speeds of a shepherd when it detects a sheep, ($s = 1$), as follows

$$v_1 = 6.449\ cm/s, \quad \text{and} \quad \omega_1 = 2.488\ rad/s, \tag{5.8}$$

and, when the shepherds detect another nothing ($s = 0$), or shepherd ($s = 2$), the linear and angular speeds are

$$\begin{aligned} v_0 &= 11.852\ cm/s, \quad \text{and} \quad \omega_0 = -0.371\ rad/s, \\ v_2 &= 7.404\ cm/s, \quad \text{and} \quad \omega_2 = 0.157\ rad/s. \end{aligned} \tag{5.9}$$

This means that, the shepherds move ‘almost’ in a straight line with a small arc when they detect nothing, or a shepherd, and make a CCW curvilinear motion when they detect a sheep, which then causes n shepherds to ‘circle’ the sheep. ■

Lemma 3. *Once circled by n shepherds, $m = 1$ sheep remain inside the circle.*

Proof. From Lemma 2, we showed that n shepherds encircle $m = 1$ sheep. The shepherds then form a uniformly symmetric ‘circular’ arrangement around the sheep. We assume that the sheep is in the centre of the virtual circle formed by shepherds, hence, the position of the sheep, $x_0 = (0, 0)$. The positions of n shepherds can be given as

$$x_i = r\cos(\theta_i)\hat{\imath} + r\sin(\theta_i)\hat{\jmath}, \tag{5.10}$$

where x_i is the position of the shepherd i, r is the distance between the shepherd to the sheep, and θ_i is the angular arrangement of the shepherd in the virtual circle.

The unit distance vector between the sheep and a shepherd, $\hat{\mathbf{r}}_i$, can be written as

$$\hat{\mathbf{r}}_i = x_i - x_0 = r\cos(\theta_i)\hat{\imath} + r\sin(\theta_i)\hat{\jmath}. \tag{5.11}$$

Using the force equation for sheep given in Equation (5.3), the net force applied to the sheep is

$$\begin{aligned}\mathbf{F} &= \sum_{i=1}^{n} \frac{c_j}{\|x_i - x_0\|^2}\hat{\mathbf{r}}_i \\ &= \sum_{i=1}^{n} \frac{c_j}{r^2}\left(r\cos(\theta_i)\hat{\imath} + r\sin(\theta_i)\hat{\jmath}\right) \\ &= \frac{c_j}{r}\left(\sum_{i=1}^{n}\cos(\theta_i)\hat{\imath} + \sum_{i=1}^{n}\sin(\theta_i)\hat{\jmath}\right).\end{aligned} \tag{5.12}$$

Due to the symmetric arrangement of the shepherds, the final summations given in Equation (5.12) is 0, implying that the net virtual force applied to the sheep is $\mathbf{F} = 0$. ■

Theorem 2. *Once circled by n shepherds, $m = 1$ sheep can be driven to the goal.*

Proof. From Lemma 3, we proved that once $m = 1$ sheep is circled it remains inside the circle. Using control parameters $(v_{\ell,3}, v_{r,3})$, the shepherd closer to the goal detects the goal ($s = 3$) and utilise the following linear and angular velocities

$$v_3 = 12.441\ cm/s, \quad \text{and} \quad \omega_3 = -0.136\ rad/s. \tag{5.13}$$

The high linear speed (almost at maximum) then results in the shepherds moving towards the goal. As a result, the virtual circle clustering the sheep stretches towards the goal. The angular speed of the shepherds allow it to turn in the CW direction (the same direction as detecting nothing), resulting it to be displaced in the circle. The other shepherds follows the same routine, resulting the swarm approaching towards the goal over time. As the circle is stretched towards the goal, the force vector applied to the sheep is stronger at back than front, resulting a gradient for the sheep to move towards the goal. ■

5.3.5 Behavioural Analysis

Figure 5.3 shows a sequence of snapshots taken from a simulation trial with the shepherds using the best controller. At the beginning, the shepherds spread out from the initial formation towards the periphery of sheep. They then cage the sheep by orbiting around them in a clockwise manner. As the sheep are repelled more by the shepherds than by each other, they assume a compact and round formation. While

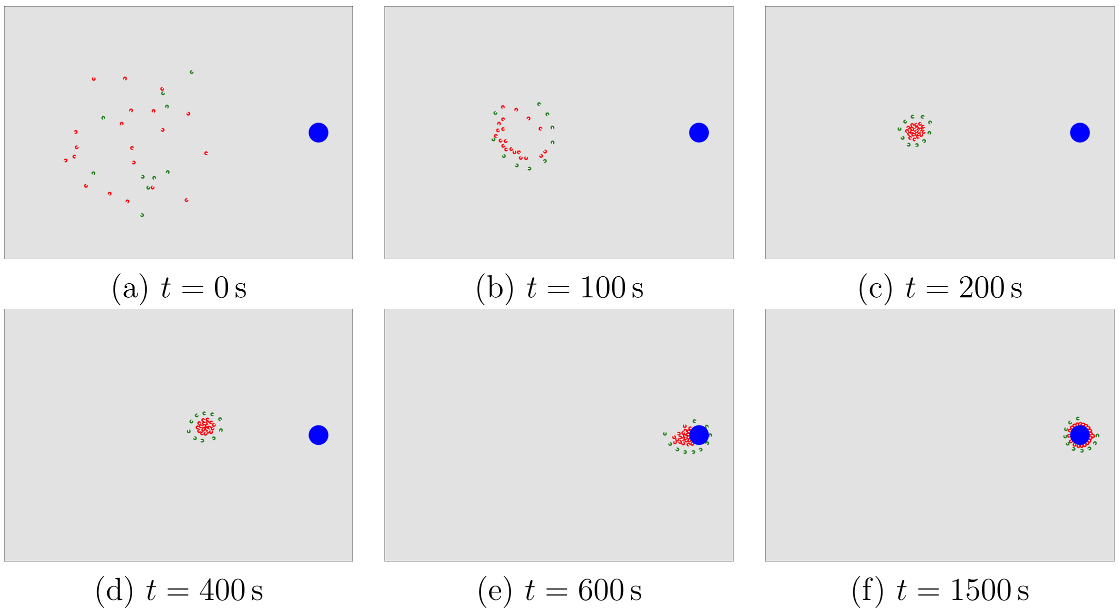

(a) $t = 0\,\mathrm{s}$
(b) $t = 100\,\mathrm{s}$
(c) $t = 200\,\mathrm{s}$
(d) $t = 400\,\mathrm{s}$
(e) $t = 600\,\mathrm{s}$
(f) $t = 1500\,\mathrm{s}$

Fig. 5.3 Sequence of snapshots showing a group of 10 shepherds (green) gathering and moving a group of 20 sheep (red) towards the goal (blue).

orbiting around the sheep, the shepherds are also attracted towards the goal. This then results in the gradual movement of the agents, that is, shepherds and sheep, towards the goal.

To gain a deeper understanding of the shepherding behaviour, we monitored the sensor reading values of $n = 10$ shepherds herding $m = 20$ sheep over 100 additional simulation trials. Note that the sensor reading values directly determine a shepherd's action, that is, wheel velocities, according to Equation (5.2). Figure 5.4 shows the average number of shepherds detecting objects of different types. Most of the time, the shepherds detect nothing. This is followed by detecting the presence of other shepherds. The detection of sheep seems to be relevant only in the initial stages—prior to the caging being completed. The goal becomes more frequently observed as the agents approach it, until the agents are caging it as well.

Note that, unlike a conventional sheep flocking model, for example Reynolds (1987) model, the sheep do not tend to stay as a flock. Thus, the clustering of sheep is solely based on the shepherd's action.

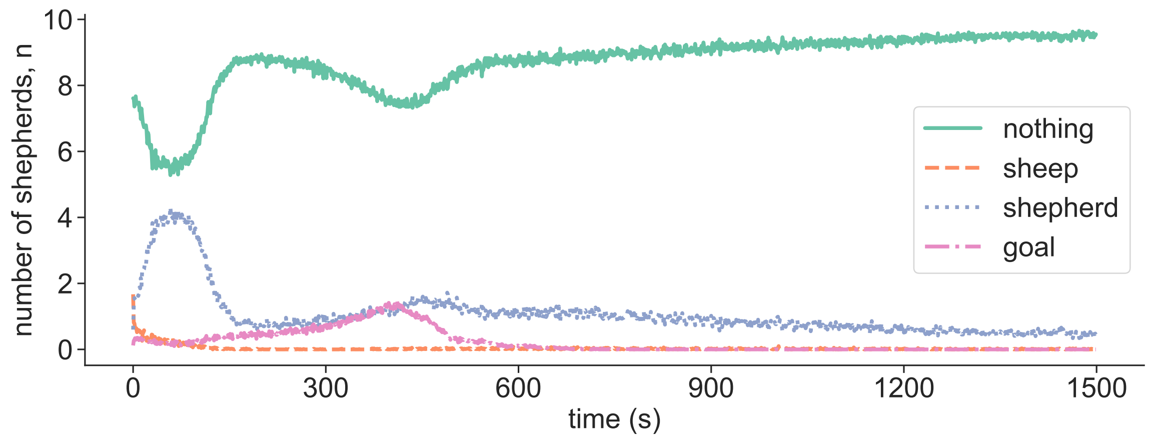


Fig. 5.4 Behavioural analysis of the shepherds. The line plots show the average number of shepherds that detect objects of particular types. The line plots are the average of 100 trial over time. The shepherds can either detect nothing ($s = 0$), a sheep ($s = 1$), another shepherd ($s = 2$), or the goal ($s = 3$).

5.4 Simulation Studies

In the following sections, we explore the capabilities of the controller through sensor noise, speed-parameter sensitivity, and scalability analyses. In these analyses, a *success rate* is used to measure the performance of the shepherds. The success rate is defined as the percentage of sheep that reside within the goal region (see Figure 5.1) after 1500 s.

5.4.1 Noise Analysis

We examine the effect of noise on the performance of shepherds. In particular, we consider false negative noise; in other words, noise preventing the detection of objects, that is, sheep, shepherd, or goal. Formally, given an unperturbed sensor reading of $s \in \{1, 2, 3\}$, the actual reading value returned by the sensor is 0 with probability p, and s otherwise. If no object is in the line of sight of the shepherd, the sensor value remains $s = 0$. We performed 100 simulation trials for each of $p \in \{0, 0.1, 0.2, \ldots, 1\}$. In addition, we performed an equivalent number of trials for the situation where only a single type of sensor reading (e.g. $s = 1$) was affected by the noise.

Figure 5.5 show the success rates for the different probability levels of noise. The success rate decreases rapidly if the sensor is subjected to noise levels of more than $p = 0.3$ on *all* readings (green curve). However, the impact of noise varies if only

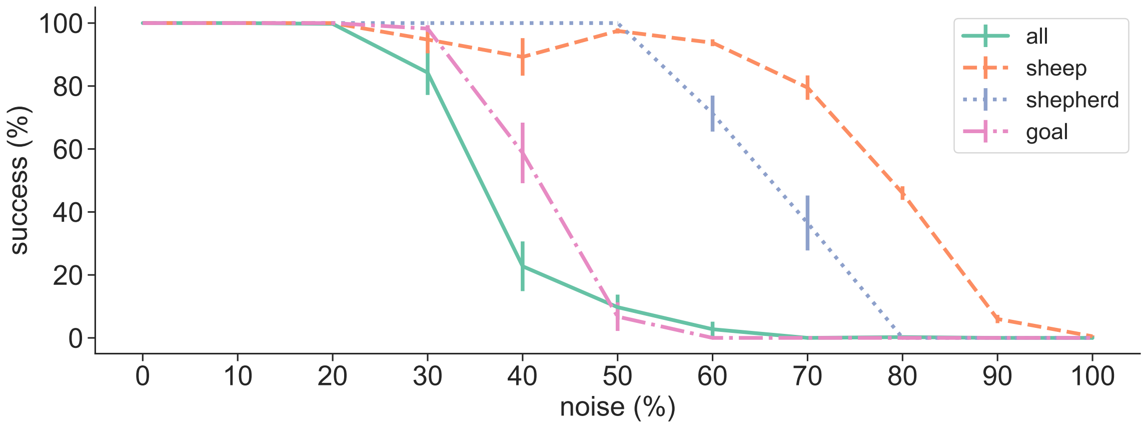


Fig. 5.5 Sensory noise analysis. The coloured curves represent different types of sensor readings that experience noise (see the text for details). Error bars represent standard deviations. The noise levels in x-axis are given in percentages.

certain readings are subjected to it. For example, if the shepherds cannot reliably detect the sheep (orange curve) the performance is better than if they cannot reliably detect the goal (pink curve). The shepherds are also tolerant to the situation where they cannot reliably detect each other (lilac curve)—even at a noise level of $p = 0.5$, they succeed in solving the task cooperatively without any significant degradation in performance.

5.4.2 Sensitivity Analysis

We examine how sensitive the controller's performance is with respect to changes in its parameters. Each of the eight controller parameters [see Table 5.1] was varied from -1 to 1 in steps of 0.05, with the other seven parameters remained fixed. For each parameter configuration, 100 trials were performed.

Figure 5.6 shows the average success rate obtained in the sensitivity analysis trials. The controller is sensitive to the parameters associated with the sensor reading for nothing ($v_{\ell,0}$ and $v_{r,0}$). This is the most commonly observed sensor reading according to Figure 5.4. On the other hand, the shepherd's motion when it detects a sheep ($s = 1$) is not highly critical, as long as $v_{\ell,1} < v_{r,1}$ (i.e. the robot turns left). When the shepherd detects the goal ($s = 3$), surprisingly, the velocity of the left wheel, $v_{\ell,3}$, is not critical. For $v_{\ell,3} < v_{r,3}$, a distinct strategy emerges, where shepherds orbit around both sheep and goal throughout the trial. When the shepherd detects another shepherd,

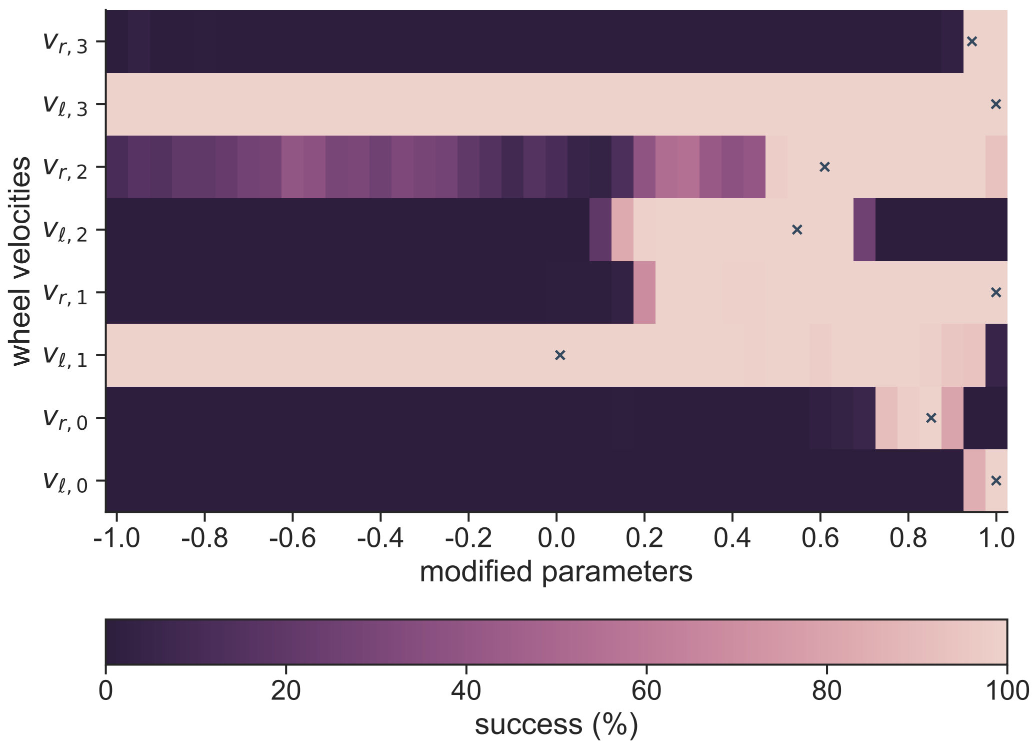


Fig. 5.6 Sensitivity analysis. Each of the eight velocity parameters is varied from -1 to 1 while the remaining seven parameters are kept constant. The colour map shows the average success rate across 100 trials. The dark grey crosses represent the unchanged parameters of the best controller, as shown in Table 5.1.

there is some leeway in sensitivity as long as $v_{\ell,2} \leq v_{r,2}$, but the margin is smaller than for the sheep and goal cases.

5.4.3 Sheep Speed Analysis

The default setup for the maximum sheep speed is half of the shepherd's maximum speed ($v_s = 0.5$). In this section, we change this condition and analyse the effect of the sheep speed to the shepherding performance. We performed 100 simulation trials for maximum velocity setting, with the default setup of $n = 10$ and $m = 20$ sheep for $T = 1500\,\mathrm{s}$. Figure 5.7 shows the average success rate obtained in the shepherding scenario.

One can see that the performance is almost 100% for the majority of the maximum speed settings. For the faster settings (i.e. $v_s > 0.5$) up to a threshold ($v_s = 0.9$), the shepherds' performance has a high success rate (over 90%). When, however, the relative

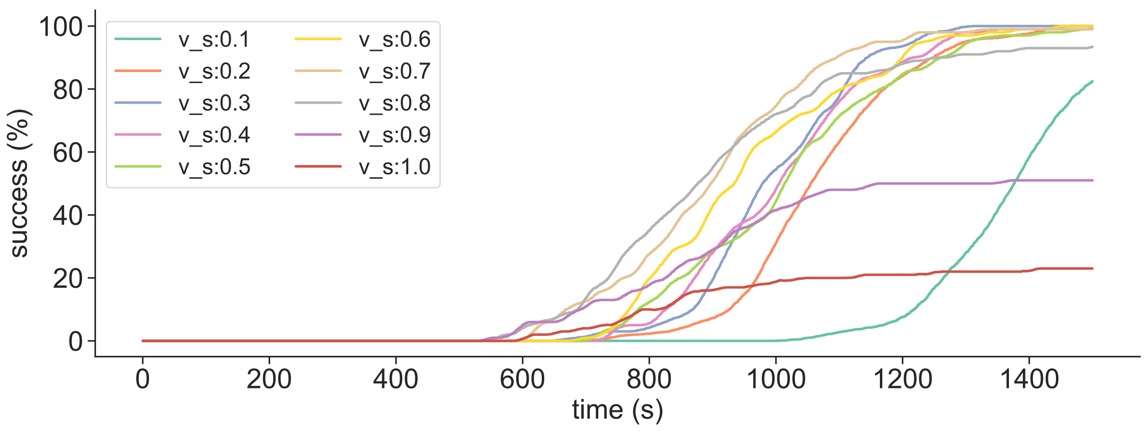


Fig. 5.7 The effect of sheep speed to the shepherding task: the success rate (as percentage) is given for each line (maximum velocity ratio, v_s, according to the shepherd's) over time.

maximum speed is above $v_s = 0.8$, the shepherds performance decreases significantly. The reason is that, if the sheep are as fast as shepherds, they cannot be caught. As the environment is unbounded, most of the $m = 20$ sheep diverge freely. On the other hand, when the sheep maximum speed is too low ($v_s = 0.1$), it takes a longer time to herd them to the goal location. As the trial has a fixed duration of $T = 1500$ s, the setting $v_s = 0.1$ could not be completed in the time period. One can see the time evolution for $v_s = 0.1$ in the dark-green line; it takes over 1000 s for it to start increasing (whereas the others take about 600 s) and it is still climbing when the trial is over (en-route reaching the goal).

5.4.4 Scalability Analysis

We examine the performance of the shepherd's controller in situations where the number of shepherds (n) and/or sheep (m) are different with respect to the default setup (i.e. $n = 10$ and $m = 20$). The numbers of shepherds considered were $n \in \{5, 10, 15, 20, 30, 40\}$. The numbers of sheep considered were $m \in \{10, 20, \ldots, 100\}$. For each combination of n and m, 100 simulation trials with the best controller were performed. Each trial lasted 1500 s. The initialisation region for all robots remained the same as shown in Figure 5.1.

Figure 5.8(top) shows the success rate. Due to the dynamic interactions, the relation between success rate and the number of agents can be nonlinear. The performance of the shepherd's controller scales well up to 70 sheep and 10 shepherds (which is the

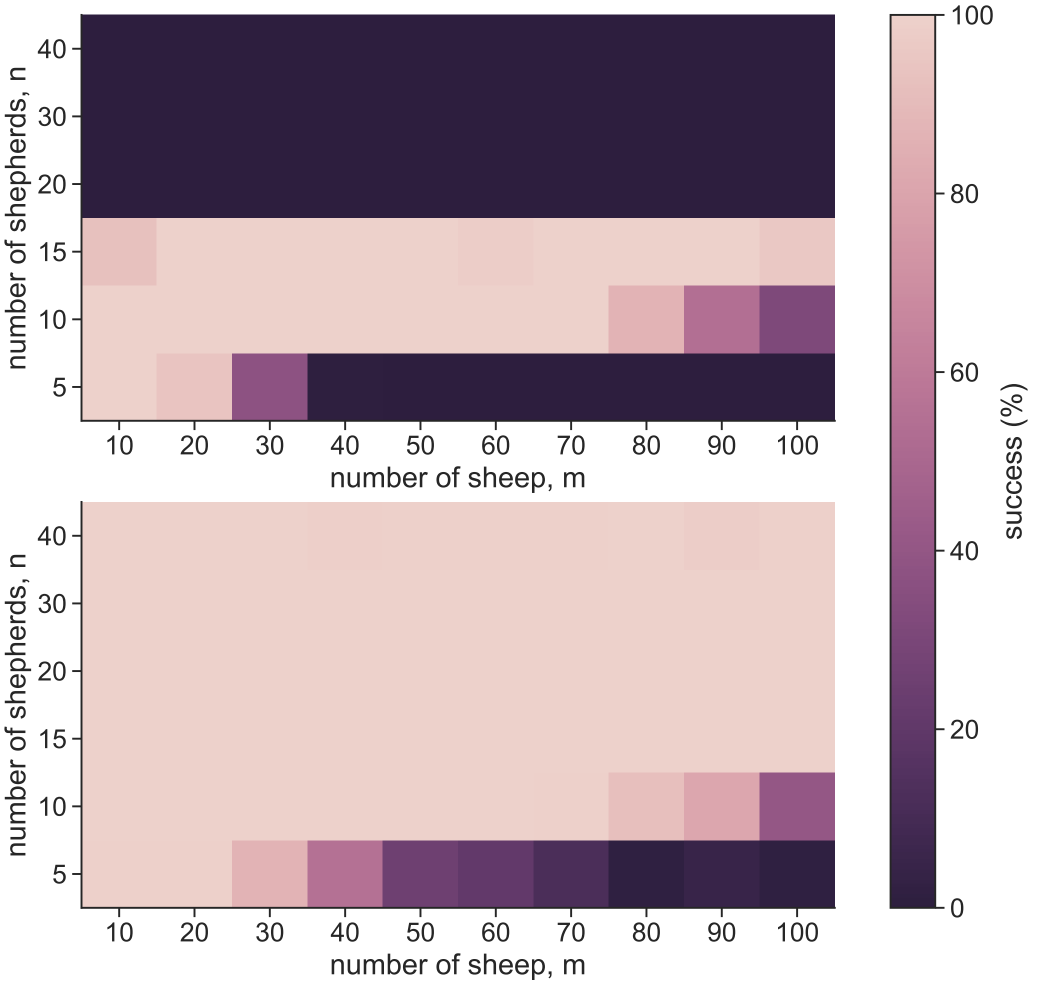


Fig. 5.8 Success rates (percentage of sheep retrieved to the goal region, averaged over 100 trials) for different numbers of shepherds and sheep. (top) Controller optimised for the 4-state sensor, given in Equation (5.1); (bottom) controller optimised for a 6-state sensor, given in Equation (5.14).

number of shepherds that was used during optimization). Beyond this point, however, the performance degrades. As the number of other agents in the environment increases, it becomes less likely for the shepherd to detect the goal. We hypothesise that the reason for the drop in performance could be that the sight of the goal is occluded for most of the time.

To alleviate the problem, we equipped the shepherd with a dedicated goal sensor, which can detect the goal even if behind some agents.[6] In this new setup, the shepherd can distinguish between six sensory states. Accordingly, the sensor reading, $s(t)$, can

[6]In practice, this is achievable if the goal object is significantly taller than the robots.

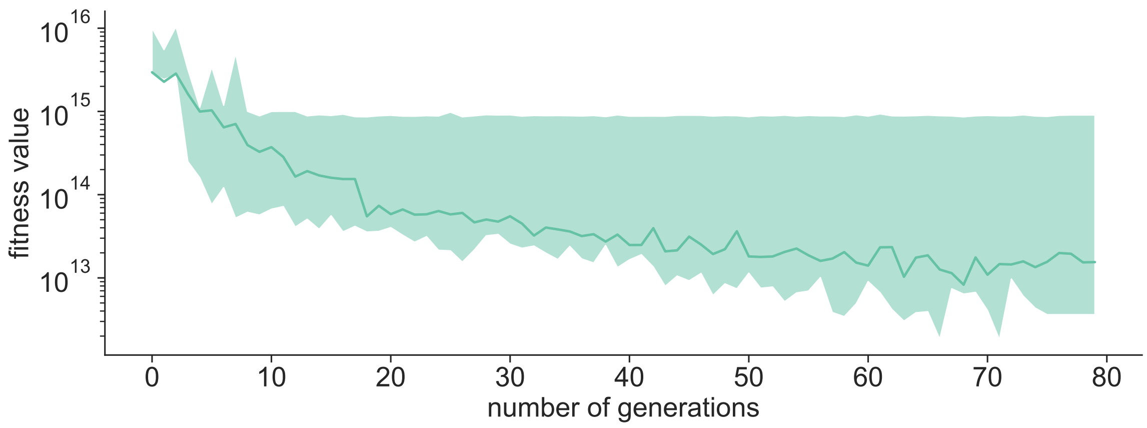


Fig. 5.9 Evolutionary dynamics of 30 runs for 80 generations for *extended* controller. The average fitness of each generation is represented by the solid line. The envelope indicates the minimum and the maximum fitness values in each generation.

be redefined by:

$$s(t) = \begin{cases} 0 & \text{if no object is detected,} \\ 1 & \text{if a sheep is detected,} \\ 2 & \text{if a shepherd is detected,} \\ 3 & \text{if only the goal is detected,} \\ 4 & \text{if both a sheep and the goal are detected,} \\ 5 & \text{if both a shepherd and the goal are detected.} \end{cases} \tag{5.14}$$

Note that the sensor is unable to detect a sheep or shepherd if located “behind” the goal. In the next section, we will explain how we obtain another controller with advanced sensing modality.

Extended Controller

We conducted 30 additional evolutions using 10 shepherds (with the 6-state sensor) and 20 sheep for a new controller. The evolutionary setup is the same as in Section 5.3.2. Figure 5.9 shows the fitness dynamics. The controller selection criteria is the same as Section 5.3.3. We refer to the controller with the lowest average fitness as the *extended controller*.

Table 5.2 shows the extended controller parameters. Two additional control parameters, “sheep and goal” and “shepherd and goal”, allow the shepherds to perform

Table 5.2 The *extended* controller velocity parameters.

nothing	sheep	shepherd	goal	sheep and goal	shepherd and goal
$v_{\ell,0}$	$v_{\ell,1}$	$v_{\ell,2}$	$v_{\ell,3}$	$v_{\ell,5}$	$v_{\ell,5}$
0.977367	-0.76925	0.568765	0.990289	-0.557135	-0.553871
$v_{r,0}$	$v_{r,1}$	$v_{r,2}$	$v_{r,3}$	$v_{r,4}$	$v_{r,5}$
0.811005	0.695187	0.485442	0.948778	0.859508	0.192682

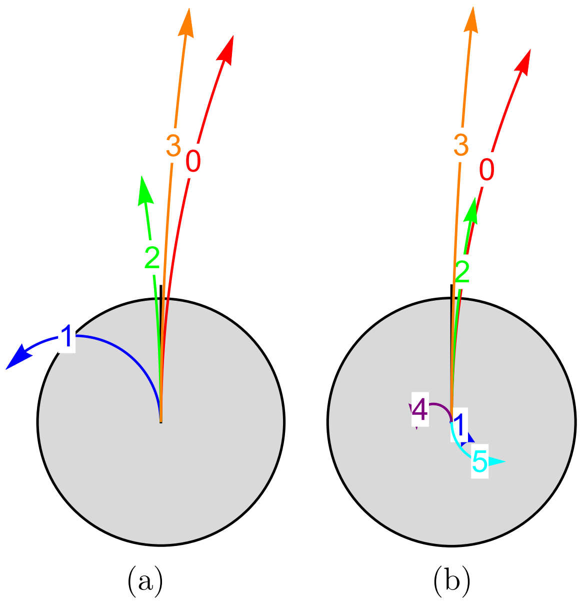
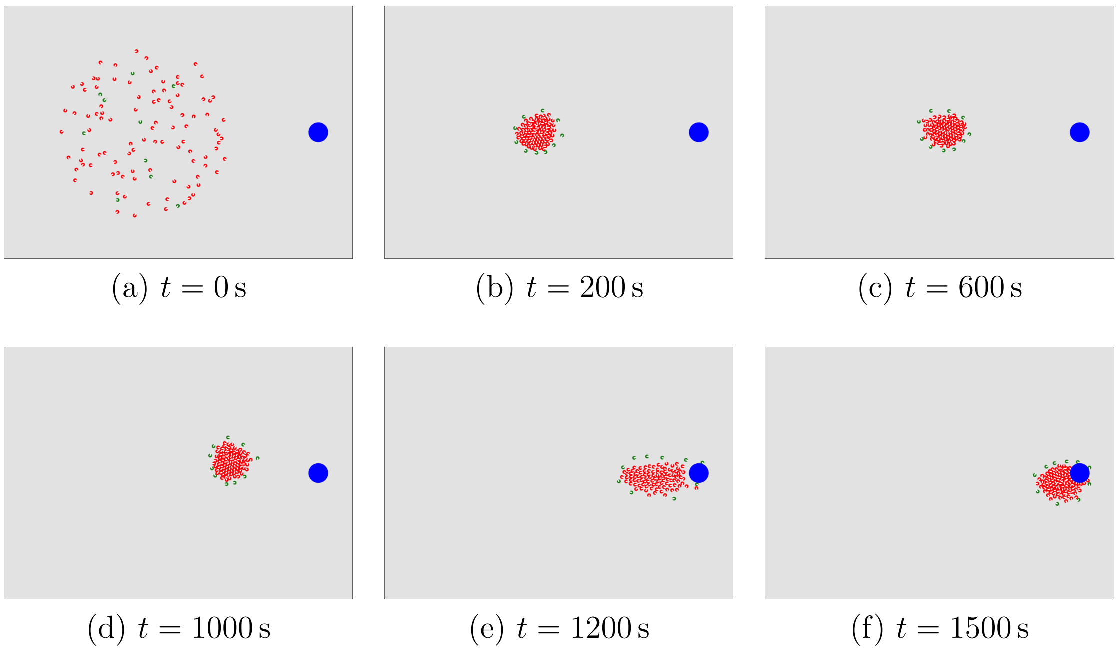


Fig. 5.10 Pictorial representations of the shepherding controllers—(a) *best* controller and (b) *extended* controller. The arcs indicate the direction the robot will head based on its sensor reading, s. The numbers on the arcs are the sensor readings.s

an enhanced behaviour. As goal occlusion is no longer an issue, due to the crowded environment, the performance of the extended controller scales far better with the number of sheep and shepherds. Figure 5.8(b) shows its success rate.

Figure 5.10 shows both the *best* and *extended* controllers as pictorial representations side-by-side. There are minor differences in the behaviour of the shepherd when it detects nothing ($s = 0$), another shepherd ($s = 2$), or the goal ($s = 3$). However, there is a major difference when it detects a sheep ($s = 1$). This is due to the difference of 'handling' sheep, as the extended controller has two additional behavioural rules.

Figure 5.11 shows the behaviour of the shepherds through a sequence of snapshots taken from a simulation trial using the extended controller. It can be observed that it is similar to the behaviour using the best controller. The process of herding 100


(a) $t = 0\,\mathrm{s}$ (b) $t = 200\,\mathrm{s}$ (c) $t = 600\,\mathrm{s}$

(d) $t = 1000\,\mathrm{s}$ (e) $t = 1200\,\mathrm{s}$ (f) $t = 1500\,\mathrm{s}$

Fig. 5.11 Sequence of snapshots showing how a group of 10 shepherds (red) gathering and moving a group of 100 sheep (green) towards the goal (blue) using the extended controller.

sheep takes about twice the time as herding 20 sheep. One can find trial videos in supplementary materials (Özdemir et al., 2017).

5.5 Discussions

This study showed for the first time that the problem of herding a group of dynamic agents can be addressed by a control group of embodied agents that lack the ability to compute. The controlling agents needed only to extract 2-bits of information from their environment—what object they first detect in their line of sight, if any. The controller directly mapped this information onto constant motor commands. This was sufficient to move the controlled group (i.e. sheep) reliably to the goal region.

The controller solution—obtained by an evolutionary algorithm—was robust with respect to sensory noise. It was also flexible with respect to moderate changes in the number of shepherds and sheep. We also investigated shepherds using 2-trits (i.e. 2 ternary digits) of information. These shepherds were able to simultaneously detect another agent and the goal, if in the line of sight. This enhanced sensing modality yielded better scalability with respect to the number of sheep and shepherds.

Although various solutions to the collective shepherding problem had been previously proposed, minimising the information that needs to be gathered and processed by the individual agents could help pave the way for applications at small scale—such as in nanomedicine—where the space and energy available for hardware is at a premium. An example of the shepherding concept in this domain can be observed in the manner in which white blood cells chase and engulf pathogens in the body. In the next chapter, we will focus on another self-organised coordination strategy: gathering.

Chapter 6

Gathering on a Grid

6.1 Introduction

Previously, we synthesised and analysed minimalist control strategies for swarms of robots. In particular, the swarms were operating in two-dimensional continuous environment, using differential-drive configuration. The robots were equipped with single line-of-sight sensor to detect the presence of an object in their environment. In the previous chapter discussing shepherding, we observed that when the environment is overcrowded with the high numbers of sheep or shepherds, the goal location is occluded. This then resulted in the robotic swarm being unable to accomplish their task. To alleviate this problem, we proposed another sensing modality in which two line-of-sight sensors were combined. The additional sensor, dedicated to perceive only the presence of the goal, allowed the enhanced swarm to achieve the task.

Getting into physical proximity is often a prerequisite for groups of autonomous robots that are collaborating to accomplish a specific task. The underlying problem, referred to as robot aggregation (Correll and Martinoli, 2011), gathering (Gordon et al., 2004), or rendezvous (Alpern, 1995), is not only relevant for groups of loosely coupled robots, but also for the units of modular reconfigurable systems that, by physically assembling with each other, form larger connected entities (Groß et al., 2006). In the previous chapters, we observed certain drawbacks when the computation-free swarms were using a deterministic controller. More importantly, due to the deterministic nature of the controllers, it is possible that there are "deadlock" situations. This means, certain configurations of the robots repeat over time, leading the robots to be in a stuck position and prevent them from accomplishing their task. We hypothesise that such situations can be overcome if a certain level of stochasticity is added to the system. It is possible that there is some randomness in the system, mostly caused by the environment, however, such randomness is uncontrollable. On the other hand, if the control system incorporate stochasticity in a controllable manner, the system could be benefit from it (e.g. avoid deadlock situations). In this chapter we investigate multi-robot gathering problem on a grid in order to identify the drawbacks of a deterministic controller in a discrete, constrained environment, design a control policy that overcomes the deadlock situations that may occur, and measure the effect of the stochasticity to the system performance.

We consider the situation in which all robots execute the same control policy, and are not allowed to exploit any cues from the environment, such as the intensity of ambient light (Claici et al., 2017; Schmickl et al., 2009). We propose novel decentralised control policies for gathering a group of embodied agents in a two-dimensional (2D) square tile environment. The policies are fully decentralised and reactive, and can be executed on anonymous, oblivious agents with chirality, but no sense of orientation. Unlike previous solutions to the gathering problem with such restricted agents, our policies are not limited to specific initial positions, take the agent's embodiment into account, and require only four trits (i.e. ternary digits) of sensory information, though the latter comes at the expense of unlimited-range sensing.

The work presented in this chapter extends the deterministic 'computation-free' swarming concept (Gauci et al., 2014c) in two directions. Firstly, it combines multiple sensor inputs an agent probes in a reactive manner. This then allows the agent to more precisely navigate in its environment. The extended concept is initially applied to the decentralised multi-agent gathering problem. It is shown that the deterministic control policy cannot succeed to gather agents in certain deadlock situations. The impossibility of guaranteeing gathering using the deterministic control policy demonstrates the need for stochasticity in the system. Secondly, the deterministic structure of the computation-free swarming is extended by allowing the agents to take an action based on a set of probability distributions. Two variations of stochastic control policies are proposed: a naïve and an optimised control policy. In the former, an agent chooses an action uniformly randomly from a set of eligible actions. In the latter, an agent chooses an action with an optimised probability distribution from a set of eligible actions. Compared to the original computation-free swarming framework, this extension allows each agent to reactively choose its action based on a few bits of sensory information in a stochastic, rather than deterministic way.

The chapter is organised as follows. Section 6.2 describes the gathering problem. Section 6.3 presents the deterministic control policy and proves the non-existence of a deterministic control solution. Section 6.4 presents a naïve, stochastic control policy and proves its convergence. Section 6.5 presents an optimised control policy, which is a refined variant of the naïve control policy and proves its convergence. Section 6.6 evaluates both control policies through numerical simulation experiments. It examines how the performance scales with the number of agents, and the robustness of the system with respect to sensory noise. Section 6.7 concludes the chapter.

6.2 Problem Definition

6.2.1 Environment and Robot Model

Consider an unbounded, obstacle-free 2D square tile environment, containing n embodied mobile agents. The agents are anonymous, that is, indistinguishable from each other, fully autonomous, and execute an identical controller. Each agent occupies a tile[1], has no orientation, but can distinguish between clockwise and counter-clockwise (chirality).

Each agent has four sensor units, one per side. Each unit provides a tuple of binary values, $s = (c, v)$. The first value indicates whether another agent is in physical *contact* with the sensor unit; c is true if another agent resides on the corresponding adjacent cell, and false, otherwise [see Fig. 6.1(a)]. The second value indicates if any other agent is *visible* from the sensor unit; v is true if at least one agent resides within the half-plane next to the unit, and false, otherwise [see Fig. 6.1(b)]. Formally,

$$\begin{aligned} c &= \text{true if } \exists j : (x_j^{\text{rel}}, y_j^{\text{rel}}) = (1, 0), \\ v &= \text{true if } \exists j : x_j^{\text{rel}} > 0, \end{aligned} \tag{6.1}$$

where $(x_j^{\text{rel}}, y_j^{\text{rel}}) \in \mathbb{Z}^2$ denotes the position of agent j in the reference frame that is (i) local to the sensing agent, and (ii) has its x-axis parallel to the sensor unit's sensing direction.[2] Note that $c = \text{true}$ implies $v = \text{true}$. In other words, the sensor unit provides a ternary digit of information, $s \in \{(\text{false}, \text{false}), (\text{false}, \text{true}), (\text{true}, \text{true})\}$.

[1] A tile can not be occupied by multiple agents.

[2] As the agent has no orientation, each of the four sensor units has its own local reference frame.

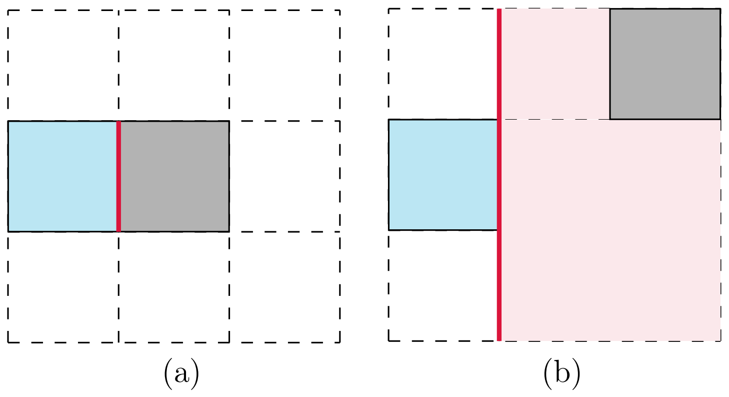


Fig. 6.1 An illustration of the sensing model showing (a) the contact and (b) the visibility sensors. The focal agent is shown as the blue cell and the perceived agent as the grey cell. (a) the grey agent is in contact (red line), that is, $c =$ true, or $s = 2$ and (b) the grey agent is in the visibility zone (red area), that is, $v =$ true, or $s = 1$.

Alternatively, we can use a trit notation;

$$s = \begin{cases} 0 & \text{if } (c, v) = (\text{false}, \text{false}), \\ 1 & \text{if } (c, v) = (\text{false}, \text{true}), \\ 2 & \text{if } (c, v) = (\text{true}, \text{true}). \end{cases} \tag{6.2}$$

Time is assumed to be discrete. In each round, every agent executes one action; the update order of agents is fixed.[3]

An agent can choose to remain in its current cell (action a_0), or move into any adjacent cell (actions a_1, a_2, a_3, a_4), provided the latter is not occupied. The sensor data, $S = (s_1, s_2, s_3, s_4)$, and actions $\mathcal{A} = \{a_0, a_1, a_2, a_3, a_4\}$ are provided in a counter-clockwise order. As the agents have no orientation, the specific starting elements are irrelevant, as long as they are consistent (e.g. s_3 and a_3). Figure 6.2 shows an illustration of scenario with $n = 5$ agents and the possible actions the focal agent can choose.

6.2.2 Objective

The configuration, $\mathcal{C}$, of a group of n agents defines their position in space. Formally, $\mathcal{C} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, $\forall i \neq j : (x_i \neq x_j) \vee (y_i \neq y_j)$, where $(x_j, y_j) \in \mathbb{Z}^2$ denotes the position of agent j in the global reference frame.

[3]Theoretical analysis presented is also valid if the order changes randomly.

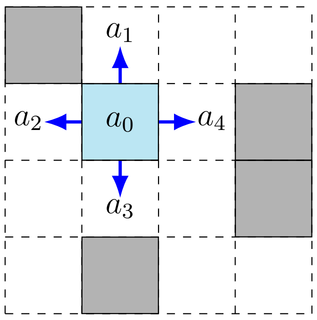


Fig. 6.2 An illustration of a gathering scenario with five agents. The focal agent is indicated by blue and the possible actions, $\mathcal{A}$, are indicated by blue arrows. The agent can choose to move any of the four unoccupied adjacent cells (actions a_1, a_2, a_3, a_4), or remain in its position (action a_0).

Given a configuration $\mathcal{C}$, let $\mathcal{B} = (b_x, b_y)$ denote the dimensions of the corresponding bounding box. Formally,

$$\begin{aligned} b_x &= 1 + \max_{i,j} \left| x_i - x_j \right| , \\ b_y &= 1 + \max_{i,j} \left| y_i - y_j \right| . \end{aligned} \tag{6.3}$$

Consider two configurations, $\mathcal{C}$ and $\bar{\mathcal{C}}$, of n agents, with bounding box dimensions $\mathcal{B}$ and $\bar{\mathcal{B}}$, respectively. Configuration $\mathcal{C}$ is said to be *preferred* to configuration $\bar{\mathcal{C}}$, denoted by $\bar{\mathcal{C}} \prec \mathcal{C}$, if $(b_x < \bar{b}_x) \wedge (b_y \leq \bar{b}_y)$ or $(b_x \leq \bar{b}_x) \wedge (b_y < \bar{b}_y)$ (see Figure 6.4). Figure 6.3 shows an example configuration of agents acting to reduce the dimensions of the bounding box.

Definition 1. *Configuration $\mathcal{C}$ is Pareto optimal, if there exists no other configuration of n agents that is preferred to $\mathcal{C}$.*

The agents start from arbitrary cells. The objective of the agents is to collectively reach, and remain indefinitely, in a Pareto optimal configuration.

Definition 2. *A control policy $\mathcal{P}$ is said to guarantee gathering if the final configuration $\mathcal{C}$ of n agents is Pareto optimal.*

6.2.3 Mathematical Analysis

In this section, we present a mathematical analysis of the gathering problem with given specifications.

Definition 3. *Let η denote the total number of empty cells within the bounding box, then $\eta = b_x b_y - n$.*

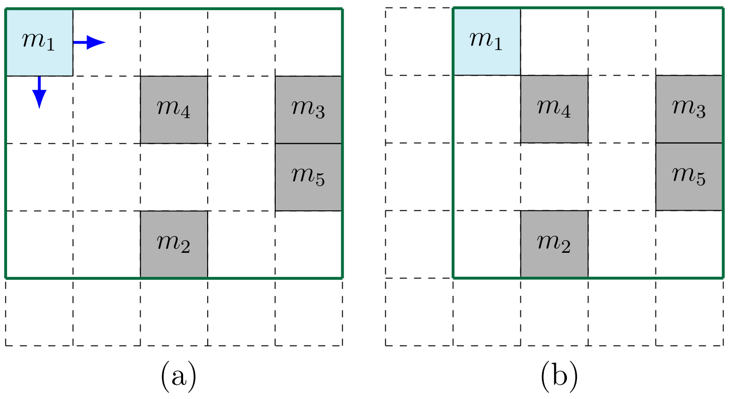


Fig. 6.3 An illustration of decentralised gathering in a 2D square tile environment. An agent can move into any empty, adjacent cell if another agent is perceived in the corresponding direction (e.g. see blue arrows for agent m_1), or remain in its current cell. Parts (a) and (b) show the situation immediately before and after agent m_1's move. In this instance, the group's spatial configuration became more compact—the dimensions of the corresponding bounding box reduced from 5×4 to 4×4 (see green frames).

Lemma 4. *A configuration of n agents contained in a bounding box of dimensions (b_x, b_y) is Pareto optimal, if and only if $\eta < \min\{b_x, b_y\}$.*

Proof. First, we consider the case that a Pareto optimal configuration, $\mathcal{C}$, is given. Without loss of generality, we assume $b_y \leq b_x$. If $\eta \geq \min\{b_x, b_y\} = b_y$, then $b_x > 1$. Let $\eta_1 \geq 0$ and $\eta_r > 0$ denote the number of empty cells within the first column and the remaining columns of the bounding box, respectively. We have $\eta = \eta_1 + \eta_r$. We can remove the $b_y - \eta_1 > 0$ agents from the first column and insert them in some of the $\eta_r = \eta - \eta_1 \geq b_y - \eta_1$ empty cells in the other columns. This would produce a configuration $\bar{\mathcal{C}}$ that has at least one fewer column and at most the same number of rows, that is, $\mathcal{C} \prec \bar{\mathcal{C}}$. This however contradicts our assumption that $\mathcal{C}$ is Pareto optimal. Consequently, $\eta = b_x b_y - n < \min\{b_x, b_y\}$.

Second, we consider the case of a configuration with $b_x b_y - n < \min\{b_x, b_y\}$. The number of empty cells within the bounding box is $\eta = b_x b_y - n < \min\{b_x, b_y\}$. In other words, neither of the dimensions of the bounding box can be reduced, without increasing the other dimension. Therefore, the configuration is Pareto optimal. ■

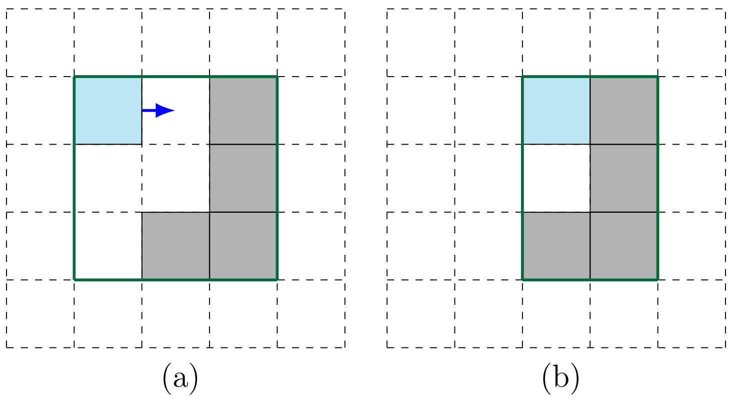


Fig. 6.4 Two example configurations, $\bar{\mathcal{C}}$ and $\mathcal{C}$, of five agents are given in (a) and (b), respectively. $\bar{\mathcal{C}} \prec \mathcal{C}$, as dimensions of $\mathcal{B}$ are smaller than $\bar{\mathcal{B}}$; $(b_x < \bar{b}_x) \wedge (b_y \leq \bar{b}_y)$ (see green frames surrounding the modules). Moreover, $\mathcal{C}$ is a Pareto optimal configuration for five agents.

6.3 Deterministic Control Policy

In this section, we first introduce the necessary concepts for designing a deterministic control policy, namely context classes and actions. Second, we introduce the concept of counter-examples in order to reduce the large search-domain. We then present 35 counter-examples that reduced the search-domain of size 331776 to 6. Finally, we theoretically prove that there exists no determinstic control policy

Algorithm 1 Deterministic Control Policy, $\mathcal{P}_D$

```
1: while true do
2:     $S \leftarrow ()$                         ▷ initialise an empty list of sensor readings
3:     for all $i \in \{1, 2, 3, 4\}$ do
4:         update $c_i$                          ▷ probe Boolean contact sensor $i$
5:         update $v_i$                          ▷ probe Boolean visibility sensor $i$
6:         $S \leftarrow (c_i, v_i)$             ▷ add sensor tuple to sensor readings
7:     end for
8:     $a \leftarrow \mathcal{T}[S]$             ▷ retrieve value of $S$
9:     execute $a$
10:    wait $\delta$ units of time
11: end while
```

Deterministic control policy assumes that there exists a predefined action for any given list of sensor quadruples. Algorithm 1 shows the formulation of the deterministic control policy. Each module executes an identical algorithm in each control cycle. The modules are assumed to be equipped with a 'lookup table' that includes a list of actions, $\mathcal{T}$, corresponding to each sensor reading. A lookup table is simply a list of $(key, value) = (S, a)$ pairs. A *key* represents a probed list of sensor readings quadruple, $S = (s_1, s_2, s_3, s_4) \in \mathcal{S}$, and *value* represents the action the module will take, $a \in \mathcal{A}$. We can now define the deterministic controller as a mapping

$$\begin{aligned} \mathcal{T}\colon \mathcal{S} &\longrightarrow \mathcal{A} \\ S &\longmapsto \mathcal{T}[S] = a \end{aligned} \tag{6.4}$$

There are four sensor tuples and each sensor can probe up to three different values. In total, there are $3^4 = 81$ different sensor quadruples. We categorise the sensor readings based on their similarity and refer to them with their unique labels to simplify the analysis. We refer to the unique sensing states of a module as a *context*. Context of a module is fully defined by its sensor reading S. In the next section, we describe the context classes in detail.

6.3.1 Context Classes

We labelled the contexts with a capital letter, starting from A, and added an enumerating digit for similar contexts. In other words, the contexts with an equal amount of $s = 1$ and $s = 2$ sensor readings, but in a different arrangement, have a digit appended. The 81 contexts are split into 19 equivalent classes. A context can have up to four rotational symmetric equivalents. We chose a representative 'base' sensor reading for each context, and call the other equivalents as 'permutations'.

Table 6.1 shows the base sensor readings and the number of permutations for each context apart from one, K.[4] There are three special contexts; A, D1, and F1. Context A has only one permutation as it is 'fully symmetric', whilst contexts D1 and F1 have two permutations as they are 'half-symmetric'. All other contexts represented in Table 6.1 have four permutations. For example, the base sensor reading for B is $S = (1, 1, 1, 2)$, and there are three other permutations; $(2, 1, 1, 1)$, $(1, 2, 1, 1)$, and $(1, 1, 2, 1)$. Figure 6.5 shows the contexts presented in Table 6.1. Each context is illustrated with its base sensor reading.

A module is in context K if all faces of the module are in contact with another module, or there is no module in the corresponding half-plane. That means, all the configurations are grouped as context K if the module is the only module in

[4]More information on context K will be provided later.

Table 6.1 Base sensor readings and number of permutations with context label.

context	sensor reading	number of permutations
A	(1,1,1,1)	1
B	(1,1,1,2)	4
C	(1,1,1,0)	4
D1	(2,1,2,1)	2
D2	(1,1,2,2)	4
E1	(2,1,0,1)	4
E2	(1,1,0,2)	4
E3	(1,1,2,0)	4
F1	(0,1,0,1)	2
F2	(1,1,0,0)	4
G	(1,2,2,2)	4
H1	(1,0,2,2)	4
H2	(1,2,0,2)	4
H3	(1,2,2,0)	4
I1	(1,0,0,2)	4
I2	(1,0,2,0)	4
I3	(1,2,0,0)	4
J	(1,0,0,0)	4

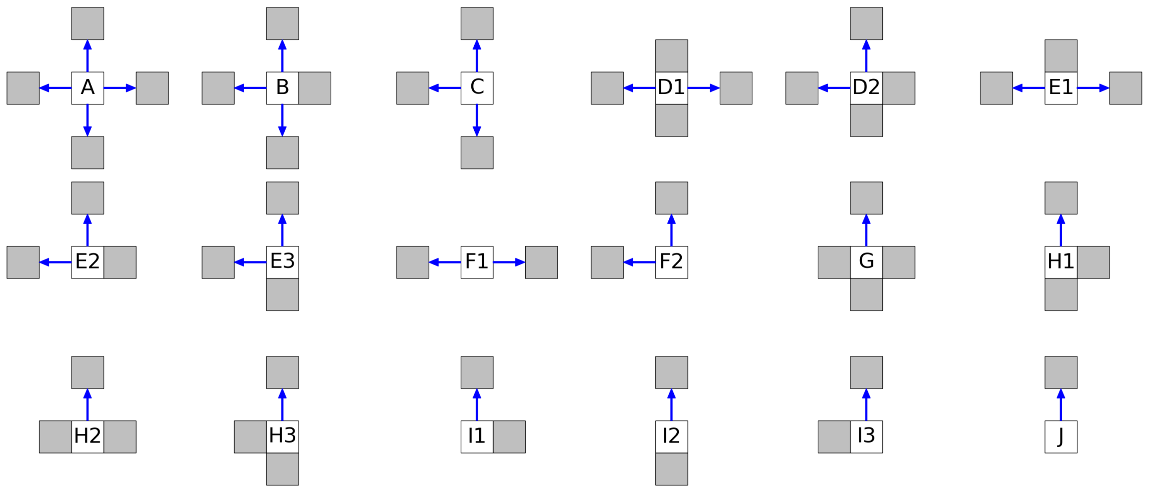


Fig. 6.5 Overview of the 18 unique contexts of the focal agent (white cell). A grey agent represents an agent in an adjacent cell, that is, an agent in physical contact ($c =$ true, or $s = 2$), or an agent that is visible ($v =$ true, or $s = 1$, for at least one of the focal agent's sides). Arrows indicate possible directions of movement. In addition, an agent may remain in its current position.

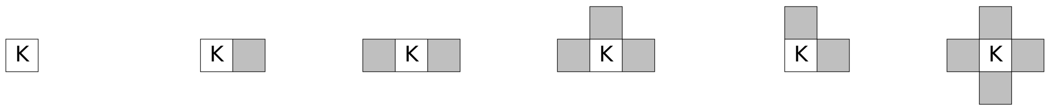


Fig. 6.6 Six example configurations for context K. The focal module (white cell) is situated as a single module, an edge module in three different configurations, a corner module, and an inner module, respectively. The former two edge configurations are only possible in a linear arrangement of the modules.

the environment, or it is at the outer edge, or outer corner, of a bounding box, or completely surrounded by other modules. Formally, a module is in context K if and only if $\forall s \in S, s \in \{0, 2\}$. There are in total 16 configurations in which a module can be in context K and Figure 6.6 shows six example configurations.

Context K has a variable number of permutations based on how many faces of a module detects $s = 0$ or $s = 2$. For example, a quadruple S with one contact detection have four permutations; $(2, 0, 0, 0)$, $(0, 2, 0, 0)$, $(0, 0, 2, 0)$, $(0, 0, 0, 2)$, whilst S with no contact detection has only one; $S = (0, 0, 0, 0)$. Contexts with two adjacent neighbours or two opposite neighbours [e.g. $(2, 0, 2, 0)$ and $(2, 2, 0, 0)$] are not the same, even though they share the same amount of contact readings. This situation is similar to any other label with a number (e.g. $\mathsf{D1}$ and $\mathsf{D2}$ have the same amount of $s = 1$ and $s = 2$), yet, we refer to both of these contexts as K.

6.3.2 Controller Design

In this section, we first present the necessary definitions, and certain assumptions for a deterministic control policy.

Definition 4. *Let* c *denote an arbitrary context. Then,* $\mathcal{A}_\mathsf{c}$ *is the set of possible actions for* c *and* a_c *is the action* c *takes (i.e.* $a_\mathsf{c} \in \mathcal{A}_\mathsf{c}$*).*

Remark. $\mathcal{A}_\mathsf{c}$ *is mutable, such that, the set of possible actions for the context* c *can be refined.*

Definition 5. *The list of actions utilised by a deterministic control policy is denoted by* $\mathcal{T}$*. Then,* $\mathcal{T} \in \prod_\mathsf{c} \mathcal{A}_\mathsf{c}$*, that is, a list of actions is an element of the Cartesian product of all sets of possible actions.*

Remark. *The disjoint union of list of actions* $\mathcal{T}$ *covers the set of all possible actions;* $\bigsqcup \mathcal{T} = \prod_\mathsf{c} \mathcal{A}_\mathsf{c}$.

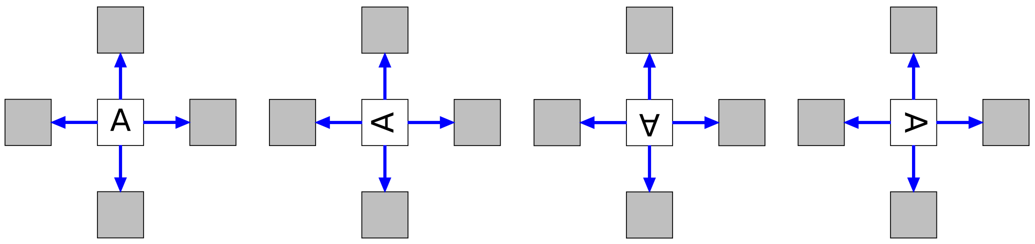


Fig. 6.7 Four possible configuration for the context A.

Definition 6. *Let* $|\cdot|$ *denote the cardinality (number of elements) of the set. Then, the number of possible controllers is* $\left|\prod_{\mathsf{c}} \mathcal{A}_{\mathsf{c}}\right| = \left|\bigsqcup \mathcal{T}\right|$.

Previously, we stated that due to lack of orientation, the order of the elements (sensor readings and actions) does not matter as long as it is consistent. However, in order to implement the control policy in numerical simulations, we are obliged to choose a starting element. We choose the first element (s_1 and a_1) as the North direction, and proceed in counter-clockwise order. Thus, the order from 1 to 4 would be set as North, West, South, and East.

Axiom 1. *A module in context* F2 *moves North, that is,* $a_{\mathsf{F2}} = a_1$.

Axiom 1 is the assumption we make on the context movement directions. Hereby, we choose a perspective and set a starter direction for F2 to North (i.e. a_1).

Axiom 2. *A module in context* K *does not move, that is,* $a_{\mathsf{K}} = a_0$.

Axiom 2 is the second assumption on the contexts. A module is in context K if it is one of the configurations as given in Figure 6.6. It is not permitted to move as it can only move out of the bounding-box, which contradicts with the control policy formulation.

Axiom 3. *A module in contexts* A, D1, *and* F1 *does not move, that is,* $a_{\mathsf{A}} = a_{\mathsf{D1}} = a_{\mathsf{F1}} = a_0$.

Axiom 3 is the final assumption made about the contexts. As the modules have no sense of orientation, they cannot choose a particular direction over another. The context does not provide any information to establish a unique orientation, and the module lacks non-determinism to choose one of multiple possibilities. Therefore, we assume that fully- and partially-symmetric contexts, A, D1, and F1, do not choose a direction to move. Figure 6.7 shows four possible configurations for the context A. As

Table 6.2 Possible actions for each of the 19 contexts.

context, c	possible actions, $\mathcal{A}_\mathsf{c}$
A	a_0
B	a_0, a_1, a_2, a_3
C	a_0, a_1, a_2, a_3
D1	a_0
D2	a_0, a_1, a_2
E1	a_0, a_2, a_4
E2	a_0, a_1, a_2
E3	a_0, a_1, a_2
F1	a_0
F2	a_1
G	a_0, a_1
H1	a_0, a_1
H2	a_0, a_1
H3	a_0, a_1
I1	a_0, a_1
I2	a_0, a_1
I3	a_0, a_1
J	a_0, a_1
K	a_0

the module has no sense of orientation, all four configurations are the same, thus, it is impossible for the module to favour one direction over another. Table 6.2 shows the possible actions for each of the contexts.

Remark. $\left|\prod_\mathsf{c} \mathcal{A}_\mathsf{c}\right| = 331776$ *for the possible actions presented in Table 6.2.*

From the remark, it is clear that the domain of possible controllers is large. One can search the whole domain to find a deterministic controller. However, this requires a proof for each $\mathcal{T} \in \prod_\mathsf{c} \mathcal{A}_\mathsf{c}$, whether the controller is successful or not. On the other hand, it is possible to reduce the size of the search domain. As such, we introduce 'counter-examples' to eliminate impossible actions. Each counter-example is independent from the other, although, they are built one after another in a sequential order. In the next section, we present 35 counter-examples, providing the necessary and sufficient conditions (assumptions and outcome) for each of them.

6.3.3 Presentation of Counter-Examples

In this section, we first define the necessary and sufficient conditions for a deterministic controller to exist. Then, we proceed by providing a definition for a counter-example and its 'type'. Finally, we present 35 counter-examples with details and summarise the set of possible actions.

Lemma 5. *A deterministic control policy $\mathcal{P}_D$ utilising $\mathcal{T}$ is a complete solution if and only if $\mathcal{P}_D$ gathers for any number of modules n, any initial configuration $\mathcal{C}_0$ with any fixed update order $\mathcal{U}$ in finite time.*

Definition 7. *A counter-example is a triple $\mathcal{E} = (\mathcal{T}, \mathcal{C}_0, \mathcal{U})$ such that $\mathcal{E}$ contradicts with the second term of the bi-conditional statement in Lemma 5.*

Remark. *A controller $\mathcal{T}$ cannot produce a complete solution if there exists at least one triple $\mathcal{E}_0$. For example, assume that there exists an initial configuration $\mathcal{C}_0$ of $n = n_0$ modules, with a certain fixed update order $\mathcal{U}_0$ that is theoretically impossible to gather the modules using the controller $\mathcal{T}$. This would contradict with the Lemma 5, thus, $\mathcal{E}_0$ would be a counter-example.*

Definition 8. *A 'static deadlock' is a non-Pareto optimal configuration $\mathcal{C}$ of modules, in which, $\mathcal{C}$ does not change over time. In other words, the modules remain in their place indefinitely, thus, gathering is theoretically impossible.*

Definition 9. *A 'dynamic deadlock' is a non-Pareto optimal configuration $\mathcal{C}$ of modules, with a particular update order, that results in fixed-periodic repetition. In other words, the modules remain in an indefinite cycle, thus, gathering is theoretically impossible.*

Algorithm 2 presents a pseudo-code for a function that reduces the space of possible controllers. The function accepts the 'current' space of possible controllers $\bigsqcup \mathcal{T}$ and a counter-example. In simple terms, every controller that has the same actions for the contexts that are *a priori* unknown and participated in the counter-example is removed from the space of controllers (line $4 - 8$). After each counter-example, we used Algorithm 2 to update the space of possible controllers.

Each counter-example is presented as a 'modified' table. We first begin with describing an 'example' counter-example. ***A Priori* Known Contexts** are the contexts that we either assumed (in one of the axioms), or discovered (using a counter-example). That means, we know what action the contexts should take, thus, the search space for such dimension is reduced before this counter-example is presented. ***A***

Algorithm 2 Function for evaluating controller $\mathcal{T}$

```
1: function ReduceControllerSpace($\bigsqcup\mathcal{T}, \mathcal{E} = (\mathcal{T}, \mathcal{C}, \mathcal{U})$)
2:     initialise $\bar{\mathsf{c}}$                ▷ initialise list of unknown contexts that participated in $\mathcal{E}$
3:     let $\overline{\mathcal{T}} \leftarrow \bigsqcup\mathcal{T}$                ▷ copy $\bigsqcup\mathcal{T}$ into $\overline{\mathcal{T}}$
4:     for all $\tau \in \bigsqcup\mathcal{T}$ do
5:         if $\forall \mathsf{c} \in \bar{\mathsf{c}}, \bigwedge(\tau[\mathsf{c}] = \mathcal{T}[\mathsf{c}])$ then        ▷ if $\bar{\mathsf{c}}$'s actions for $\tau$ are the same as $\mathcal{T}$
6:             $\overline{\mathcal{T}} \leftarrow \overline{\mathcal{T}} \setminus \tau$                ▷ remove $\tau$ from $\overline{\mathcal{T}}$
7:         end if
8:     end for
9:     return $\overline{\mathcal{T}}$                ▷ return to the updated union of controllers
10: end function
```

Counter-example 0	
***A Priori* Known Contexts**	The contexts with only one action
***A Priori* Unknown Contexts**	The contexts that have multiples actions, i.e. unknown contexts
Unknown Contexts Participated	The actions used for the unknown contexts that are participated
Knowledge Gained	The outcome of the counter-example
$\lvert\bigsqcup\mathcal{T}\rvert$ **Update**	The change in the number of possible controllers
Type of Counter-Example	Static or Dynamic Deadlock
Reference in Appendix	Caption number in Appendix

Illustration of initial configuration, $\mathcal{C}_0$

***Priori* Unknown Contexts**, on the other hand, are the contexts that have multiple possible actions so that currently we do not know which action is correct.

Unknown Contexts Participated are the actions used for the unknown contexts that participated in the counter-example. Note that the 'unknown contexts participated' is a subset of '*a priori* unknown contexts'. We can only determine an action is valid or not if the unknown context is actually present in the counter-example. **Knowledge Gained** is a logical statement, that is, the outcome of the counter-example. In other words, as the counter-example contradicts with the assumption on $\mathcal{T}$, it is the negation of the logical statement given in **Unknown Contexts Participated**.

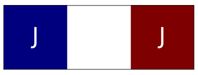
$\lvert\bigsqcup\mathcal{T}\rvert$ **Update** is the change in the search space dimensions. That means, it shows the number of possible controllers that exist before the counter-example, and after. **Type of Counter-Example** refers to one of the deadlock situations given in

Definition 8-9. It is either static or dynamic deadlock. **Reference in Appendix** includes all the images of the counter-examples. There are some counter-examples with a very long period, with the longest being 360. Thus, we only present the full list of images in Appendix A, as we present an image for each update.

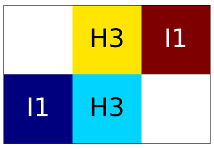
`Illustration of initial configuration,` $\mathcal{C}_0$ is the image of an initial configuration for the counter-example. For a counter-example with a type of static deadlock, this is the only image, hence, it is not repeated in the Appendix. Each module is represented by a different colour. The module colour-spectrum starts with dark blue and ends with dark red.

We designed counter-examples $1-16$ manually. They are solely based on a situation where if neither of the modules move, gathering becomes impossible.

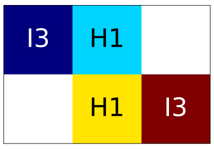
Counter-example 1	
***A Priori* Known Contexts**	A, D1, F1, F2, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I2, I3, J
Unknown Contexts Participated	$a_{\mathsf{J}} = a_0$
Knowledge Gained	$a_0 \notin \mathcal{A}_{\mathsf{J}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ Update	$331776 \to 165888$
Type of Counter-Example	Static Deadlock



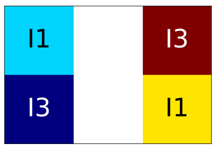
Counter-example 2	
***A Priori* Known Contexts**	A, D1, F1, F2, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I2, I3
Unknown Contexts Participated	$a_{\mathsf{H3}} = a_0 \wedge a_{\mathsf{I1}} = a_0$
Knowledge Gained	$a_0 \notin \mathcal{A}_{\mathsf{H3}} \vee a_0 \notin \mathcal{A}_{\mathsf{I1}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ Update	$165888 \to 124416$
Type of Counter-Example	Static Deadlock



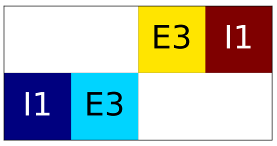
Counter-example 3	
***A Priori* Known Contexts**	A, D1, F1, F2, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I2, I3
Unknown Contexts Participated	$a_{\mathsf{H1}} = a_0 \wedge a_{\mathsf{I3}} = a_0$
Knowledge Gained	$a_0 \notin \mathcal{A}_{\mathsf{H1}} \vee a_0 \notin \mathcal{A}_{\mathsf{I3}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ Update	$124416 \to 93312$
Type of Counter-Example	Static Deadlock



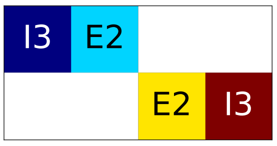
Counter-example 4	
***A Priori* Known Contexts**	A, D1, F1, F2, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I2, I3
Unknown Contexts Participated	$a_{\mathsf{I1}} = a_0 \wedge a_{\mathsf{I3}} = a_0$
Knowledge Gained	$a_0 \notin \mathcal{A}_{\mathsf{I1}} \vee a_0 \notin \mathcal{A}_{\mathsf{I3}}$
$\lvert \bigsqcup \mathcal{T} \rvert$ **Update**	$93312 \rightarrow 82944$
Type of Counter-Example	Static Deadlock



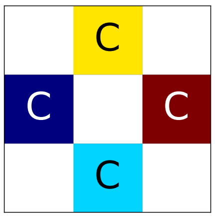
Counter-example 5	
***A Priori* Known Contexts**	A, D1, F1, F2, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I2, I3
Unknown Contexts Participated	$a_{\mathsf{E3}} = a_0 \wedge a_{\mathsf{I1}} = a_0$
Knowledge Gained	$a_0 \notin \mathcal{A}_{\mathsf{E3}} \vee a_0 \notin \mathcal{A}_{\mathsf{I1}}$
$\lvert \bigsqcup \mathcal{T} \rvert$ **Update**	$82944 \rightarrow 76032$
Type of Counter-Example	Static Deadlock



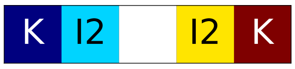
Counter-example 6	
***A Priori* Known Contexts**	A, D1, F1, F2, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I2, I3
Unknown Contexts Participated	$a_{\mathsf{E2}} = a_0 \wedge a_{\mathsf{I3}} = a_0$
Knowledge Gained	$a_0 \notin \mathcal{A}_{\mathsf{E2}} \vee a_0 \notin \mathcal{A}_{\mathsf{I3}}$
$\lvert \bigsqcup \mathcal{T} \rvert$ **Update**	$76032 \rightarrow 69120$
Type of Counter-Example	Static Deadlock



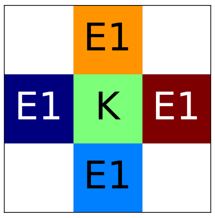
Counter-example 7	
***A Priori* Known Contexts**	A, D1, F1, F2, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I2, I3
Unknown Contexts Participated	$a_{\mathsf{C}} = a_0$
Knowledge Gained	$a_0 \notin \mathcal{A}_{\mathsf{C}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ Update	$69120 \to 51840$
Type of Counter-Example	Static Deadlock



Counter-example 8	
***A Priori* Known Contexts**	A, D1, F1, F2, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I2, I3
Unknown Contexts Participated	$a_{\mathsf{I2}} = a_0$
Knowledge Gained	$a_0 \notin \mathcal{A}_{\mathsf{I2}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ Update	$51840 \to 25920$
Type of Counter-Example	Static Deadlock

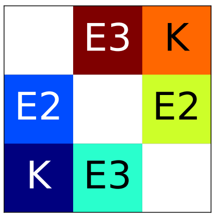


Counter-example 9	
***A Priori* Known Contexts**	A, D1, F1, F2, I2, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I3
Unknown Contexts Participated	$a_{\mathsf{E1}} = a_0$
Knowledge Gained	$a_0 \notin \mathcal{A}_{\mathsf{E1}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ Update	$25920 \to 17280$
Type of Counter-Example	Static Deadlock



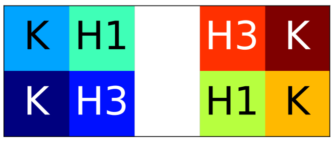
Counter-example 10

***A Priori* Known Contexts**	A, D1, F1, F2, I2, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I3
Unknown Contexts Participated	$a_{\mathsf{E2}} = a_0 \wedge a_{\mathsf{E3}} = a_0$
Knowledge Gained	$a_0 \notin \mathcal{A}_{\mathsf{E2}} \vee a_0 \notin \mathcal{A}_{\mathsf{E3}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ Update	$17280 \rightarrow 16128$
Type of Counter-Example	Static Deadlock



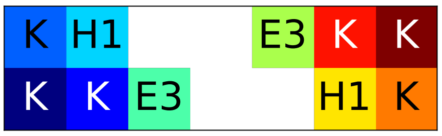
Counter-example 11

***A Priori* Known Contexts**	A, D1, F1, F2, I2, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I3
Unknown Contexts Participated	$a_{\mathsf{H1}} = a_0 \wedge a_{\mathsf{H3}} = a_0$
Knowledge Gained	$a_0 \notin \mathcal{A}_{\mathsf{H1}} \vee a_0 \notin \mathcal{A}_{\mathsf{H3}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ Update	$16128 \rightarrow 13824$
Type of Counter-Example	Static Deadlock

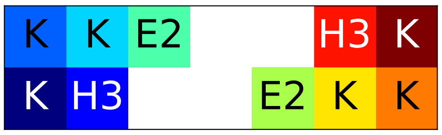


Counter-example 12

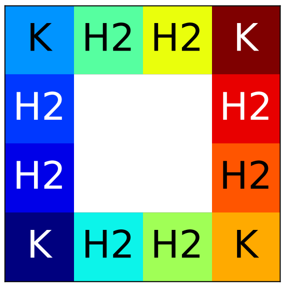
***A Priori* Known Contexts**	A, D1, F1, F2, I2, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I3
Unknown Contexts Participated	$a_{\mathsf{E3}} = a_0 \wedge a_{\mathsf{H1}} = a_0$
Knowledge Gained	$a_0 \notin \mathcal{A}_{\mathsf{E3}} \vee a_0 \notin \mathcal{A}_{\mathsf{H1}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ Update	$13824 \rightarrow 13248$
Type of Counter-Example	Static Deadlock



Counter-example 13	
***A Priori* Known Contexts**	A, D1, F1, F2, I2, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I3
Unknown Contexts Participated	$a_{\mathsf{E2}} = a_0 \wedge a_{\mathsf{H3}} = a_0$
Knowledge Gained	$a_0 \notin \mathcal{A}_{\mathsf{E2}} \vee a_0 \notin \mathcal{A}_{\mathsf{H3}}$
$\lvert\bigsqcup \mathcal{T}\rvert$ Update	$13248 \to 12672$
Type of Counter-Example	Static Deadlock

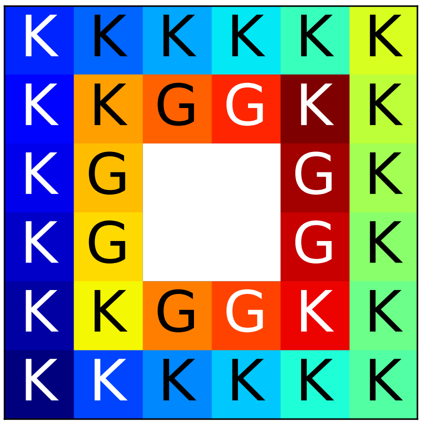


Counter-example 14	
***A Priori* Known Contexts**	A, D1, F1, F2, I2, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I3
Unknown Contexts Participated	$a_{\mathsf{H2}} = a_0$
Knowledge Gained	$a_0 \notin \mathcal{A}_{\mathsf{H2}}$
$\lvert\bigsqcup \mathcal{T}\rvert$ Update	$12672 \to 6336$
Type of Counter-Example	Static Deadlock



Counter-example 15

***A Priori* Known Contexts**	A, D1, F1, F2, H2, I2, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, G, H1, H3, I1, I3
Unknown Contexts Participated	$a_{\mathsf{G}} = a_0$
Knowledge Gained	$a_0 \notin \mathcal{A}_{\mathsf{G}}$
$\left\lvert \bigsqcup \mathcal{T} \right\rvert$ **Update**	$6336 \to 3168$
Type of Counter-Example	Static Deadlock



The number of possible controllers, $\left|\bigsqcup \mathcal{T}\right|$, was reduced from 331776 to 3168, meaning the search space is reduced by 99.045%. In terms of the knowledge we gained of the contexts, we list them by the order; $a_{\mathsf{J}} \neq a_0, a_{\mathsf{C}} \neq a_0, a_{\mathsf{E1}} \neq a_0, a_{\mathsf{G}} \neq a_0, a_{\mathsf{H2}} \neq a_0, a_{\mathsf{I2}} \neq a_0$. Even though we reduced the search space significantly, we only gained full knowledge on $\mathsf{G}, \mathsf{H2}$, and J. Prior to the next counter-example, we still do not know what are the specific actions of the 11 remaining contexts; $\mathsf{B}, \mathsf{C}, \mathsf{D2}, \mathsf{E1}, \mathsf{E2}, \mathsf{E3}, \mathsf{H1}, \mathsf{H3}, \mathsf{I1}, \mathsf{I2}$, and $\mathsf{I3}$.

Algorithm 3 Generating a counter-example $\mathcal{E}$

```
 1: function SIMULATE($\mathcal{T}$,$\mathcal{C}_0$,$\mathcal{U}$)
 2:     $\overline{\mathcal{C}} \leftarrow ()$                      ▷ initialise empty set of configurations
 3:     while $(t < T_D) \wedge (h! = 0)$ do                 ▷ loop until the trial ends
 4:         update $\mathcal{C}_t$             ▷ using Algorithm 1 utilising $\mathcal{T}$ for each module
 5:         $\overline{\mathcal{C}} \leftarrow \mathcal{C}_t$                          ▷ append the list with $\mathcal{C}_t$
 6:         update $\eta$                ▷ calculate using bounding-box dimensions
 7:         $t \leftarrow t + 1$                           ▷ increase the time-step counter
 8:     end while
 9:     return $\overline{\mathcal{C}}$         ▷ return to the list of configurations with length $T_D$
10: end function
11: function ISPERIODIC($\overline{\mathcal{C}}$)
12:     if $\exists! T_P$ s.t. $\forall t < (\lfloor T_D/2 \rfloor - T_P)$, $\bigwedge(\overline{\mathcal{C}}_t = \overline{\mathcal{C}}_{t+T_P})$ then
13:         return true                          ▷ $\overline{\mathcal{C}}$ has a period $T_P$, terminate
14:     end if
15:     return false                                         ▷ $\overline{\mathcal{C}}$ is not periodic
16: end function
17: initialise $\mathcal{C}_0$,$\mathcal{U}$
18: $\overline{\mathcal{C}} \leftarrow$ SIMULATE($\mathcal{T}$,$\mathcal{C}_0$,$\mathcal{U}$)
19: if ISPERIODIC($\overline{\mathcal{C}}$) = true then                 ▷ i.e. there exists a period $T_P$
20:     $\mathcal{E} \leftarrow (\mathcal{T},\mathcal{C}_0,\mathcal{U})$                   ▷ counter-example $\mathcal{E}$ is generated
21: end if
```

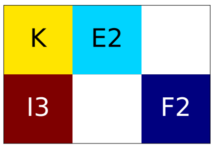
We then designed a computer algorithm that can 'generate' a counter-example by searching through initial configurations and update orders comprehensively. Algorithm 3 presents a pseudo-code for generating a counter-example from a given set of inputs. Line $1 - 8$ and $9 - 18$ define two salient functions for generating a counter-example. The first function, Simulate, simulates a group of modules using the deterministic control policy $\mathcal{P}_D$ (see Algorithm 1) utilising $\mathcal{T}$, starting from $\mathcal{C}_0$, with an update order $\mathcal{U}$ for maximum of T_D time-steps.[5] Then the function returns $\overline{\mathcal{C}}$, a list of configurations (of group of modules) for each time-step. The second function, IsPeriodic, determines whether the given $\overline{\mathcal{C}}$ comprises a periodic structure.

[5] T_D is set to a sufficiently high number. The successful trials, where gathering is possible, terminated immediately.

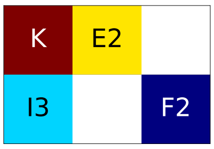
For a given $\mathcal{T}$, Algorithm 3 has to be executed using multiple pairs of $(\mathcal{C}_0, \mathcal{U})$ until a counter-example is generated. Ideally, one can search for a counter-example in the whole domain of $(\mathcal{C}_0, \mathcal{U})$ for a given number of modules n. It may not be a trivial task to search for all pairs of $(\mathcal{C}_0, \mathcal{U})$ for a large number n, however, it is feasible to search over the whole domain for a small number of modules. Considering that, we systematically searched all possible $(\mathcal{C}_0, \mathcal{U})$ pairs for $n = 4, 5, 6,$ and 8 in sufficiently large environments.

For a given number of modules n and environment size (l_x, l_y), one can calculate the total number of order updates by $n!$ and the total number of initial configurations by $\binom{l_x l_y}{n}$. For example, in order to generate a counter-example in a $(3, 2)$ environment with $n = 4$ modules, one needs to perform $\binom{3 \cdot 2}{4} 4! = 360$ simulations for each controller $\mathcal{T}$. The number of simulations can be tremendously extensive for larger n, mostly due to the factorial term. For example, for $n = 8$ in a $(5, 2)$ environment, the number of simulations would be 1814400. This is the reason why we gradually increased the number of modules over the different counter-examples. Once we searched through all pairs, we then changed the (l_x, l_y) and n. We first performed Algorithm 3 on lower number of n when we had large $\bigsqcup \mathcal{T}$, and increased n when $\bigsqcup \mathcal{T}$ decreased. Counter-examples $16 - 35$ followed the same procedure—they all are generated using Algorithm 3. A challenge is that the same counter-example may appear for different pairs of $(\mathcal{C}_0, \mathcal{U})$. We only present those that are novel and contribute to the overall knowledge we gained.

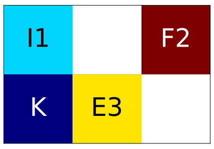
Counter-example 16	
***A Priori* Known Contexts**	A, D1, F1, F2, G, H2, I2, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, H1, H3, I1, I3
Unknown Contexts Participated	$a_{\mathsf{E2}} = a_1 \wedge a_{\mathsf{H1}} = a_1 \wedge a_{\mathsf{I1}} = a_1 \wedge a_{\mathsf{I3}} = a_0$
Knowledge Gained	$a_1 \notin \mathcal{A}_{\mathsf{E2}} \vee a_1 \notin \mathcal{A}_{\mathsf{H1}} \vee a_1 \notin \mathcal{A}_{\mathsf{I1}} \vee a_0 \notin \mathcal{A}_{\mathsf{I3}}$
$\lvert\bigsqcup \mathcal{T}\rvert$ Update	$3168 \rightarrow 2736$
Reference in Appendix	Figure A.1
Type of Counter-Example	Dynamic deadlock with period of 20



Counter-example 17	
***A Priori* Known Contexts**	A, D1, F1, F2, G, H2, I2, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, H1, H3, I1, I3
Unknown Contexts Participated	$a_{\mathsf{E2}} = a_2 \wedge a_{\mathsf{H1}} = a_1 \wedge a_{\mathsf{I1}} = a_1$
Knowledge Gained	$a_2 \notin \mathcal{A}_{\mathsf{E2}} \vee a_1 \notin \mathcal{A}_{\mathsf{H1}} \vee a_1 \notin \mathcal{A}_{\mathsf{I1}}$
$\lvert\bigsqcup \mathcal{T}\rvert$ Update	$2736 \rightarrow 1872$
Reference in Appendix	Figure A.2
Type of Counter-Example	Dynamic deadlock with period of 24

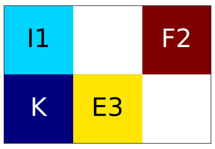


Counter-example 18	
***A Priori* Known Contexts**	A, D1, F1, F2, G, H2, I2, I3, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, H1, H3, I1
Unknown Contexts Participated	$a_{\mathsf{E3}} = a_1 \wedge a_{\mathsf{H3}} = a_1$
Knowledge Gained	$a_1 \notin \mathcal{A}_{\mathsf{E3}} \vee a_1 \notin \mathcal{A}_{\mathsf{H3}}$
$\lvert\bigsqcup \mathcal{T}\rvert$ Update	$1872 \rightarrow 1080$
Reference in Appendix	Figure A.3-A.4
Type of Counter-Example	Dynamic deadlock with period of 36



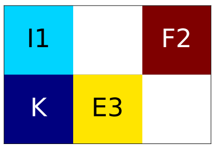
Counter-example 19

***A Priori* Known Contexts**	A, D1, F1, F2, G, H2, I2, I3, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, H1, H3, I1
Unknown Contexts Participated	$a_{\mathsf{E1}} = a_2 \wedge a_{\mathsf{E3}} = a_2 \wedge a_{\mathsf{H1}} = a_0$ $\wedge a_{\mathsf{H3}} = a_1 \wedge a_{\mathsf{I1}} = a_1$
Knowledge Gained	$a_2 \notin \mathcal{A}_{\mathsf{E1}} \vee a_2 \notin \mathcal{A}_{\mathsf{E3}} \vee a_0 \notin \mathcal{A}_{\mathsf{H1}}$ $\vee a_1 \notin \mathcal{A}_{\mathsf{H3}} \vee a_1 \notin \mathcal{A}_{\mathsf{I1}}$
$\lvert\bigsqcup \mathcal{T}\rvert$ Update	$1080 \to 972$
Reference in Appendix	Figure A.5
Type of Counter-Example	Dynamic deadlock with period of 28



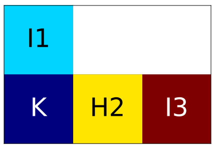
Counter-example 20

***A Priori* Known Contexts**	A, D1, F1, F2, G, H2, I2, I3, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, H1, H3, I1
Unknown Contexts Participated	$a_{\mathsf{E1}} = a_2 \wedge a_{\mathsf{E2}} = a_0 \wedge a_{\mathsf{E3}} = a_2$ $\wedge a_{\mathsf{H1}} = a_1 \wedge a_{\mathsf{H3}} = a_1 \wedge a_{\mathsf{I1}} = a_1$
Knowledge Gained	$a_2 \notin \mathcal{A}_{\mathsf{E1}} \vee a_0 \notin \mathcal{A}_{\mathsf{E2}} \vee a_2 \notin \mathcal{A}_{\mathsf{E3}}$ $\vee a_1 \notin \mathcal{A}_{\mathsf{H1}} \vee a_1 \notin \mathcal{A}_{\mathsf{H3}} \vee a_1 \notin \mathcal{A}_{\mathsf{I1}}$
$\lvert\bigsqcup \mathcal{T}\rvert$ Update	$972 \to 936$
Reference in Appendix	Figure A.6-A.7
Type of Counter-Example	Dynamic deadlock with period of 52

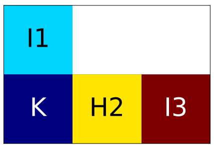


Counter-example 21

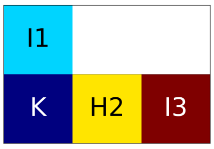
***A Priori* Known Contexts**	A, D1, F1, F2, G, H2, I2, I3, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, H1, H3, I1
Unknown Contexts Participated	$a_{\mathsf{E1}} = a_2 \wedge a_{\mathsf{E2}} = a_1 \wedge a_{\mathsf{H1}} = a_1 \wedge a_{\mathsf{I1}} = a_1$
Knowledge Gained	$a_2 \notin \mathcal{A}_{\mathsf{E1}} \vee a_1 \notin \mathcal{A}_{\mathsf{E2}} \vee a_1 \notin \mathcal{A}_{\mathsf{H1}} \vee a_1 \notin \mathcal{A}_{\mathsf{I1}}$
$\lvert\bigsqcup \mathcal{T}\rvert$ Update	$936 \to 756$
Reference in Appendix	Figure A.8
Type of Counter-Example	Dynamic deadlock with period of 16



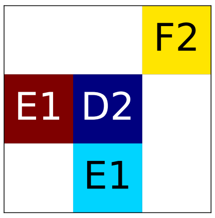
Counter-example 22	
***A Priori* Known Contexts**	A, D1, F1, F2, G, H2, I2, I3, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, H1, H3, I1
Unknown Contexts Participated	$a_{\mathsf{E1}} = a_4 \wedge a_{\mathsf{E2}} = a_1 \wedge a_{\mathsf{E3}} = a_0$ $\wedge a_{\mathsf{H1}} = a_1 \wedge a_{\mathsf{I1}} = a_1$
Knowledge Gained	$a_4 \notin \mathcal{A}_{\mathsf{E1}} \vee a_1 \notin \mathcal{A}_{\mathsf{E2}} \vee a_0 \notin \mathcal{A}_{\mathsf{E3}}$ $\vee a_1 \notin \mathcal{A}_{\mathsf{H1}} \vee a_1 \notin \mathcal{A}_{\mathsf{I1}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ Update	$756 \rightarrow 684$
Reference in Appendix	Figure A.9-A.10
Type of Counter-Example	Dynamic deadlock with period of 44



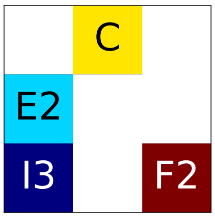
Counter-example 23	
***A Priori* Known Contexts**	A, D1, F1, F2, G, H2, I2, I3, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, E3, H1, H3, I1
Unknown Contexts Participated	$a_{\mathsf{E1}} = a_4 \wedge a_{\mathsf{E2}} = a_1 \wedge a_{\mathsf{H1}} = a_1$ $\wedge a_{\mathsf{H3}} = a_0 \wedge a_{\mathsf{I1}} = a_1$
Knowledge Gained	$a_4 \notin \mathcal{A}_{\mathsf{E1}} \vee a_1 \notin \mathcal{A}_{\mathsf{E2}} \vee a_1 \notin \mathcal{A}_{\mathsf{H1}}$ $\vee a_0 \notin \mathcal{A}_{\mathsf{H3}} \vee a_1 \notin \mathcal{A}_{\mathsf{I1}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ Update	$684 \rightarrow 612$
Reference in Appendix	Figure A.11
Type of Counter-Example	Dynamic deadlock with period of 28



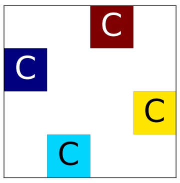
Counter-example 24	
***A Priori* Known Contexts**	A, D1, E3, F1, F2, G, H2, H3, I2, I3, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, H1, I1
Unknown Contexts Participated	$a_{\mathsf{B}} = a_0 \wedge a_{\mathsf{D2}} = a_0 \wedge a_{\mathsf{E1}} = a_4$
Knowledge Gained	$a_0 \notin \mathcal{A}_{\mathsf{B}} \vee a_0 \notin \mathcal{A}_{\mathsf{D2}} \vee a_4 \notin \mathcal{A}_{\mathsf{E1}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ Update	$612 \rightarrow 579$
Reference in Appendix	Figure A.12-A.13
Type of Counter-Example	Dynamic deadlock with period of 32



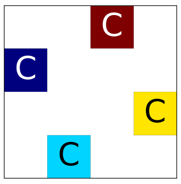
Counter-example 25	
***A Priori* Known Contexts**	A, D1, E3, F1, F2, G, H2, H3, I2, I3, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, H1, I1
Unknown Contexts Participated	$a_{\mathsf{E2}} = a_2$
Knowledge Gained	$a_2 \notin \mathcal{A}_{\mathsf{E2}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ Update	$579 \rightarrow 408$
Reference in Appendix	Figure A.14
Type of Counter-Example	Dynamic deadlock with period of 8



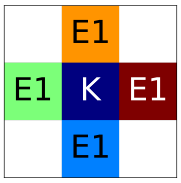
Counter-example 26	
***A Priori* Known Contexts**	A, D1, E3, F1, F2, G, H2, H3, I2, I3, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, H1, I1
Unknown Contexts Participated	$a_{\mathsf{C}} = a_1$
Knowledge Gained	$a_1 \notin \mathcal{A}_{\mathsf{C}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ Update	$408 \rightarrow 272$
Reference in Appendix	Figure A.15-A.19
Type of Counter-Example	Dynamic deadlock with period of 48



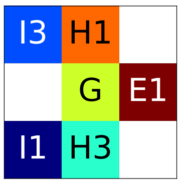
Counter-example 27	
***A Priori* Known Contexts**	A, D1, E3, F1, F2, G, H2, H3, I2, I3, J, K
***A Priori* Unknown Contexts**	B, C, D2, E1, E2, H1, I1
Unknown Contexts Participated	$a_{\mathsf{C}} = a_3$
Knowledge Gained	$a_3 \notin \mathcal{A}_{\mathsf{C}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ **Update**	$272 \to 136$
Reference in Appendix	Figure A.20
Type of Counter-Example	Dynamic deadlock with period of 8



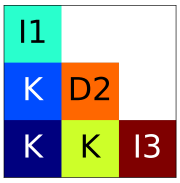
Counter-example 28	
***A Priori* Known Contexts**	A, C, D1, E3, F1, F2, G, H2, H3, I2, I3, J, K
***A Priori* Unknown Contexts**	B, D2, E1, E2, H1, I1
Unknown Contexts Participated	$a_{\mathsf{E1}} = a_4$
Knowledge Gained	$a_4 \notin \mathcal{A}_{\mathsf{E1}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ **Update**	$136 \to 48$
Reference in Appendix	Figure A.21-A.23
Type of Counter-Example	Dynamic deadlock with period of 40



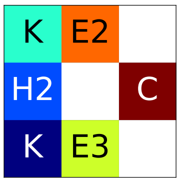
Counter-example 29	
***A Priori* Known Contexts**	A, C, D1, E1, E3, F1, F2, G, H2, H3 I1, I2, I3, J, K
***A Priori* Unknown Contexts**	B, D2, E2, H1
Unknown Contexts Participated	$a_{\mathsf{B}} = a_0$
Knowledge Gained	$a_0 \notin \mathcal{A}_{\mathsf{B}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ **Update**	$48 \to 36$
Reference in Appendix	Figure A.24-A.26
Type of Counter-Example	Dynamic deadlock with period of 48



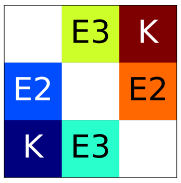
Counter-example 30	
***A Priori* Known Contexts**	A, C, D1, E1, E3, F1, F2, G, H2, H3 I1, I2, I3, J, K
***A Priori* Unknown Contexts**	B, D2, E2, H1
Unknown Contexts Participated	$a_{\mathsf{D2}} = a_2 \wedge a_{\mathsf{E2}} = a_1 \wedge a_{\mathsf{H1}} = a_1$
Knowledge Gained	$a_2 \notin \mathcal{A}_{\mathsf{D2}} \vee a_1 \notin \mathcal{A}_{\mathsf{E2}} \vee a_1 \notin \mathcal{A}_{\mathsf{H1}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ **Update**	$36 \rightarrow 33$
Reference in Appendix	Figure A.27-A.33
Type of Counter-Example	Dynamic deadlock with period of 126



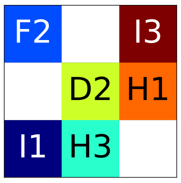
Counter-example 31	
***A Priori* Known Contexts**	A, C, D1, E1, E3, F1, F2, G, H2, H3 I1, I2, I3, J, K
***A Priori* Unknown Contexts**	B, D2, E2, H1
Unknown Contexts Participated	$a_{\mathsf{D2}} = a_2 \wedge a_{\mathsf{H1}} = a_1$
Knowledge Gained	$a_2 \notin \mathcal{A}_{\mathsf{D2}} \vee a_1 \notin \mathcal{A}_{\mathsf{H1}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ **Update**	$33 \rightarrow 30$
Reference in Appendix	Figure A.34-A.35
Type of Counter-Example	Dynamic deadlock with period of 36



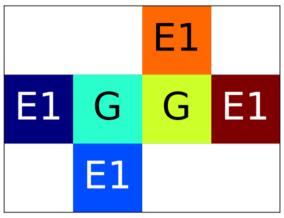
Counter-example 32	
***A Priori* Known Contexts**	A, C, D1, E1, E3, F1, F2, G, H2, H3 I1, I2, I3, J, K
***A Priori* Unknown Contexts**	B, D2, E2, H1
Unknown Contexts Participated	$a_{\mathsf{H1}} = a_1$
Knowledge Gained	$a_1 \notin \mathcal{A}_{\mathsf{H1}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ Update	$30 \rightarrow 18$
Reference in Appendix	Figure A.36-A.47
Type of Counter-Example	Dynamic deadlock with period of 228



Counter-example 33	
***A Priori* Known Contexts**	A, C, D1, E1, E3, F1, F2, G, H1, H2, H3 I1, I2, I3, J, K
***A Priori* Unknown Contexts**	B, D2, E2
Unknown Contexts Participated	$a_{\mathsf{B}} = a_2$
Knowledge Gained	$a_2 \notin \mathcal{A}_{\mathsf{B}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ Update	$18 \rightarrow 12$
Reference in Appendix	Figure A.48-A.51
Type of Counter-Example	Dynamic deadlock with period of 60



Counter-example 34	
***A Priori* Known Contexts**	A, C, D1, E1, E3, F1, F2, G, H1, H2, H3 I1, I2, I3, J, K
***A Priori* Unknown Contexts**	B, D2, E2
Unknown Contexts Participated	$a_{\mathsf{D2}} = a_0$
Knowledge Gained	$a_0 \notin \mathcal{A}_{\mathsf{D2}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ **Update**	$12 \to 8$
Reference in Appendix	Figure A.52
Type of Counter-Example	Dynamic deadlock with period of 12



Counter-example 35	
***A Priori* Known Contexts**	A, C, D1, E1, E3, F1, F2, G, H1, H2, H3 I1, I2, I3, J, K
***A Priori* Unknown Contexts**	B, D2, E2
Unknown Contexts Participated	$a_{\mathsf{B}} = a_1 \wedge a_{\mathsf{D2}} = a_1$
Knowledge Gained	$a_1 \notin \mathcal{A}_{\mathsf{B}} \vee a_1 \notin \mathcal{A}_{\mathsf{D2}}$
$\lvert\bigsqcup\mathcal{T}\rvert$ **Update**	$8 \to 6$
Reference in Appendix	Figure A.53-A.56
Type of Counter-Example	Dynamic deadlock with period of 40

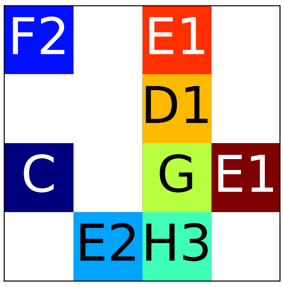


Table 6.3 Actions for *a priori* known contexts and the six combinations of actions for the remaining unknown contexts.

$a_{\mathsf{A}} = a_0$	$a_{\mathsf{C}} = a_2$	$a_{\mathsf{D1}} = a_0$	$a_{\mathsf{E1}} = a_2$
$a_{\mathsf{E3}} = a_2$	$a_{\mathsf{F1}} = a_0$	$a_{\mathsf{F2}} = a_1$	$a_{\mathsf{G}} = a_1$
$a_{\mathsf{H1}} = a_0$	$a_{\mathsf{H2}} = a_1$	$a_{\mathsf{H3}} = a_1$	$a_{\mathsf{I1}} = a_0$
$a_{\mathsf{I2}} = a_1$	$a_{\mathsf{I3}} = a_1$	$a_{\mathsf{J}} = a_1$	$a_{\mathsf{K}} = a_0$
$(a_{\mathsf{B}} = a_1) \wedge (a_{\mathsf{D2}} = a_2) \wedge (a_{\mathsf{E2}} = a_0)$			
$(a_{\mathsf{B}} = a_1) \wedge (a_{\mathsf{D2}} = a_2) \wedge (a_{\mathsf{E2}} = a_1)$			
$(a_{\mathsf{B}} = a_3) \wedge (a_{\mathsf{D2}} = a_1) \wedge (a_{\mathsf{E2}} = a_0)$			
$(a_{\mathsf{B}} = a_3) \wedge (a_{\mathsf{D2}} = a_1) \wedge (a_{\mathsf{E2}} = a_1)$			
$(a_{\mathsf{B}} = a_3) \wedge (a_{\mathsf{D2}} = a_2) \wedge (a_{\mathsf{E2}} = a_0)$			
$(a_{\mathsf{B}} = a_3) \wedge (a_{\mathsf{D2}} = a_2) \wedge (a_{\mathsf{E2}} = a_1)$			

In Table 6.3, we present the six possible controllers after the last counter-example has been applied. Amongst the six remaining controllers, 16 of 19 contexts have the same action, that is, we have full knowledge of 16 of the contexts. The contexts that are still unknown are $\mathsf{B}, \mathsf{D2}, \mathsf{E2}$. Each of them have two possible actions, though not all combinations are possible.

Table 6.4 shows the summary. The first row of the table shows the details prior to the first counter-example being applied. Whilst we gain insights after each counter-example, some of them reveal the 'full' knowledge of a context. This can be clearly seen from the table, where the list of remaining unknown contexts reduces. For example, after the first counter-example, we fully discovered that $a_{\mathsf{J}} \neq a_0$, thus, J is no longer unknown. On the other hand, the rest of the counter-examples reveal a 'partial' knowledge of contexts. That means the knowledge is conditional, for example, $(a_{\mathsf{H3}} \neq a_0) \vee (a_{\mathsf{I1}} \neq a_0)$. Below, we list the full and 'explicit' knowledge gained from counter-examples (denoted as $\mathcal{E}_{\#}$) in an alphabetical order.

- $a_{\mathsf{C}} \neq a_0$ $(\mathcal{E}_7)$, $a_{\mathsf{C}} \neq a_1$ $(\mathcal{E}_{26})$, $a_{\mathsf{C}} \neq a_3$ $(\mathcal{E}_{27})$ $\implies$ $a_{\mathsf{C}} = a_2$
- $a_{\mathsf{B}} \neq a_0$ $(\mathcal{E}_{29})$, $a_{\mathsf{B}} \neq a_2$ $(\mathcal{E}_{33})$
- $a_{\mathsf{D2}} \neq a_0$ $(\mathcal{E}_{34})$
- $a_{\mathsf{E1}} \neq a_0$ $(\mathcal{E}_9)$, $a_{\mathsf{E1}} \neq a_4$ $(\mathcal{E}_{28})$ $\implies$ $a_{\mathsf{E1}} = a_2$
- $a_{\mathsf{E2}} \neq a_2$ $(\mathcal{E}_{25})$
- $a_{\mathsf{G}} \neq a_0$ $(\mathcal{E}_{15})$ $\implies$ $a_{\mathsf{G}} = a_1$
- $a_{\mathsf{H1}} \neq a_1$ $(\mathcal{E}_{32})$ $\implies$ $a_{\mathsf{H1}} = a_0$
- $a_{\mathsf{H2}} \neq a_0$ $(\mathcal{E}_{14})$ $\implies$ $a_{\mathsf{H2}} = a_1$
- $a_{\mathsf{I2}} \neq a_0$ $(\mathcal{E}_8)$ $\implies$ $a_{\mathsf{I2}} = a_1$
- $a_{\mathsf{J}} \neq a_0$ $(\mathcal{E}_1)$ $\implies$ $a_{\mathsf{J}} = a_1$

Table 6.4 Summary of counter-examples showing n and (l_x, l_y) used, the remaining number of possible controllers and unknown contexts.

CE Index	n	(l_x, l_y)	$\lvert\bigsqcup\mathcal{T}\rvert$	remaining unknown contexts, c_u
N/A	N/A	N/A	331776	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I2, I3, J
1	2	$(3, 1)$	165888	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I2, I3
2	4	$(3, 2)$	124416	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I2, I3
3	4	$(3, 2)$	93312	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I2, I3
4	4	$(3, 2)$	82944	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I2, I3
5	4	$(4, 2)$	76032	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I2, I3
6	4	$(4, 2)$	69120	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I2, I3
7	4	$(3, 3)$	51840	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I2, I3
8	4	$(5, 1)$	25920	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I3
9	5	$(3, 3)$	17280	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I3
10	6	$(3, 3)$	16128	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I3
11	8	$(5, 2)$	13824	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I3
12	10	$(7, 2)$	13248	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I3
13	10	$(7, 2)$	12672	B, C, D2, E1, E2, E3, G, H1, H2, H3, I1, I3
14	12	$(4, 4)$	6336	B, C, D2, E1, E2, E3, G, H1, H3, I1, I3
15	32	$(6, 6)$	3168	B, C, D2, E1, E2, E3, H1, H3, I1, I3
16	4	$(3, 2)$	2736	B, C, D2, E1, E2, E3, H1, H3, I1, I3
17	4	$(3, 2)$	1872	B, C, D2, E1, E2, E3, H1, H3, I1
18	4	$(3, 2)$	1080	B, C, D2, E1, E2, E3, H1, H3, I1
19	4	$(3, 2)$	972	B, C, D2, E1, E2, E3, H1, H3, I1
20	4	$(3, 2)$	936	B, C, D2, E1, E2, E3, H1, H3, I1
21	4	$(3, 2)$	756	B, C, D2, E1, E2, E3, H1, H3, I1
22	4	$(3, 2)$	684	B, C, D2, E1, E2, E3, H1, H3, I1
23	4	$(3, 2)$	612	B, C, D2, E1, E2, H1, I1
24	4	$(3, 3)$	579	B, C, D2, E1, E2, H1, I1
25	4	$(3, 3)$	408	B, C, D2, E1, E2, H1, I1
26	4	$(4, 4)$	272	B, C, D2, E1, E2, H1, I1
27	4	$(4, 4)$	136	B, D2, E1, E2, H1, I1
28	5	$(3, 3)$	48	B, D2, E2, H1
29	6	$(3, 3)$	36	B, D2, E2, H1
30	6	$(3, 3)$	33	B, D2, E2, H1
31	6	$(3, 3)$	30	B, D2, E2, H1
32	6	$(3, 3)$	18	B, D2, E2
33	6	$(3, 3)$	12	B, D2, E2
34	6	$(4, 3)$	8	B, D2, E2
35	8	$(4, 4)$	6	B, D2, E2

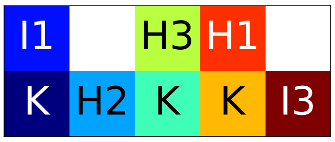
6.3.4 Theoretical Analysis

We start this section with a theorem proving the impossibility of finding a deterministic controller.

Theorem 3. *For a given environment and robotic system in Section 6.2, there exists no deterministic control policy that can guarantee gathering in 2D.*

Proof. We reduced the domain of possible controllers, starting from 331776 to 6 using 35 counter-examples. Hereby, we present an additional counter-example in below.

Counter-example 36	
***A Priori* Known Contexts**	A, C, D1, E1, E3, F1, F2, G, H1, H2, H3 I1, I2, I3, J, K
***A Priori* Unknown Contexts**	B, D2, E2
Unknown Contexts Participated	N/A
$\lvert\bigsqcup\mathcal{T}\rvert$ **Update**	$6 \rightarrow 0$
Reference in Appendix	Figure A.57-A.79
Type of Counter-Example	Dynamic deadlock with period of 360



This counter-example has a special meaning. It *contradicts* with all of the previously discovered contexts. The participated contexts are E3, H1, H2, H3, I3, K. Even though we discovered them previously, they failed in gathering the group of modules, yielding a contradiction with Lemma 5. Therefore, there can exist no deterministic control policy $\mathcal{P}_D$ utilising $\mathcal{T}$ that is complete. ■

Theorem 4. *There exists a deterministic control policy $\mathcal{P}_D$ that can guarantee gathering in $1D$.*

Proof. In the first case, if $\eta = 0$, i.e. if there is no hole in the bounding box, we are done. Let's assume there is $\eta \geq 1$ holes in the centre of the bounding box of the modules. In a $1D$ grid environment, only the contexts F1, I2, J, and K can be defined. From Axiom 2 and 3, recall that $a_{\mathsf{K}} = a_0$, and $a_{\mathsf{F1}} = a_0$. In this proof, we assume that $a_{\mathsf{I2}} = a_1$ and $a_{\mathsf{J}} = a_1$. Figure 6.8 shows the contexts in an example configuration.

As $a_{\mathsf{F1}} = a_0$, a central module cannot increase or decrease η. Similarly, a module on the edge—context K, cannot increase the bounding box. On the other hand, actions

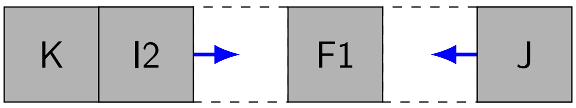


Fig. 6.8 An example illustration of four possible contexts: F1, I2, J, and K.

of both I2 and J are towards the centre of the configuration. After the movement of I2, a context K becomes J, thus contributes to decreasing η. The only module that initially cannot move is F1, however, when another module moves to the adjacent cell, F1 becomes I2. The movement of both I2 and J reduce η by 1, hence, $\eta[k+1] = \eta[k] - 1$. At any moment, either η is decreasing by the actions of I2 and J, or remaining the same (after the updates of K and F1). Thus, after a round of updates for each module independent of the order; $\eta[k+1] \leq \eta[k]$. Therefore, it is guaranteed that gathering will be achieved after a finite round of updates. ■

When a deadlock occurs, it becomes theoretically impossible to gather the modules. We believe that a 'deadlock' situation can be avoided if certain level of stochasticity introduced to the system. In order to test our hypothesis, in the next section, we design a stochastic control policy. We then present mathematical analysis on the completeness of gathering.

6.4 Naïve Stochastic Control Policy

In this section, we propose a 'naïve' stochastic control policy. Algorithm 4 describes the naïve stochastic control policy, hereafter referred to as $\mathcal{P}_N$.

In each control cycle, the agent chooses uniformly random from the following set of actions: $a_0 \cup \{a_i \in \mathcal{A} | i \in \{1,2,3,4\} \wedge (\neg c_i \wedge v_i)\}$. In other words, the agent can rest in place, but may move in up to four directions (see Figure 6.3). It is prevented from moving into a direction that is blocked by an adjacent agent (i.e. $c_i = \text{true}$), or in which no other agent is seen (i.e. $v_i = \text{false}$). The control cycle is here assumed to have some finite length δ. Note that Algorithm 4 is fully reactive, as the agent does not store any information from the previous cycle.

Compared to the deterministic control policy $\mathcal{P}_D$, the stochastic control policy $\mathcal{P}_N$ is less detailed. There is no concept of a context, rather, every action is solely determined by uniform random choice over the eligible actions. There is also no definition of a controller, however, one can interpret the uniform random choice is a controller. In the

Algorithm 4 Naïve Stochastic Control Policy, $\mathcal{P}_N$

```
while true do
    $\mathcal{A}_e \leftarrow \{a_0\}$                      ▷ initialise set of eligible actions
    for all $i \in \{1, 2, 3, 4\}$ do
        update $c_i$                            ▷ probe Boolean contact sensor $i$
        update $v_i$                            ▷ probe Boolean visibility sensor $i$
        if $\neg c_i \wedge v_i$ then
            $\mathcal{A}_e \leftarrow \mathcal{A}_e \cup \{a_i\}$           ▷ add eligible action $\{a_i\}$
        end if
    end for
    $a \leftarrow$ select uniformly random from $\mathcal{A}_e$
    execute $a$
    wait $\delta$ units of time
end while
```

next section, we will formally prove that the group of modules using $\mathcal{P}_N$ can almost surely reach a Pareto optimal configuration.

6.4.1 Mathematical Analysis

Lemma 6. *Consider n agents using policy $\mathcal{P}_N$. Let $\mathcal{C}[k]$ and $\mathcal{C}[k+1]$ denote the configurations at time steps k and $k+1$, respectively, which is immediately before and after one of the agents was considered. Then, $b_x[k+1] \leq b_x[k]$ and $b_y[k+1] \leq b_y[k]$.*

Proof. At time step k only one agent, say agent j_1, was considered. All other agents will not have moved, that is, $\forall j_2 \neq j_1 : x_{j_2}[k+1] = x_{j_2}[k]$ and $y_{j_2}[k+1] = y_{j_2}[k]$. The x-coordinate of the "leftmost" agent at time k is given as $x_{\text{left}} = \min_{j_2}\{x_{j_2}[k]\}$. If agent j_1 was at the left boundary ($x_{j_1}[k] = x_{\text{left}}$), no agent would have been visible towards the "left" ($v =$ false), which would prevent the agent from moving in that direction. Otherwise ($x_{j_1}[k] > x_{\text{left}}$), agent j_1 may have moved, but at most by 1 cell. In both cases, we have $x_{j_1}[k+1] \geq x_{\text{left}}$. The same argument can be used for the lower, right and upper boundaries. From this, it follow that $b_x[k+1] \leq b_x[k]$ and $b_y[k+1] \leq b_y[k]$. ■

Corollary 1. *Consider n agents using policy $\mathcal{P}_N$. Let $\mathcal{C}[k]$ denote the configuration at time step k. Then, $\forall l > k : b_x[l] \leq b_x[k]$ and $b_y[l] \leq b_y[k]$.*

Theorem 5. *Using policy $\mathcal{P}_N$, n agents almost surely reach a Pareto optimal configuration in finite time.*

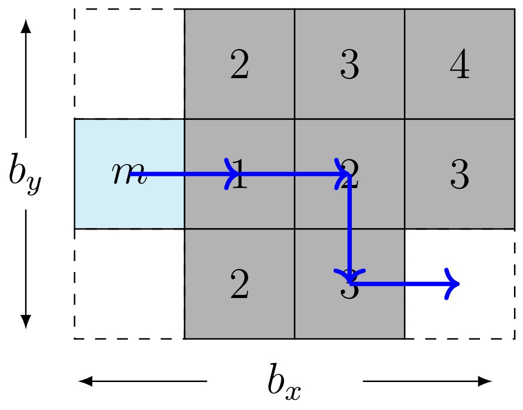


Fig. 6.9 Example configuration of 9 agents with a 4×3 bounding box. The configuration is not Pareto optimal, as the same number of agents could be contained in a 3×3 bounding box. The numbers indicate the Manhattan distance between the corresponding cell and the reference agent, m, of the first column. The blue arcs illustrate one of the shortest paths from the reference agent to the empty cell in the last column. If the four agents on this path choose to move in the indicated direction, whereas all other agents choose not to move, the empty cell is pushed into the first column, causing the new configuration to be Pareto optimal.

Proof. Let for all $k \geq 0$, $\eta[k] = b_x[k]b_y[k] - n$ denote the number empty cells within the bounding box at time step k. We prove the theorem by induction.

Base case: $\eta[k] = 0$. As $\eta[k] = b_x[k]b_y[k] - n = 0 < \min\{b_x, b_y\}$, from Lemma 4 it follows that the configuration is Pareto optimal. From Corollary 1 it follows that the configuration remains Pareto optimal indefinitely.

Induction step: $\eta[k] > 0$. Without loss of generality, we assume $b_y[k] \leq b_x[k]$. Moreover, $b_x[k] > 1$, as otherwise, $\eta[k] = 0$. If $\eta[k] = b_x[k]b_y[k] - n < b_y[k] = \min\{b_x[k], b_y[k]\}$, then it follows from Lemma 4 and Corollary 1 that the configuration is Pareto optimal and remains so indefinitely. For the case $\eta[k] \geq b_y[k]$, as in the proof of Lemma 4, we consider that the agents of the leftmost column all relocate to the other columns (see Figure 6.9). We obtain a positive lower bound for the probability for this to happen in constant time. We assume that every round one agent chooses to move, while all others choose not to move. The probability for this to happen in a given round is at least ϵ^n, where $\epsilon = \frac{1}{5}$ is a lower bound for the probability for any eligible action to be chosen (note that actions are chosen uniformly random). Let us consider the shortest path from an arbitrary agent of the leftmost column to an empty cell in one of the other columns (see Figure 6.9). The length of path candidates are determined by the Manhattan distance, and thus reflecting how the agents may move. Multiple shortest paths may exist, but in any case only the last cell of a path is an empty cell. At any round let only the agent that is closest to the empty cell (but part of the remaining path) move. At most d rounds are required for the empty cell to reach the leftmost column, where d is the length of the shortest path. As d is bounded by $b_x[k] + b_y[k] - 2$, the probability for the agent relocation to have occurred after $b_x[k] + b_y[k] - 2$ rounds is at least $\epsilon^{(b_x[k]+b_y[k]-2)n}$. As there could be up to $b_y[k]$ agents in the leftmost column, the probability to reach the preferred configuration after $U = b_y[k](b_x[k] + b_y[k] - 2)$ rounds is at least $p = \epsilon^{b_y[k](b_x[k]+b_y[k]-2)n}$. From Lemma 6, it follows that the bounding box dimensions, b_x and b_y, are monotonically decreasing with time, k. In other words, our lower bound, p, monotonically increases with time, k, and the number of rounds, U, required for an improvement to occur with at least probability p, monotonically decreases with time, k. If an improvement occurred, the new configuration would have at least $b_y \geq 1$ fewer empty cells, resulting in $\eta[k + Un] \leq \eta[k] - b_y[k] \leq \eta[k] - 1$. The probability that an improved (preferred) configuration is found within τU rounds is at least $p_\tau = 1 - (1 - p)^\tau$. We have $\lim_{\tau\to\infty} p_\tau = 1$. In other words, a preferred configuration is found almost surely in finite time, reducing η by at least 1. As $\forall k : \eta[k] \geq 0$, only a finite number of improvements are possible. A Pareto optimal configuration is hence obtained almost surely in finite time. ■

Remark. *Time required to reach a Pareto optimal configuration using policy $\mathcal{P}_N$ can be arbitrarily large.*

6.5 Optimised Stochastic Control Policy

In the previous section, we showed that a group of agents almost surely reach a Pareto optimal configuration in finite time when using the naïve stochastic control policy, $\mathcal{P}_N$. The policy determines the set of eligible actions and then chooses uniformly random from this set.

In this section, we consider an alternative stochastic control policy, $\mathcal{P}_O$, which is not restricted to using uniform distributions, but rather takes into account an agent's *context* (see Figure 6.3.1). Depending on the context, an agent can choose between 1 and 5 actions (note that an agent can always choose the option to rest, that is, a_0). An agent in context A can either remain in its current position (action a_0) or move into any direction (a_1, a_2, a_3, a_4). As the agent has no orientation, it has to choose either direction with equal probability. An agent in context B has four possible options to choose from, and, due to chirality, for each option a dedicated probability can be chosen. In the following, we optimise the probabilities of choosing the eligible actions for each specific context. Note that, the Axiom 2 is still valid for this section, however, Axiom 2 and Axiom 2 are now relaxed. Thus, there is no assumption on the action of the contexts $\mathsf{A}, \mathsf{D1}, \mathsf{F1},$ and $\mathsf{F2}$.

6.5.1 Representation of Candidate Solutions

A candidate solution is represented by 27 real-valued parameters in range $[0, 1]$, which, following normalization, determine the motion probabilities. For example, for context A, all five actions are possible. However, a single parameter is sufficient, as the probabilities, p_1, p_2, p_3, and p_4, for choosing actions a_1, a_2, a_3, and a_4 must be identical (due to the lack of orientation). Moreover, the probability of choosing a_0 must be $p_0 = 1 - p_1 - p_2 - p_3 - p_4$.

6.5.2 Evolutionary Algorithm

We employ an evolutionary algorithm for the optimisation process, namely Covariance Matrix Adaptation Evolution Strategy (CMA-ES) Hansen et al. (2019). CMA-ES is a derivation-free, black-box optimisation method. It starts with a random population of λ candidate solutions, and uses a fitness function to select promising solutions for producing the subsequent generations.

Definition 10. *Compactness, denoted as $\mathcal{H}$, of a configuration is the number of empty cells in the corresponding bounding box, if the configuration is not Pareto optimal, and 0, otherwise. Formally, compactness $\mathcal{H}$ at round r can be given as,*

$$\mathcal{H}[r] = \begin{cases} \eta[r], & \textit{if } \eta[r] \geq \min\{b_x[r], b_y[r]\}; \\ 0, & \textit{otherwise.} \end{cases} \tag{6.5}$$

In each generation, every candidate solution is tested against the same set of $T = 20$ starting configurations. A new set of configurations is produced at the beginning of every generation. For each configuration, the number of agents is chosen as $n = 2 + m$, where m is generated randomly using the exponential cumulative distribution function given by,

$$P(m;\lambda) = \begin{cases} 1 - e^{-\lambda m}, & \text{if } m \geq 0; \\ 0, & \text{otherwise,} \end{cases} \tag{6.6}$$

where $\frac{1}{\lambda} = 6$ represents the expected value, $E[m]$. Hence, the expected number of agents is $E[n] = 2 + E[m] = 8$. The agents are all placed in random positions within a grid size of $L = 2\left\lceil \sqrt{n} \right\rceil$ and are assigned a randomly generated *fixed* update order.

The *fitness* function to be minimised by the evolutionary algorithm is,

$$F = \sum_{r=1}^{R} r\mathcal{H}[r], \tag{6.7}$$

where $R = 100$ is the maximum number of rounds. The performance measure $\mathcal{H}[r]$ is multiplied by round number r to promote faster solutions.

6.5.3 Controller Selection

The source code was written by the author, incuding simulation setup, environment model, robotic model (sensing and actuation), counter-example algorithms. Implementation of CMA-ES is different than Chapter 3, 4, and 5, using a Python package instead of C++ library. Batch-job submission, visualiser (renderer), and data analysis source code was written by the author.

We conducted 100 evolutionary runs with a population size of $\lambda = 30$.[6] Each run was terminated after 3000 generations. Figure 6.10 shows the fitness of the highest-rated candidate solution per run.

To select the *best* solution, we considered the solutions that exhibited the lowest fitness value in the final generation of each run. Each of these 100 solutions was post-evaluated on $T = 200$ random configurations using a grid size of $L = 100$ and $R = 10000$ rounds. In the following, we refer to the solution that exhibited the best mean performance as the optimised stochastic control policy, $\mathcal{P}_O$.

Table 6.5 shows the optimised probability distributions for each context. The context K is omitted, as it is not part of the optimisation process. N/A in probability parameter p_i indicates the impossiblity of choosing the action a_i due to contact with

[6]As in Section 3.3.2, we follow the same advise to choose the population size.

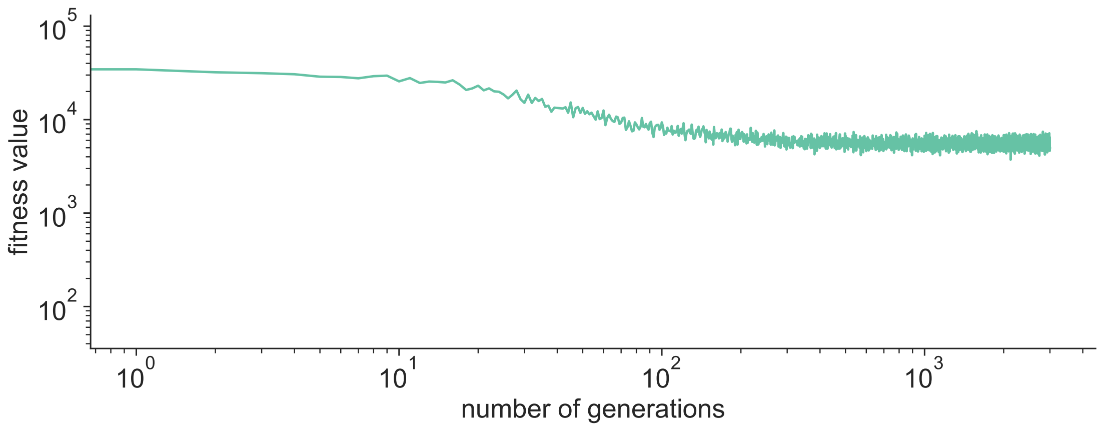


Fig. 6.10 Fitness dynamics showing the performance of 100 evolutionary runs over 3000 generations. The green envelope represents the minimum and maximum average fitness values of $\lambda = 30$ solutions for each run, and the green solid line represents the median fitness values. Both axes are in logarithmic scale.

another module (i.e. $s_i = 2$). Hence, the probability parameter, $p_i = 0$ for the action a_i.

Theorem 6. *Using policy $\mathcal{P}_O$, n agents almost surely reach a Pareto optimal configuration in finite time.*

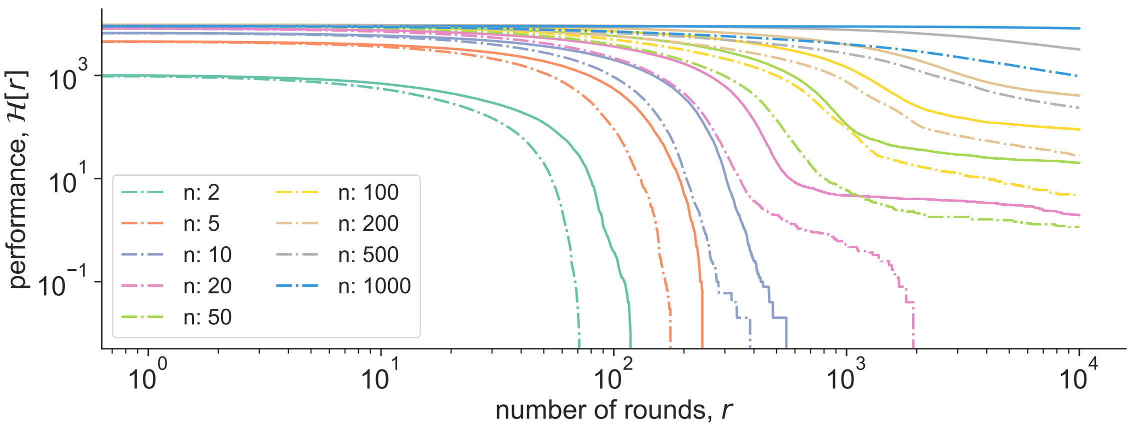
Proof. We choose ϵ to be the lowest probability of the $\mathcal{P}_O$ controller. If $\epsilon \neq 0$, then the proof for Theorem 5 for $\mathcal{P}_N$ naturally extends to $\mathcal{P}_O$. The lowest probability value, $\epsilon = \min\{p_i\} = 0.000015$. Therefore, a Pareto optimal configuration is obtained almost surely in finite time. ■

6.6 Simulation Studies

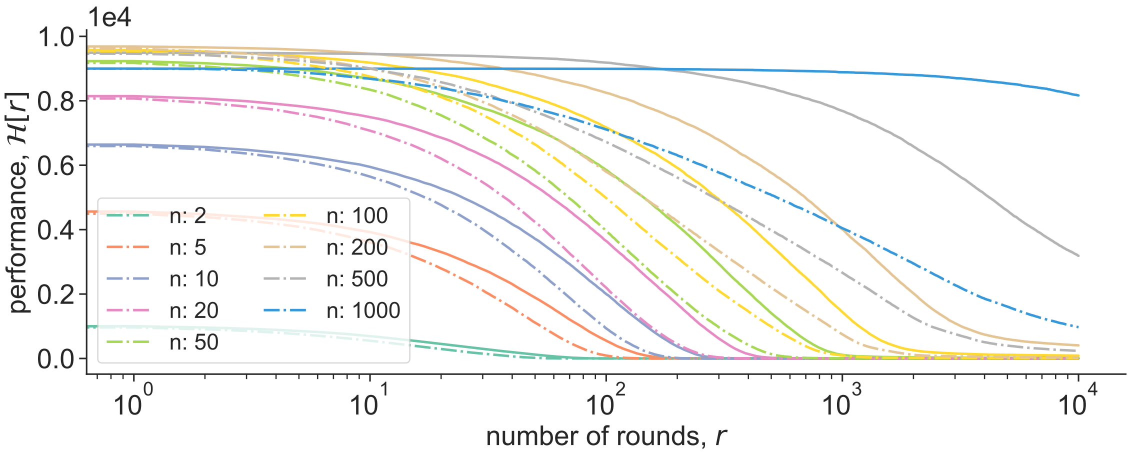
In this section, we analyse the performance of the control policies, $\mathcal{P}_N$ and $\mathcal{P}_O$, through computer simulations using measure $\mathcal{H}[r]$, as defined in Equation (6.5).

6.6.1 Scalability Analysis

We investigate the scalability of the control policies. We consider a 2D square tile environment of size $L = 100$ containing $n \in \mathcal{N} = \{2, 5, 10, 20, 50, 100, 200, 500, 1000\}$ agents. In the beginning of a simulation trial, the agents are uniformly randomly



(a)



(b)

Fig. 6.11 Results of the scalability study with up to 1000 agents in a 100×100 environment. Dashed and solid lines show, respectively, the performance using the naïve ($\mathcal{P}_N$) and optimised ($\mathcal{P}_O$) control policies. Each line represents the average, across 100 trials, of $\mathcal{H}[r]$, which is 0 for Pareto optimal configurations, and otherwise equals the number of empty cells in the bounding box. (a) and (b) plot the compactness using logarithmic and linear axes, respectively.

Table 6.5 The optimised control parameters for each context.

c	p_0	p_1	p_2	p_3	p_4
A	0.992677	0.001831	0.001831	0.001831	0.001831
B	0.941329	0.000070	0.044790	0.013810	N/A
C	0.664613	0.001601	0.333195	0.000591	N/A
D1	0.008827	N/A	0.495587	N/A	0.495587
D2	0.996154	0.000463	0.003383	N/A	N/A
E1	0.043784	N/A	0.465886	N/A	0.490330
E2	0.165319	0.499262	0.335420	N/A	N/A
E3	0.496369	0.004311	0.499320	N/A	N/A
F1	0.003692	N/A	0.498154	N/A	0.498154
F2	0.002682	0.499544	0.497773	N/A	N/A
G	0.998558	0.001442	N/A	N/A	N/A
H1	0.556494	0.443506	N/A	N/A	N/A
H2	0.009498	0.990502	N/A	N/A	N/A
H3	0.999902	0.000098	N/A	N/A	N/A
I1	0.000015	0.999985	N/A	N/A	N/A
I2	0.373871	0.626129	N/A	N/A	N/A
I3	0.984537	0.015463	N/A	N/A	N/A
J	0.005356	0.994644	N/A	N/A	N/A

placed. For each $n \in \mathcal{N}$ and each control policy ($\mathcal{P}_N$ and $\mathcal{P}_O$), 100 trials of $R = 10000$ rounds are performed.

Figure 6.11 reports the average performance over the number of rounds. Groups of $n \in \{2, 5, 10\}$ agents consistently reached a Pareto optimal configuration using either of the control policies. For these group sizes, the optimised controller, $\mathcal{P}_O$, took on average 39.50%, 26.97% and 30.07% less time to reach a Pareto optimal configuration. For larger group sizes, it became increasingly unlikely for the agents to reach a Pareto optimal configuration in the provided time period. However, for groups of $n = 1000$ agents, the compactness, $\mathcal{H}[r]$, still improved during the trial, on average by 9.26% and 89.20% for $\mathcal{P}_N$ and $\mathcal{P}_O$, respectively. One can find scalability trial videos in supplementary materials (Özdemir et al., 2019c).

6.6.2 Sensory Noise Analysis

Our analysis in Section 6.4 assumed the absence of sensory noise. As a consequence, it could be shown that the modules' bounding box dimensions were monotonically decreasing with time. We now investigate the effect of sensory noise on the performance of both

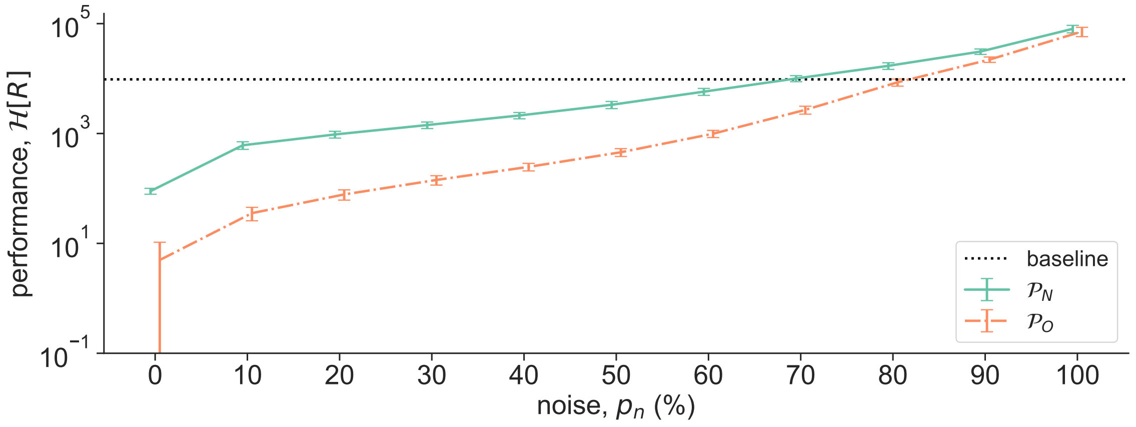


Fig. 6.12 Effect of sensory noise on the performance at the end of 100 simulation trials ($\mathcal{H}[R]$, defined in Equation (6.5), in logarithmic scale). With probability p_n, each sensor unit provides a purely random reading value. Dashed and solid lines represent, respectively, the naïve ($\mathcal{P}_N$) and optimised ($\mathcal{P}_O$) control policies. The error bars show the standard deviation. The dotted line represents a baseline which corresponds to no movement ($\mathcal{H}[0]$).

control policies. As mentioned in Section 6.2.1, each of the module's four sensor units provides one ternary digit of information: $s \in \{(\text{false}, \text{false}), (\text{false}, \text{true}), (\text{true}, \text{true})\}$. In the following, we assume that each sensor unit reports a uniformly randomly chosen ternary digit with probability p_n, and reports the original reading value otherwise. We increase the noise-level from 0% to 100% by 10% increments. For each level of noise and control policy, 100 trials are performed with $n = 100$ agents in a 100×100 grid environment. When a module decides to move onto an occupied cell, no action is taken for the corresponding round. Each trial is run for a constant duration of $R = 10000$ rounds.

Figure 6.12 shows the performance measure $\mathcal{H}[r]$ for all levels of noise and both policies. For a 10% noise-level, the compactness, $\mathcal{H}[r]$, improved during the trial, on average by 93.68% and 99.63% for $\mathcal{P}_N$ and $\mathcal{P}_O$, respectively. The system improved its compactness by at least 90% for any noise level up to 20% and 50% for $\mathcal{P}_N$ and $\mathcal{P}_O$, respectively. For a noise-level of 100%, implying purely random sensor readings, the system diverged.

6.6.3 Comparison with Deterministic Policy

In this section, we to compare the control policies against multiple other variations. The first variation is a hybrid deterministic control policy; we take the action with

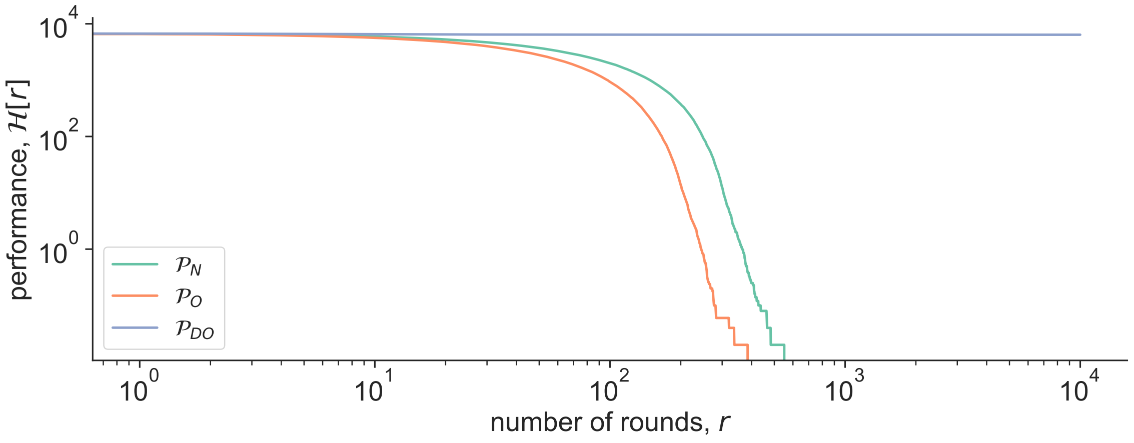


Fig. 6.13 The comparison of naïve (green), optimised (orange), and deterministic-optimised (lilac) control policies for $n = 10$ agents. Each line represents the average, across 100 trials, of $\mathcal{H}[r]$. Both axes are in logarithmic scale.

the highest probability value for each context in Table 6.5 as a deterministic action.[7] Formally,

$$a_{\mathsf{c}} = \operatorname{argmax}\{p_0, p_1, p_2, p_3, p_4\}, \tag{6.8}$$

where a_{c} is the action for the context c. For instance, the highest probability for context B is $p_0 = 0.941329$, thus the action is a_0. We refer to this hybrid control policy as deterministic-optimised and denote by $\mathcal{P}_{DO}$.

We performed 100 trials in a grid size of $L = 100$ for $n = 10$ robots using $\mathcal{P}_N$, $\mathcal{P}_O$, and $\mathcal{P}_{DO}$. Figure 6.13 shows the results. The lilac line is the average performance of $\mathcal{P}_{DO}$. As it is clearly seen, there is no improvement on the bounding box by the agents using $\mathcal{P}_{DO}$. The reason is the optimised controller was optimised for the "stochastic" scenario, where it can choose between multiple options with certain probability. Whilst our assumption for $\mathcal{P}_{DO}$ is valid for the high probability actions, it becomes problematic for probability values close to 0.5. For instance, context $\mathsf{E1}$ chooses the action a_4 (as $p_4 = 0.49$), even though the action a_2 has probability $p_2 = 0.47$. As a result of the accumulative deterministic decisions, the agents remain stuck in deadlock situations where no improvement is possible.

Recall that in Table 6.3 we presented six deterministic controllers before we concluded that none of them could solve the problem in general. Nevertheless, we can still measure their average performance, and how fast it converges until a deadlock

[7]Recall that in Section 6.3.4 we show that there is no deterministic controller can solve the given gathering problem in general, though, local solutions may exist.

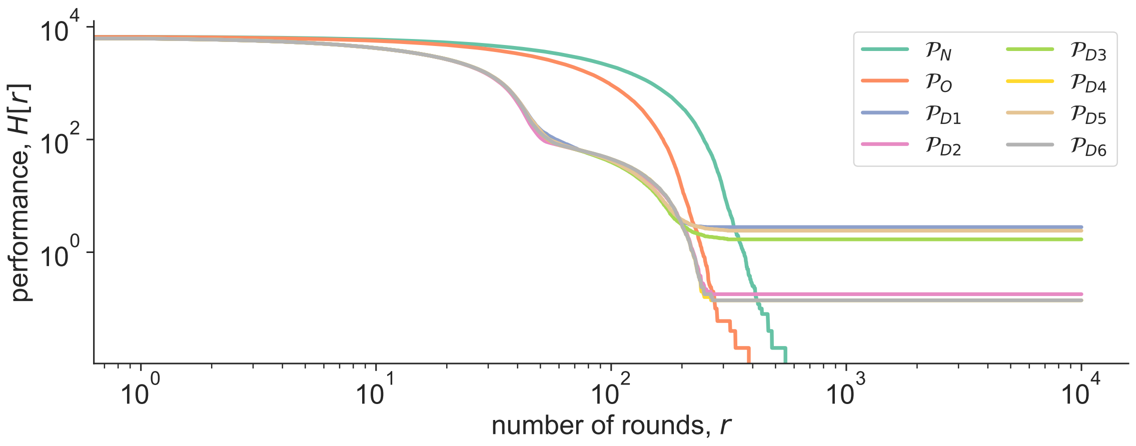


Fig. 6.14 The comparison of naïve (green), optimised (orange), and six deterministic control policies for $n = 10$ agents. Each line represents the average, across 100 trials, of $\mathcal{H}[r]$. Both axes are in logarithmic scale.

situation happens.[8] The six deterministic control policies are referred as $\mathcal{P}_D$ with an additional subscript to refer the number, e.g. $\mathcal{P}_{D1}$.

Figure 6.14 shows the average simulation results of 100 trials for $n = 10$ agents using eight different control policies; $\mathcal{P}_N$, $\mathcal{P}_O$, $\mathcal{P}_{D1}$ ... $\mathcal{P}_{D6}$. Compared to the hybrid policy $\mathcal{P}_{DO}$, the deterministic control policies, $\mathcal{P}_D$s, achieved a much better performance. Every one of the six $\mathcal{P}_D$ improved the compactness significantly, and they did faster than the stochastic policies as expected. However, as they could not avoid deadlock situations in some of the trials, none of them fully succeeded on average. As they share common actions (16 out of 19 actions are the same) and they are deterministic, we observe the same trend. An interesting effect, however, occurs after $r = 200$ rounds, where the performance of the deterministic policies branch out. After this point, they show two separate characteristics—each with three controllers.

In Figure 6.14 we show how six deterministic control policies perform compared to two stochastic controllers for $n = 10$ agents. However, we did not see how they scale. As all six controllers follow the same trend, we arbitrarily picked the first one, $\mathcal{P}_{D1}$, and performed a scalability analysis for $n \in \{2, 5, 10, 20, 50, 100, 200, 500, 1000\}$.

Figure 6.15 shows the result of the scalability study for $\mathcal{P}_O$ and $\mathcal{P}_{D1}$. One can see that for the low number of agents (i.e. $n = 2, 5$), the deterministic policy has full success. Moreover, the number of rounds to take to gather is faster than the optimised

[8]In general, deadlocks happen in a relatively small area, after the agents improve the bounding box.

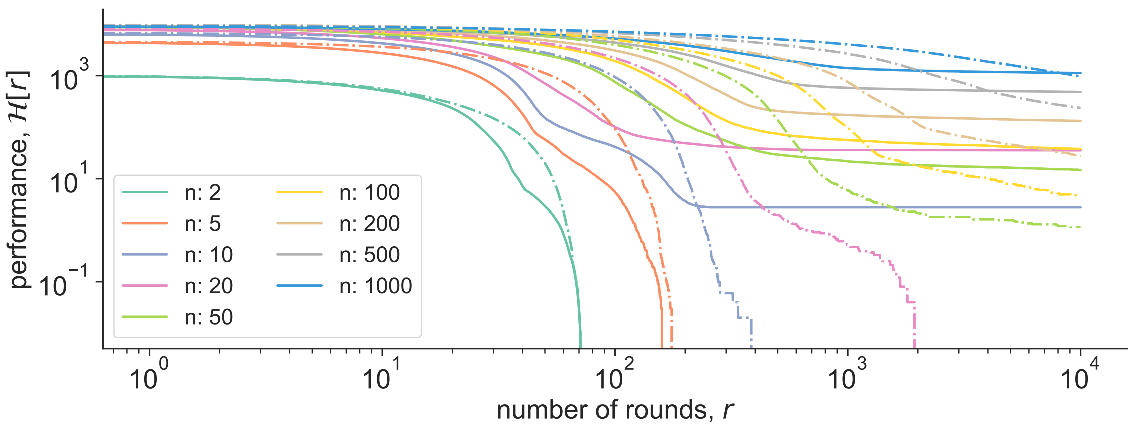


Fig. 6.15 The comparison of $\mathcal{P}_O$ (dashed), and $\mathcal{P}_{D1}$ in a scalability study with up to 1000 agents in a 100×100 environment. Each line represents the average, across 100 trials, of $\mathcal{H}[r]$. Both axes are in logarithmic scale.

stochastic control policy. Additionally, one can see that the deterministic control policy improves the bounding box significantly faster than the stochastic policy (for every other number of agent settings). However, it eventually reaches a deadlock situation and cannot improve any further.

6.7 Discussions

This chapter introduced an extended version of the computation-free swarming framework: (i) multiple sensor readings were combined in a reactive manner, (ii) stochasticity introduced to the system in terms of probability distributions. The first extension allowed the robots to perform more precise actions by enabling it to retrieve more information from its environment. The agents used binary sensors that indicate whether other modules are physically in contact or otherwise visible. It has been applied to design a deterministic control policy for gathering oblivious, embodied agents in a 2D square tile environment. We proved by aided exhaustive computer simulations that the gathering problem is unsolvable with current deterministic framework.

The second extension was necessary to overcome observed deadlock situations. We then proposed two variations of a stochastic control policies: naïve and optimised. Naïve stochastic control policy was proposed, considering the uniform probability distribution for the agents to choose from a set of eligible actions. The agents were mathematically proven to assume a Pareto optimal spatial arrangement almost surely in finite time. Optimised stochastic control policy was proposed. It took the unique context of an agent into account, and optimised a probability distribution for each contexts. The mathematical proof was naturally extended for the latter control policy. Computer simulation were examined the performance for different number of agents, or in the presence of sensory noise using either of the stochastic control policies. In addition, the policies' performance were compared with each other.

The next chapter will conclude the thesis by summarising the works and presenting future directions.

Chapter 7

Conclusions and Future Work

In this thesis, we synthesised multiple control strategies for swarms of minimal information processing robots. The robots required only discrete sensor readings, no run-time memory, and all executed an identical controller. We followed an evolutionary robotics approach and "evolved" the swarms' behavioural rules based on group level objective function. The control solutions we proposed are the simplest solution to date, making it applicable to robots that lack arithmetic computation units and memory storage. The swarms' collective behaviours emerged from simple local interactions and produced solutions to four challenging tasks. We then analysed the performance of the swarms by changing the number of individuals within the groups, by modifying their environment, and by introducing noise in their sensor units. Additionally, two of the control strategies (for spatial coverage and finding consensus) were ported onto a swarm of physical e-puck robots (Mondada et al., 2009).

In the first study, we investigated how a swarm of simple robots can cooperatively cover an unknown environment without using run-time memory, or sharing information. The robots used only a single-bit of discrete sensor reading, whether another robot was present in their direct line of sight or not. Using computer simulations, we showed that our control strategy can outperform a random walk strategy that was obtained through the same optimisation setup. Further simulation studies revealed that using the same control strategy, the swarm can navigate a maze provided by a constant-rate inflow of robots up to a certain critical rate. We then ported the controller into 25 e-puck robots, operating in an identical physical environment setup, to demonstrate the feasibility of the computation-free coverage controller. The performance of the physical swarm was fairly similar to the simulated robots, dropping on average only 6% and 11% (depending on the performance measure used).

In the second study, we explored the consensus finding abilities of swarms of extremely simple robots. Solely based on ternary discrete sensory input, the majority of the swarm was able to commit to either of the options present in the environment in 97.3% of the trials. At the expense of a longer sensing range, the robotic swarms required significantly lower information processing capabilities compared to the previously proposed control strategies. We also investigated the capabilities of the strategy in more challenging situations, where either more than two options were present or false-negative noise affected their sensors when they perceived an option. We also examined the control strategy in choosing between unequal alternatives. The quality of an option was represented by the size of the physical objects, hence, the higher quality object was larger, providing a stronger stimulus. As expected, the computer simulations showed that the swarm tended to choose the larger of the two options. We tested the control strategy using a swarm of 20 physical e-puck robots in 50 experimental trials. Even though some robots ceased motion during operation, the swarm succeeded in committing to an option in 48 of the trials.

In the third study, we examined the shepherding task in which the shepherds—the controlled swarm—cage and herd another 'sheep-like' group of robots. The shepherds used only a single line-of-sight sensor to detect the presence of a sheep, another shepherd, or the goal location. We observed that, although the shepherds have constant motor commands, the strategy allowed the shepherds to display three distinct behaviours; (i) caging sheep, (ii) herding sheep towards the goal location, and (iii) keeping sheep in the goal. Computer simulation studies showed that the shepherds were robust against sensory noise and scalable up to a certain extent. When the number of sheep in the environment was high, the goal became occluded, as a result, the shepherds could cage the sheep but could not herd them. To alleviate this problem, we proposed an extended version of our controller. We added another line-of-sight sensor dedicated to perceiving only the goal. We evolved this extended controller in the same setup and tested in environments with high numbers of sheep. The enhanced version of the computation-free controller yielded better scalability for the large number of sheep and shepherds. The findings regarding the extended controller showed that in challenging situations, such as overcrowded environments, the limited sensing ability can be overcome by including an additional dedicated sensor.

In the fourth study, contrastingly to the previous three studies, we modelled and deployed a modular robotic system operating in a 2D square grid environment to explore the gathering task. We investigated deterministic and stochastic gathering strategies for oblivious modular robots requiring only a contact and a visibility sensor on each of their faces. The modules had chirality but no sense of orientation, meaning that they could only sense the difference between clockwise and counter-clockwise directions. First, we proved, aided by exhaustive computer simulations, that there exists no deterministic control policy for the gathering problem with aforementioned desiderata. This indicated that stochasticity was required to be present in the system. Therefore, we extended the computation-free swarming framework and proposed two stochastic control policies: naïve and optimised. Naïve stochastic control policy allowed the modules to uniformly randomly choose an action from a set of eligible actions. In the latter case, optimised stochastic control policy considered the configuration of a module and provided an optimised probability distribution for action selection. We mathematically proved that the modules reach a Pareto optimal configuration almost surely in finite time using either of the control policies. Computer simulations showed that, for up to 1000 robots, the optimised stochastic control policy gathers the modules faster than the naïve policy. The system also tested for sensory noise, and it improved compactness up to 20% and 50% of noise level, respectively, however, the system diverged when there was 100% noise meaning that the sensor readings were purely random.

The controllers we proposed in this thesis are the simplest solutions for their own tasks. They provide a baseline against which one can quantify the performance improvements that more advanced and expensive techniques may offer. One of the promising directions of the minimalist strategies is enabling simplistic robotic swarms to be deployed in nanomedical applications. In the future, with the aid of recent technological advances, swarms of simplistic robots can be miniaturised to micro- or nano-scale. In the next section, we list future directions for the work presented in this thesis to be explored further.

7.1 Future Work

In this thesis, we presented minimalist control strategies for a range of bio-inspired tasks for swarms of robots. In three of the four studies, we used differential-drive e-puck robots for proof-of-concept robotic implementation. Transferring the control strategy onto the physical platform required processing information extracted by the robots' camera. This was necessary for the e-puck robot to emulate the line-of-sight sensor. In the future, there could be bespoke robotic systems that consider both the controller and the physical hardware design. The future systems could make benefit of fully integrated 'sense-and-act' paradigm and reduce complexities of porting the controller. In the future, one could address this by studying physical swarm robotics platforms with reduced hardware complexity (Li et al., 2017; Requicha, 2003; Sitti, 2017), which could benefit from our low-capability control strategy.

Restrictions in the control strategy are necessary when the environment is constrained to a small scale—for example in a nanomedical scenario. While producing a synthetic nanomachine is still a challenge, researchers have been able to synthesise chemical and biological nanomachines (Cerofolini and Amato, 2013; Requicha, 2013). This was led to increasing interest for controlling the nanomachines to perform a multitude of tasks. One particular area of interest is detecting and removing pathogens from the body using nanorobots. We believe that there are similarities between this operation and the shepherding problem we presented. The nanorobots—similar to the shepherds—can identify, approach, attach, and guide pathogens to an outlet. In the current state, we do not expect the control solutions to work straight away in a nanorobotic platform. Although, after suitable modelling of the nanomachines, the pathogens, and the body, one should be able to synthesise a bespoke control solution.

Minimal information acquisition results in less energy or power consumption compared to processing more detailed information. Thus, future studies could take energy consumption of the robots into account when designing the strategies. In addition, the minimal control strategies can be seen of as last resort solutions. The robots can be hard-coded to switch to the minimal controller in cases of malfunctioning. For example, in the case of their power sources running out of energy, the robots, by switching to a minimalist energy efficient control strategy, could still gather at a recharge point.

In Chapters 3 and 4, we demonstrated the control strategies using physical e-puck robots. The robots were equipped with RGB CMOS cameras, and we emulated a line-of-sight sensor by selecting a few pixels in the centre of the camera frame. We acknowledged that the proof-of-concept implementation of the control strategies required image preprocessing, however, the controller logic remained free of arithmetic computation. In principle, a line-of-sight sensor does not require arithmetic computation, for example, one can use a single-pixel RGB CMOS sensor with hardwired logic to return one out of N-values—one per sensor state—based on predefined thresholds. It is also possible to build bespoke sensors that directly function as a line-of-sight sensor, and it is not required to be an optical sensor. Chemical or touch sensors can be considered and might enable the control solution to be ported onto sub-millimeter scale robots. This can then lead to implementation of a computation-free controller for a robot that does not feature arithmetic logic unit.

For physical implementation of spatial coverage controller, one needs only a binary line-of sight sensor to detect the presence of a robot. For the decision making controller, one needs a ternary line-of-sight sensor which can also distinguish between a robot and an option. For the shepherding controller, a sensor that can distinguish between three objects—a robot, a sheep and the goal target—is required. One can use two different sensors, if necessary, and hardwire them to output a single value. For the gathering controller, using two different sensors is essential, as each robot is required to detect another robot in distance (e.g. using a binary vision sensor) and a robot that is in contact (e.g. a binary touch sensor).

Another future direction can include further testing gathering strategies on physical robotics platforms (Özdemir et al., 2019b; Romanishin et al., 2015) to assess the feasibility of the strategies and compare them. We proved that there exists no deterministic control policy in 2D to guarantee a complete solution, there could be, however, deterministic control policies that guarantee solutions in restricted subsets of the environment. We hypothesise that a deterministic control policy would provide a faster solution, as unlike a stochastic policy, it has a certain action to take. Thus, one could explore designing a deterministic control policy for a bounded environment. Another future direction would be further studying gathering for predefined ensemble shapes in two scenarios. First, the modules in a grid can gather into a certain area and create a 3D assembled structures—for example a chair or a desk. Second, after the modules assembled, they could collectively move into a direction of interest.

A limitation of minimalist control strategies is that the swarms of robots perform only one particular task—the one that the controllers were designed for. As an example, a spatial coverage strategy could not directly be used for finding consensus, as the robots require an additional sensor state to detect an option. However, multiple swarms of simplistic robots could be designed, or evolved, integrating cooperation with each other to accomplish a common task. In nature, this is known as cooperative co-evolution. For instance, two swarms could be co-evolved to allow one to find consensus on choosing a site to inspect (e.g. based on chemical density) and herd the other to the area, enabling them to cover and monitor the area of interest. Hence, cooperation amongst different swarms can result in elaborate solutions to complex scenarios.

References

3atoms (2012). School of fish. http://3atoms.tumblr.com/post/25223888588/schools-of-fish.

Alpern, S. (1995). The rendezvous search problem. *SIAM Journal on Control and Optimization*, 33(3):673–683.

Bak, P. (1996). *How nature works: the science of self-organized criticality.* Copernicus, New York, NY Copernicus, New York, NY.

Balch, T. (2000). Hierarchic social entropy: An information theoretic measure of robot group diversity. *Autonomous Robots*, 8(3):209–238.

Ballerini, M., Cabibbo, N., Candelier, R., Cavagna, A., Cisbani, E., Giardina, I., Lecomte, V., Orlandi, A., Parisi, G., Procaccini, A., et al. (2008). Interaction ruling animal collective behavior depends on topological rather than metric distance: Evidence from a field study. *Proceedings of the national academy of sciences*, 105(4):1232–1237.

Barel, A., Manor, R., and Bruckstein, A. M. (2017). Probabilistic gathering of agents with simple sensors. Technical report, Technion – Israel Institute of Technology.

Becker, A., Habibi, G., Werfel, J., Rubenstein, M., and McLurkin, J. (2013). Massive uniform manipulation: Controlling large populations of simple robots with a common input signal. In *Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 520–527. IEEE.

Beckers, R., Deneubourg, J. L., Goss, S., and Pasteels, J. M. (1990). Collective decision making through food recruitment. *Insectes Sociaux*, 37(3):258–267.

Beni, G. (2005). From swarm intelligence to swarm robotics. In *Swarm Robotics*, pages 1–9. Springer Berlin Heidelberg.

Beni, G. and Wang, J. (1993). Swarm intelligence in cellular robotic systems. In *Robots and Biological Systems: Towards a New Bionics?*, pages 703–712. Springer.

Benson, J. (2007). V-formation in flock of birds. https://commons.wikimedia.org/wiki/File:CanadianGeeseFlyingInVFormation.jpg.

Birattari, M., Delhaisse, B., Francesca, G., and Kerdoncuff, Y. (2016). Observing the effects of overdesign in the automatic design of control software for robot swarms. In *International Conference on Swarm Intelligence*, pages 149–160. Springer.

Bonabeau, E., Marco, D., and Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press.

Bonabeau, E. and Meyer, C. (2001). Swarm intelligence: A whole new way to think about business. *Harvard Business Review*, 79(5):106–115.

Bonabeau, E., Sobkowski, A., Theraulaz, G., Deneubourg, J.-L., et al. (1997). Adaptive task allocation inspired by a model of division of labor in social insects. In *BCEC*, pages 36–45.

Bose, T., Reina, A., and Marshall, J. A. (2017). Collective decision-making. *Current Opinion in Behavioral Sciences*, 16:30–34.

Bradski, G. (2000). Open source computer vision library.

Braitenberg, V. (1986). *Vehicles: Experiments in synthetic psychology.* MIT press.

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

Brewbooks (2009). Cathedral termite mound. https://www.flickr.com/photos/brewbooks/3491315062/.

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, 2(1):14–23.

Brown, D. S., Turner, R., Hennigh, O., and Loscalzo, S. (2018). Discovery and exploration of novel swarm behaviors given limited robot capabilities. In *Proceedings of the 13th International Symposium on Distributed Autonomous Robotic Systems (DARS 2016)*, pages 447–460, Berlin, Germany. Springer.

Camazine, S., Deneubourg, J.-L., Franks, N. R., Sneyd, J., Bonabeau, E., and Theraula, G. (2001). *Self-Organization in Biological Systems.* Princeton University Press.

Carnemolla, J. (2016). Flock of sheep in roadway. http://www.allposters.com/-sp/Flock-of-Sheep-in-Roadway-Posters_i8668862_.htm.

Cech, J. J. and Moyle, P. B. (2000). *Fishes: an introduction to ichthyology.* Prentice-Hall.

Çelikkanat, H. and Şahin, E. (2010). Steering self-organized robot flocks through externally guided individuals. *Neural Computing and Applications*, 19(6):849–865.

Cerofolini, G. and Amato, P. (2013). Sensing strategies for early diagnosis of cancer by swarm of nanorobots: An evidential paradigm. In *Nanorobotics*, pages 331–352. Springer.

Chaimowicz, L., Campos, M. F., and Kumar, V. (2002). Dynamic role assignment for cooperative robots. In *Proceedings of 2002 IEEE International Conference on Robotics and Automation (ICRA 2002)*, volume 1, pages 293–298. IEEE.

Choset, H. (2001). Coverage for robotics – a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31(1):113–126.

Claici, S., Romanishin, J., Lipton, J. I., Bonardi, S., Gilpin, K. W., and Rus, D. (2017). Distributed aggregation for modular robots in the pivoting cube model. In *Proceedings of 2017 IEEE International Conference on Robotics and Automation (ICRA 2017)*, pages 1489–1496. IEEE.

Conradt, L. and Roper, T. J. (2005). Consensus decision making in animals. *Trends in Ecology & Evolution*, 20(8):449–456.

Cord-Landwehr, A., Fischer, M., Jung, D., and Meyer auf der Heide, F. (2016). Asymptotically optimal gathering on a grid. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2016)*, pages 301–312. ACM.

Correll, N. and Martinoli, A. (2006). Collective inspection of regular structures using a swarm of miniature robots. In *Experimental Robotics IX*, pages 375–386. Springer, Berlin, Germany.

Correll, N. and Martinoli, A. (2011). Modeling and designing self-organized aggregation in a swarm of miniature robots. *The International Journal of Robotics Research*, 30(5):615–626.

Crespi, V., Galstyan, A., and Lerman, K. (2008). Top-down vs bottom-up methodologies in multi-agent system design. *Autonomous Robots*, 24(3):303–313.

Deneubourg, J.-L., Grégoire, J.-C., and Le Fort, E. (1990). Kinetics of larval gregarious behavior in the bark beetledendroctonus micans (coleoptera: Scolytidae). *Journal of Insect Behavior*, 3(2):169–182.

Deneubourg, J.-L., Pasteels, J. M., and Verhaeghe, J.-C. (1983). Probabilistic behaviour in ants: a strategy of errors? *Journal of Theoretical Biology*, 105(2):259–271.

Dimidov, C., Oriolo, G., and Trianni, V. (2016). Random walks in swarm robotics: An experiment with Kilobots. In *Proceedings of the 10th International Conference on Swarm Intelligence (ANTS 2016)*, pages 185–196, Cham, Switzerland. Springer.

Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms.* PhD thesis, Politecnico di Milano.

Dorigo, M. and Blum, C. (2005). Ant colony optimization theory: A survey. *Theoretical Computer Science*, 344(2-3):243–278.

Dorigo, M. and Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66.

Dorigo, M. and Şahin, E. (2004). Autonomous robots: guest editorial. *Autonomous Robots*, 17(2-3):111–113.

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

Dudek, G. and Jenkin, M. (2010). *Computational Principles of Mobile Robotics*. Cambridge University Press, Cambridge, UK.

Dudek, G., Jenkin, M., Milios, E., and Wilkes, D. (1993). A taxonomy for swarm robots. In *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 1993)*, volume 1, pages 441–447. IEEE.

Eiben, A. and Smith, J. (2008). *Introduction to Evolutionary Computing*. Springer.

Escalera, J. A., Doyle, M. J., Mondada, F., and Groß, R. (2018). Evo-bots: A simple, stochastic approach to self-assembling artificial organisms. In *Proceedings of the 13th International Symposium on Distributed Autonomous Robotic Systems (DARS 2016)*, pages 373–385.

Fatès, N. (2010). Solving the decentralised gathering problem with a reaction–diffusion–chemotaxis scheme. *Swarm Intelligence*, 4(2):91–115.

Fax, J. A. and Murray, R. M. (2004). Information flow and cooperative control of vehicle formations. *IEEE Transactions on Automatic Control*, 49(9):1465–1476.

Ferrante, E., Turgut, A. E., Huepe, C., Stranieri, A., Pinciroli, C., and Dorigo, M. (2012). Self-organized flocking with a mobile robot swarm: A novel motion control method. *Adaptive Behavior*, 20(6):460–477.

Fischer, M., Jung, D., and auf der Heide, F. M. (2017). Gathering anonymous, oblivious robots on a grid. In *Algorithms for Sensor Systems (ALGOSENSORS 2017)*, pages 168–181. Springer.

Floreano, D., Husbands, P., and Nolfi, S. (2008). Evolutionary robotics. In *Springer Handbook of Robotics*, pages 1423–1451. Springer.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial Intelligence Through Simulated Evolution*. John Wiley.

Francesca, G., Brambilla, M., Trianni, V., Dorigo, M., and Birattari, M. (2012). Analysing an evolved robotic behaviour using a biological model of collegial decision making. In *From Animals to Animats 12*, volume 7426, pages 381–390. Springer-Verlag Berlin Heidelberg.

Franks, N. R., Hooper, J. W., Gumn, M., Bridger, T. H., Marshall, J. A. R., Groß, R., and Dornhaus, A. (2007). Moving targets: Collective decisions and flexible choices in house-hunting ants. *Swarm Intelligence*, 1(2):81–94.

Gauci, M. (2014). *Swarm robotic systems with minimal information processing*. PhD thesis, The University of Sheffield.

Gauci, M., Chen, J., Dodd, T. J., and Groß, R. (2014a). Evolving aggregation behaviors in multi-robot systems with binary sensors. In *Proceedings of the 11th International Symposium on Distributed Autonomous Robotic Systems (DARS 2012)*, volume 104, pages 355–367, Berlin, Germany. Springer.

Gauci, M., Chen, J., Li, W., Dodd, T. J., and Groß, R. (2014b). Clustering objects with robots that do not compute. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014)*, pages 421–428.

Gauci, M., Chen, J., Li, W., Dodd, T. J., and Groß, R. (2014c). Self-organized aggregation without computation. *The International Journal of Robotics Research*, 33(8):1145–1161.

Gazi, V. and Fidan, B. (2007). Coordination and control of multi-agent dynamic systems: Models and approaches. In *Swarm Robotics*, pages 71–102. Springer.

Gordon, D. (2013). What ant colony networks can tell us about what's next for digital networks. https://www.nextnature.net/2013/07/what-ant-colony-networks-can-tell-us-about-what%E2%80%99s-next-for-digital-networks/.

Gordon, N., Wagner, I. A., and Bruckstein, A. M. (2004). Gathering multiple robotic a(ge)nts with limited sensing capabilities. In Dorigo, M., Birattari, M., Blum, C., Gambardella, L. M., Mondada, F., and Stützle, T., editors, *Ant Colony Optimization and Swarm Intelligence*, pages 142–153, Berlin, Heidelberg. Springer Berlin Heidelberg.

Graham, J. M., Kao, A. B., Wilhelm, D. A., and Garnier, S. (2017). Optimal construction of army ant living bridges. *Journal of Theoretical Biology*, 435:184–198.

Grassé, P.-P. et al. (1984). *Termitology. Termite anatomy-physiology-biology-systematics. Vol. II. Colony foundation-construction.* Masson.

Groß, R., Bonani, M., Mondada, F., and Dorigo, M. (2006). Autonomous self-assembly in swarm-bots. *IEEE Transactions on Robotics*, 22(6):1115–1130.

Groß, R. and Dorigo, M. (2008). Evolution of solitary and group transport behaviors for autonomous robots capable of self-assembling. *Adaptive Behavior*, 16(5):285–305.

Halloy, J., Sempo, G., Caprari, G., Rivault, C., Asadpour, M., Tâche, F., Saïd, I., Durier, V., Canonge, S., Amé, J. M., Detrain, C., Correll, N., Martinoli, A., Mondada, F., Siegwart, R., and Deneubourg, J. L. (2007). Social integration of robots into groups of cockroaches to control self-organized choices. *Science*, 318(5853):1155–1158.

Hamann, H., Meyer, B., Schmickl, T., and Crailsheim, K. (2010). A model of symmetry breaking in collective decision-making. In *From Animals to Animats 11*, volume 6226, pages 639–648. Springer-Verlag Berlin Heidelberg.

Hansen, N., Akimoto, Y., and Baudis, P. (2019). CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634.

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195.

Holland, J. H. et al. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* MIT press.

Hornby, G., Globus, A., Linden, D., and Lohn, J. (2006). Automated antenna design with evolutionary algorithms. In *Space 2006*, pages 19–21.

Howard, A., Matarić, M. J., and Sukhatme, G. S. (2002). Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Proceedings of the 6th International Symposium on Distributed Autonomous Robotic Systems (DARS 2002)*, pages 299–308, Berlin, Germany. Springer.

Howse, P. E. (1970). *Termites: A Study in Social Behaviour.* Hutchinson.

Hsieh, M. A., Halász, Á., Berman, S., and Kumar, V. (2008). Biologically inspired redistribution of a swarm of robots among multiple sites. *Swarm Intelligence*, 2(2):121–141.

Ijspeert, A. J., Martinoli, A., Billard, A., and Gambardella, L. M. (2001). Collaboration through the exploitation of local interactions in autonomous collective robotics: The stick pulling experiment. *Autonomous Robots*, 11(2):149–171.

Jakobi, N., Husbands, P., and Harvey, I. (1995). *Noise and the reality gap: The use of simulation in evolutionary robotics*, pages 704–720. Springer Berlin Heidelberg, Berlin, Heidelberg.

Jalili, N. (2013). Nanomechanical cantilever-based manipulation for sensing and imaging. In *Nanorobotics*, pages 29–40. Springer.

Ji, M. and Egerstedt, M. (2007). Distributed coordination control of multiagent systems while preserving connectedness. *IEEE Transactions on Robotics*, 23(4):693–703.

Johnson, M. and Brown, D. S. (2015). Evolving and controlling perimeter, rendezvous, and foraging behaviors in a computation-free robot swarm. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies*, pages 311–314. ICST, Brussels, Belgium.

Jones, C. and Mataric, M. J. (2003). Adaptive division of labor in large-scale minimalist multi-robot systems. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 1969–1974. IEEE.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of International Conference on Neural Networks (ICNN 1995)*, volume 4, pages 1942–1948.

Kershner, R. (1939). The number of circles covering a set. *American Journal of Mathematics*, 61(3):665–671.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous Robot Vehicles*, pages 396–404. Springer.

Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press.

Krause, J., Cordeiro, J., Parpinelli, R. S., and Lopes, H. S. (2013). A survey of swarm algorithms applied to discrete optimization problems. In *Swarm Intelligence and Bio-Inspired Computation*, pages 169–191. Elsevier.

Krause, J. and Ruxton, G. D. (2002). *Living in Groups.* Oxford University Press.

Lee, W. and Kim, D. (2017). Autonomous shepherding behaviors of multiple target steering robots. *Sensors*, 17(12):2729.

Li, J., Esteban-Fernández de Ávila, B., Gao, W., Zhang, L., and Wang, J. (2017). Micro/nanorobots for biomedicine: Delivery, surgery, sensing, and detoxification. *Science Robotics*, 2(4).

Lien, J.-M., Rodriguez, S., Malric, J.-P., and Amato, N. M. (2005). Shepherding behaviors with multiple shepherds. In *Proceedings of 2005 IEEE International Conference on Robotics and Automation (ICRA 2005)*, pages 3402–3407. IEEE.

Lin, Y.-y. and Chen, Y.-p. (2007). Crowd control with swarm intelligence. In *2007 IEEE Congress on Evolutionary Computation*, pages 3321–3328. IEEE.

Magnenat, S., Waibel, M., and Beyeler, A. (2009). Enki: An open source fast 2D robot simulator. https://github.com/enki-community/enki.

Martens, D., Baesens, B., and Fawcett, T. (2011). Editorial survey: swarm intelligence for data mining. *Machine Learning*, 82(1):1–42.

Martinoli, A. and Mondada, F. (1997). Collective and cooperative group behaviours: Biologically inspired experiments in robotics. In *Experimental Robotics IV*, pages 1–10. Springer-Verlag.

Mataric, M. J. (1998). Using communication to reduce locality in distributed multiagent learning. *Journal of Experimental & Theoretical Artificial Intelligence*, 10(3):357–369.

Mavroidis, C. and Ferreira, A. (2013). *Nanorobotics: past, present, and future.* Springer.

McLurkin, J. and Smith, J. (2007). Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots. In *Proceedings of the 7th International Symposium on Distributed Autonomous Robotic Systems (DARS 2004)*, pages 399–408, Tokyo, Japan. Springer.

Minsky, M. L. (1967). *Computation.* Prentice-Hall Englewood Cliffs.

Mitchell, M. (2009). *Complexity: A guided tour.* Oxford University Press.

Mitrano, P., Burklund, J., Giancola, M., and Pinciroli, C. (2019). A minimalistic approach to segregation in robot swarms. In *Proceedings of the 2nd IEEE International Symposium on Multi-Robot and Multi-Agent Systems (MRS 2019).*

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D., and Martinoli, A. (2009). The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, volume 1, pages 59–65.

Mondada, F., Guignard, A., Bonani, M., Bar, D., Lauria, M., and Floreano, D. (2003). Swarm-bot: From concept to implementation. In *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, volume 2, pages 1626–1631. IEEE.

Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics: The Biology, Intelligence, and Technology.* MIT Press, Cambridge, MA, USA.

Norrman, R. (1999). Creating the world in our image: A new theory of love of symmetry and iconicist desire. *Form miming meaning: Iconicity in language and literature*, pages 59–82.

Özdemir, A., Gauci, M., Bonnet, S., and Groß, R. (2018). Online supplementary material. http://naturalrobotics.group.shef.ac.uk/supp/2018-002/.

Özdemir, A., Gauci, M., and Groß, R. (2017). Online supplementary material. http://naturalrobotics.group.shef.ac.uk/supp/2017-002/.

Özdemir, A., Gauci, M., Kolling, A., Hall, M. D., and Groß, R. (2019a). Online supplementary material. http://naturalrobotics.group.shef.ac.uk/supp/2019-001/.

Özdemir, A., Romanishin, J. W., Groß, R., and Rus, D. (2019b). Decentralized gathering of stochastic, oblivious agents on a grid: A case study with 3d m-blocks. In *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*.

Özdemir, A., Romanishin, J. W., Groß, R., and Rus, D. (2019c). Online supplementary material. https://doi.org/10.6084/m9.figshare.8527148.

Ozsoyeller, D., Beveridge, A., and Isler, V. (2019). Rendezvous in planar environments with obstacles and unknown initial distance. *Artificial Intelligence*, 273:19 – 36.

Panait, L. and Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434.

Parker, C. A. and Zhang, H. (2009). Cooperative decision-making in decentralized multiple-robot systems: The best-of-n problem. *IEEE/ASME Transactions on Mechatronics*, 14(2):240–251.

Parker, L. E. (2008). Multiple mobile robot systems. In *Springer Handbook of Robotics*, pages 921–941. Springer.

Parpinelli, R. S., Lopes, H. S., and Freitas, A. A. (2001). An ant colony based system for data mining: applications to medical data. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 791–797. Morgan Kaufmann Publishers Inc.

Parrish, J. K. and Edelstein-Keshet, L. (1999). Complexity, pattern, and evolutionary trade-offs in animal aggregation. *Science*, 284(5411):99–101.

Partridge, B. L. (1982). The structure and function of fish schools. *Scientific American*, 246(6):114–123.

Pierson, A. and Schwager, M. (2015). Bio-inspired non-cooperative multi-robot herding. In *Proceedings of 2015 IEEE International Conference on Robotics and Automation (ICRA 2015)*, pages 1843–1849.

Pierson, A. and Schwager, M. (2017). Controlling noncooperative herds with robotic herders. *IEEE Transactions on Robotics*, 34(2):517–525.

Piggins, D. and Phillips, C. (1996). The eye of the domesticated sheep with implications for vision. *Animal Science*, 62(2):301–308.

Pitcher, T., Magurran, A., and Winfield, I. (1982). Fish in larger shoals find food faster. *Behavioral Ecology and Sociobiology*, 10(2):149–151.

Portugal, D. and Rocha, R. (2011). A survey on multi-robot patrolling algorithms. In Camarinha-Matos, L. M., editor, *Technological Innovation for Sustainability*, pages 139–146, Berlin, Germany. Springer.

Prorok, A., Correll, N., and Martinoli, A. (2011). Multi-level spatial modeling for stochastic distributed robotic systems. *International Journal of Robotics Research*, 30(5):574–589.

Ramaithitima, R., Whitzer, M., Bhattacharya, S., and Kumar, V. (2015). Sensor coverage robot swarms using local sensing without metric information. In *Proceedings of 2015 IEEE International Conference on Robotics and Automation (ICRA 2015)*, pages 3408–3415. IEEE.

Razali, S., Meng, Q., and Yang, S.-H. (2010). A refined immune systems inspired model for multi-robot shepherding. In *Nature and Biologically Inspired Computing (NaBIC), 2010 Second World Congress on*, pages 473–478. IEEE.

Rechenberg, I. (1978). Evolutionsstrategien. In *Simulationsmethoden in der Medizin und Biologie*, pages 83–114. Springer.

Requicha, A. (2013). Swarms of self-organized nanorobots. In *Nanorobotics*, pages 41–49. Springer.

Requicha, A. A. (2003). Nanorobots, NEMS, and nanoassembly. *Proceedings of the IEEE*, 91(11):1922–1933.

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. In *ACM Siggraph Computer Graphics*, volume 21, pages 25–34. ACM.

Romanishin, J. W., Gilpin, K., Claici, S., and Rus, D. (2015). 3d m-blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions. In *Proceedings of 2015 IEEE International Conference on Robotics and Automation (ICRA 2015)*, pages 1925–1932. IEEE.

Rutishauser, S., Correll, N., and Martinoli, A. (2009). Collaborative coverage using a swarm of networked miniature robots. *Robotics and Autonomous Systems*, 57(5):517–525.

Safonov, A. (2008). Sardine run. https://bit.ly/2YXTA7y.

Şahin, E. (2005). Swarm robotics: From sources of inspiration to domains of application. In *Swarm Robotics*, pages 10–20. Springer Berlin Heidelberg.

Şahin, E., Girgin, S., Bayindir, L., and Turgut, A. E. (2008). Swarm robotics. *Swarm intelligence*, 1:87–100.

Schmickl, T., Thenius, R., Moeslinger, C., Radspieler, G., Kernbach, S., Szymanski, M., and Crailsheim, K. (2009). Get in touch: cooperative decision making based on robot-to-robot collisions. *Autonomous Agents and Multi-Agent Systems*, 18(1):133–155.

Schwager, M., McLurkin, J., and Rus, D. (2006). Distributed coverage control with sensory feedback for networked robots. In *Robotics: Science and Systems*, pages 49–56.

Seyfried, J., Szymanski, M., Bender, N., Estana, R., Thiel, M., and Wörn, H. (2004). The i-swarm project: Intelligent small world autonomous robots for micro-manipulation. In *International Workshop on Swarm Robotics*, pages 70–83. Springer.

Sitti, M. (2017). *Mobile Microrobotics.* MIT Press, Cambridge, MA.

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

Spears, W. M., Spears, D. F., Hamann, J. C., and Heil, R. (2004). Distributed, physics-based control of swarms of vehicles. *Autonomous Robots*, 17(2-3):137–162.

Stass, J. (2015). Why do birds flock together? https://www.howitworksdaily.com/why-do-birds-flock-together/.

Strömbom, D., Mann, R. P., Wilson, A. M., Hailes, S., Morton, A. J., Sumpter, D. J., and King, A. J. (2014). Solving the shepherding problem: Heuristics for herding autonomous, interacting agents. *Journal of the Royal Society Interface*, 11(100):20140719.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337.

Tan, Y. and Zheng, Z.-y. (2013). Research advance in swarm robotics. *Defence Technology*, 9(1):18–39.

Trianni, V. (2008). *Evolutionary Swarm Robotics: Evolving Self-Organising Behaviours in Groups of Autonomous Robots.* Springer, Berlin, Germany.

Trianni, V. and Nolfi, S. (2009). Self-organizing sync in a robotic swarm: a dynamical system view. *Evolutionary Computation, IEEE Transactions on*, 13(4):722–741.

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

Tsitsiklis, J. N. (1984). *Problems in Decentralized Decision making and Computation.* PhD thesis, MIT, Cambridge, MA.

Valentini, G., Ferrante, E., and Dorigo, M. (2017). The best-of-n problem in robot swarms: Formalization, state of the art, and novel perspectives. *Frontiers in Robotics and AI*, 4(9).

Valentini, G., Ferrante, E., Hamann, H., and Dorigo, M. (2016). Collective decision with 100 kilobots: speed versus accuracy in binary discrimination problems. *Autonomous Agents and Multiagent Systems*, 30(3):553–580.

Vaughan, R., Sumpter, N., Henderson, J., Frost, A., and Cameron, S. (2000). Experiments in automatic flock control. *Robotics and Autonomous Systems*, 31(1):109–117.

Visscher, P. K. (2007). Group decision making in nest-site selection among social insects. *Annual Review of Entomology*, 52(1):255–275.

Walter, J. E. (2018). Sensor-driven algorithm for self-reconfiguration of modular robots. In *2018 International Conference on Reconfigurable Mechanisms and Robots (ReMAR)*, pages 1–7. IEEE.

Wareham, T. and Vardy, A. (2018). Viable algorithmic options for designing reactive robot swarms. *ACM Transactions on Autonomous and Adaptive Systems*, 13(1):5:1–5:23.

Wolpert, D. H. and Macready, W. G. (1995). No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute.

Wurman, P. R., D'Andrea, R., and Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29(1):9–9.

Yu, J., LaValle, S. M., and Liberzon, D. (2012). Rendezvous without coordinates. *IEEE Transactions on Automatic Control*, 57(2):421–434.

Appendix A

Counter-Examples

This appendix presents the counter-example images for the dynamic deadlock situations presented in Section 6.3.3 in Chapter 6 for deterministic multi-agent gathering problem. The caption states the counter-example ID number, the period of the dynamic deadlock, and the interval of images in that page. Additionaly, if the images span over one page, a roman numeral indicates the number of page in the beginning of the caption. Each module is represented by a different colour. The 'hatch' over a module indicates that it is that module's turn to take an action.

Table A.1 shows a summary list of 21 counter-examples. It lists the counter-example ID number, the starting figure reference number, the period of the counter-example, and *a priori* unknown contexts that are participated.

Table A.1 The list of counter-examples presented in Appendix A.

ID	Fig.Ref.	Period	Unknown participated contexts
16	Fig. A.1	20	E2, H1, I1, I3
17	Fig. A.2	24	E2, H1, I1
18	Fig. A.3-A.4	36	E3, H3
19	Fig. A.5	28	E1, E3, H1, H3, I1
20	Fig. A.6-A.7	52	E1, E2, E3, H1, H3, I1
21	Fig. A.8	16	E1, E2, H1, I1
22	Fig. A.9-A.10	44	E1, E2, E3, H1, I1
23	Fig. A.11	28	E1, E2, H1, H3, I1
24	Fig. A.12-A.13	32	B, D2, E1
25	Fig. A.14	8	E2
26	Fig. A.15-A.19	48	C
27	Fig. A.20	8	C
28	Fig. A.21-A.23	40	E1
29	Fig. A.24-A.26	48	B
30	Fig. A.27-A.33	126	D2, E2, H1
31	Fig. A.34-A.35	36	D2, H1
32	Fig. A.36-A.47	228	H1
33	Fig. A.48-A.51	60	B
34	Fig. A.52	12	D2
35	Fig. A.53-A.56	40	B, D2
36	Fig. A.57-A.79	360	N/A

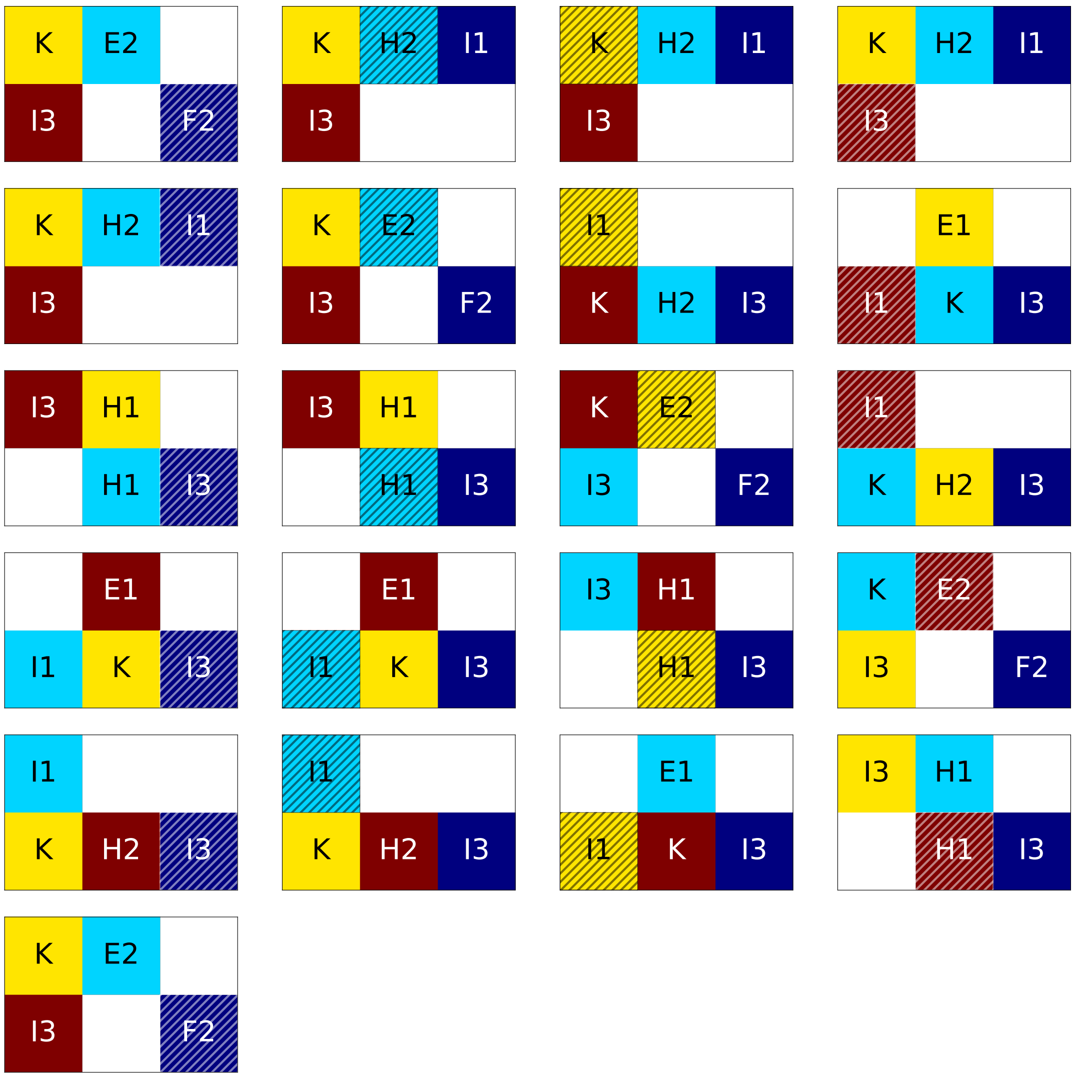


Fig. A.1 Counter-example 16. Dynamic deadlock of period 20. Images show 0-20 of 20.

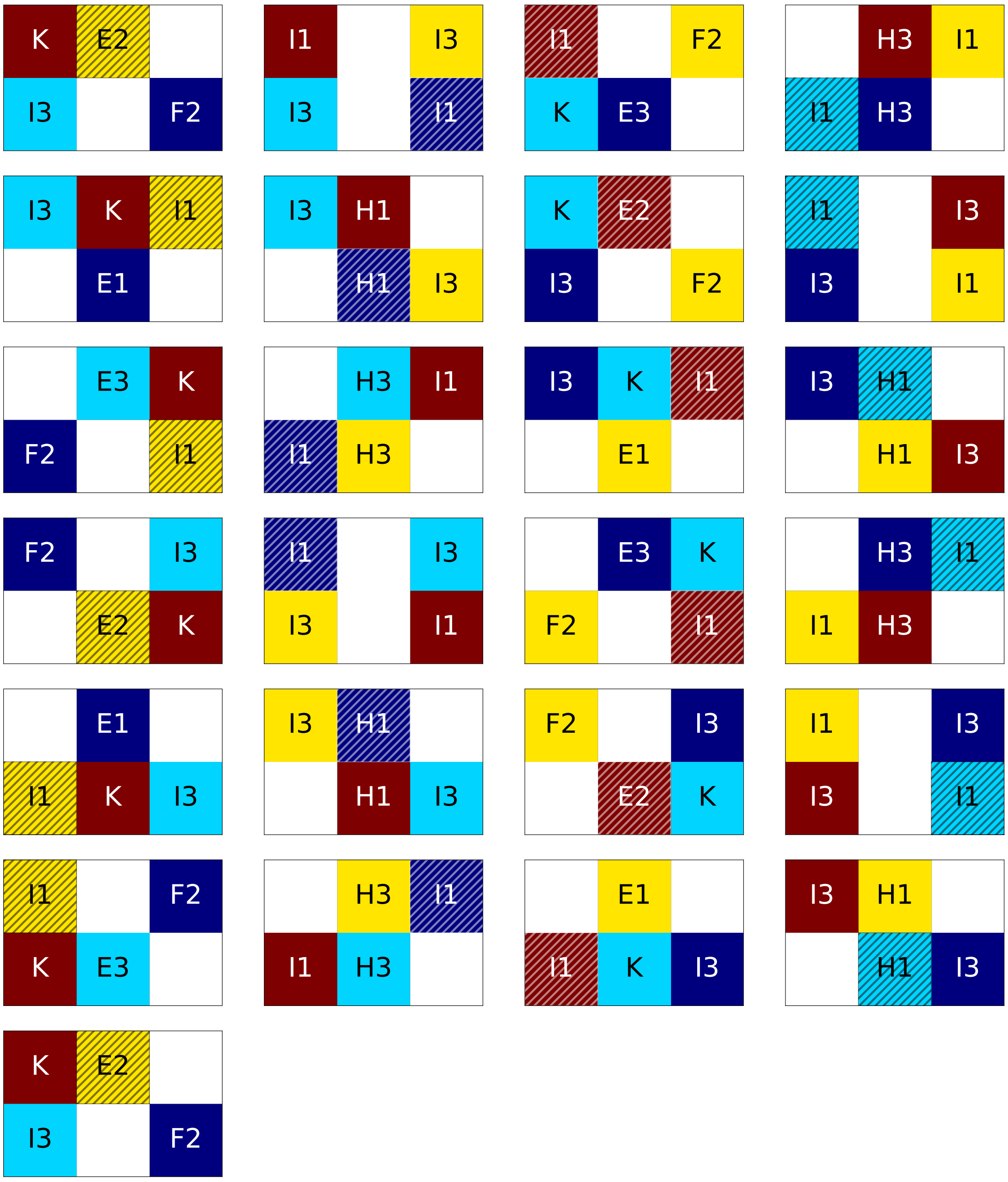


Fig. A.2 Counter-example 17. Dynamic deadlock of period 24. Images show 0-24 of 24.

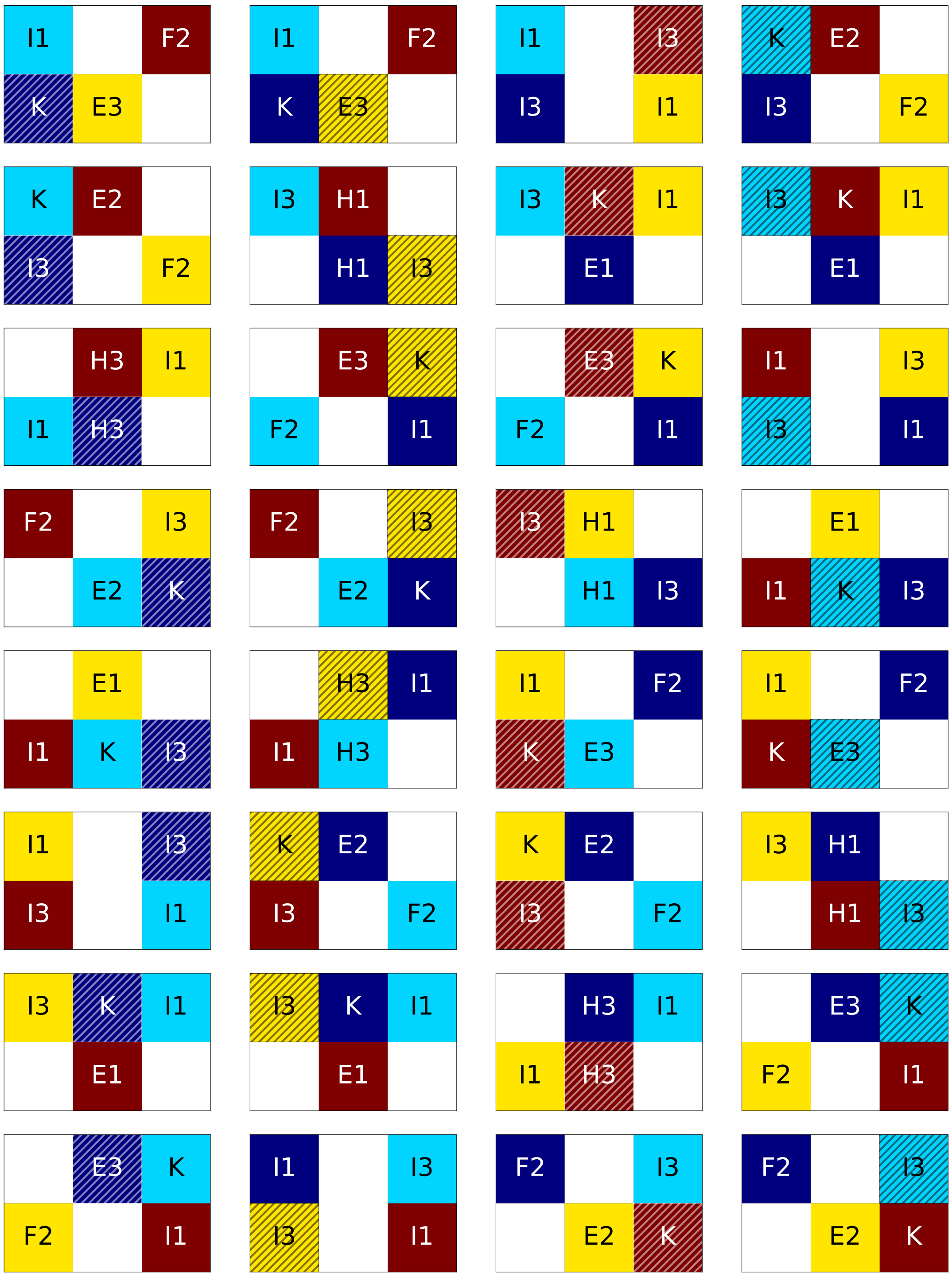


Fig. A.3 (I) Counter-example 18. Dynamic deadlock of period 36. Images show 0-31 of 36.

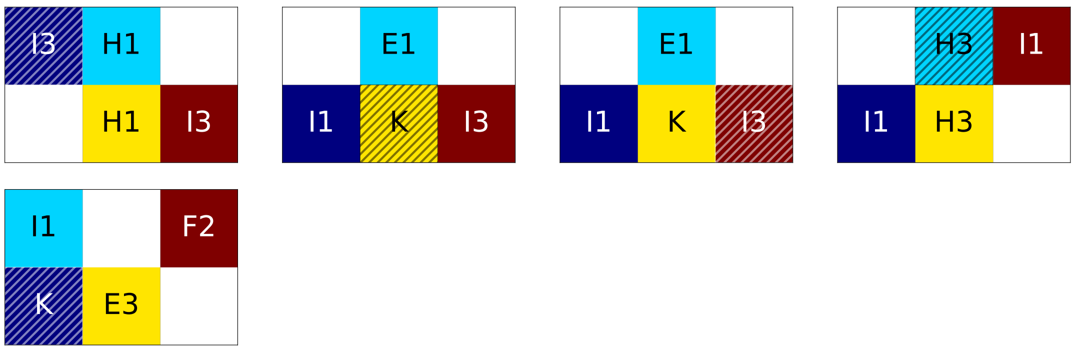


Fig. A.4 (II) Counter-example 18. Dynamic deadlock of period 36. Images show 32-36 of 36.

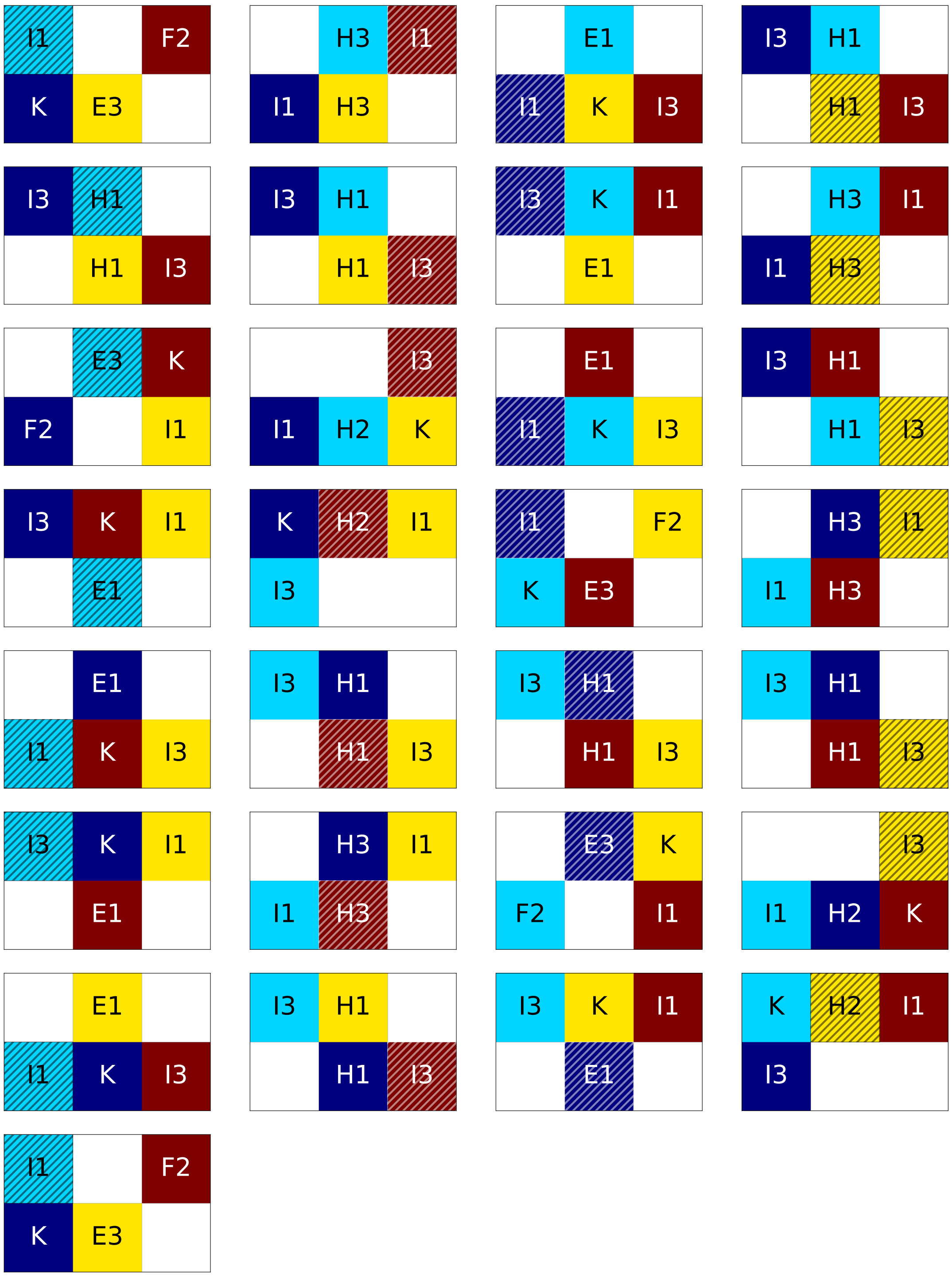


Fig. A.5 Counter-example 19. Dynamic deadlock of period 28. Images show 0-28 of 28.

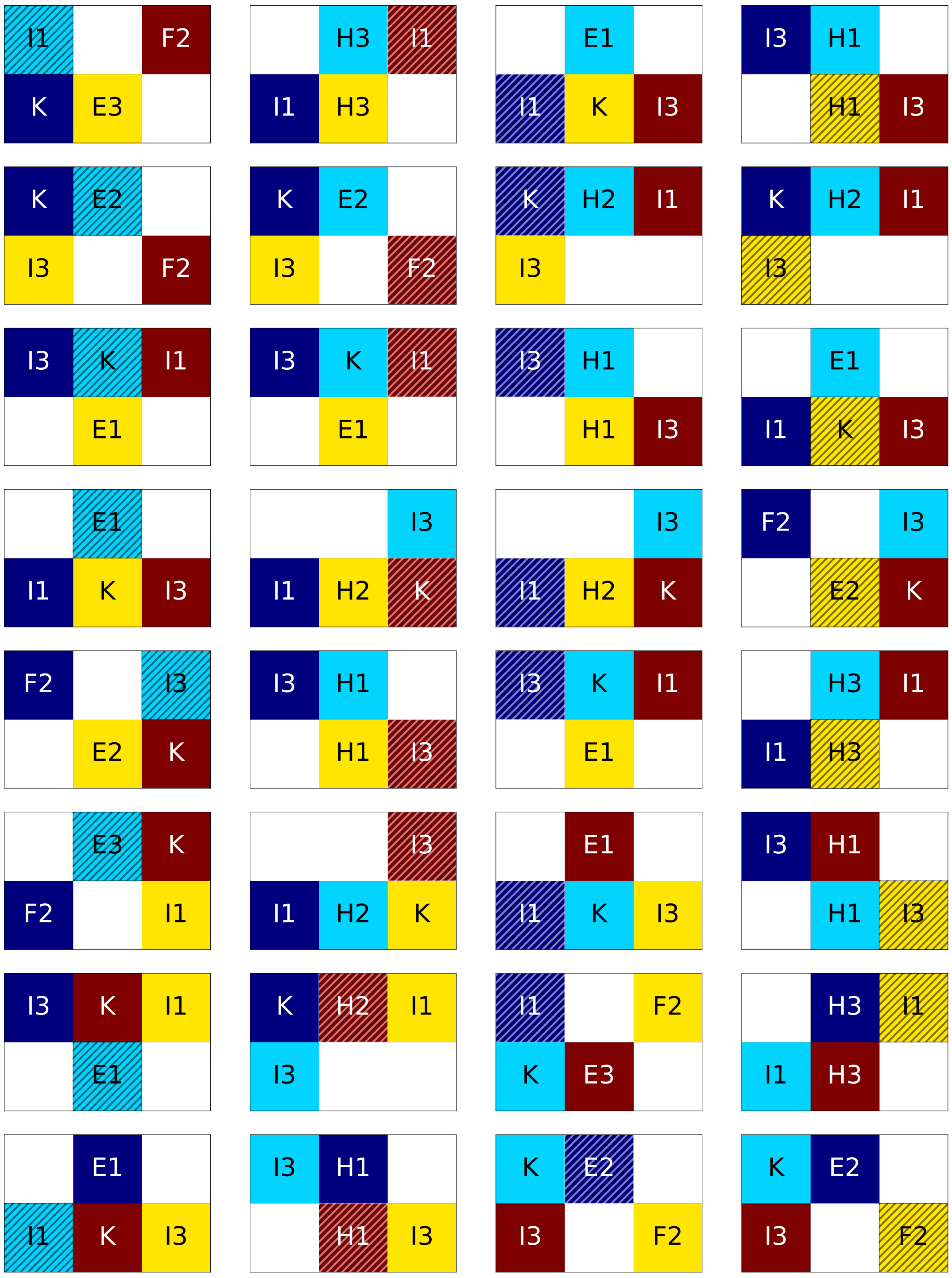


Fig. A.6 (I) Counter-example 20. Dynamic deadlock of period 52. Images show 0-31 of 52.

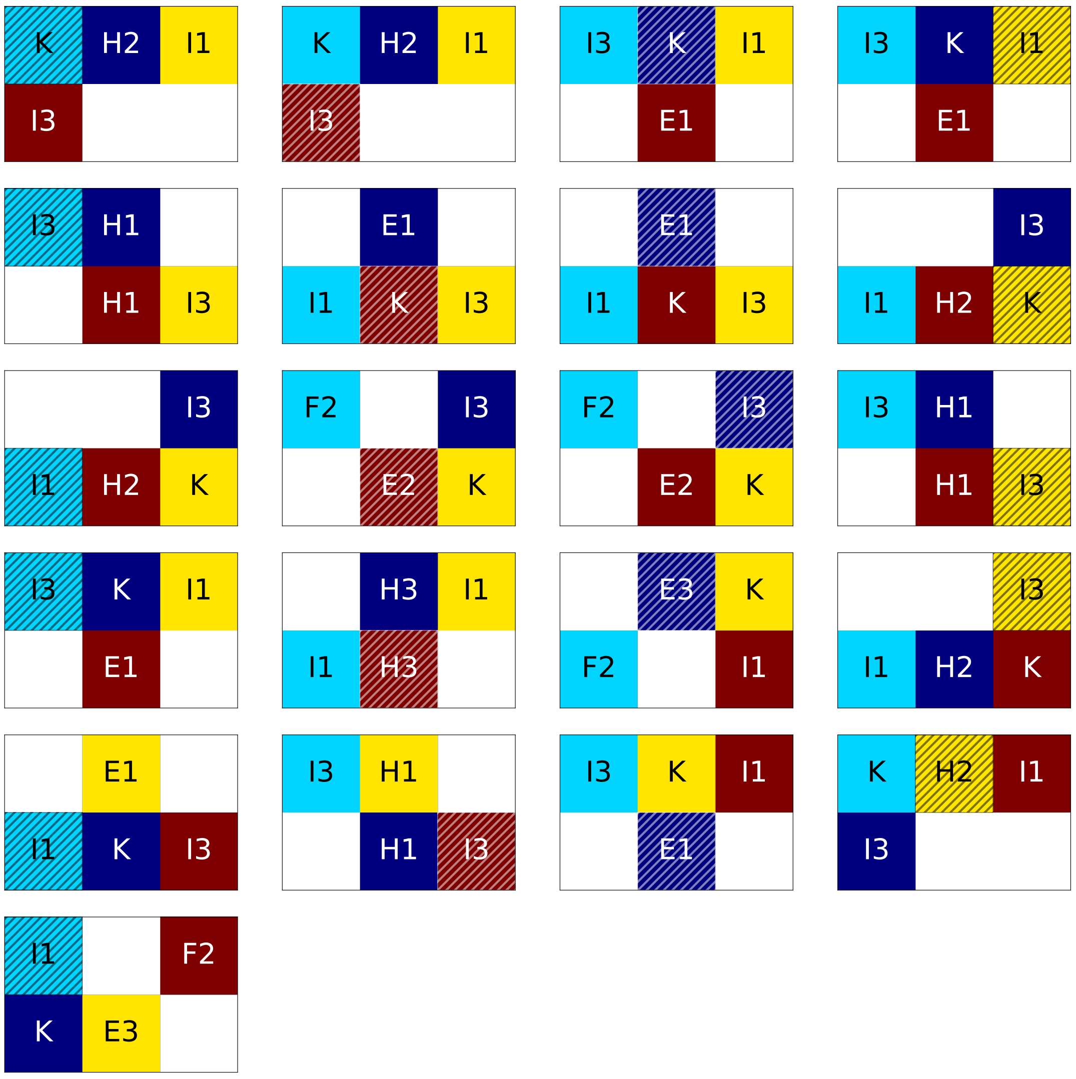


Fig. A.7 (II) Counter-example 20. Dynamic deadlock of period 52. Images show 32-52 of 52.

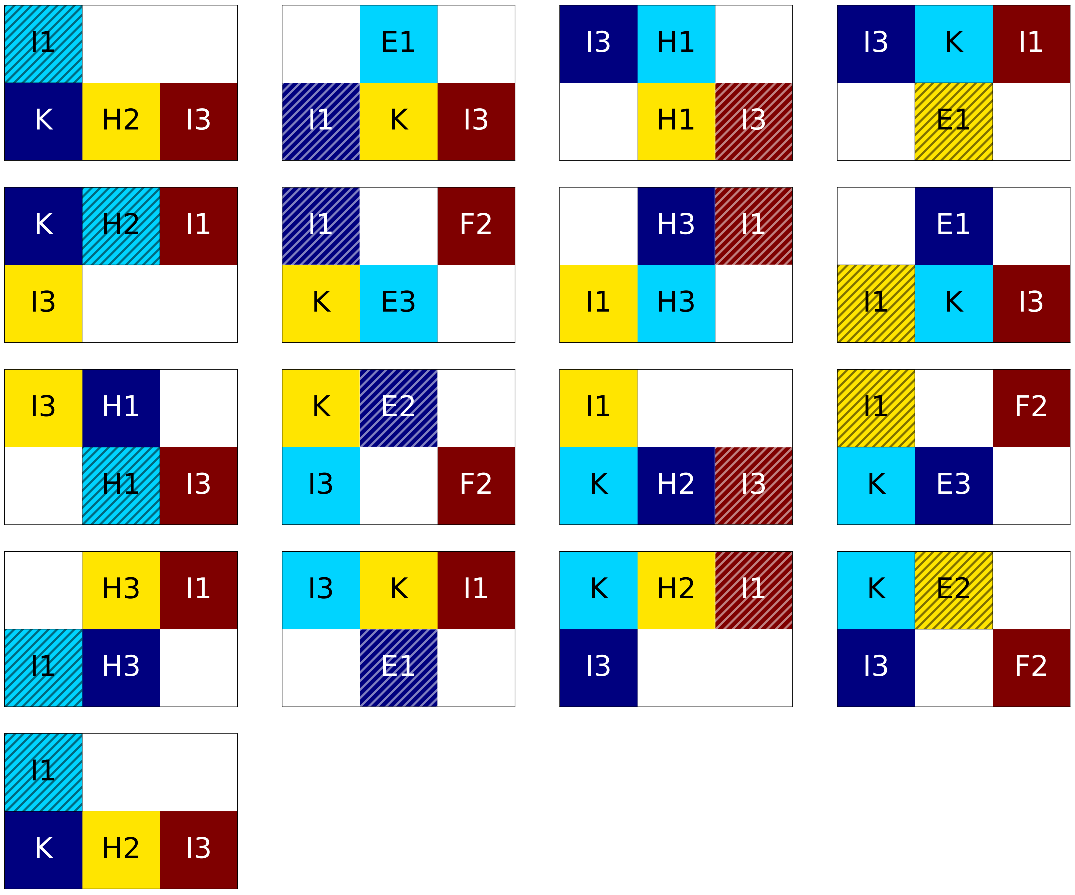


Fig. A.8 Counter-example 21. Dynamic deadlock of period 16. Images show 0-16 of 16.

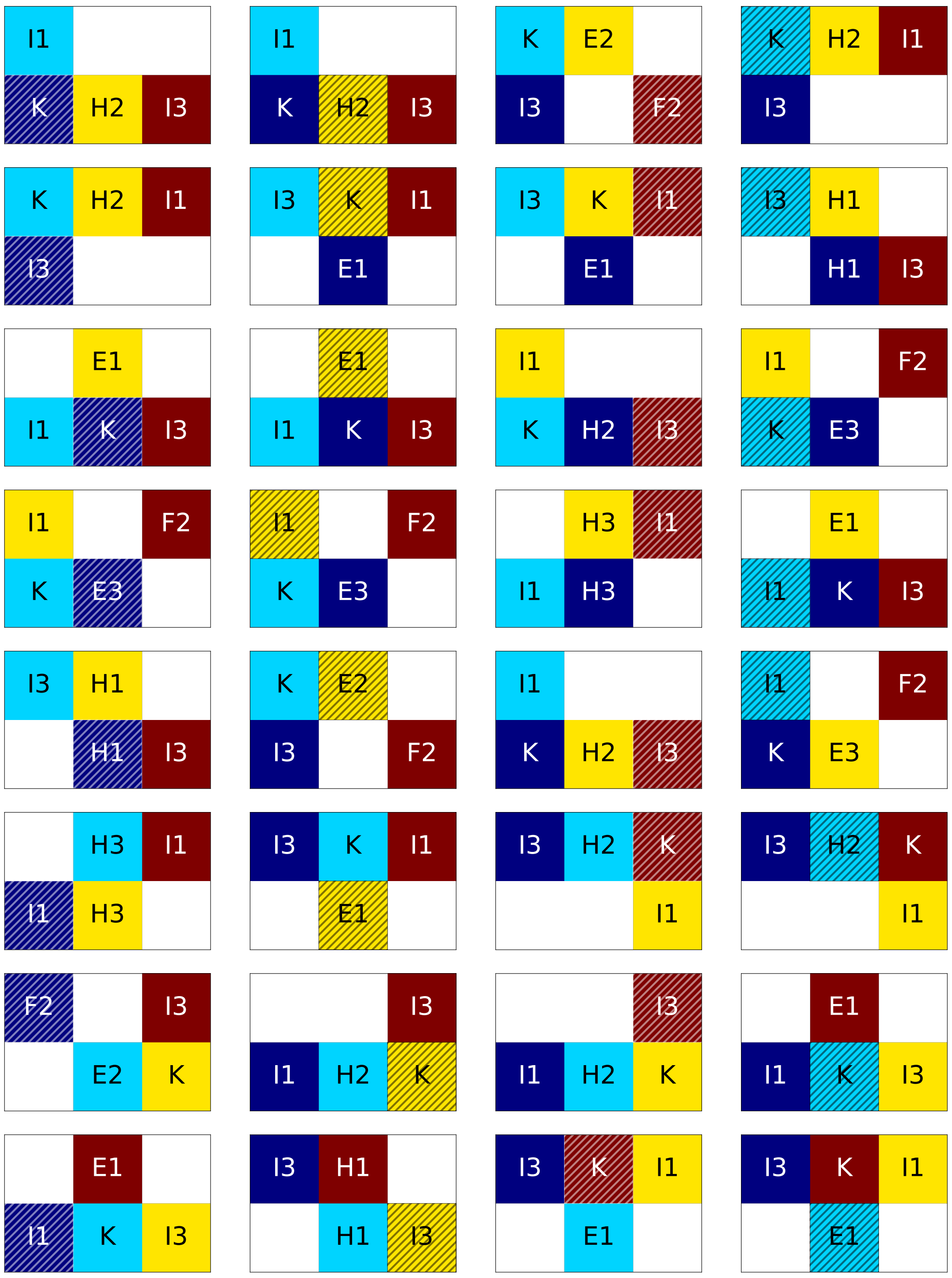


Fig. A.9 (I) Counter-example 22. Dynamic deadlock of period 44. Images show 0-31 of 44.

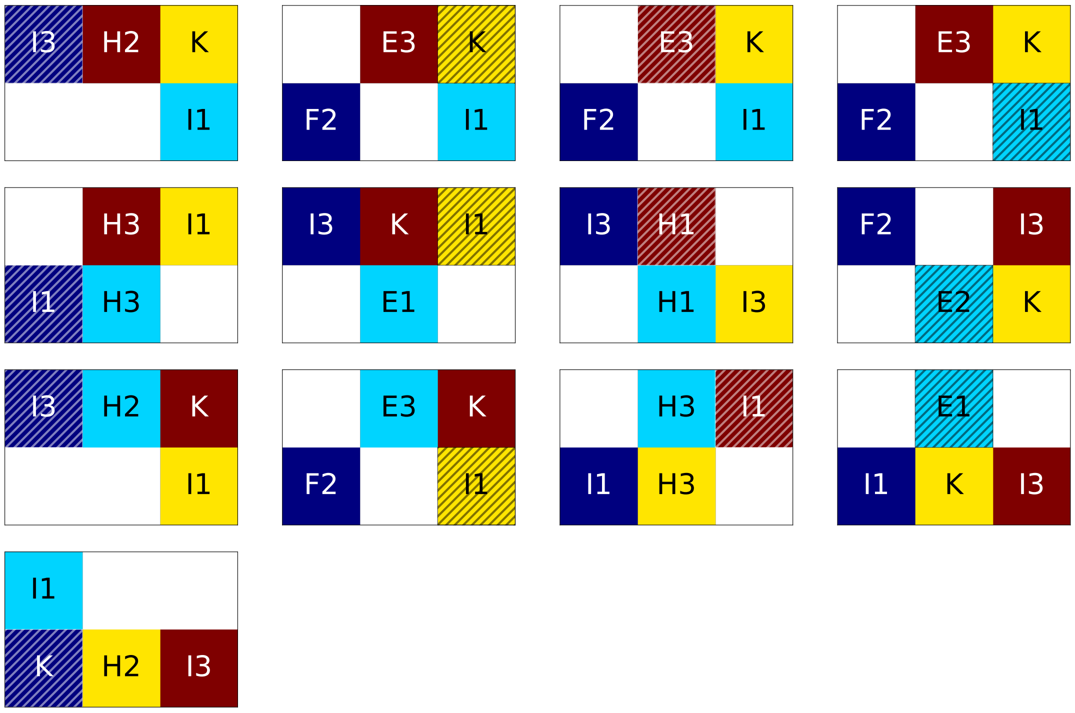


Fig. A.10 (II) Counter-example 22. Dynamic deadlock of period 44. Images show 32-44 of 44.

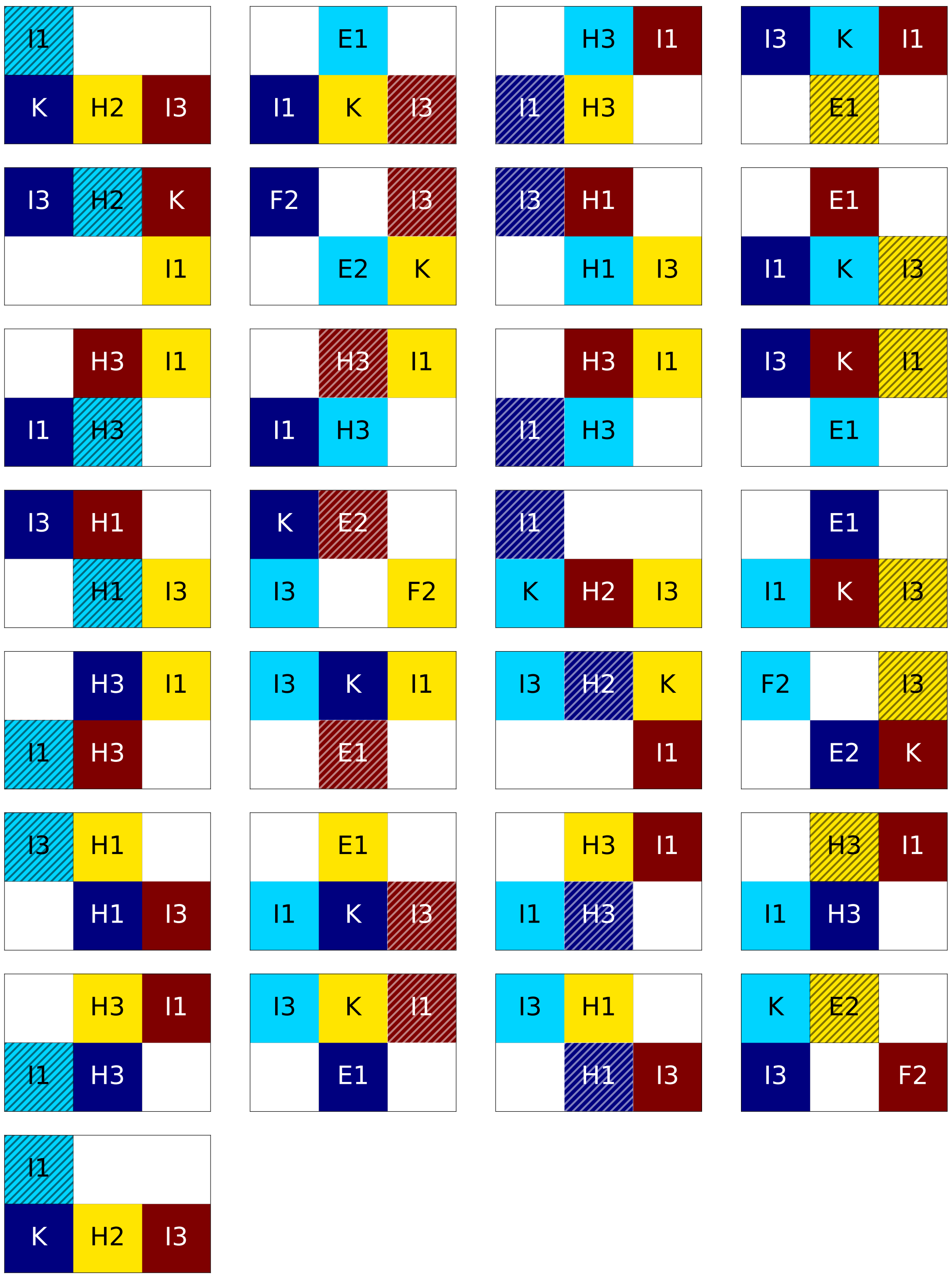


Fig. A.11 Counter-example 23. Dynamic deadlock of period 28. Images show 0-28 of 28.

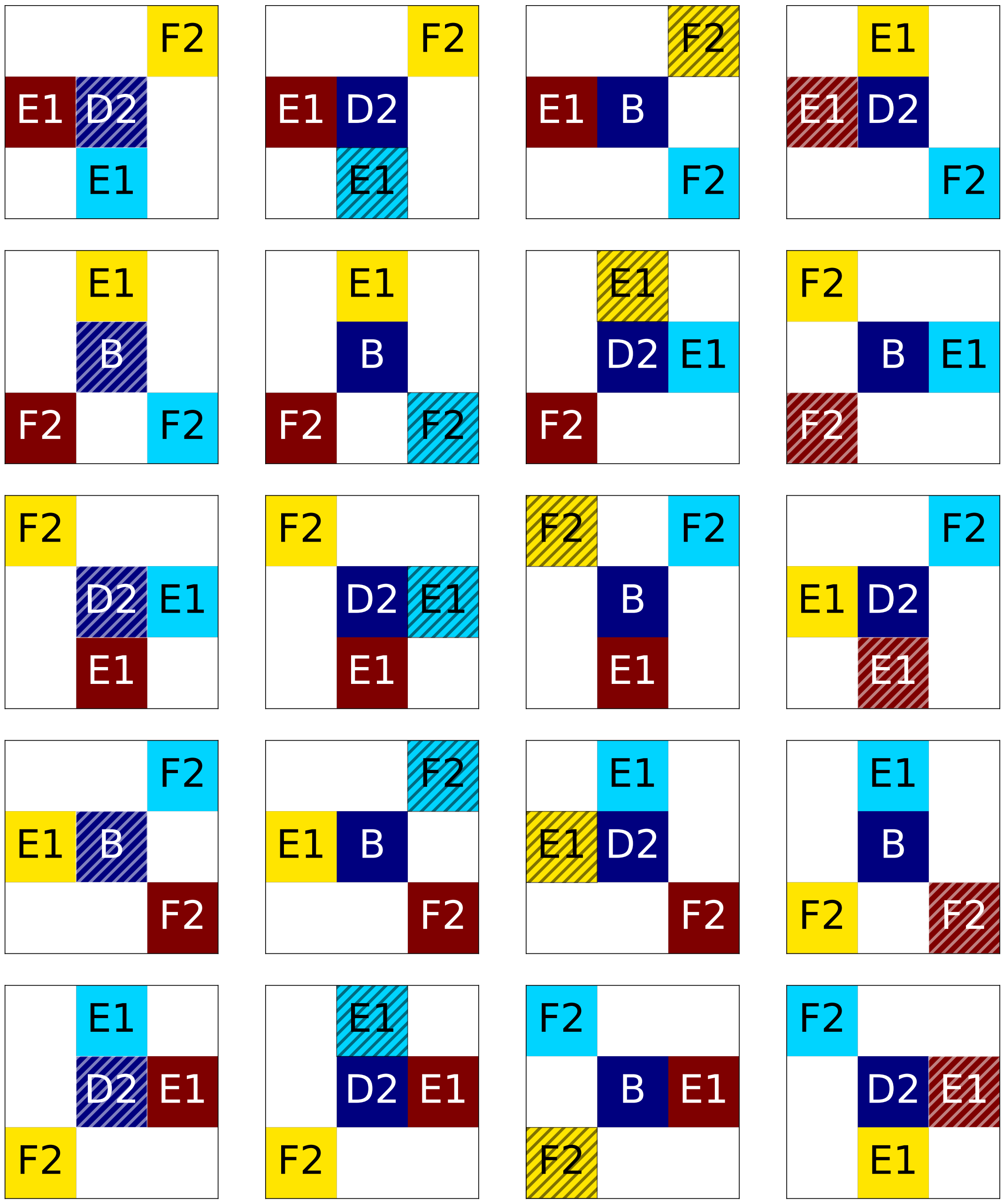


Fig. A.12 (I) Counter-example 24. Dynamic deadlock of period 32. Images show 0-19 of 32.

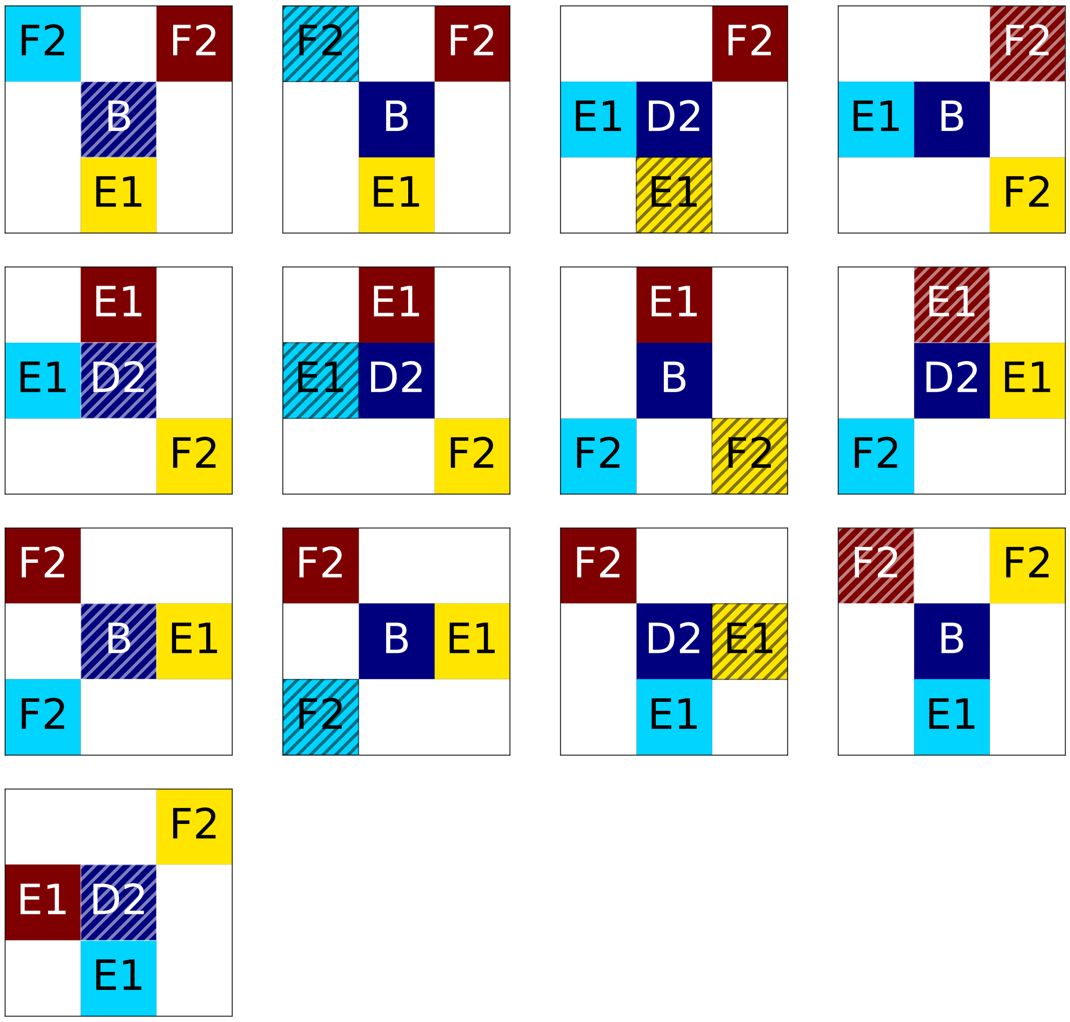


Fig. A.13 (II) Counter-example 24. Dynamic deadlock of period 32. Images show 20-32 of 32.

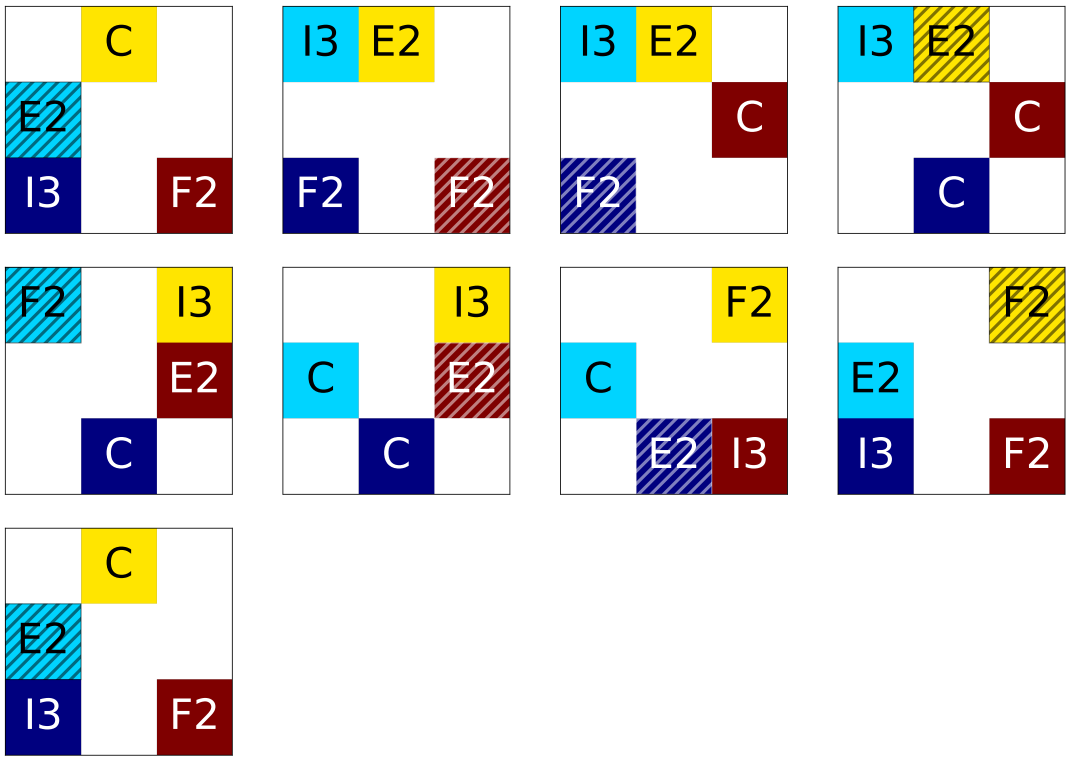


Fig. A.14 Counter-example 25. Dynamic deadlock of period 8. Images show 0-8 of 8.

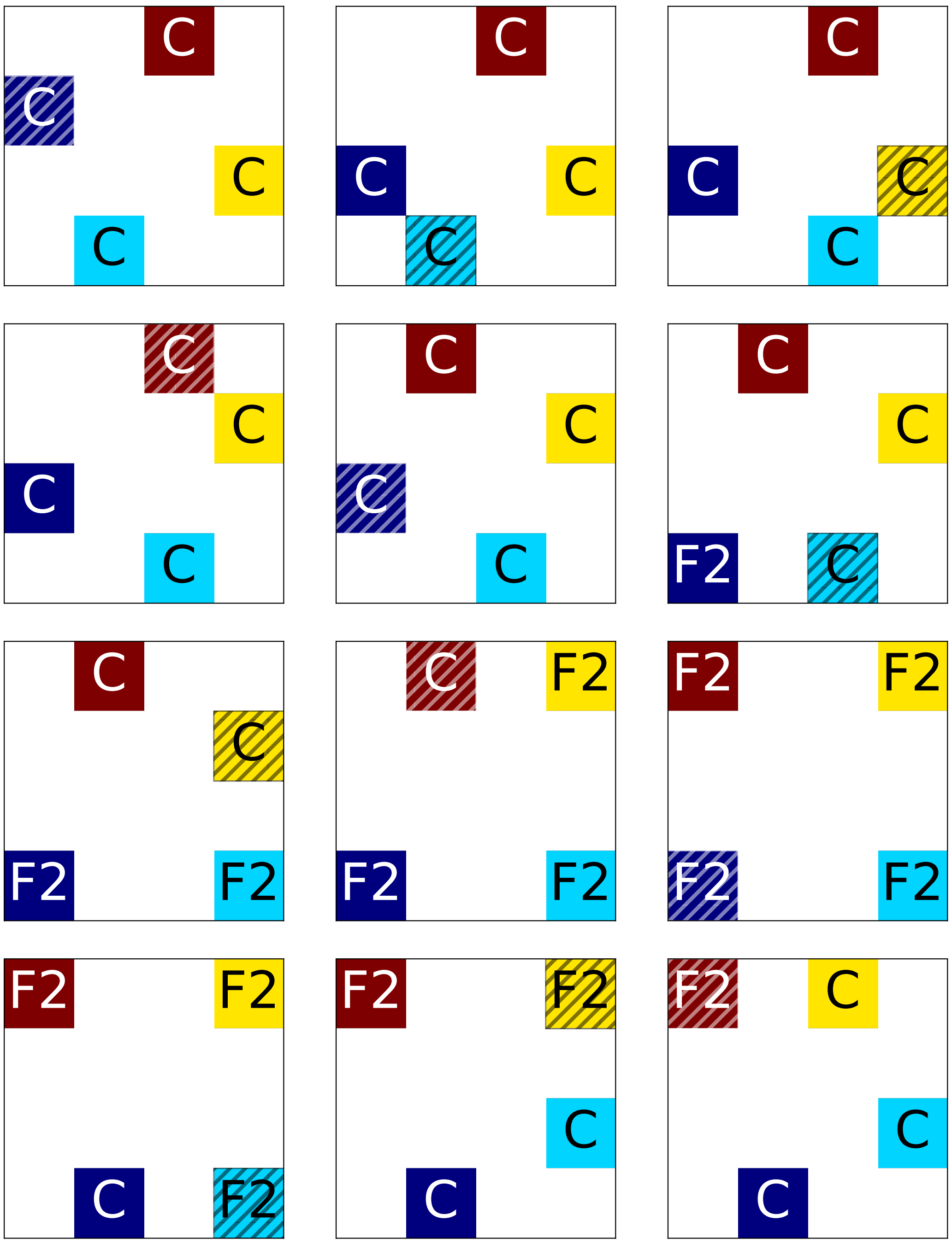


Fig. A.15 (I) Counter-example 26. Dynamic deadlock of period 48. Images show 0-11 of 48.

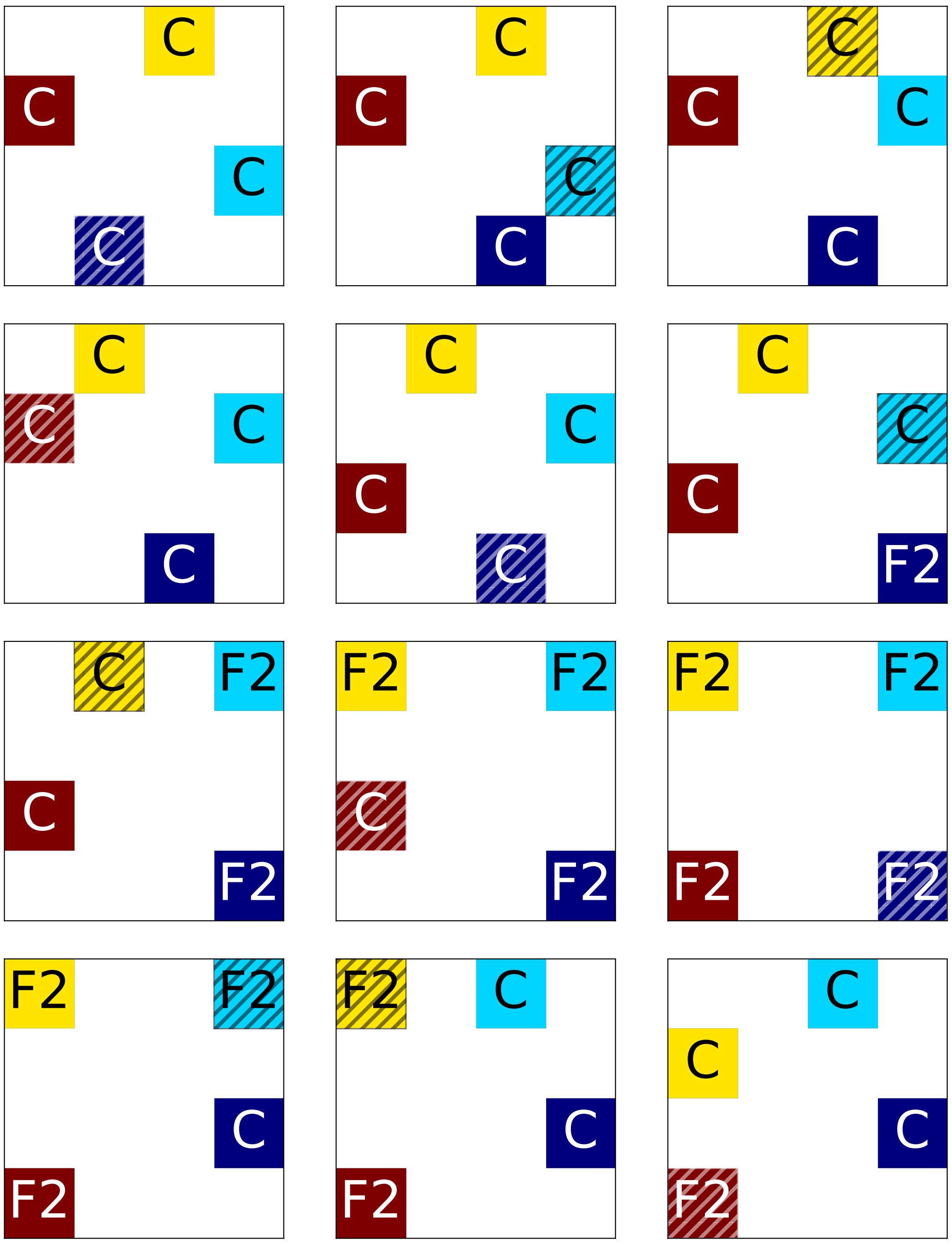


Fig. A.16 (II) Counter-example 26. Dynamic deadlock of period 48. Images show 12-23 of 48.

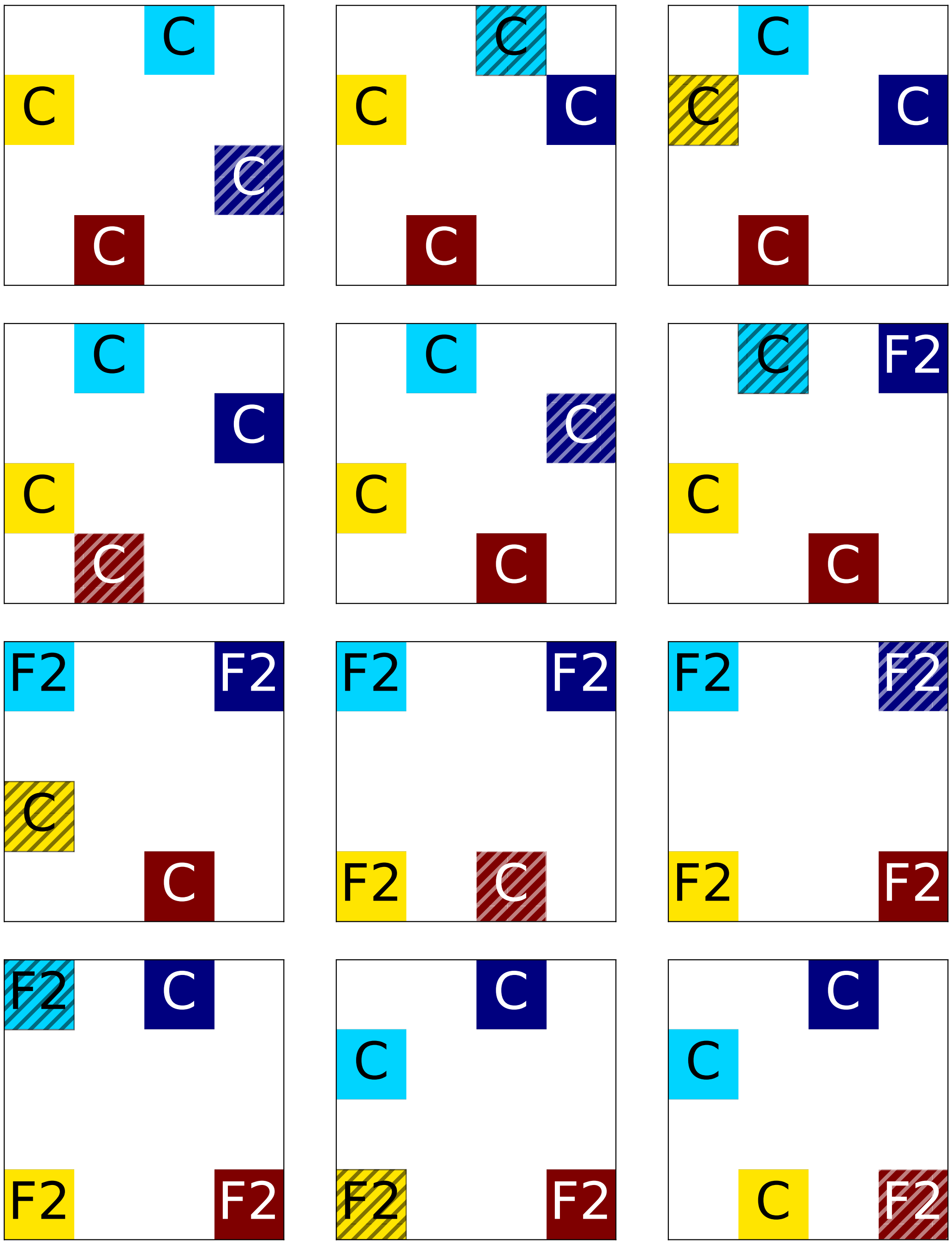


Fig. A.17 (III) Counter-example 26. Dynamic deadlock of period 48. Images show 24-35 of 48.

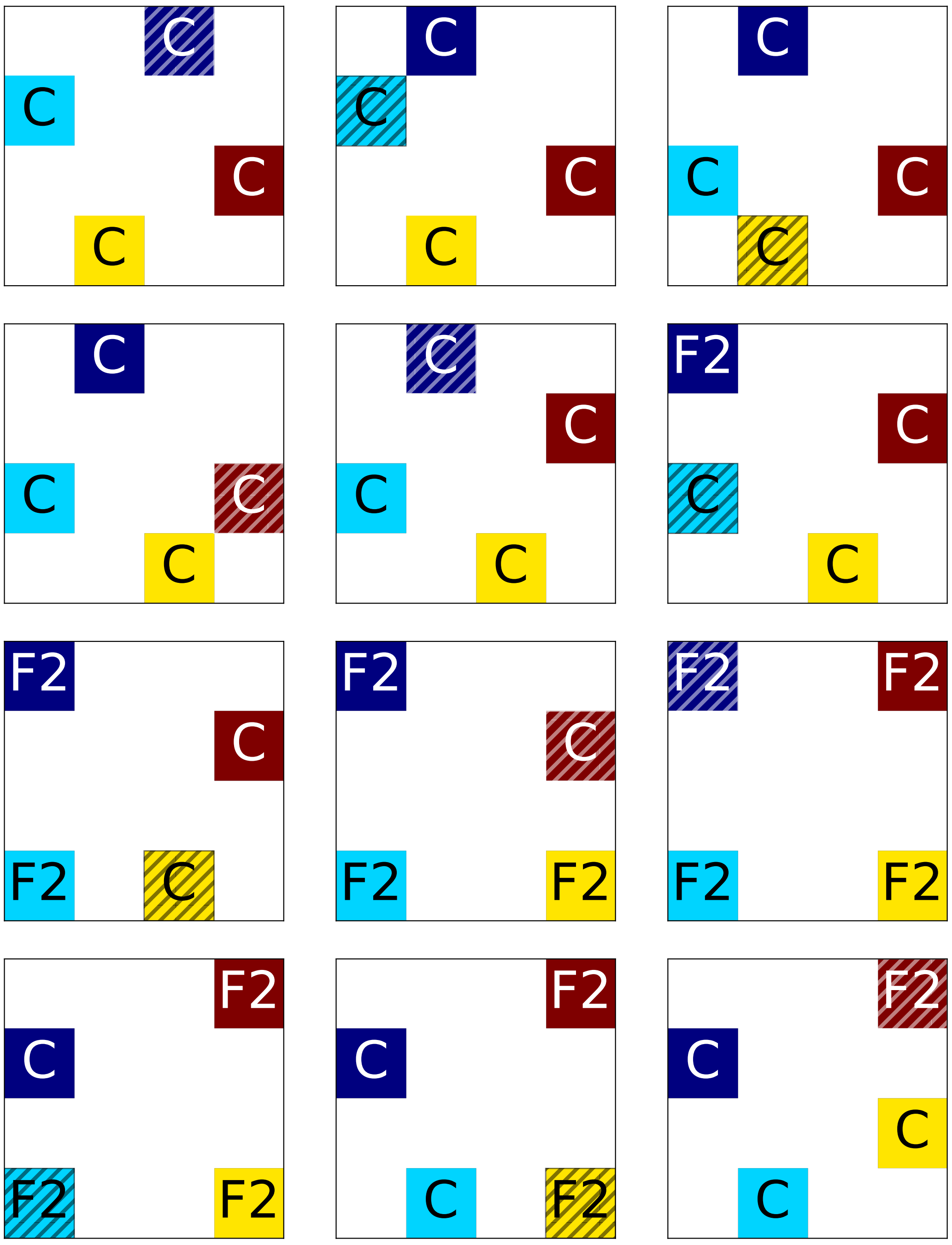


Fig. A.18 (IV) Counter-example 26. Dynamic deadlock of period 48. Images show 36-47 of 48.

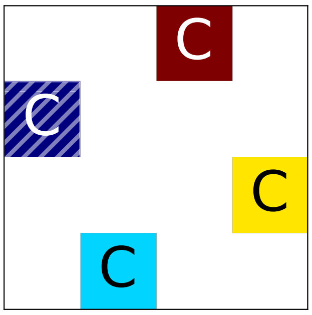


Fig. A.19 (V) Counter-example 26. Dynamic deadlock of period 48. Images show 48-48 of 48.

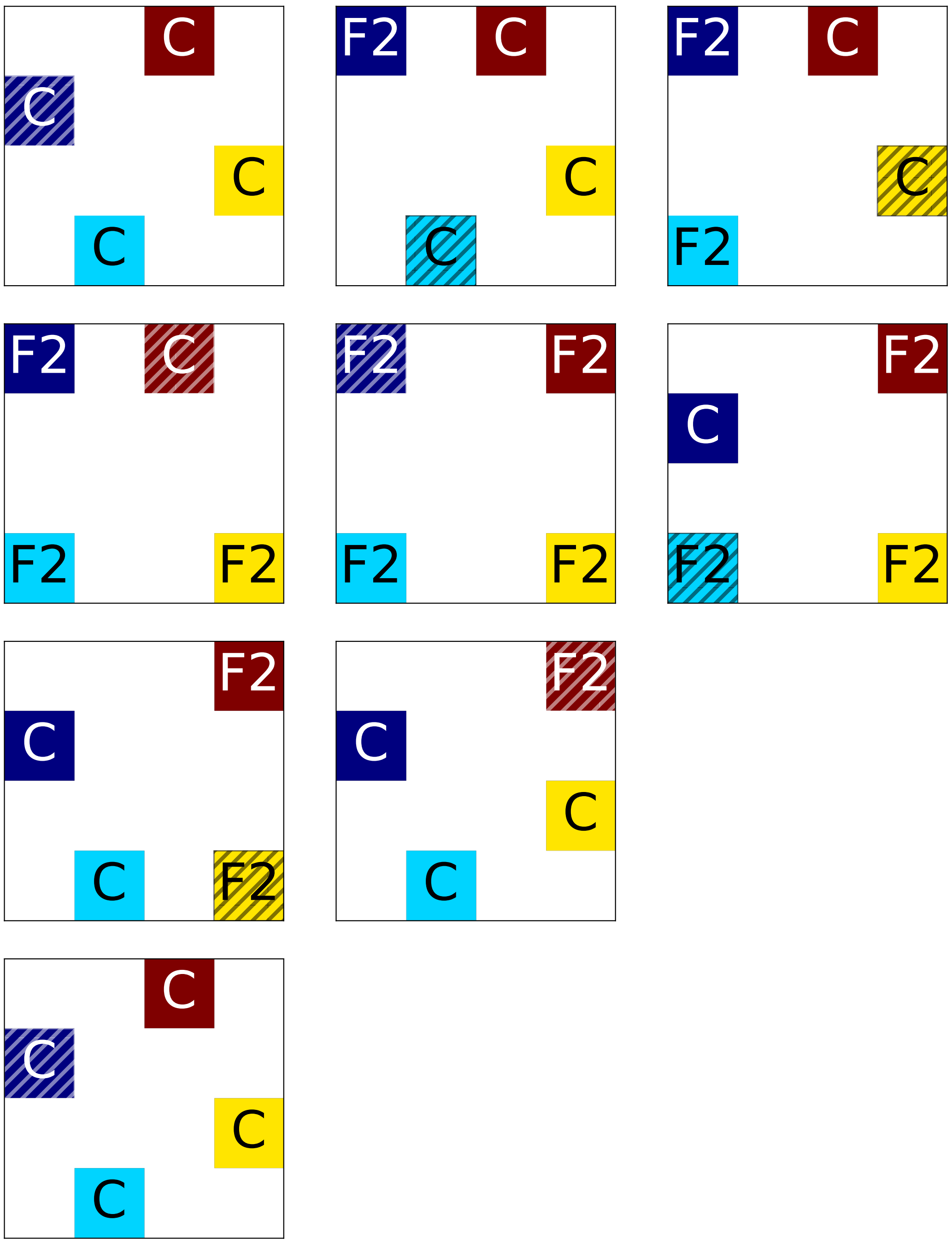


Fig. A.20 Counter-example 27. Dynamic deadlock of period 8. Images show 0-8 of 8.

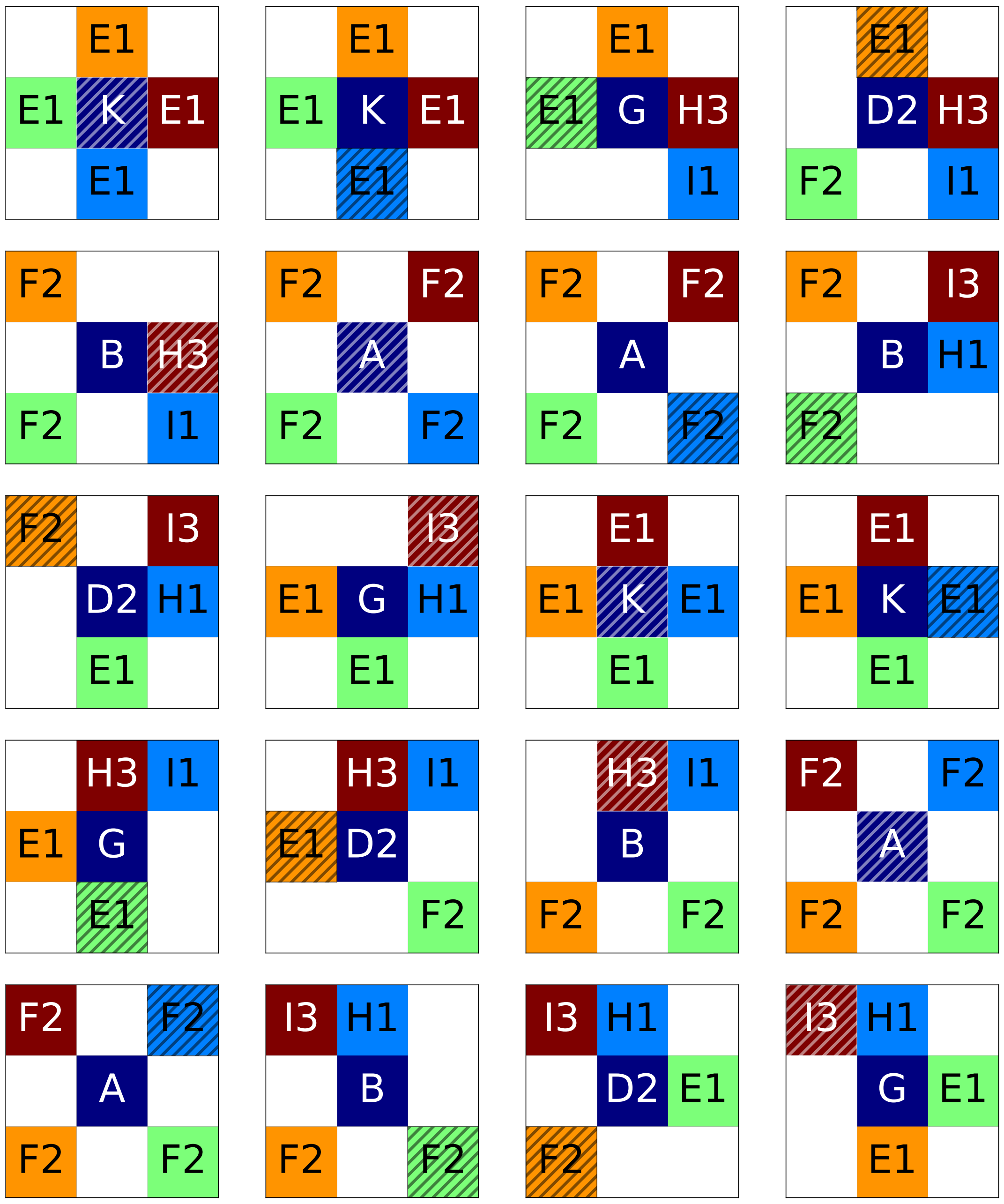


Fig. A.21 (I) Counter-example 28. Dynamic deadlock of period 40. Images show 0-19 of 40.

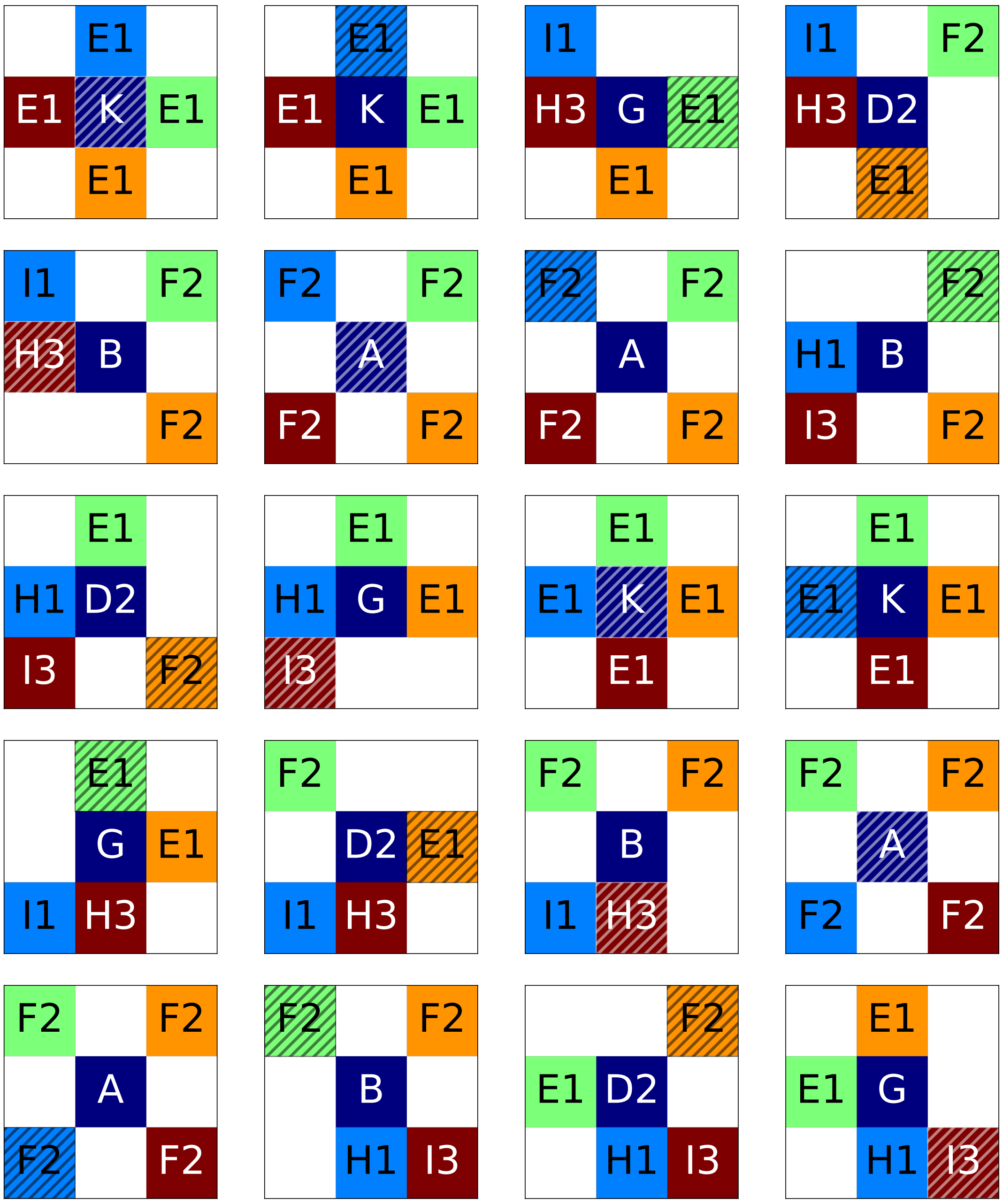


Fig. A.22 (II) Counter-example 28. Dynamic deadlock of period 40. Images show 20-39 of 40.

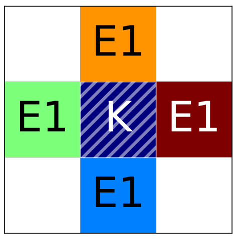


Fig. A.23 (III) Counter-example 28. Dynamic deadlock of period 40. Images show 40-40 of 40.

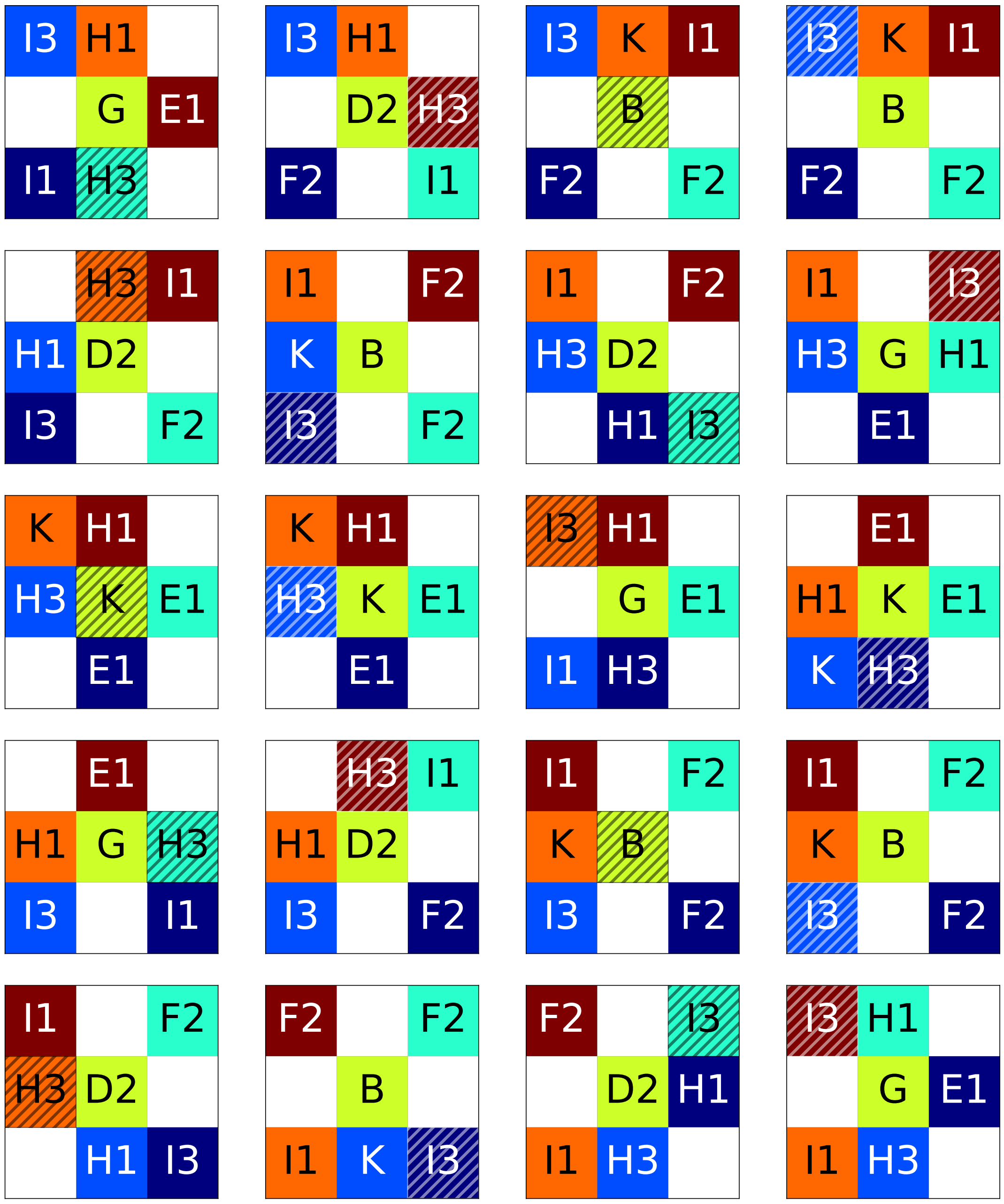


Fig. A.24 (I) Counter-example 29. Dynamic deadlock of period 48. Images show 0-19 of 48.

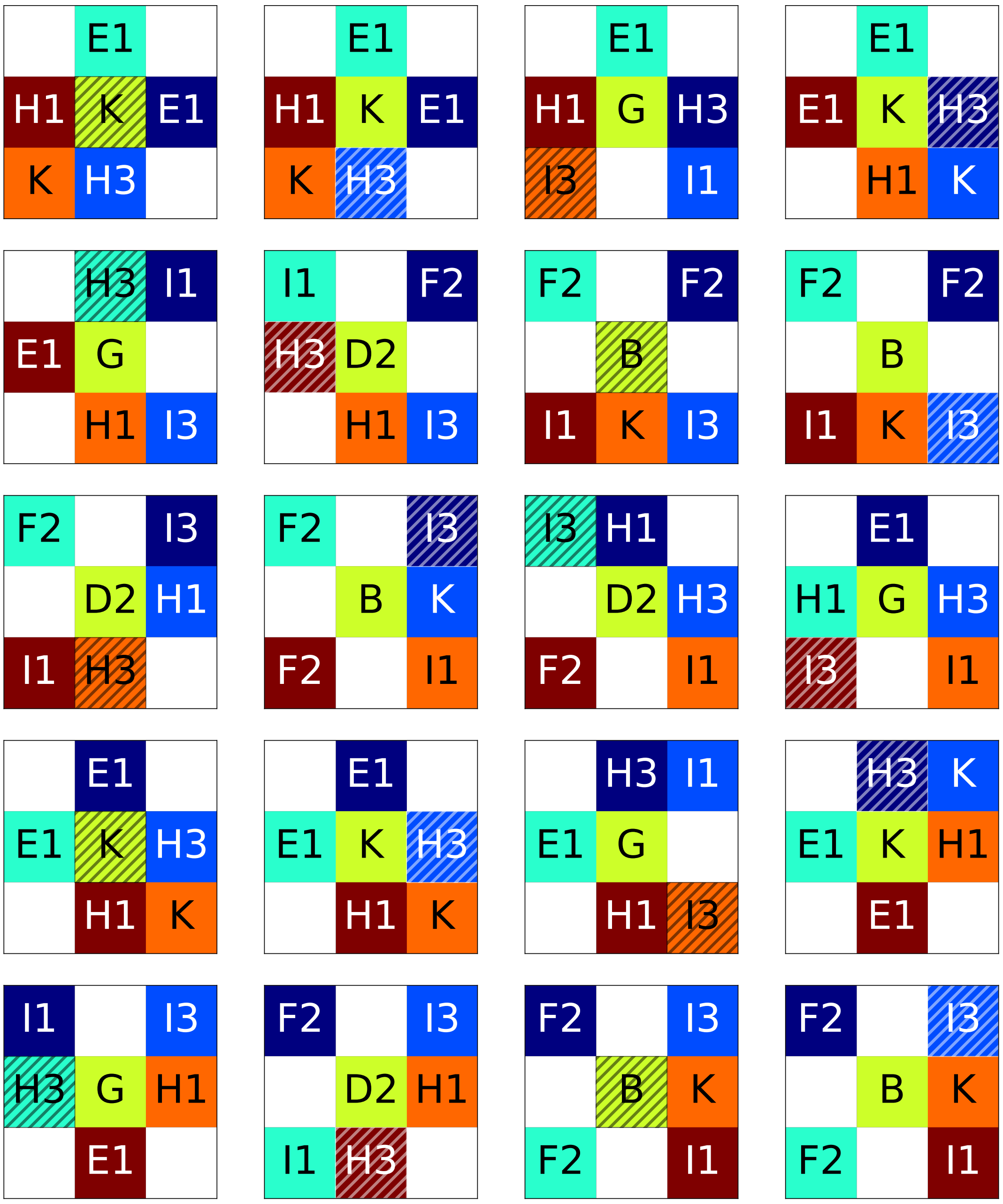


Fig. A.25 (II) Counter-example 29. Dynamic deadlock of period 48. Images show 20-39 of 48.

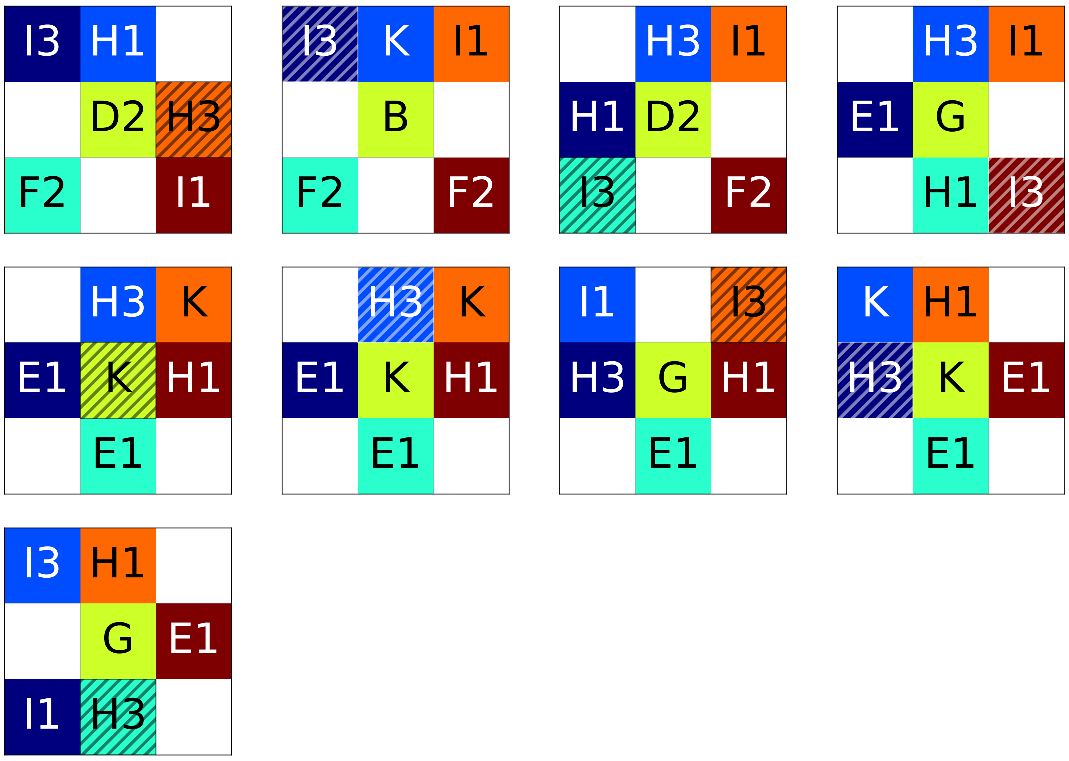


Fig. A.26 (III) Counter-example 29. Dynamic deadlock of period 48. Images show 40-48 of 48.

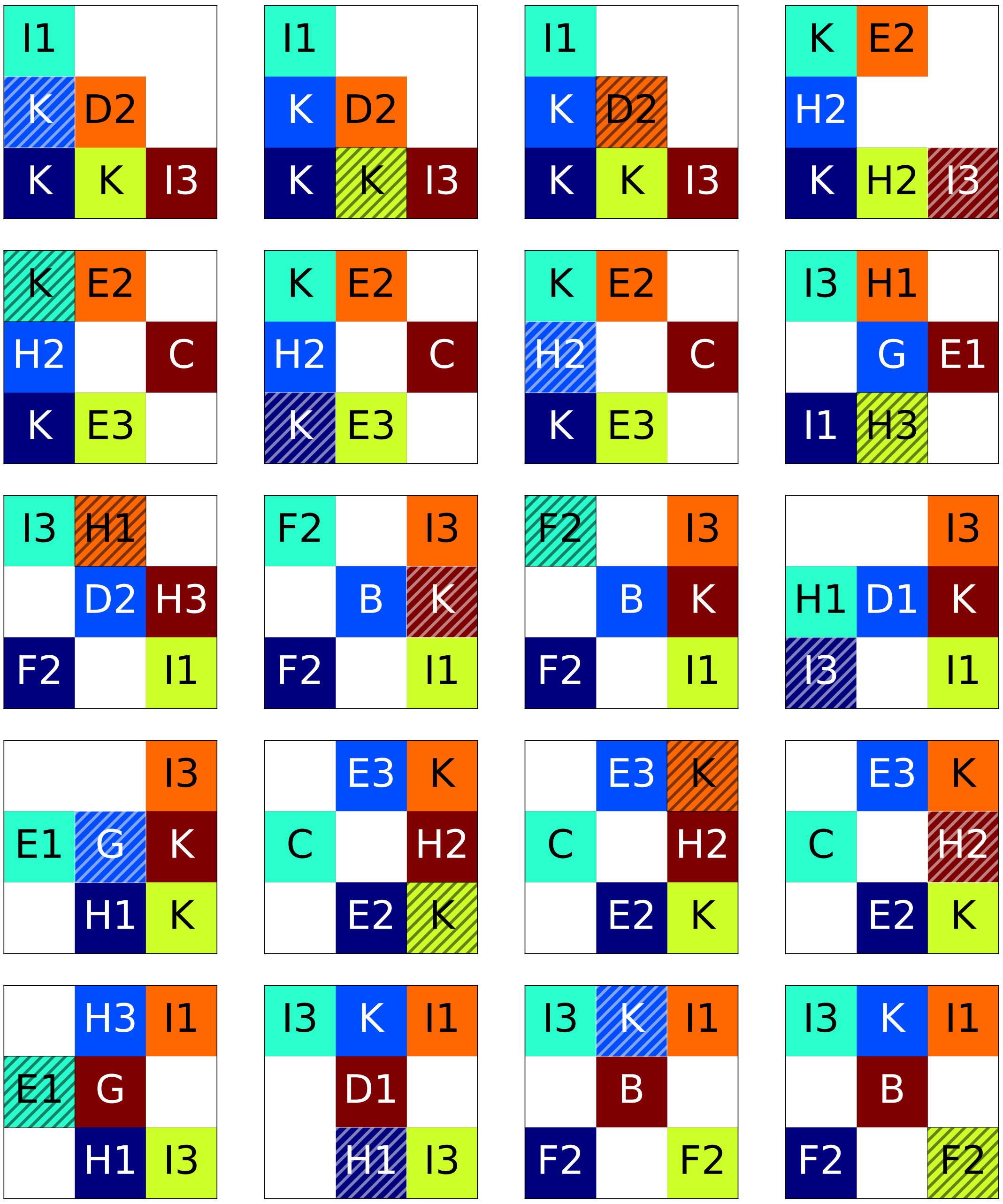


Fig. A.27 (I) Counter-example 30. Dynamic deadlock of period 126. Images show 0-19 of 126.

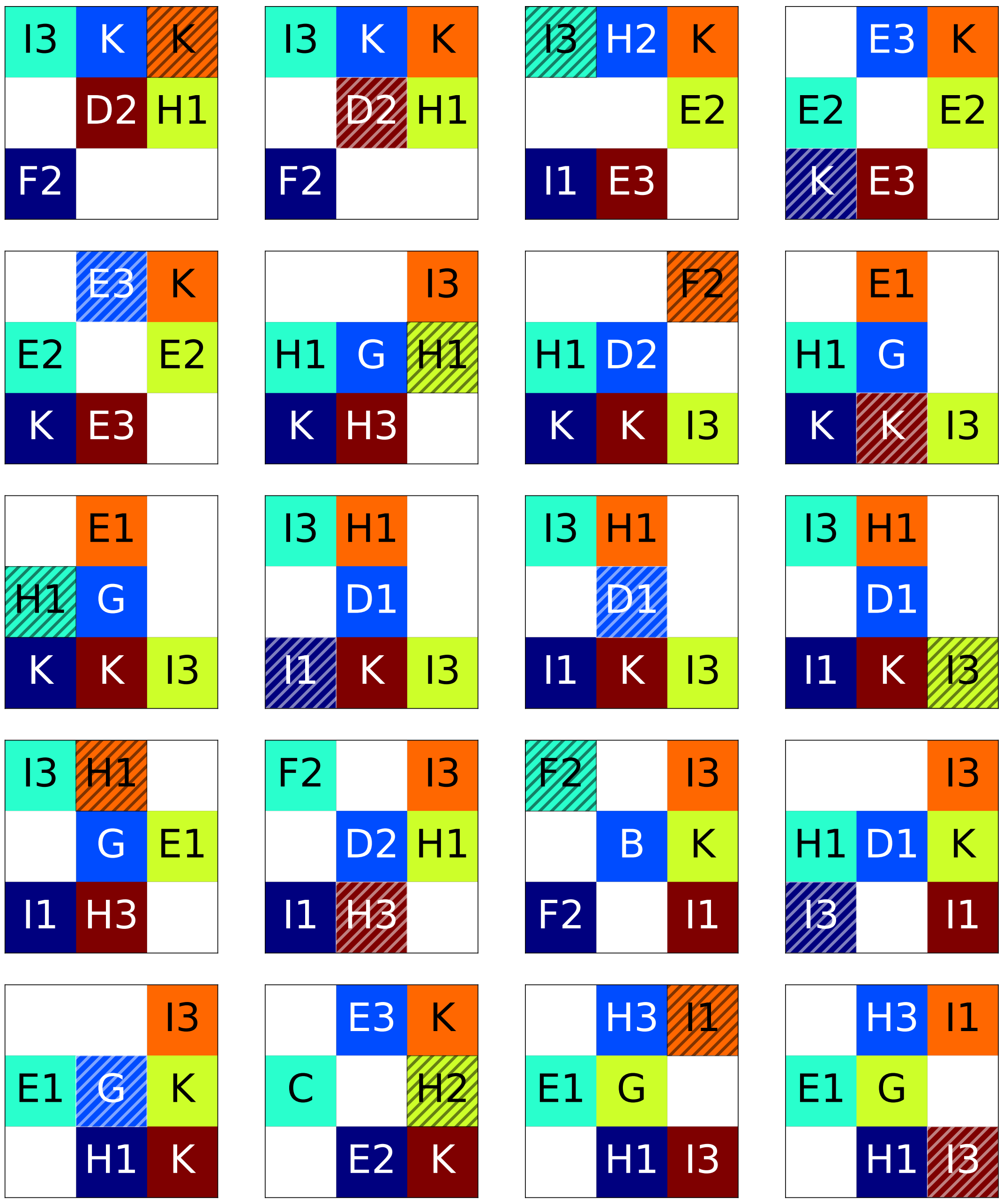


Fig. A.28 (II) Counter-example 30. Dynamic deadlock of period 126. Images show 20-39 of 126.

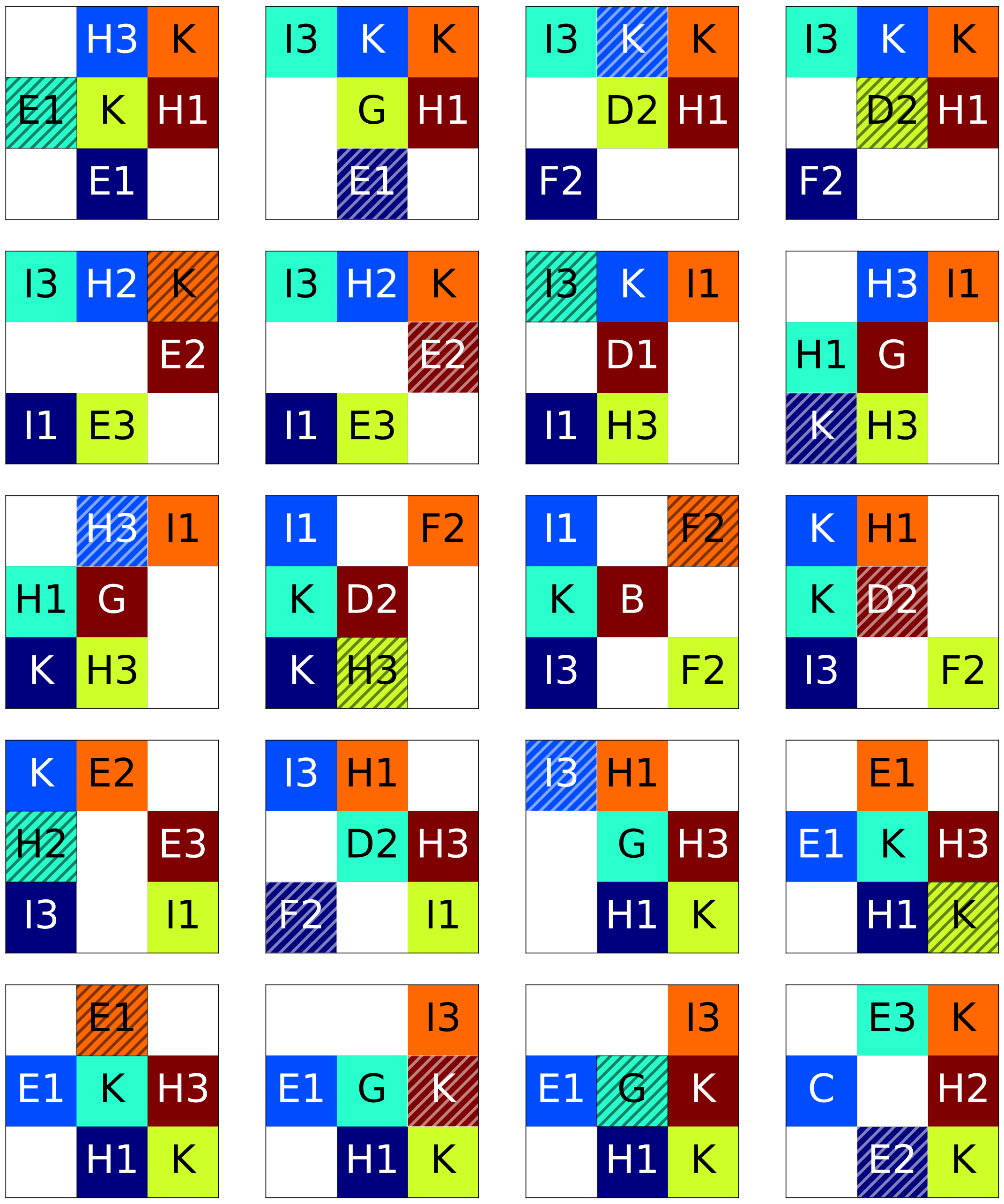


Fig. A.29 (III) Counter-example 30. Dynamic deadlock of period 126. Images show 40-59 of 126.

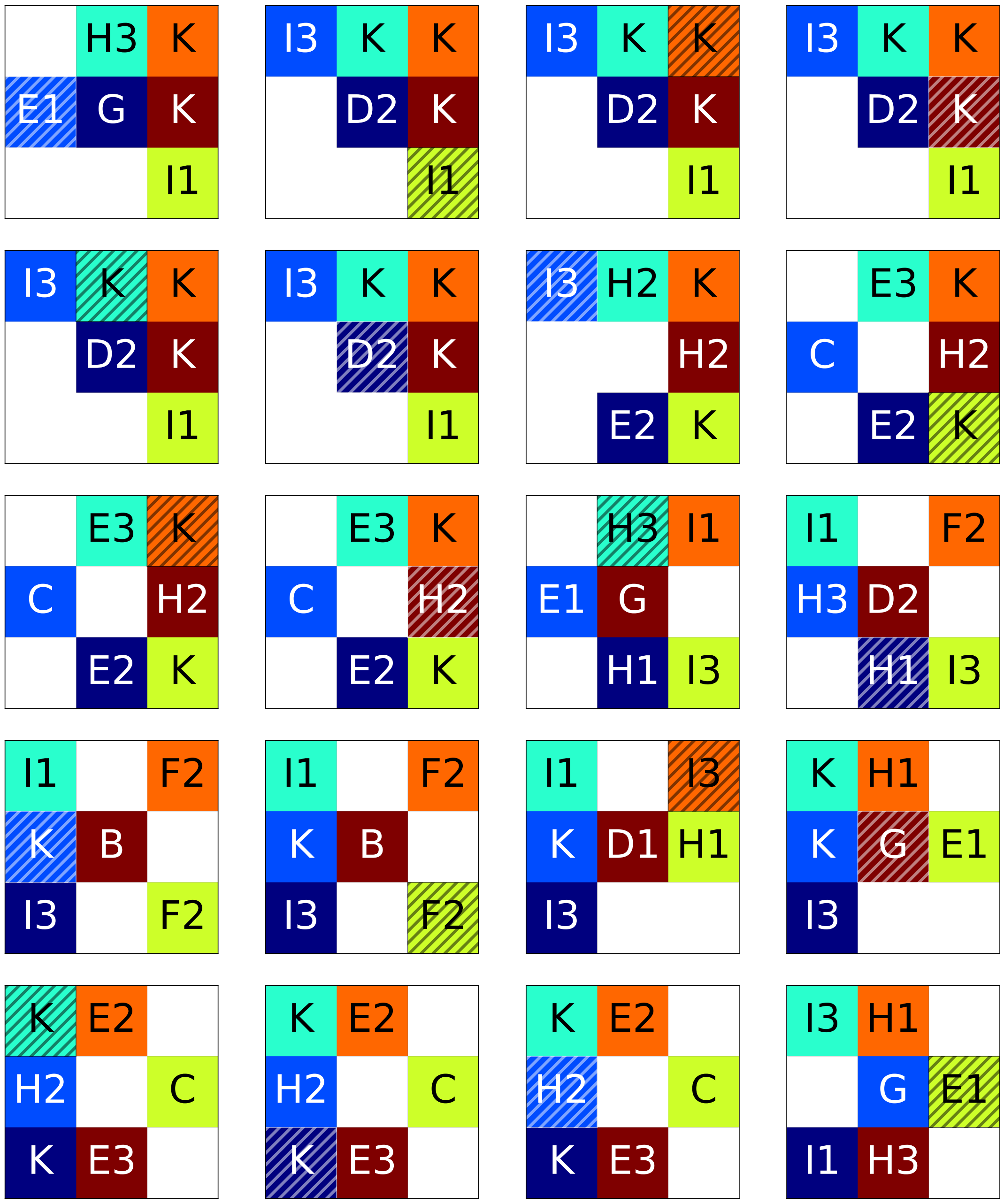


Fig. A.30 (IV) Counter-example 30. Dynamic deadlock of period 126. Images show 60-79 of 126.

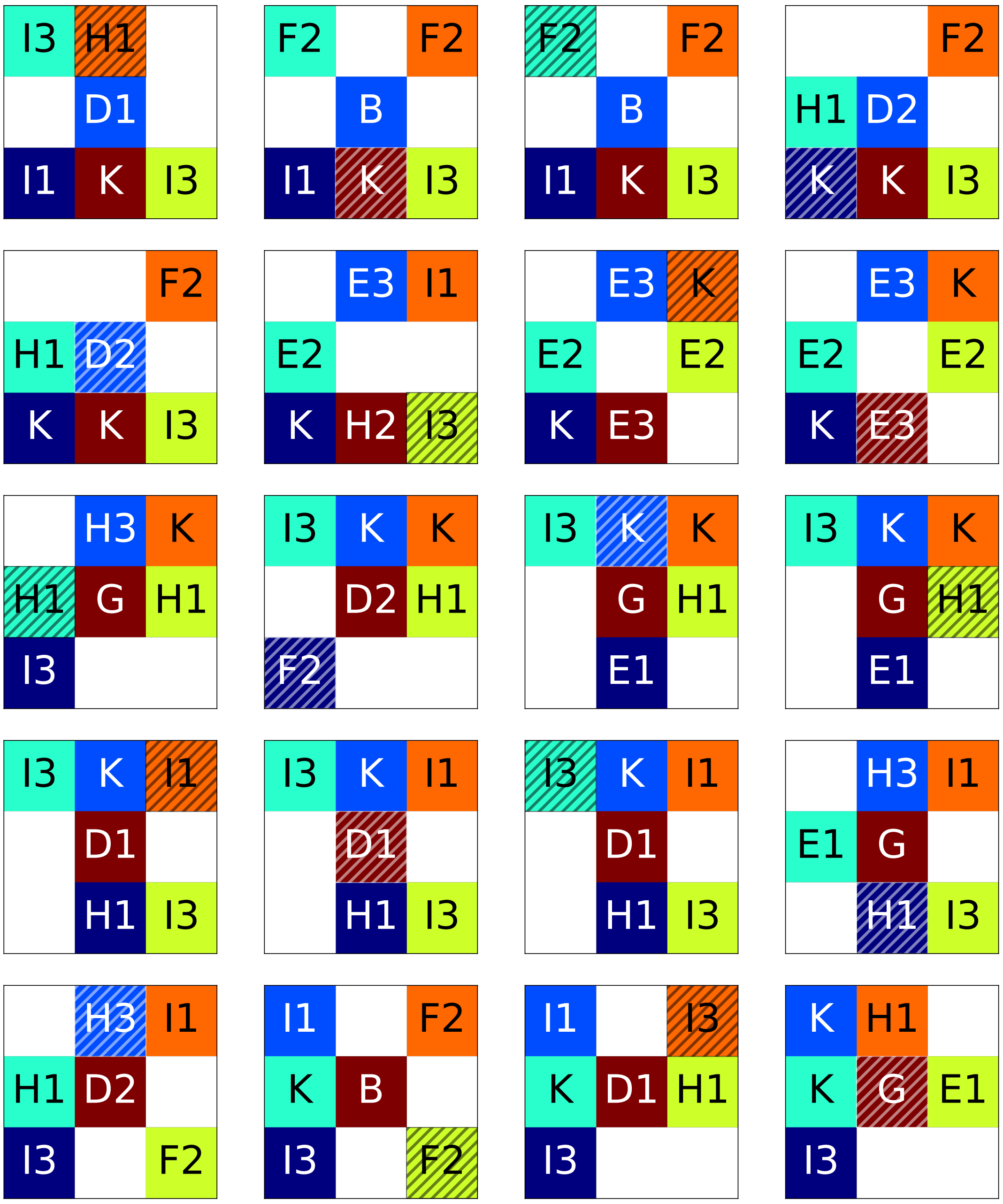


Fig. A.31 (V) Counter-example 30. Dynamic deadlock of period 126. Images show 80-99 of 126.

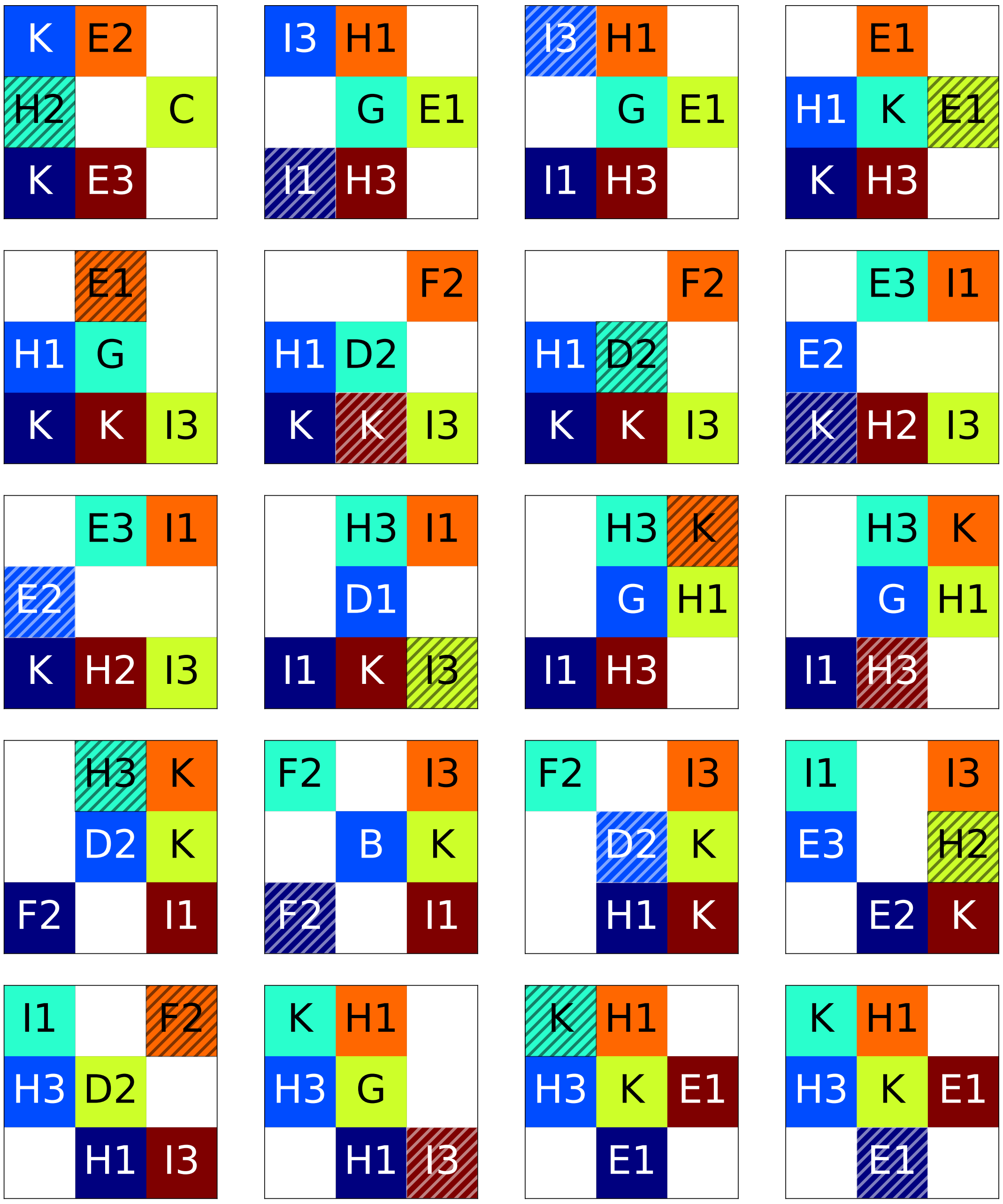


Fig. A.32 (VI) Counter-example 30. Dynamic deadlock of period 126. Images show 100-119 of 126.

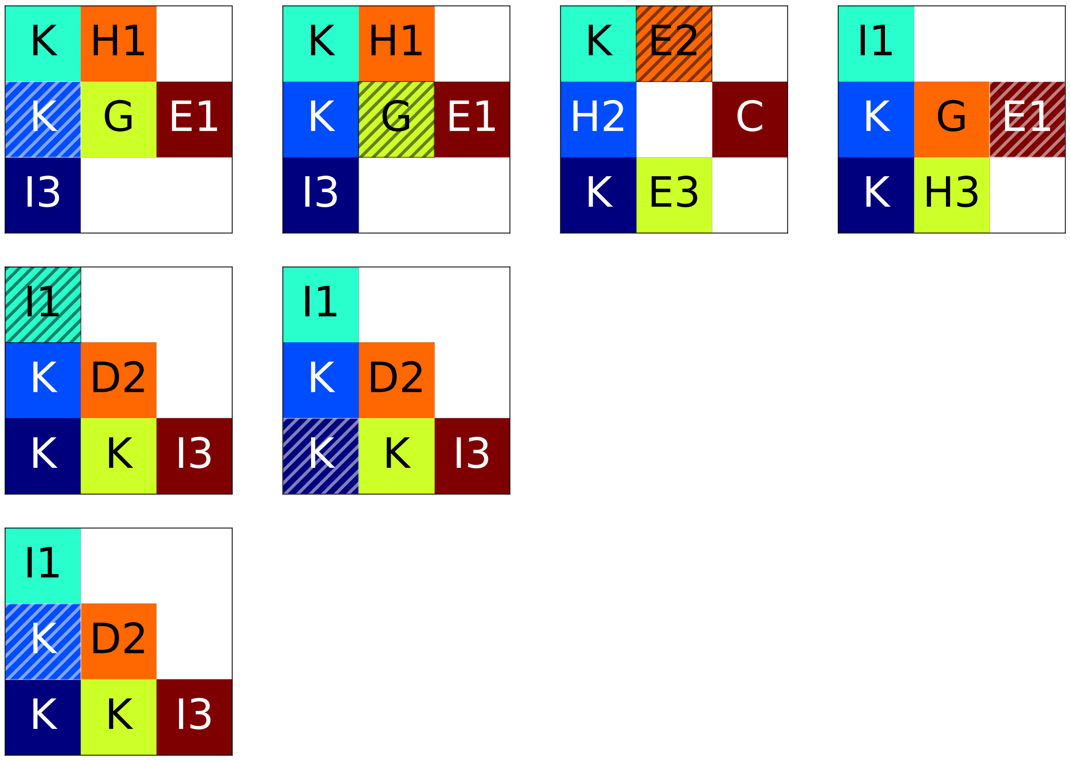


Fig. A.33 (VII) Counter-example 30. Dynamic deadlock of period 126. Images show 120-126 of 126.

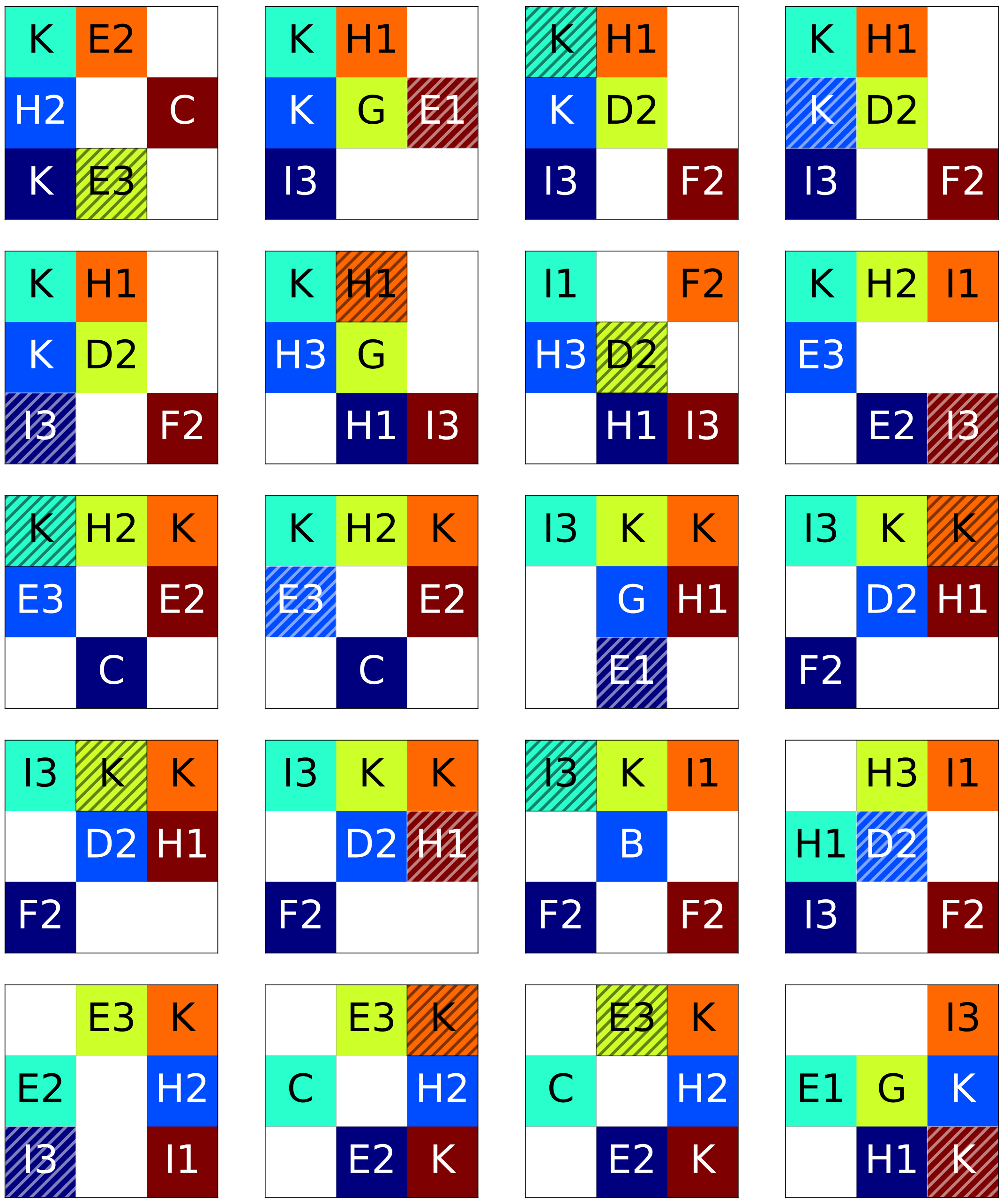


Fig. A.34 (I) Counter-example 31. Dynamic deadlock of period 36. Images show 0-19 of 36.

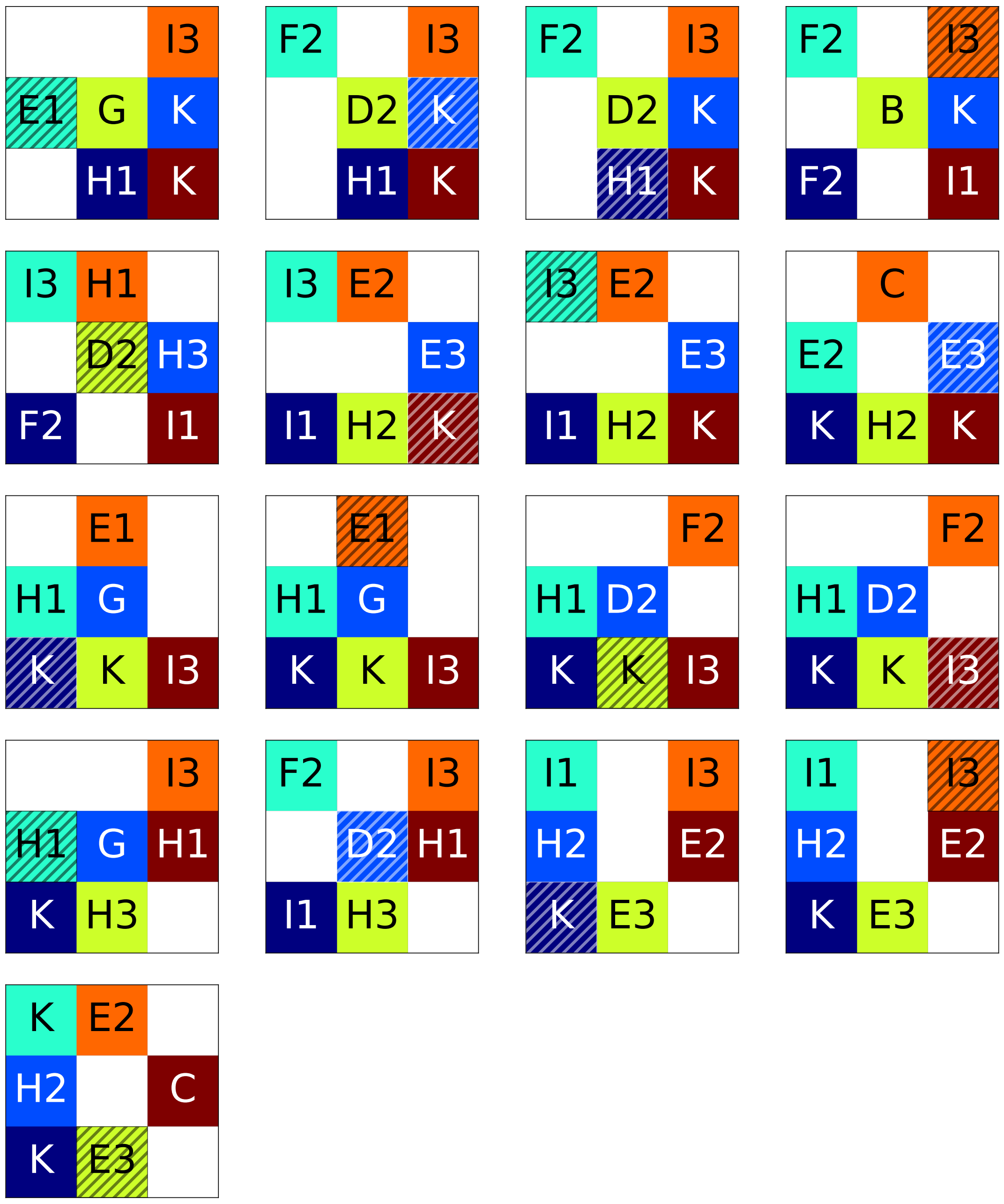


Fig. A.35 (II) Counter-example 31. Dynamic deadlock of period 36. Images show 20-36 of 36.

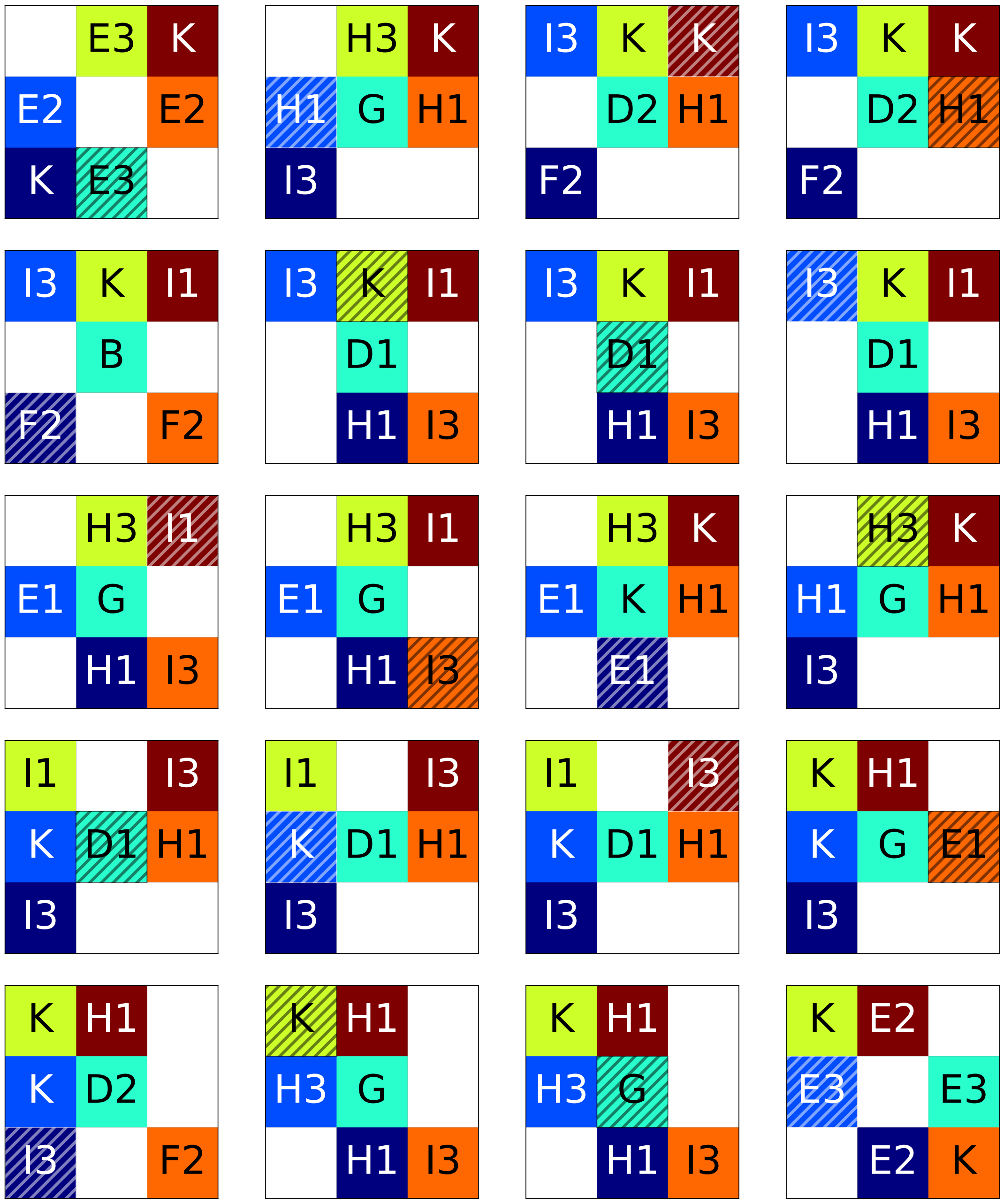


Fig. A.36 (I) Counter-example 32. Dynamic deadlock of period 228. Images show 0-19 of 228.

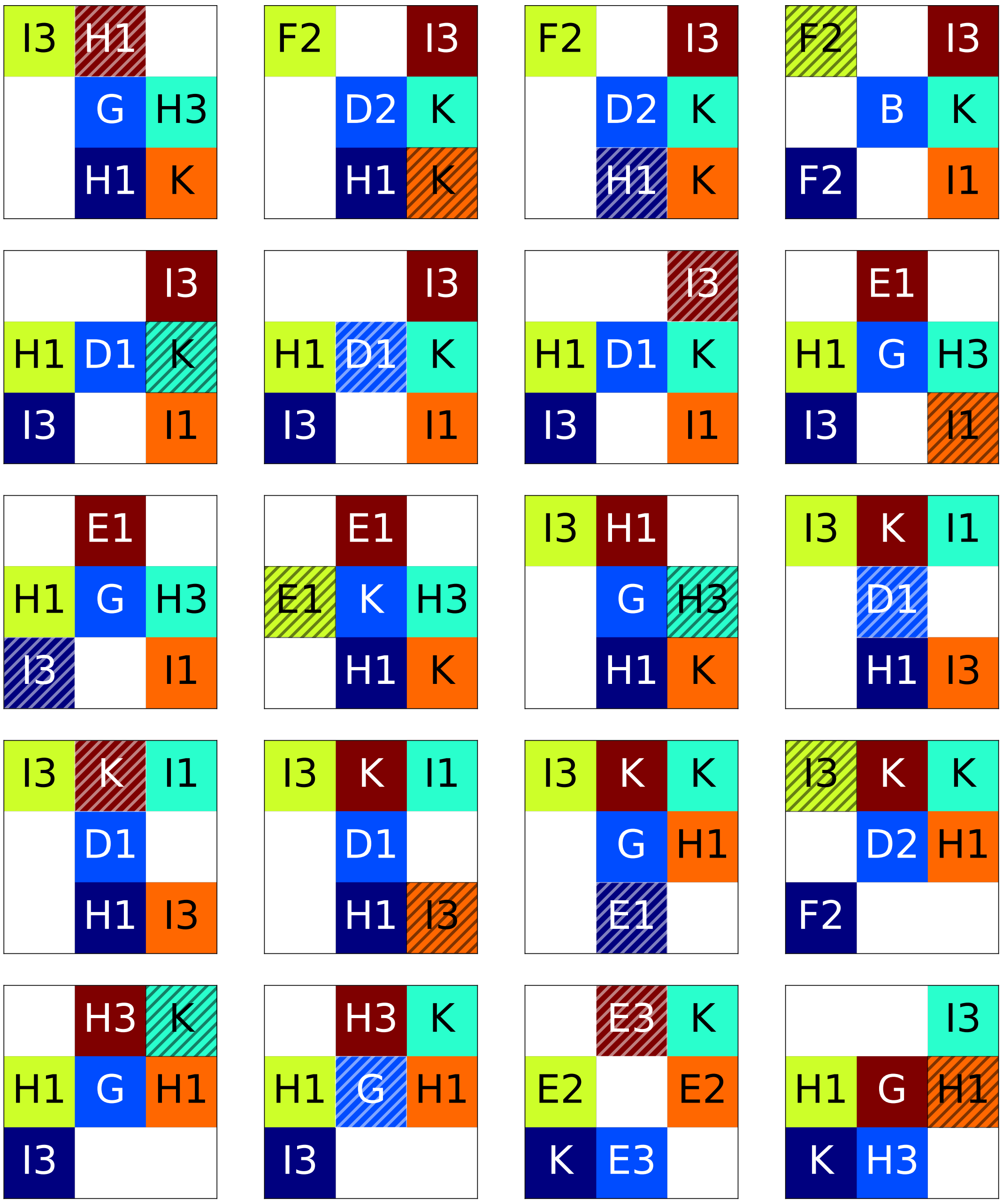


Fig. A.37 (II) Counter-example 32. Dynamic deadlock of period 228. Images show 20-39 of 228.

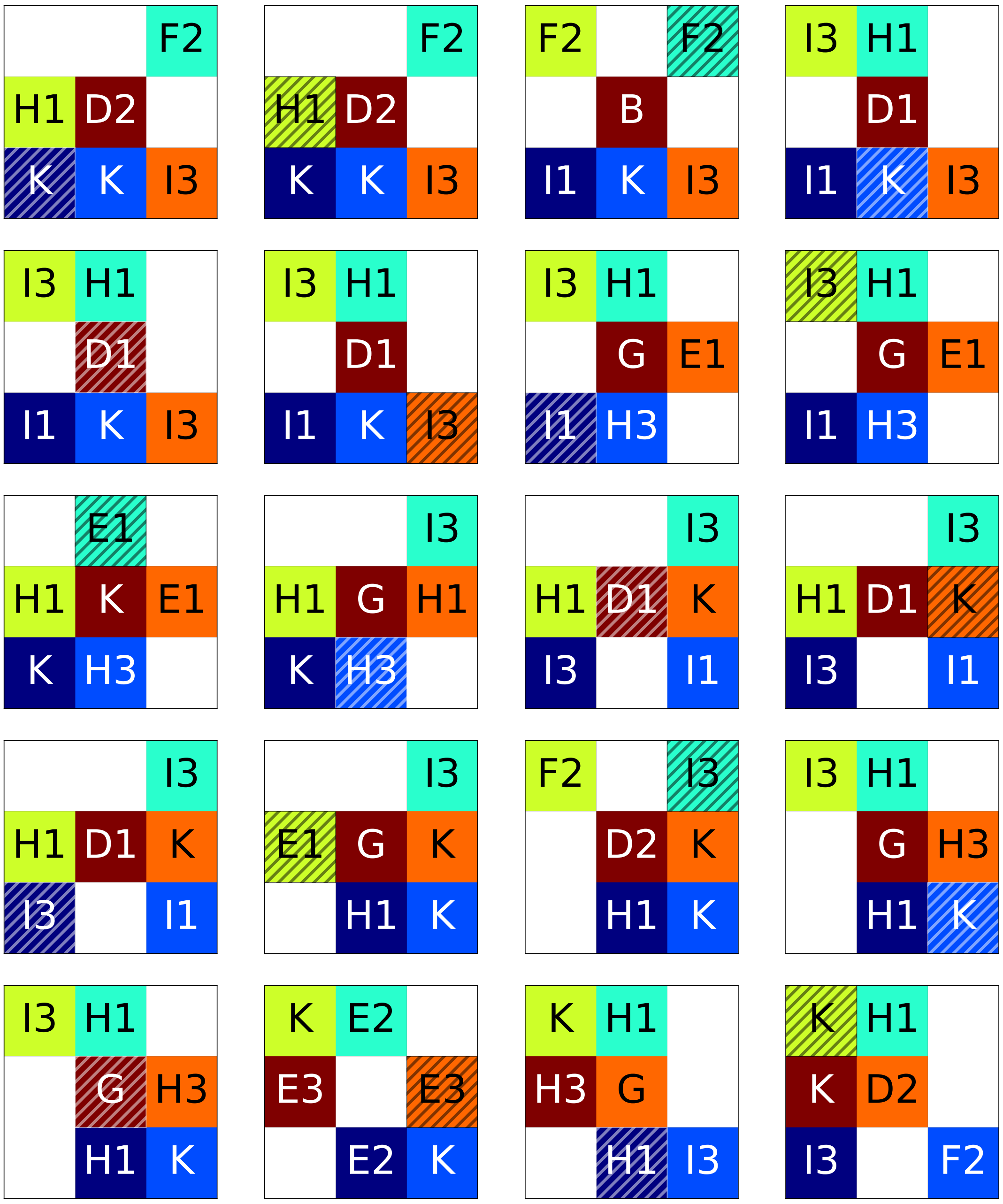


Fig. A.38 (III) Counter-example 32. Dynamic deadlock of period 228. Images show 40-59 of 228.

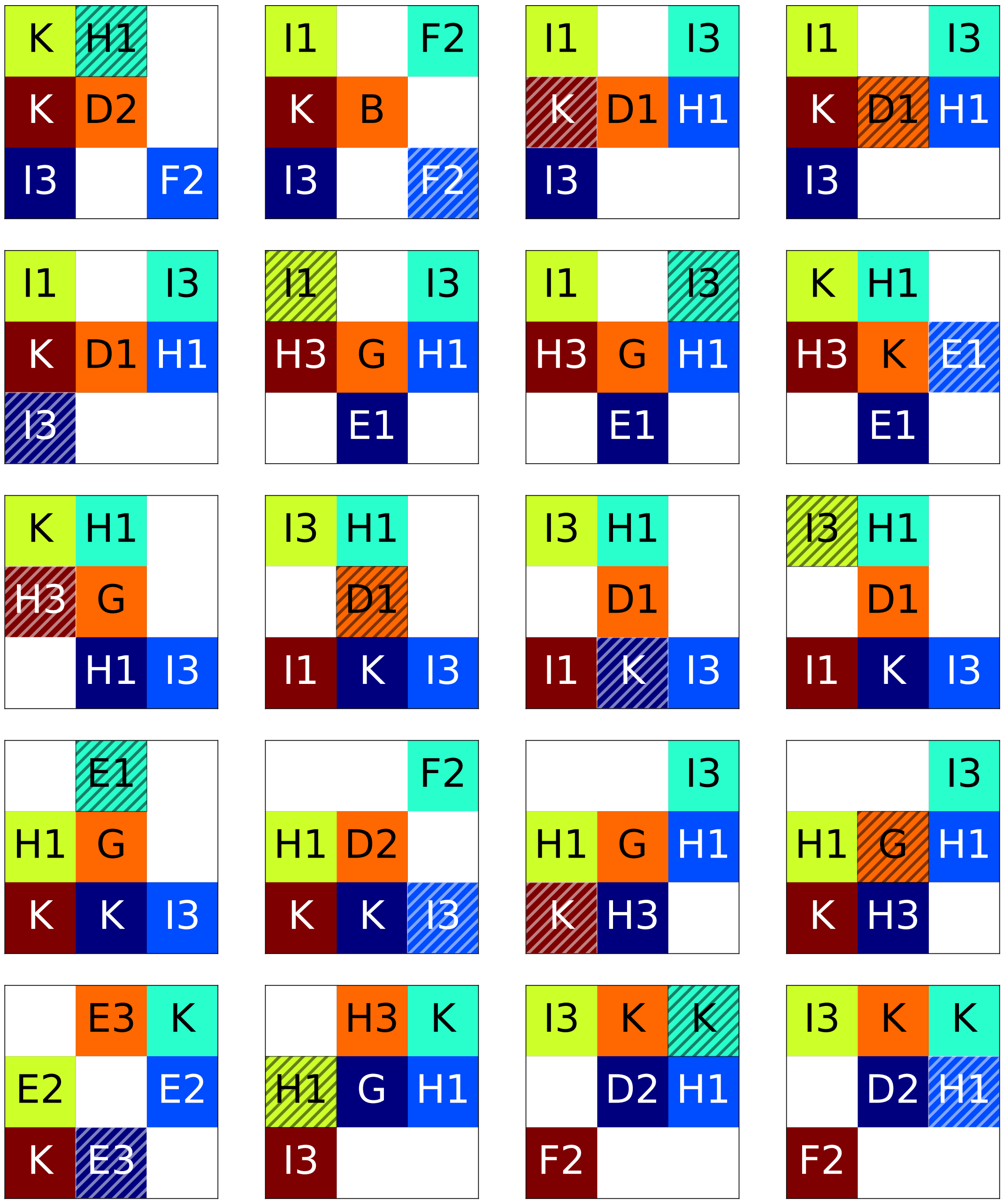


Fig. A.39 (IV) Counter-example 32. Dynamic deadlock of period 228. Images show 60-79 of 228.

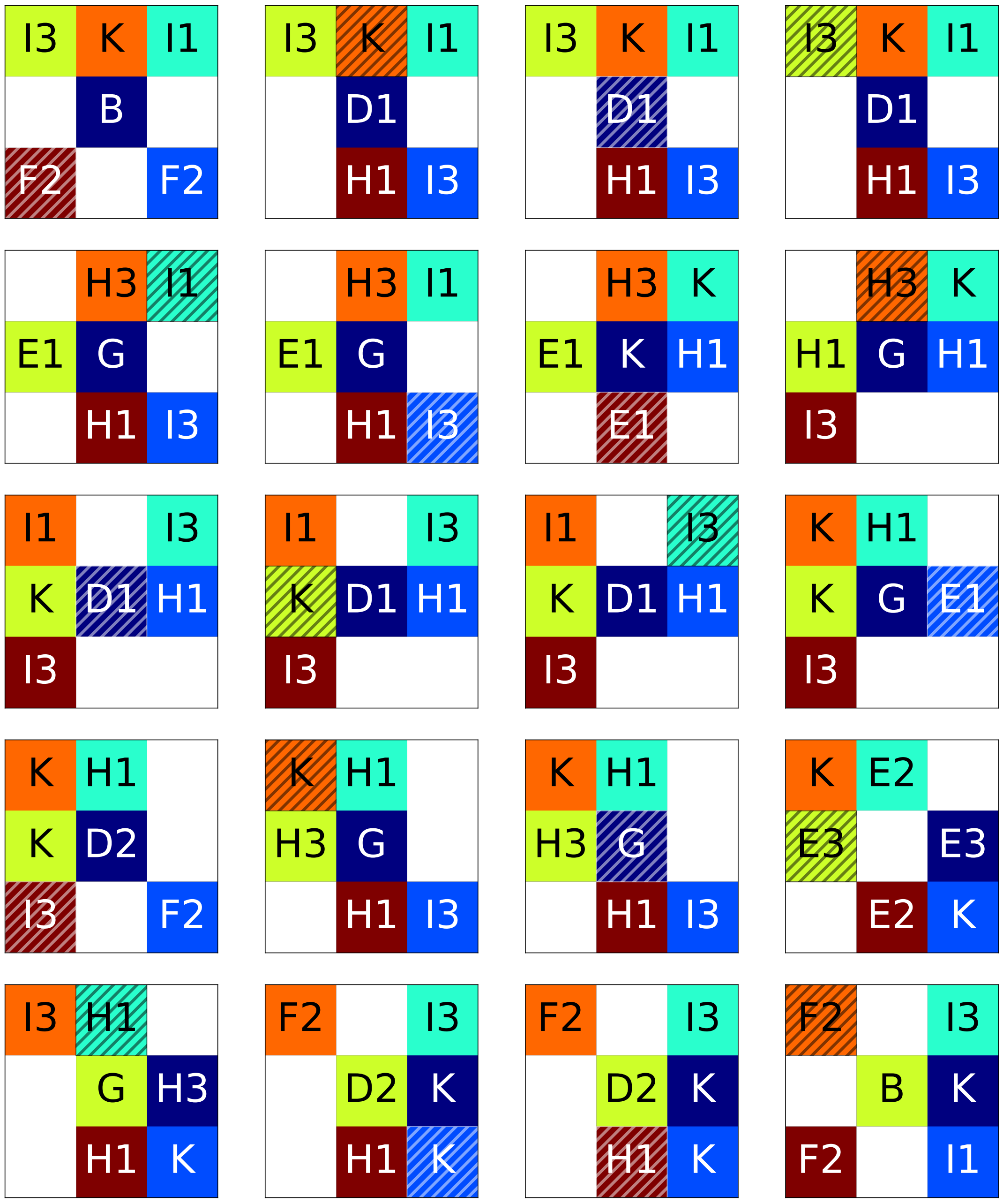


Fig. A.40 (V) Counter-example 32. Dynamic deadlock of period 228. Images show 80-99 of 228.

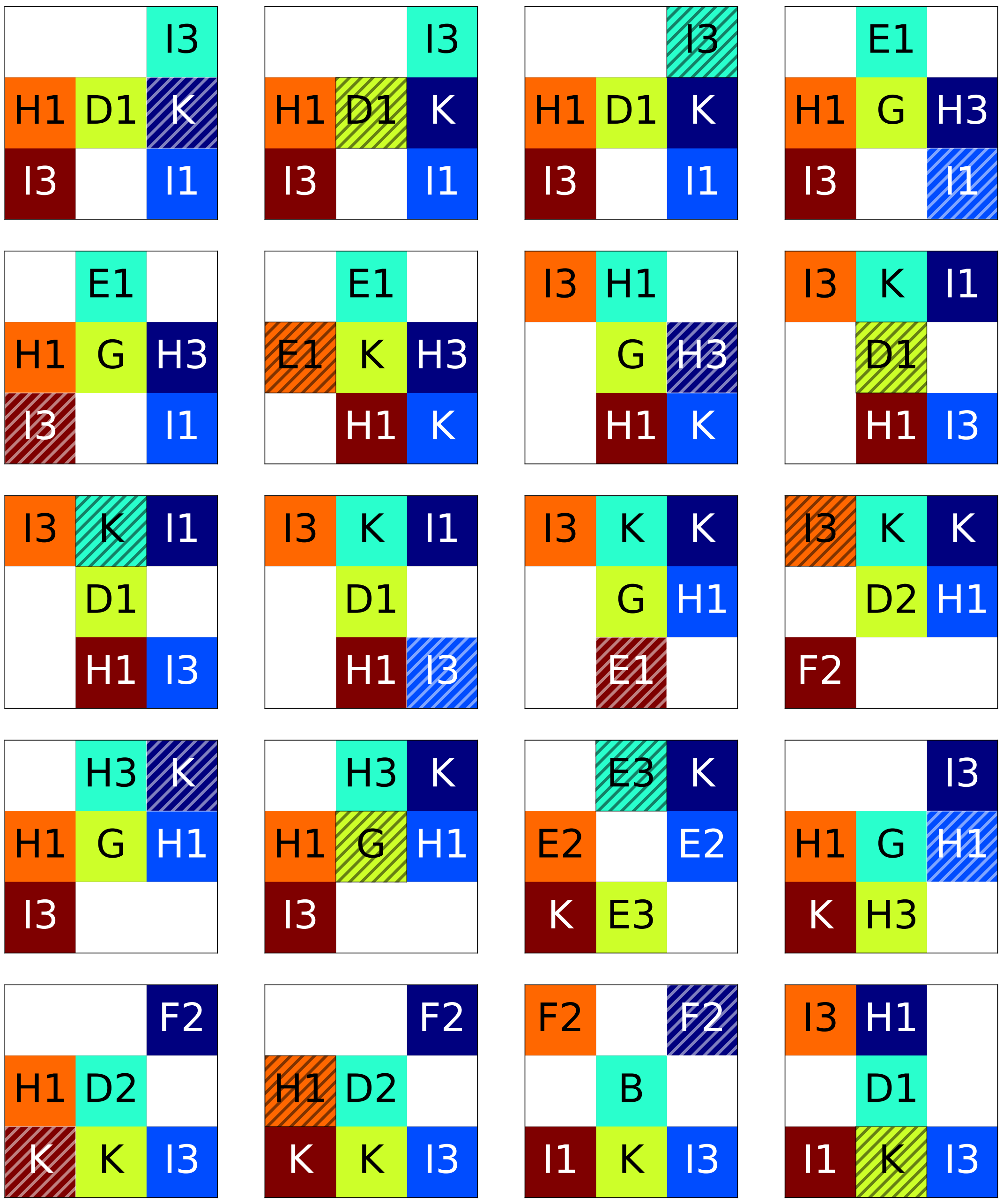


Fig. A.41 (VI) Counter-example 32. Dynamic deadlock of period 228. Images show 100-119 of 228.

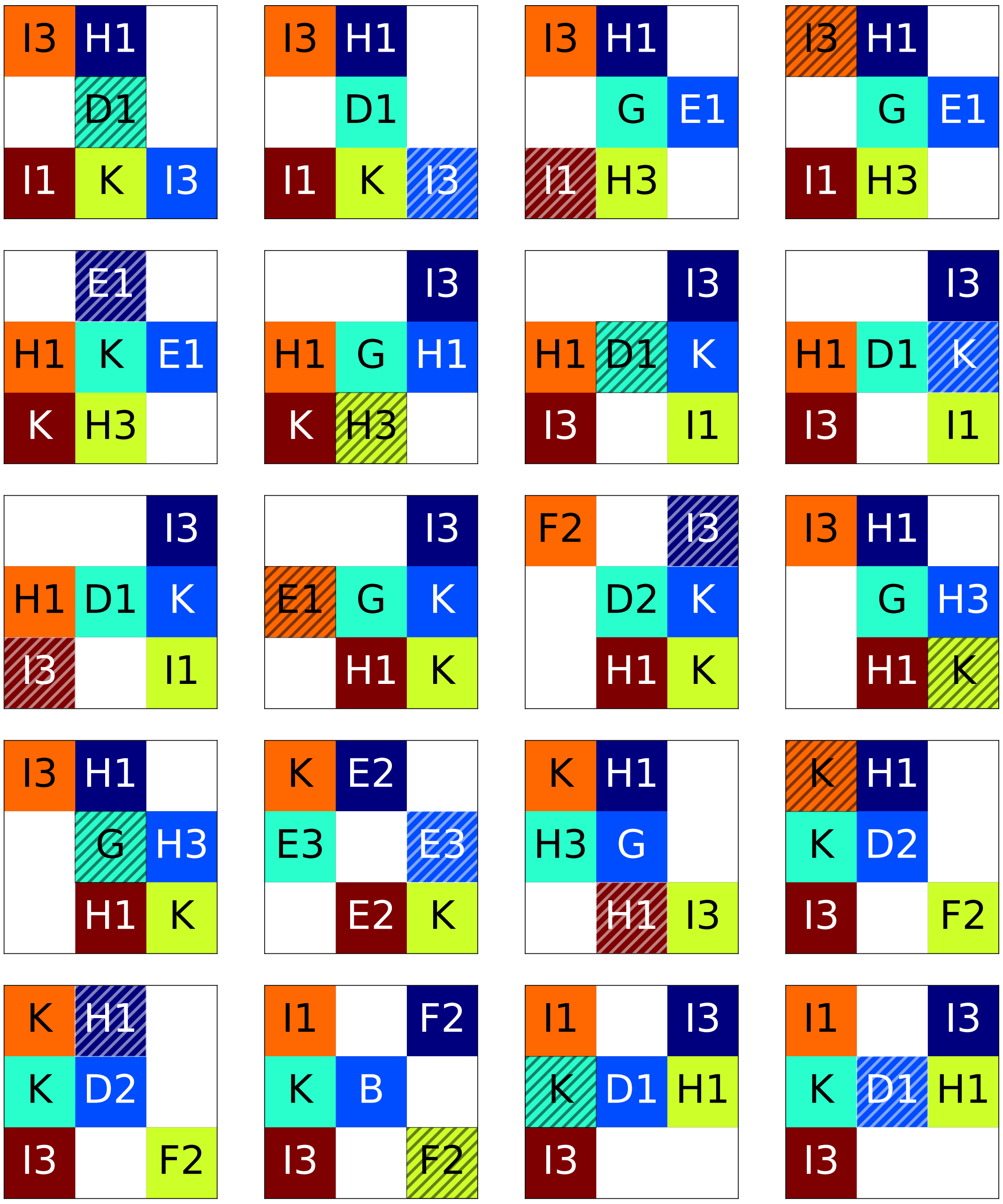


Fig. A.42 (VII) Counter-example 32. Dynamic deadlock of period 228. Images show 120-139 of 228.

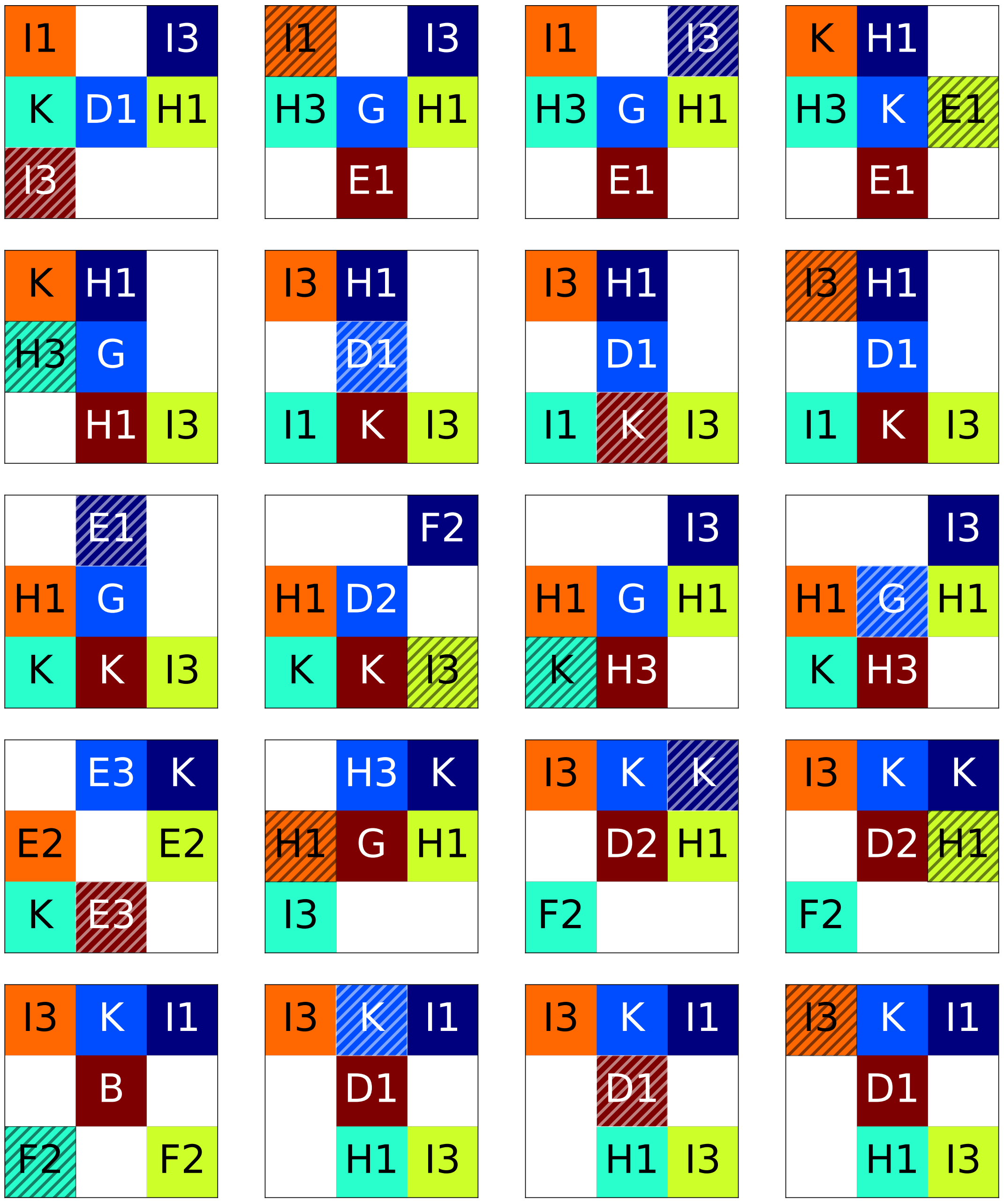


Fig. A.43 (VIII) Counter-example 32. Dynamic deadlock of period 228. Images show 140-159 of 228.

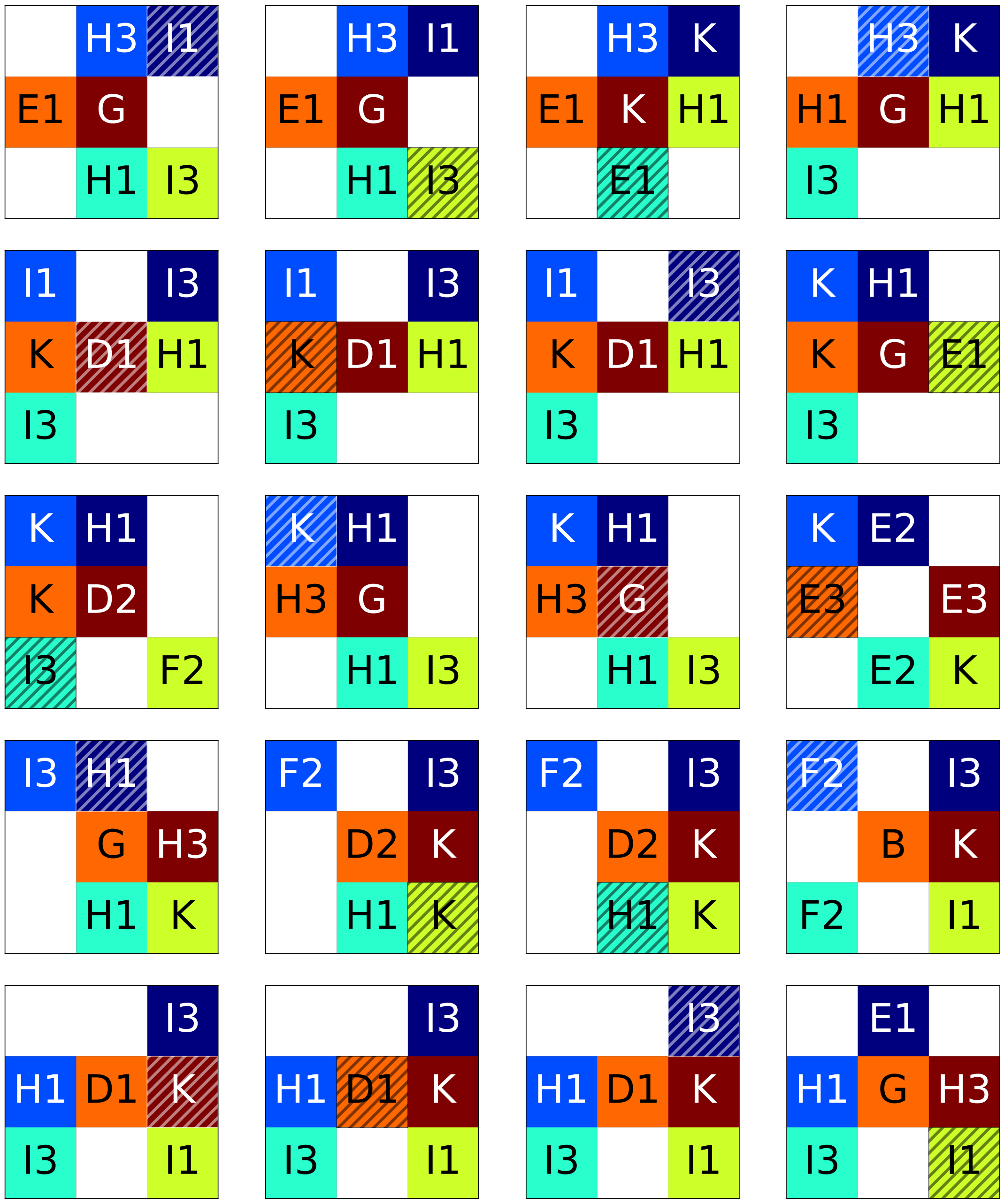


Fig. A.44 (IX) Counter-example 32. Dynamic deadlock of period 228. Images show 160-179 of 228.

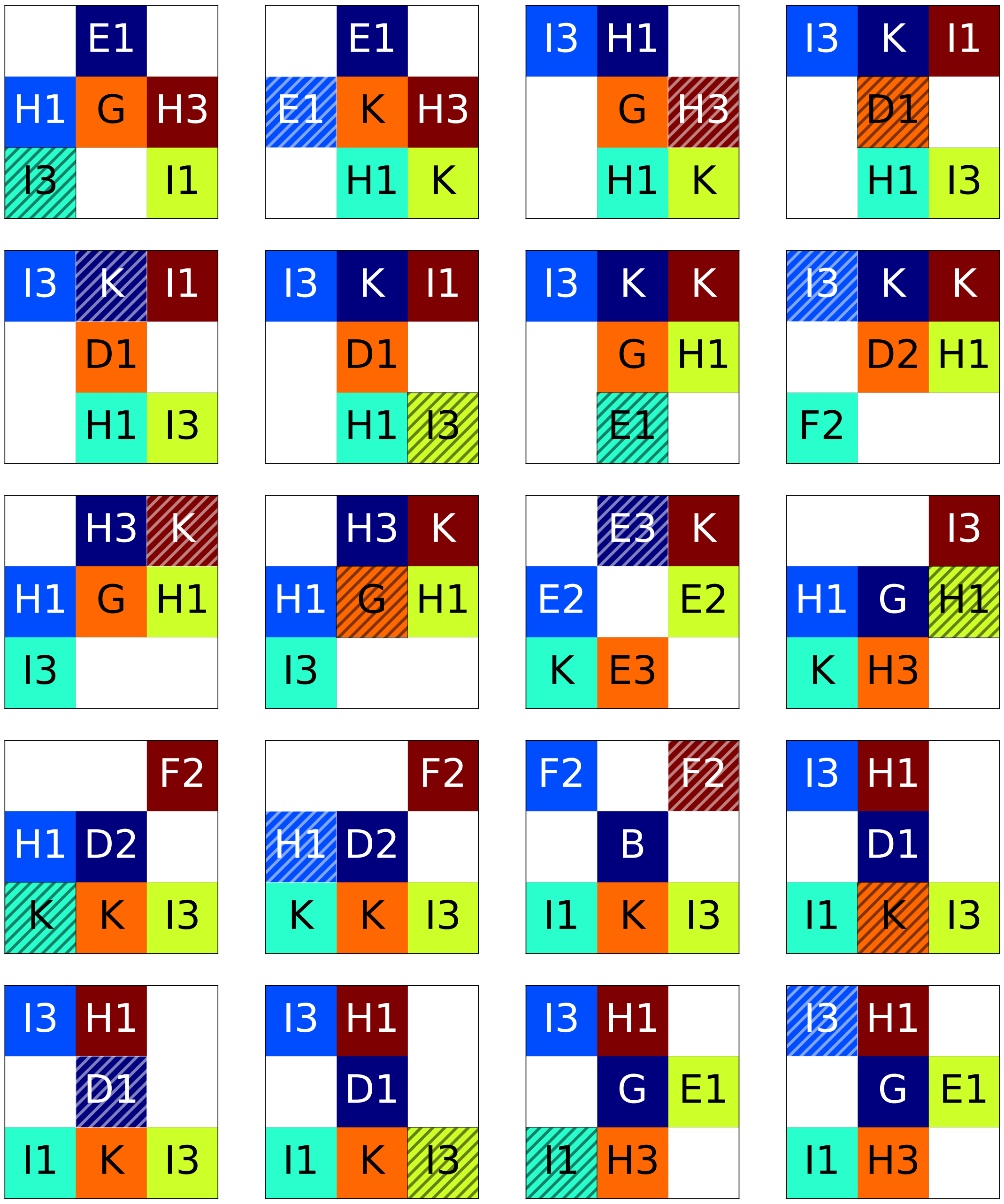


Fig. A.45 (X) Counter-example 32. Dynamic deadlock of period 228. Images show 180-199 of 228.

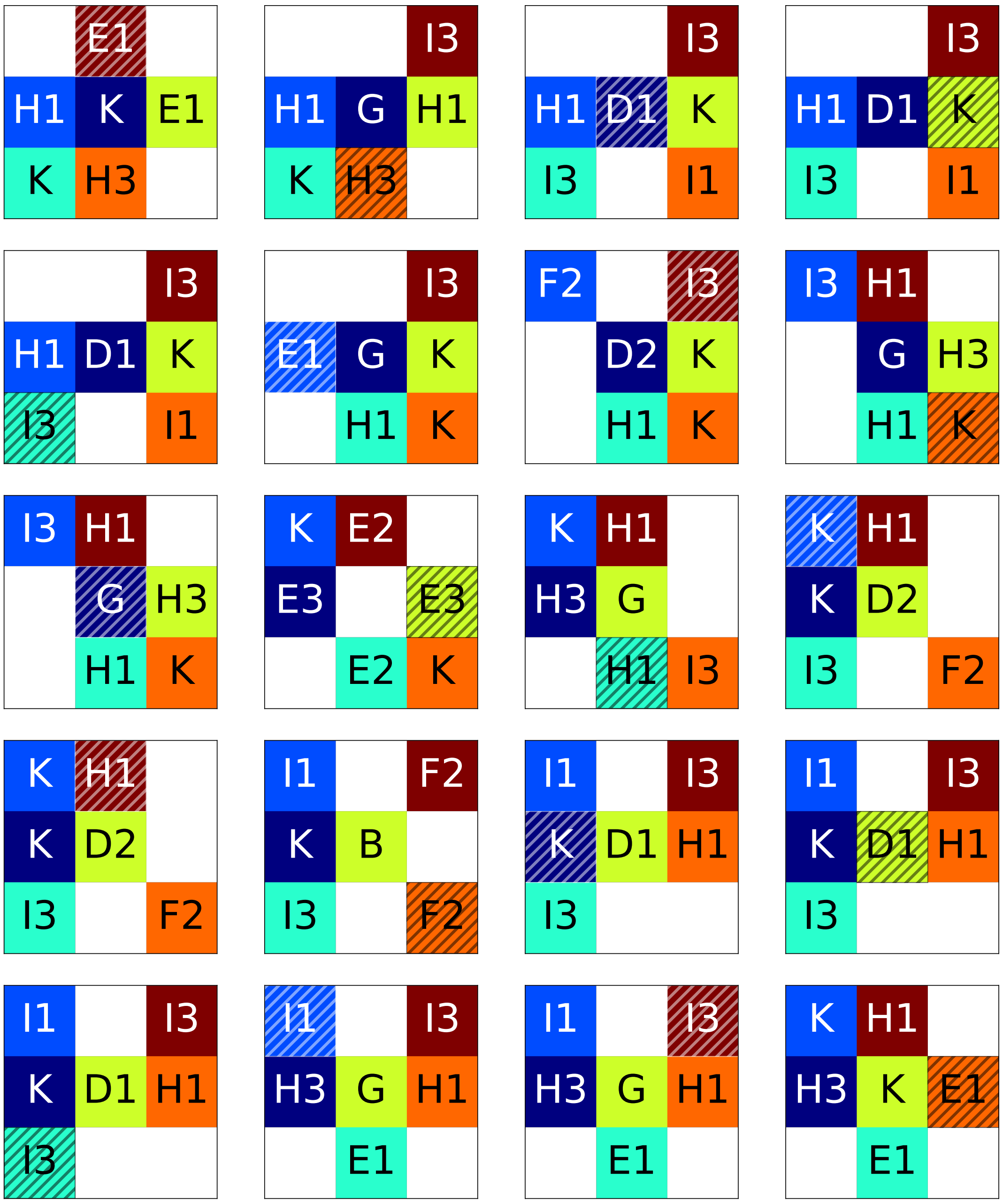


Fig. A.46 (XI) Counter-example 32. Dynamic deadlock of period 228. Images show 200-219 of 228.

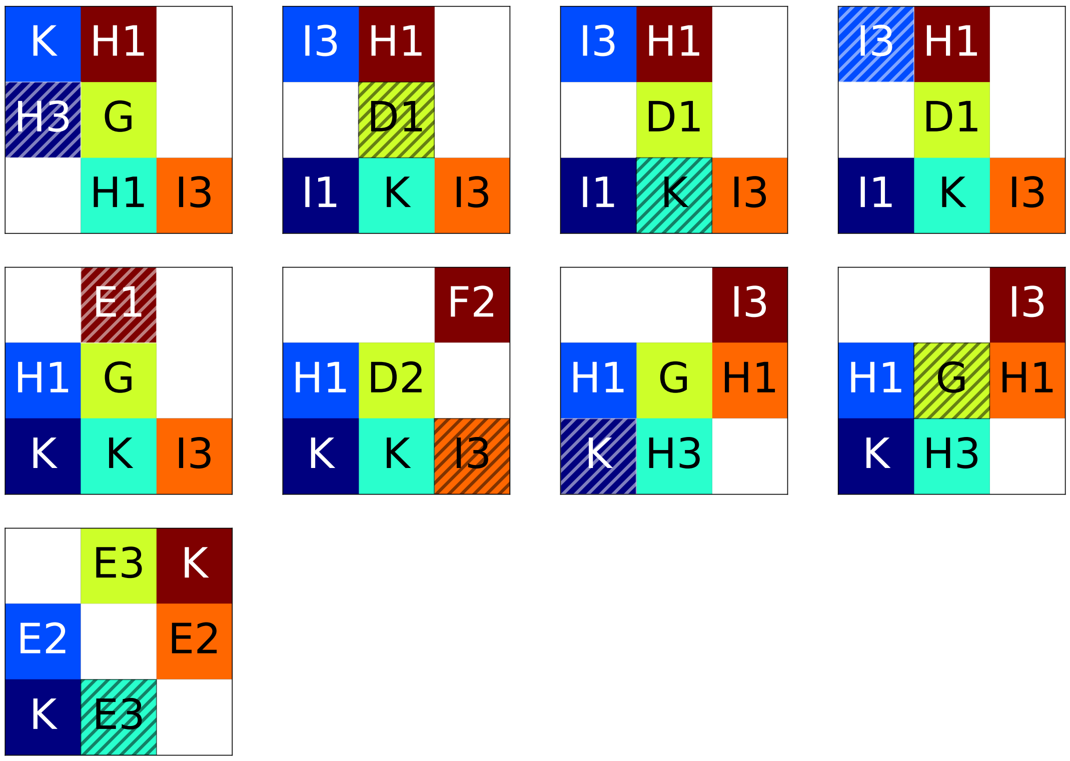


Fig. A.47 (XII) Counter-example 32. Dynamic deadlock of period 228. Images show 220-228 of 228.

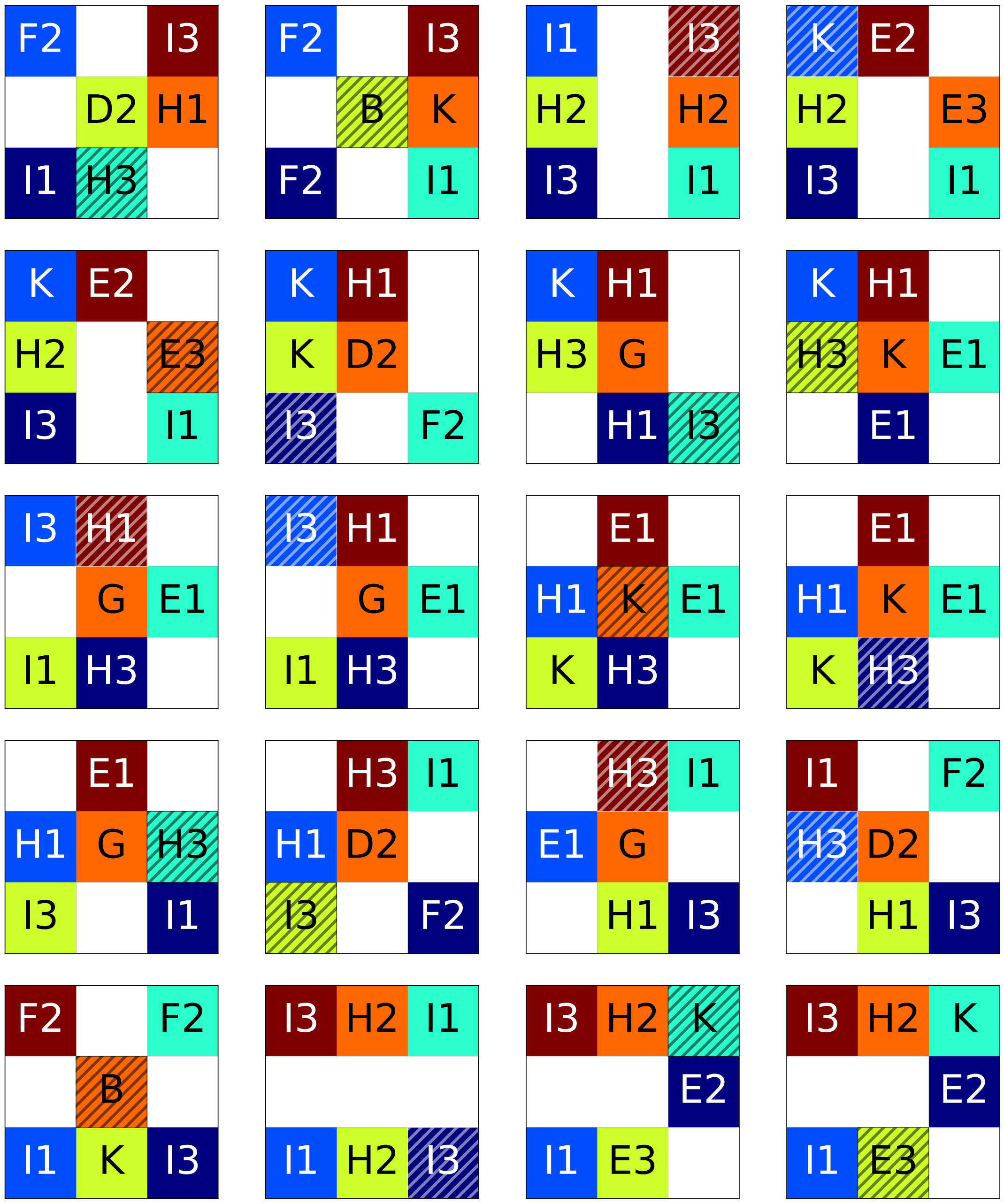


Fig. A.48 (I) Counter-example 33. Dynamic deadlock of period 60. Images show 0-19 of 60.

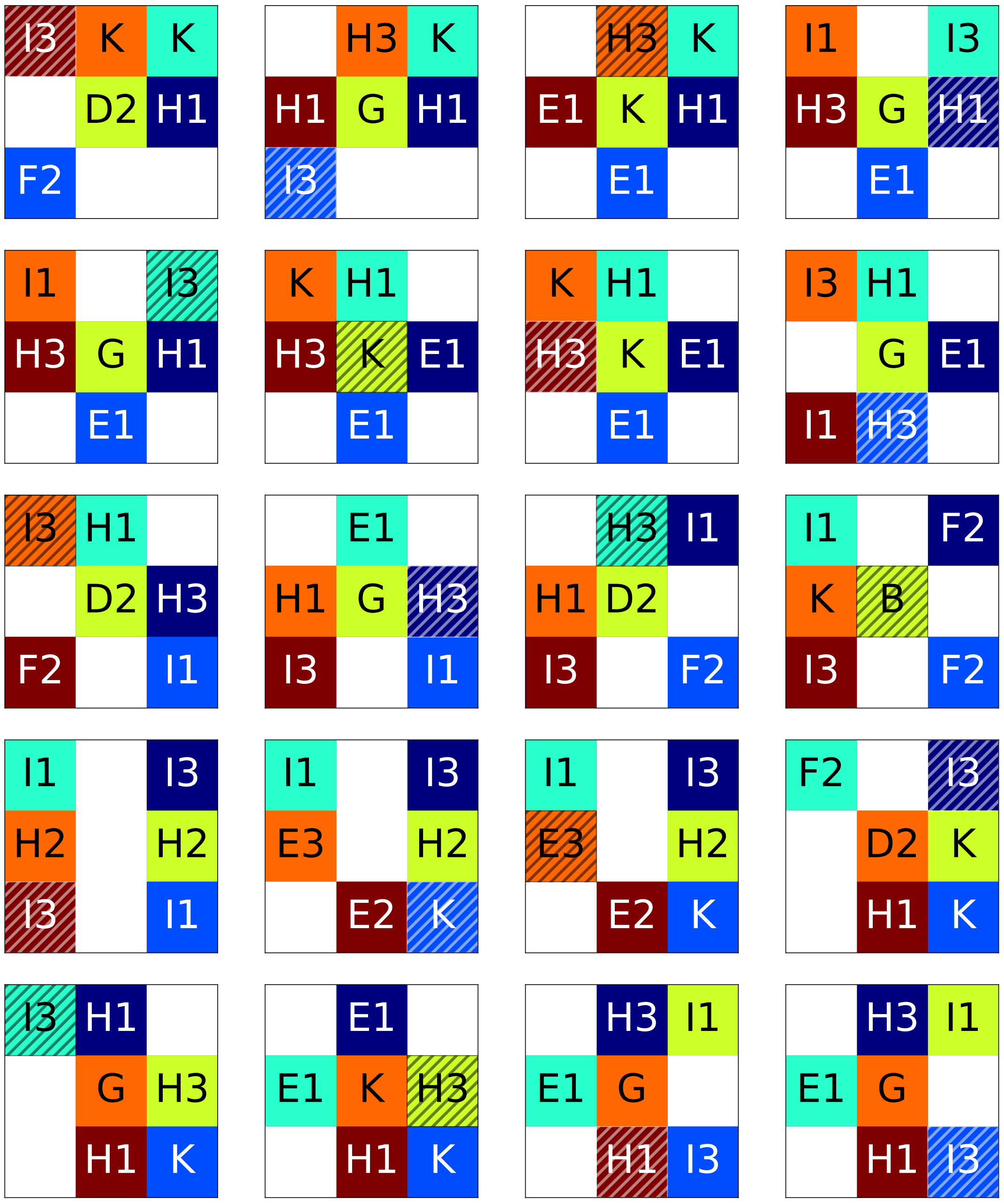


Fig. A.49 (II) Counter-example 33. Dynamic deadlock of period 60. Images show 20-39 of 60.

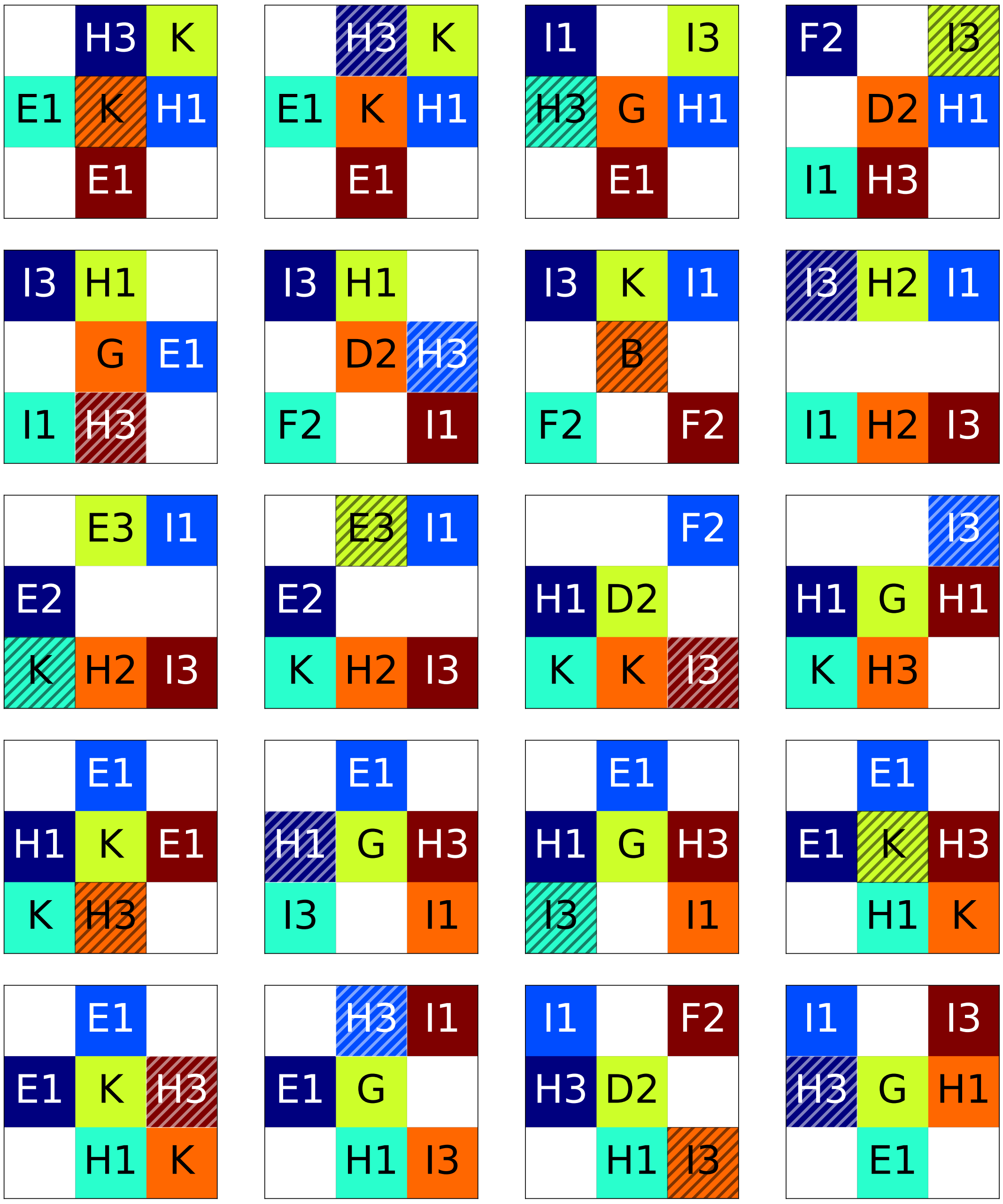


Fig. A.50 (III) Counter-example 33. Dynamic deadlock of period 60. Images show 40-59 of 60.

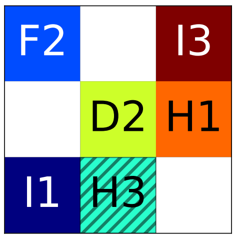


Fig. A.51 (IV) Counter-example 33. Dynamic deadlock of period 60. Images show 60-60 of 60.

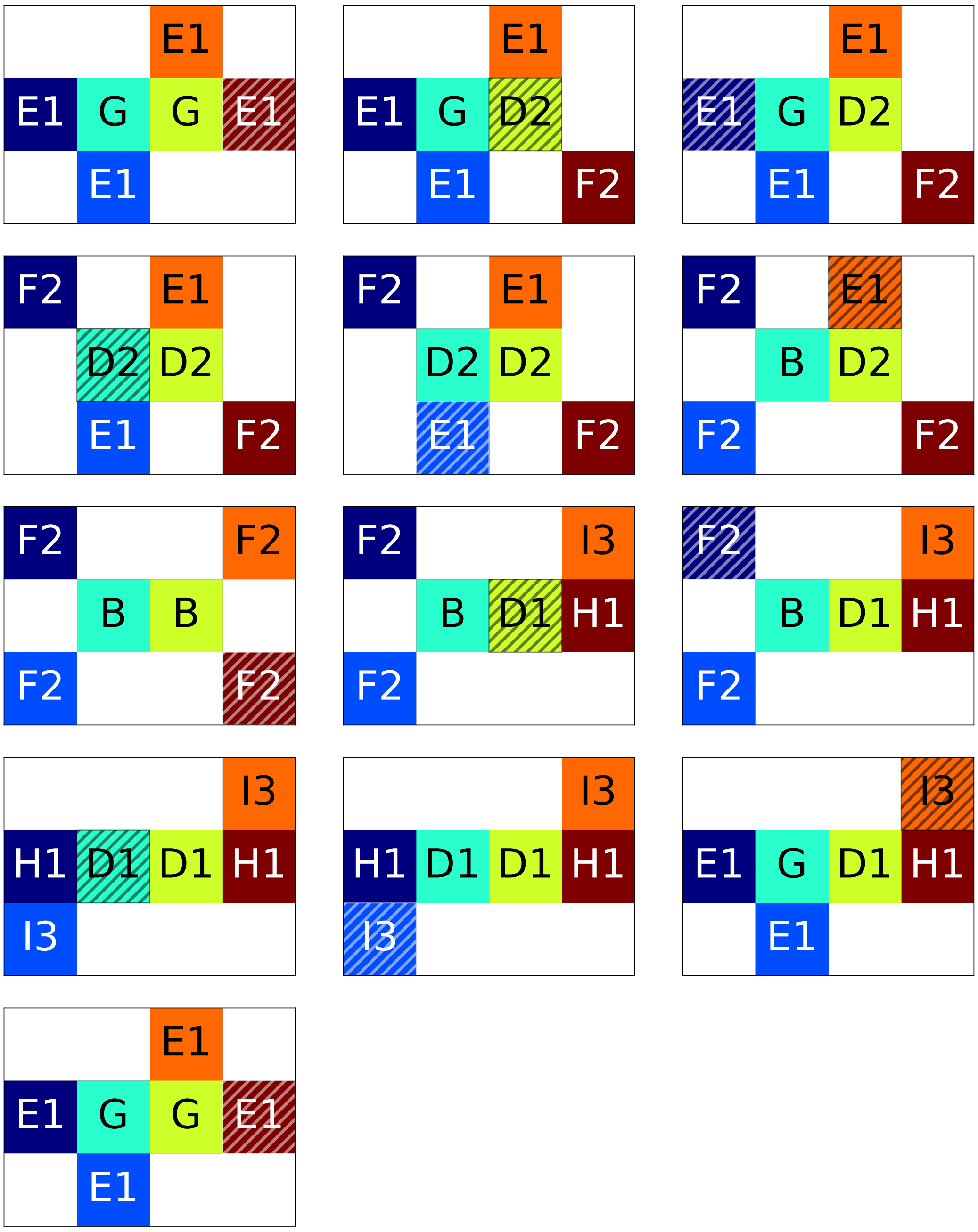


Fig. A.52 Counter-example 34. Dynamic deadlock of period 12. Images show 0-12 of 12.

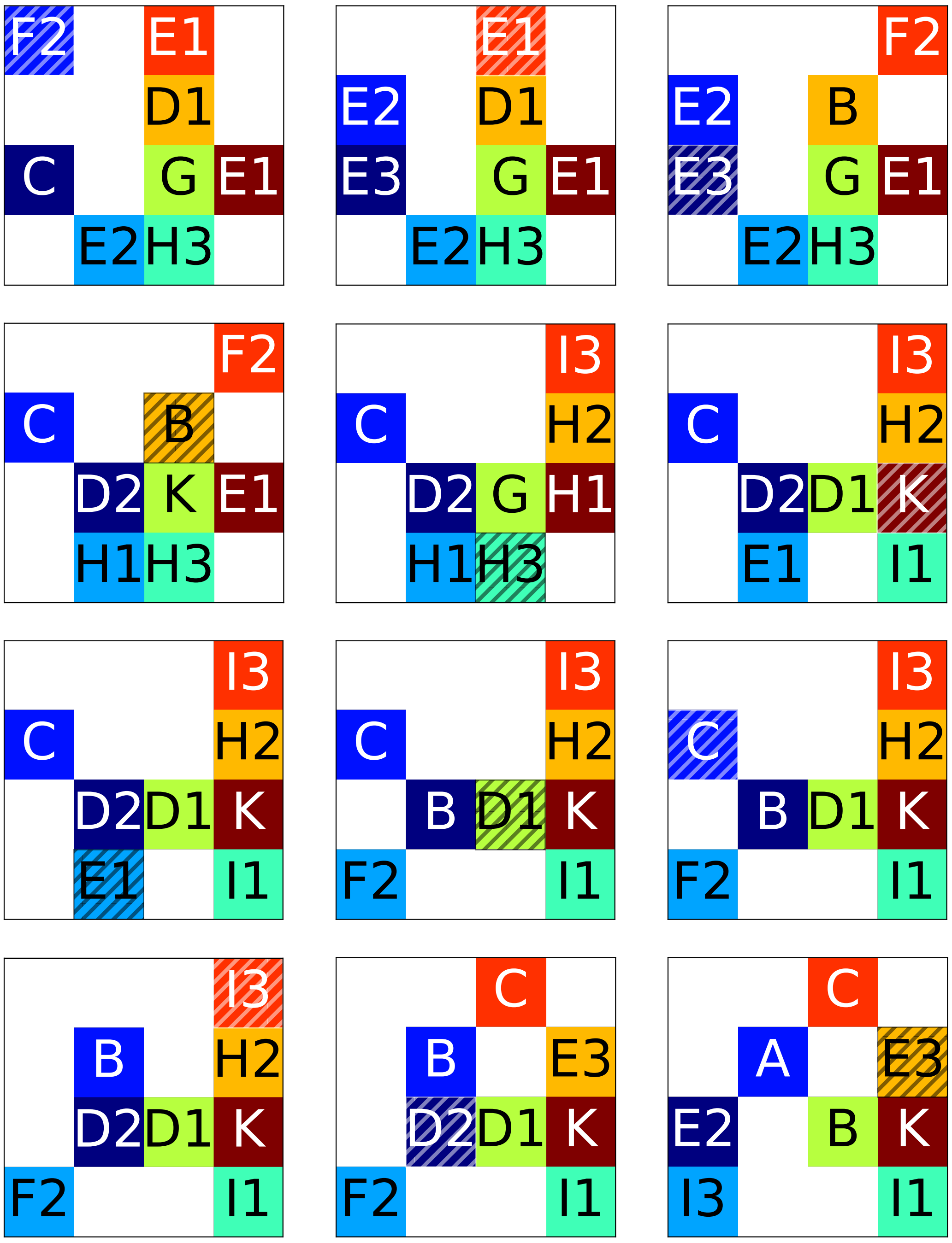


Fig. A.53 (I) Counter-example 35. Dynamic deadlock of period 40. Images show 0-11 of 40.

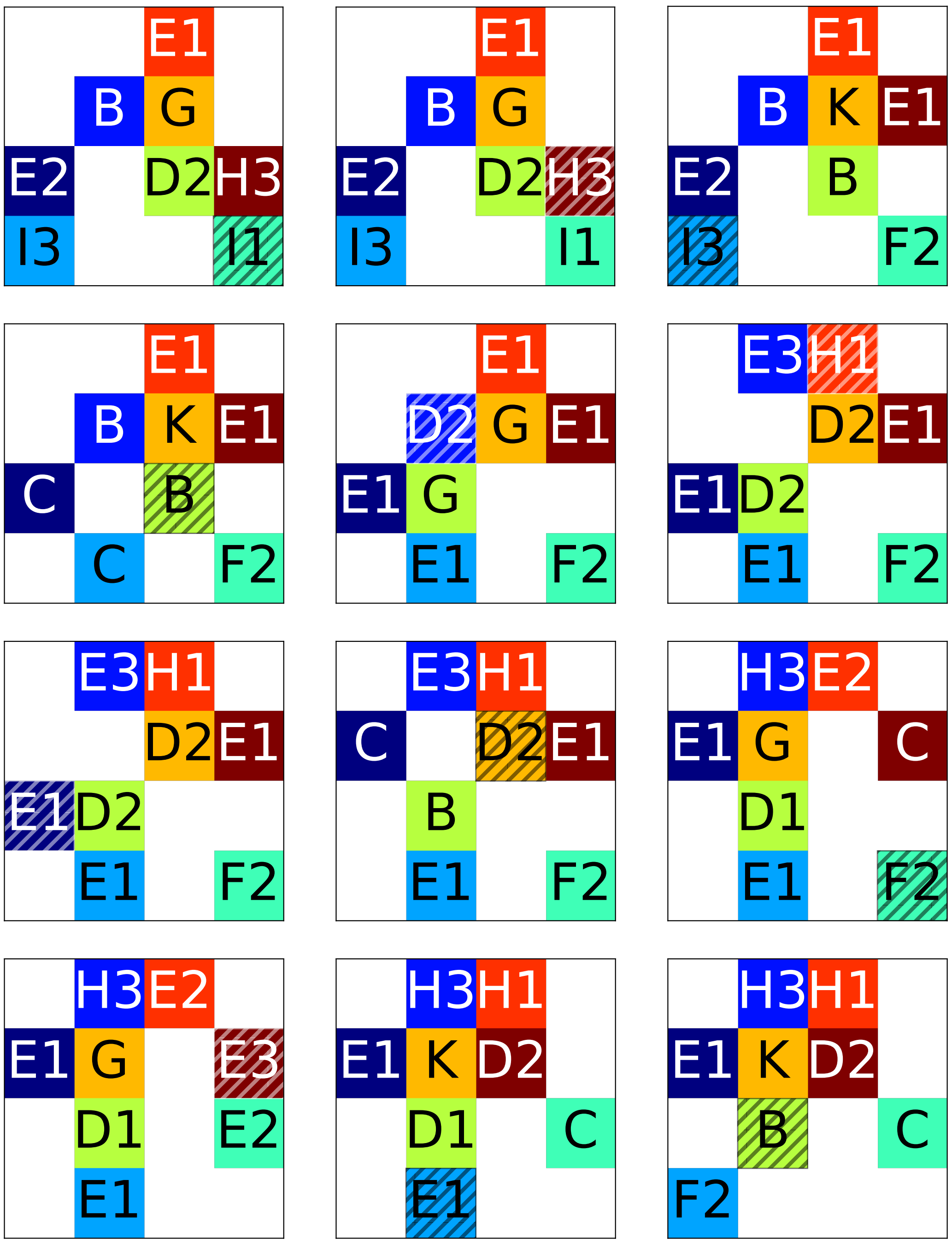


Fig. A.54 (II) Counter-example 35. Dynamic deadlock of period 40. Images show 12-23 of 40.

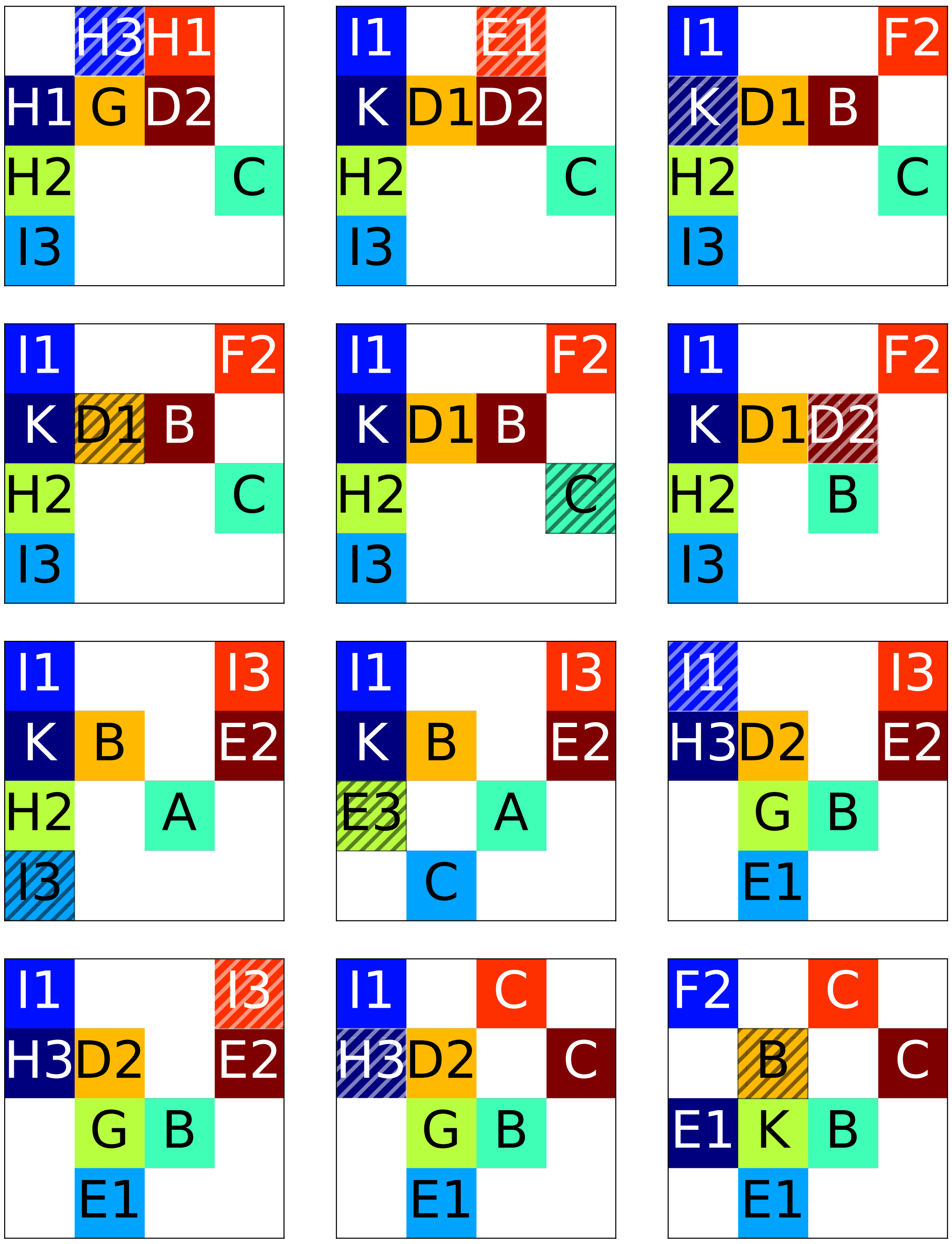


Fig. A.55 (III) Counter-example 35. Dynamic deadlock of period 40. Images show 24-35 of 40.

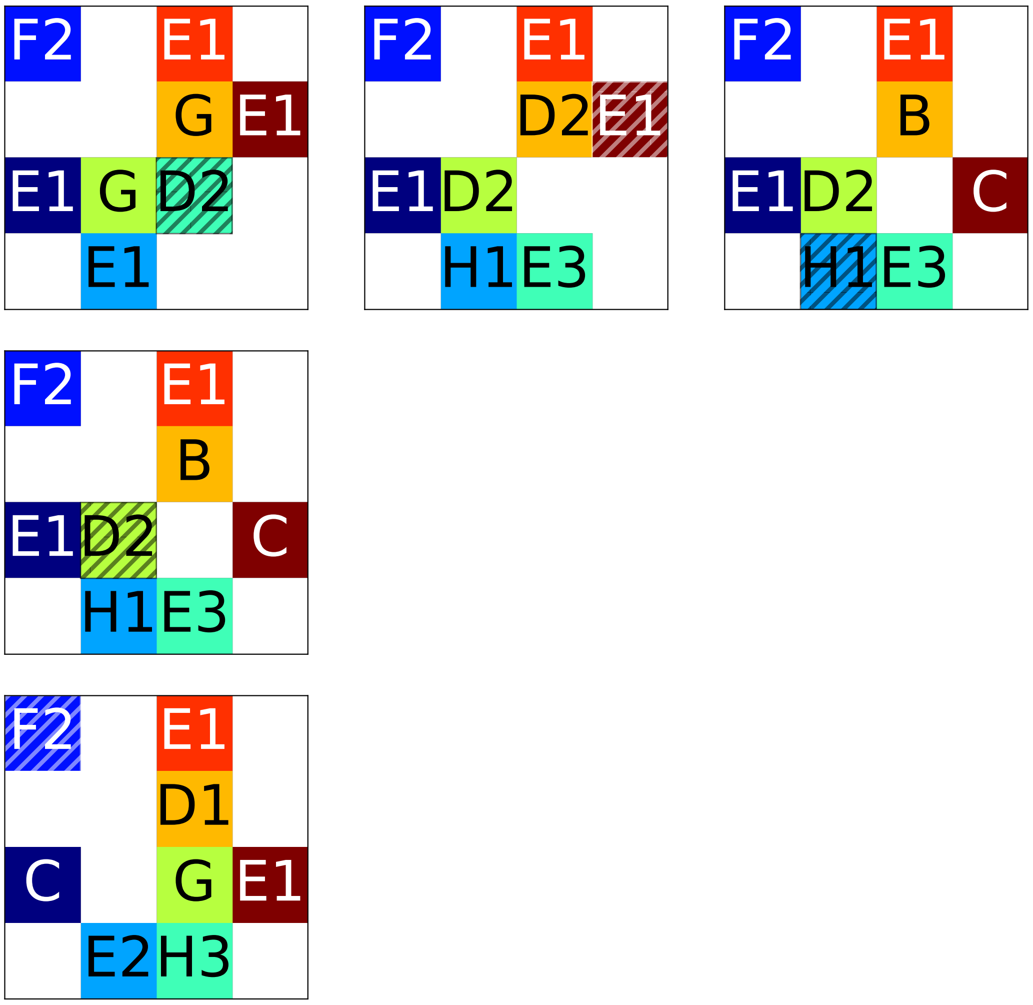


Fig. A.56 (IV) Counter-example 35. Dynamic deadlock of period 40. Images show 36-40 of 40.

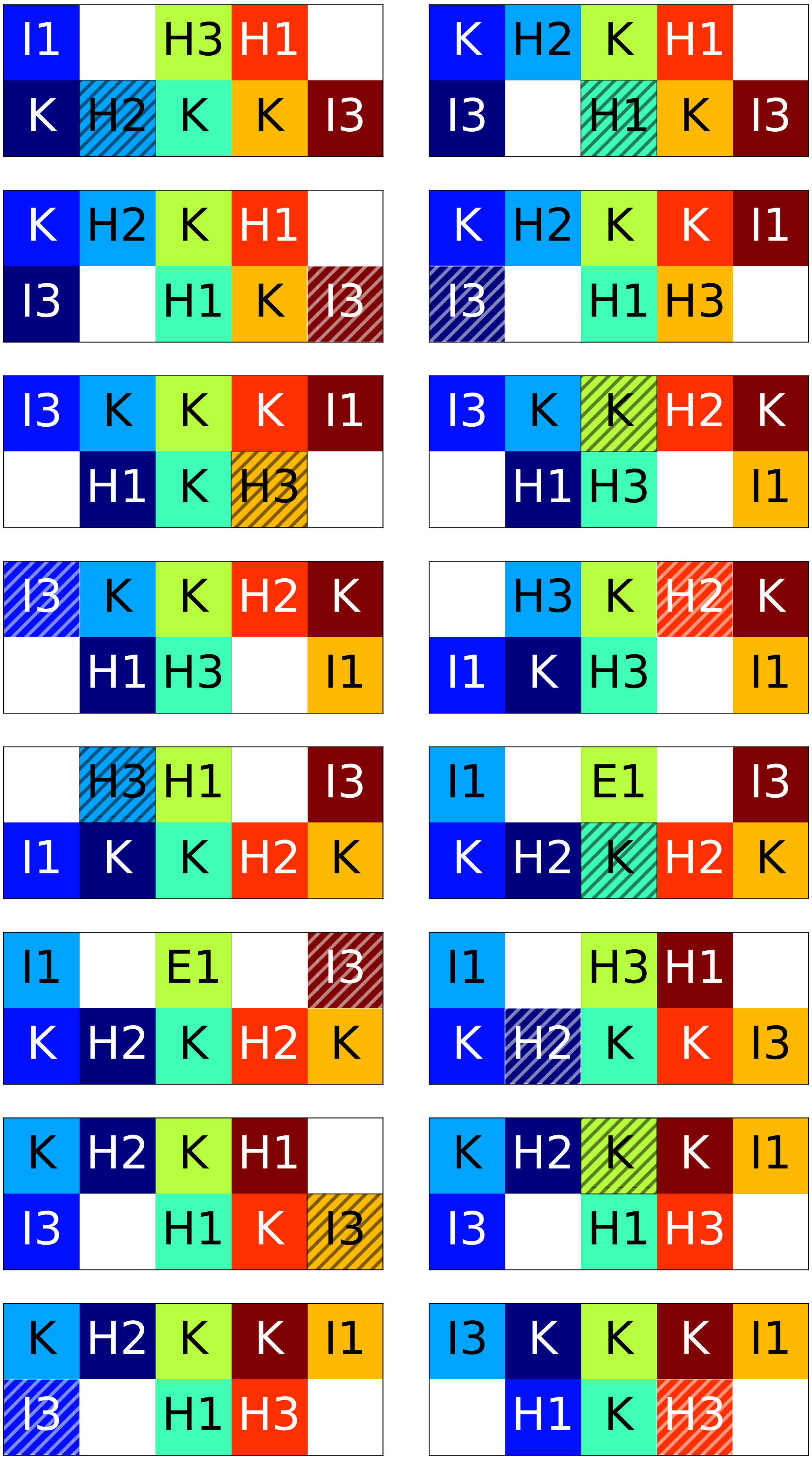


Fig. A.57 (I) Counter-example 36. Dynamic deadlock of period 360. Images show 0-15 of 360.

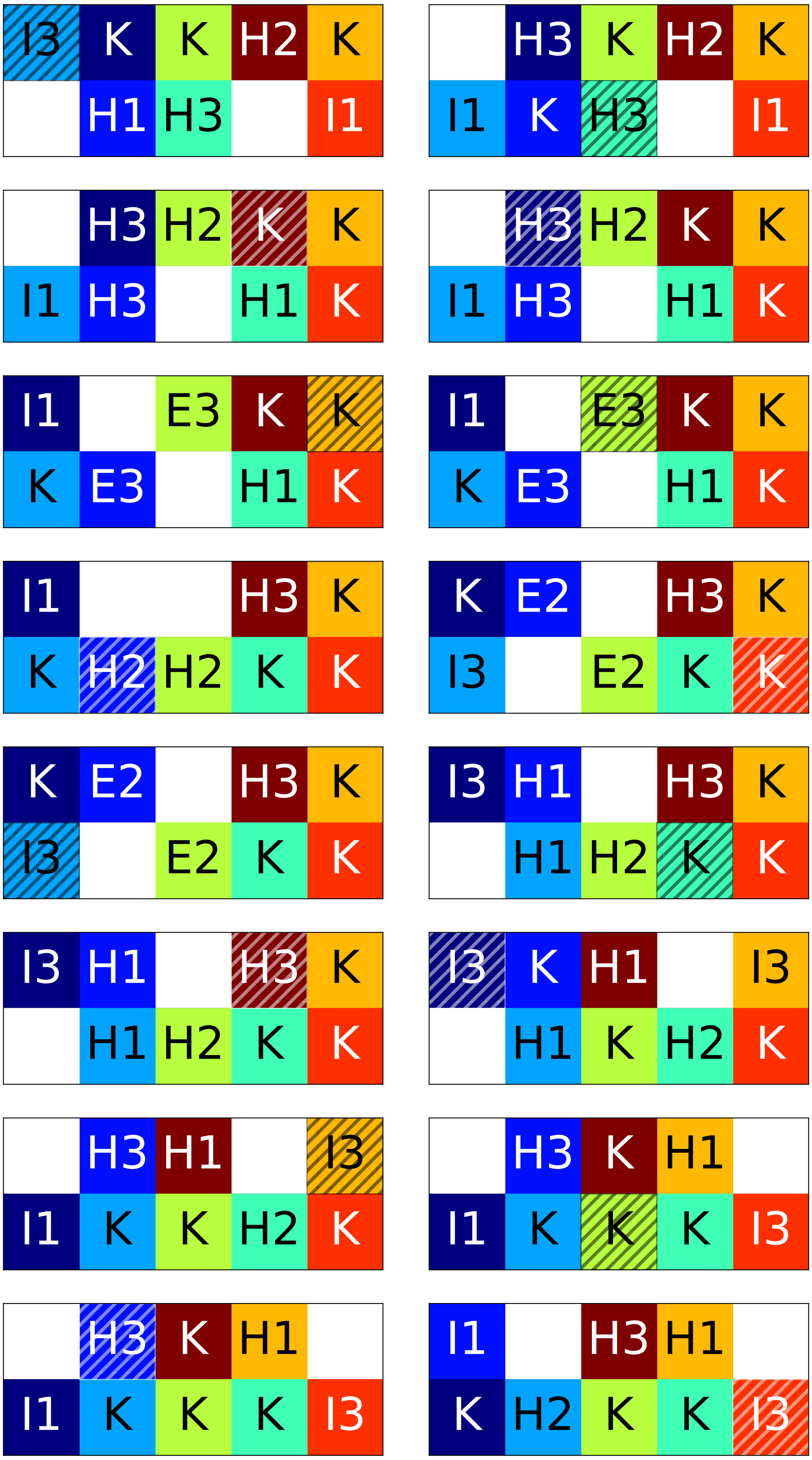


Fig. A.58 (II) Counter-example 36. Dynamic deadlock of period 360. Images show 16-31 of 360.

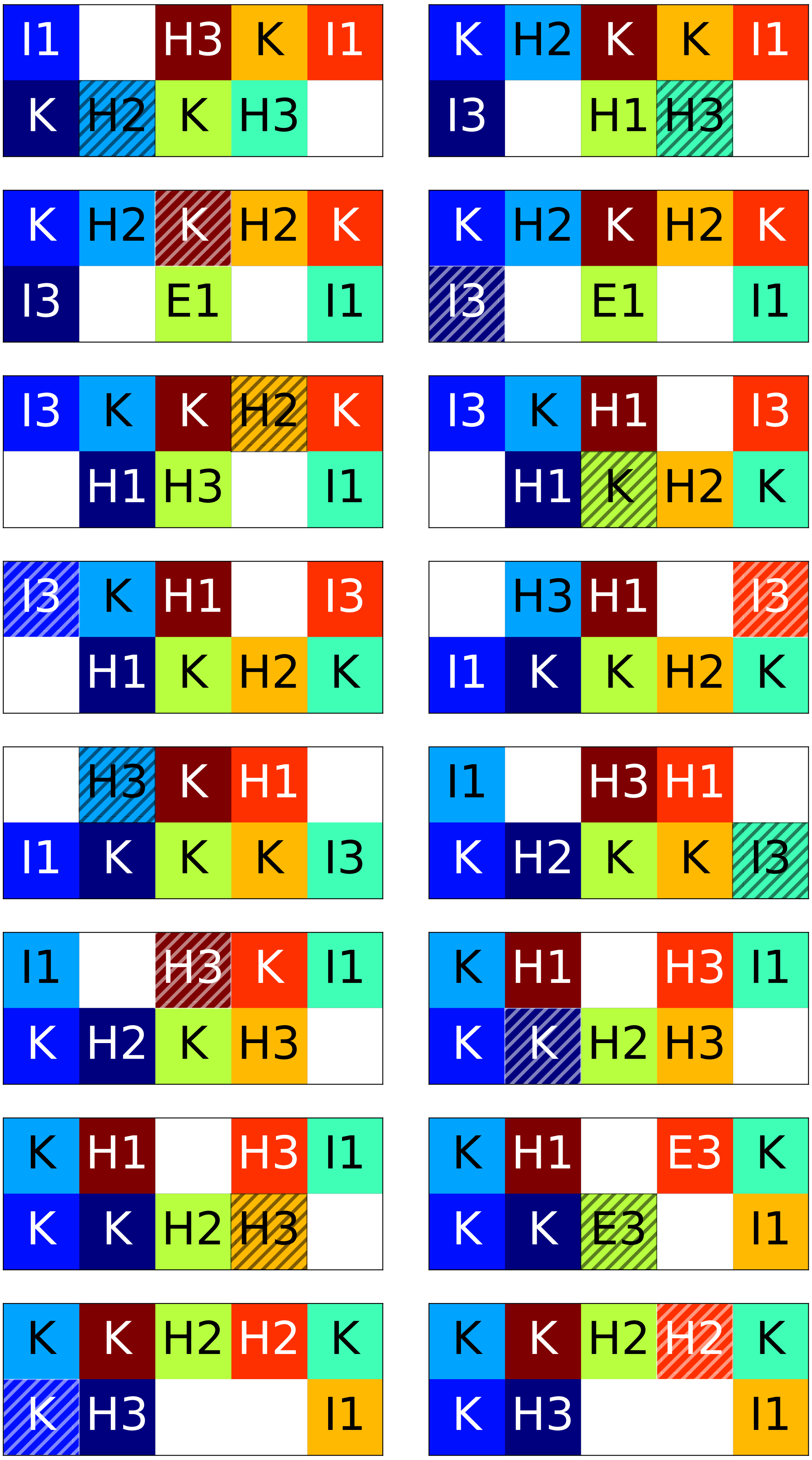


Fig. A.59 (III) Counter-example 36. Dynamic deadlock of period 360. Images show 32-47 of 360.

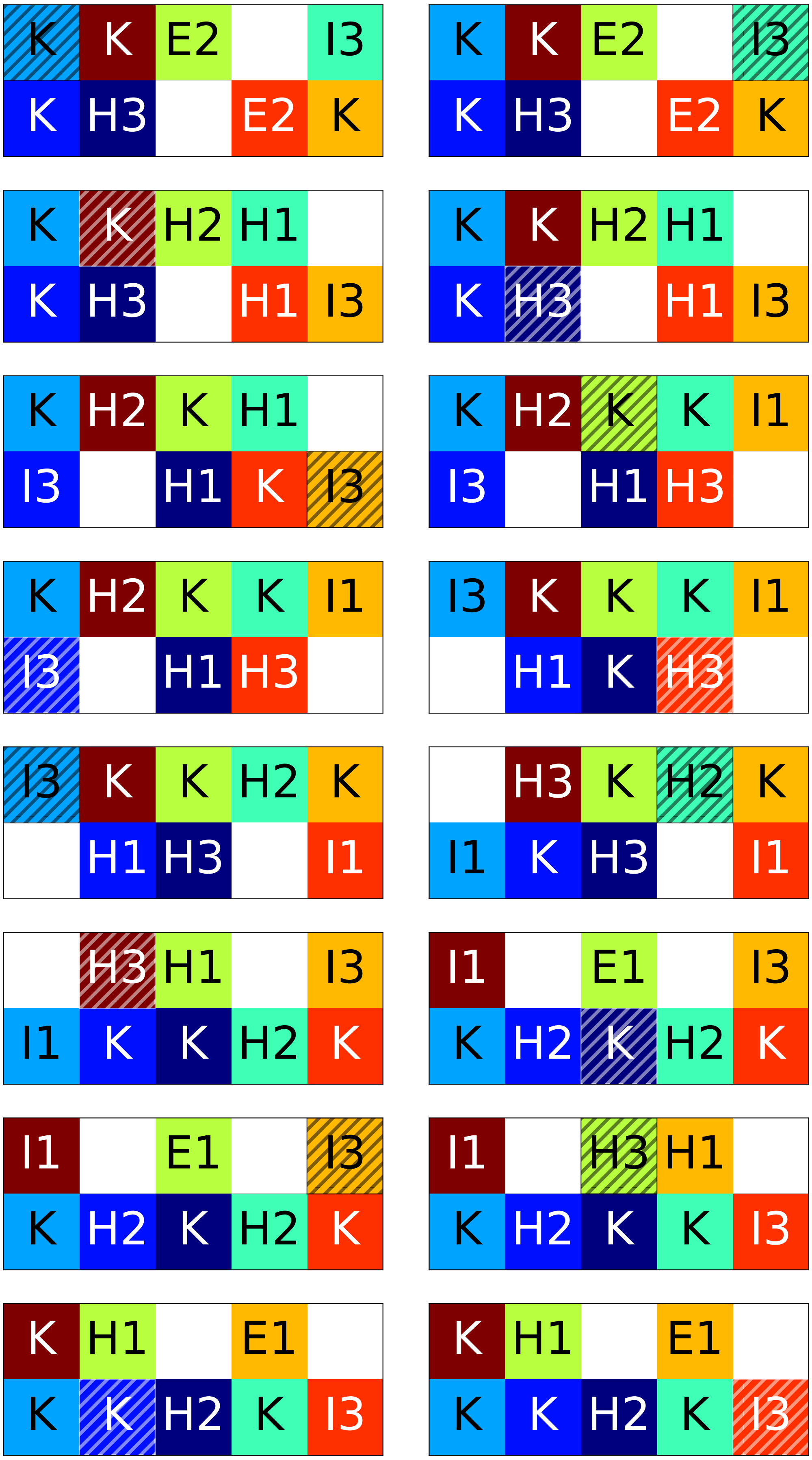


Fig. A.60 (IV) Counter-example 36. Dynamic deadlock of period 360. Images show 48-63 of 360.

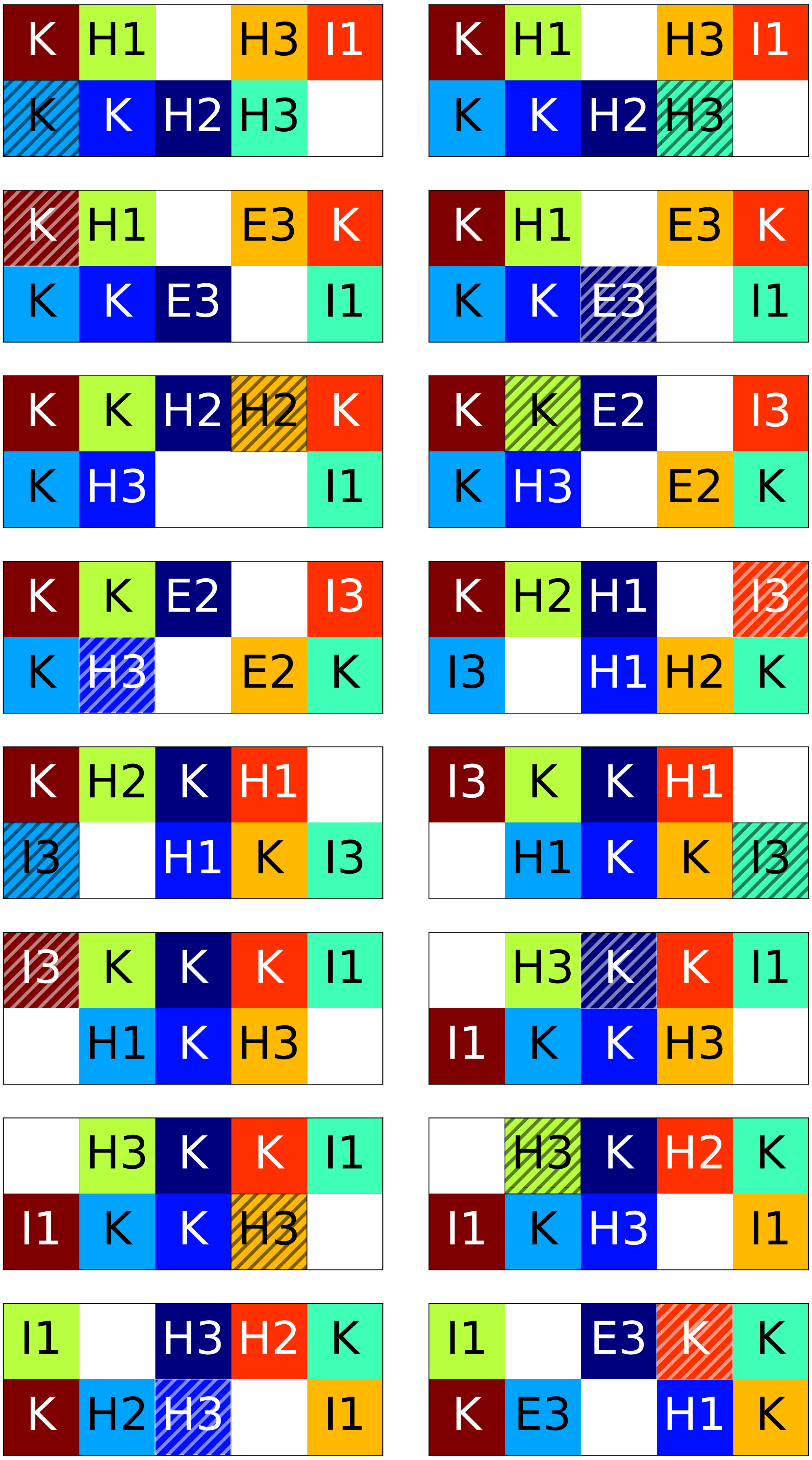


Fig. A.61 (V) Counter-example 36. Dynamic deadlock of period 360. Images show 64-79 of 360.

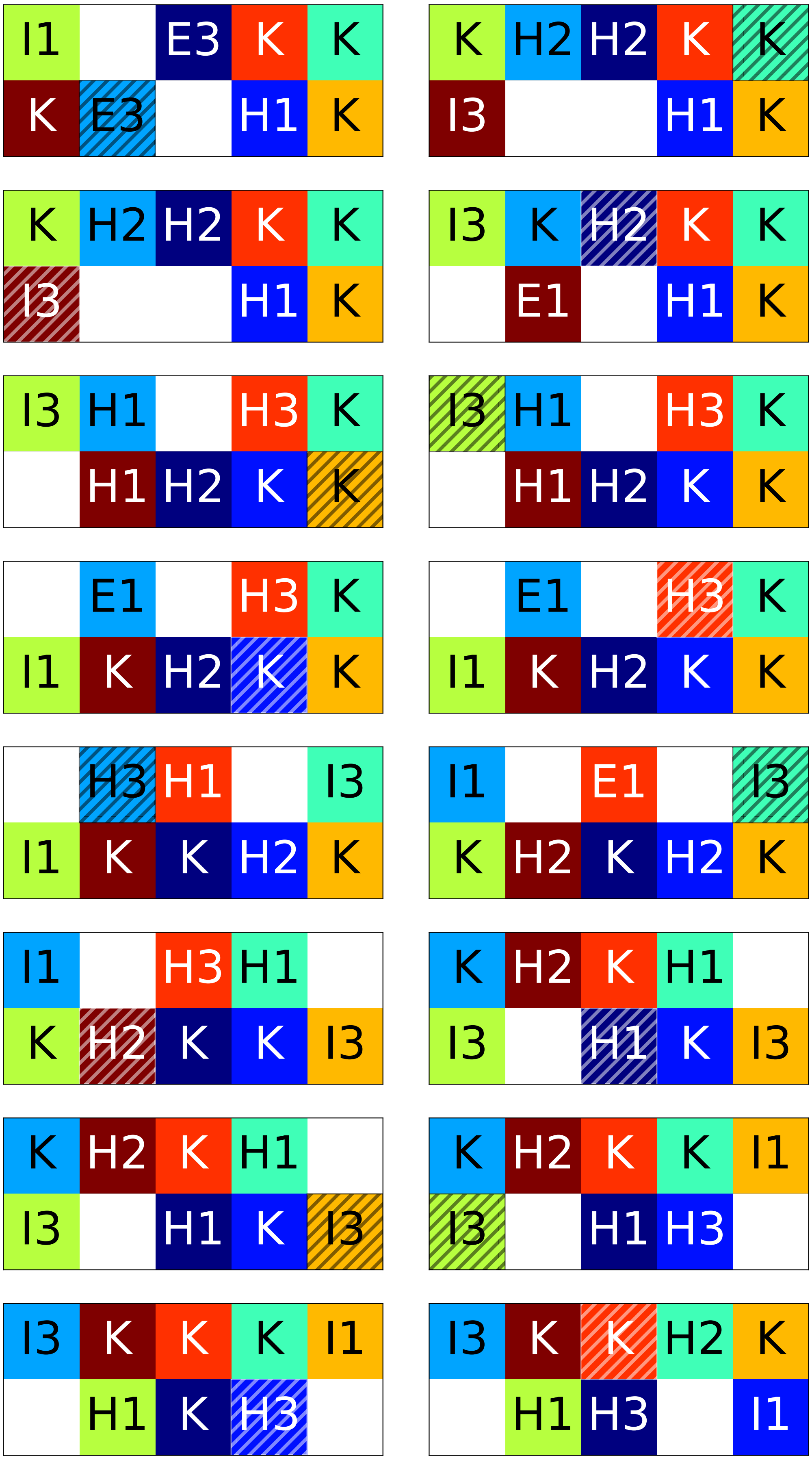


Fig. A.62 (VI) Counter-example 36. Dynamic deadlock of period 360. Images show 80-95 of 360.

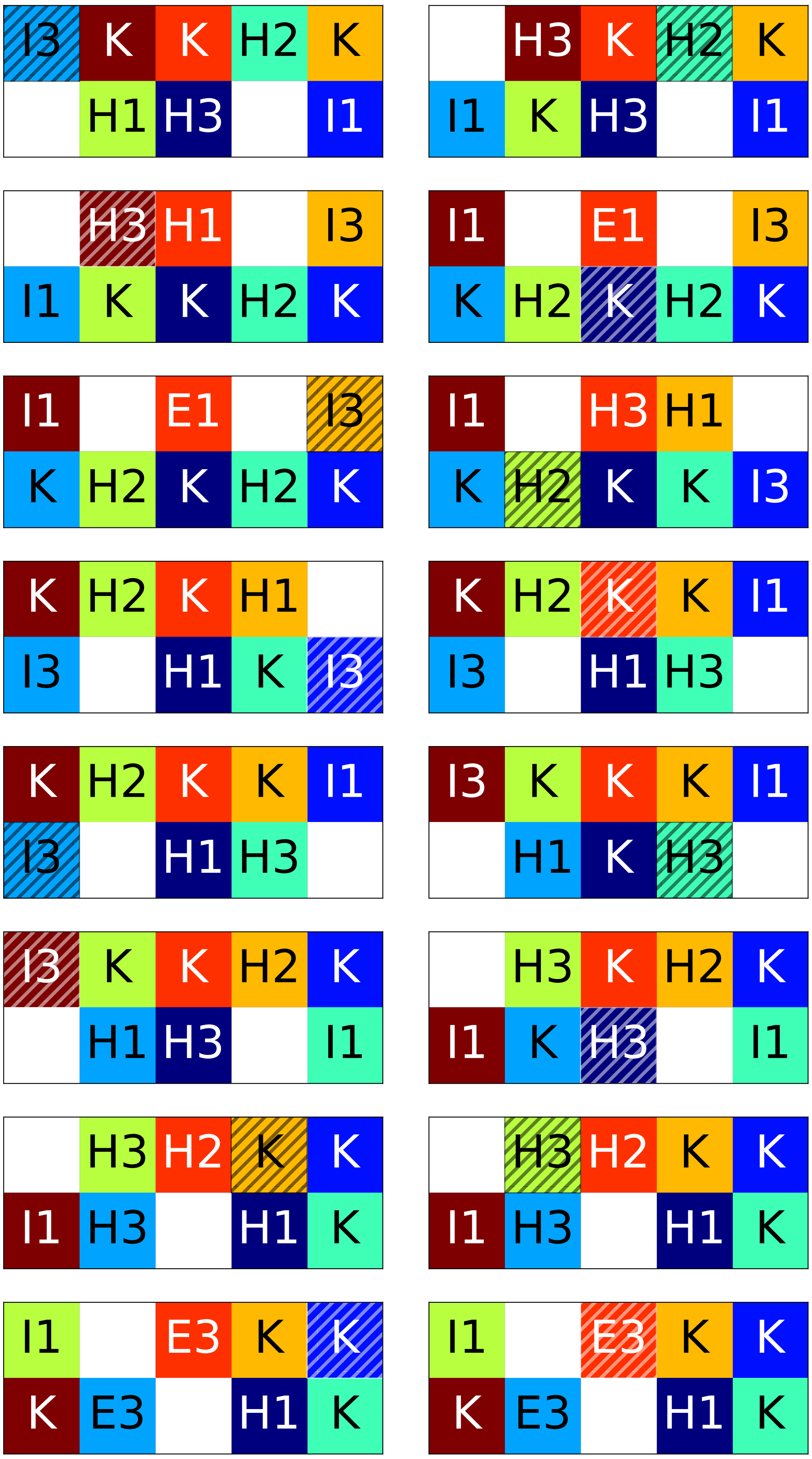


Fig. A.63 (VII) Counter-example 36. Dynamic deadlock of period 360. Images show 96-111 of 360.

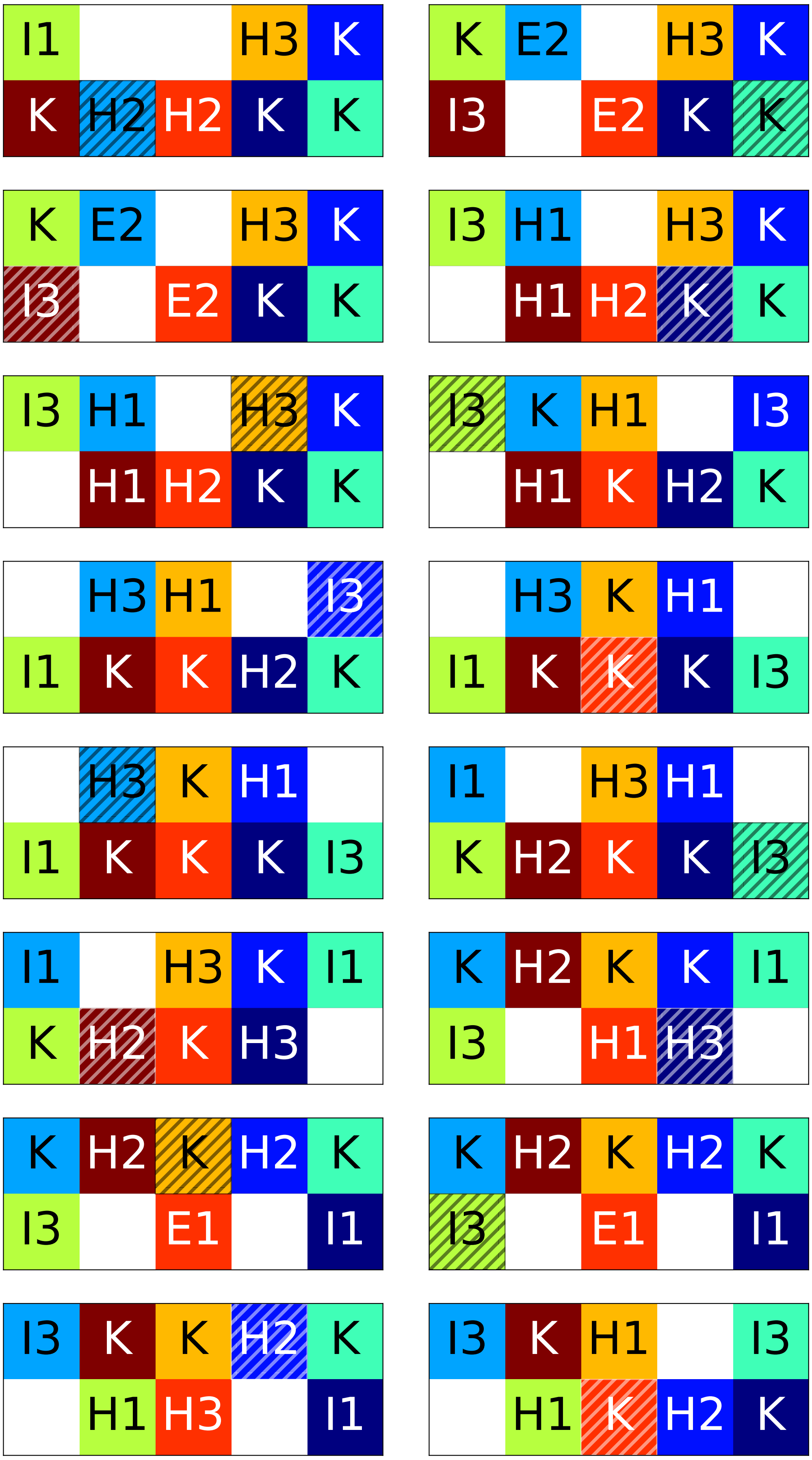


Fig. A.64 (VIII) Counter-example 36. Dynamic deadlock of period 360. Images show 112-127 of 360.

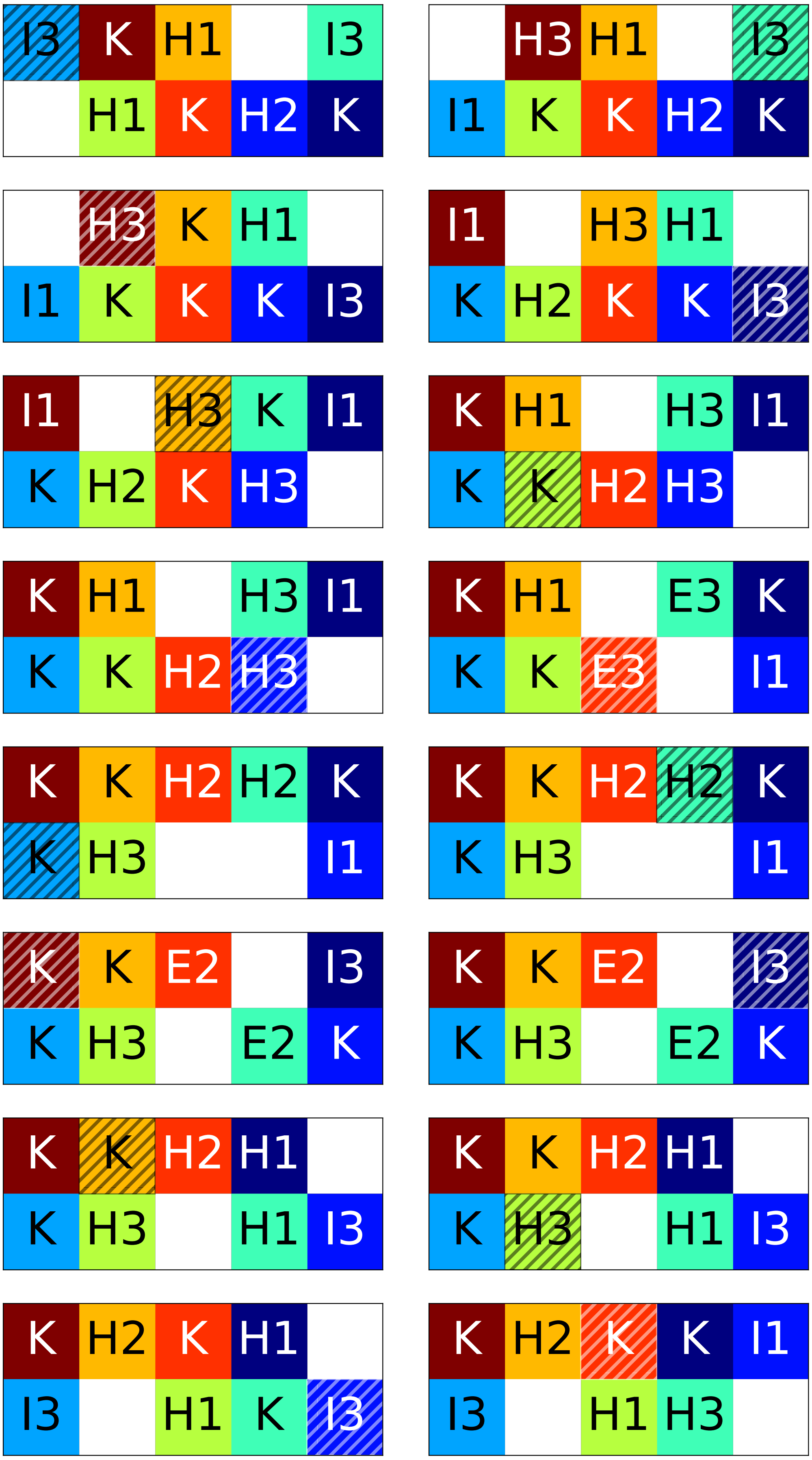


Fig. A.65 (IX) Counter-example 36. Dynamic deadlock of period 360. Images show 128-143 of 360.

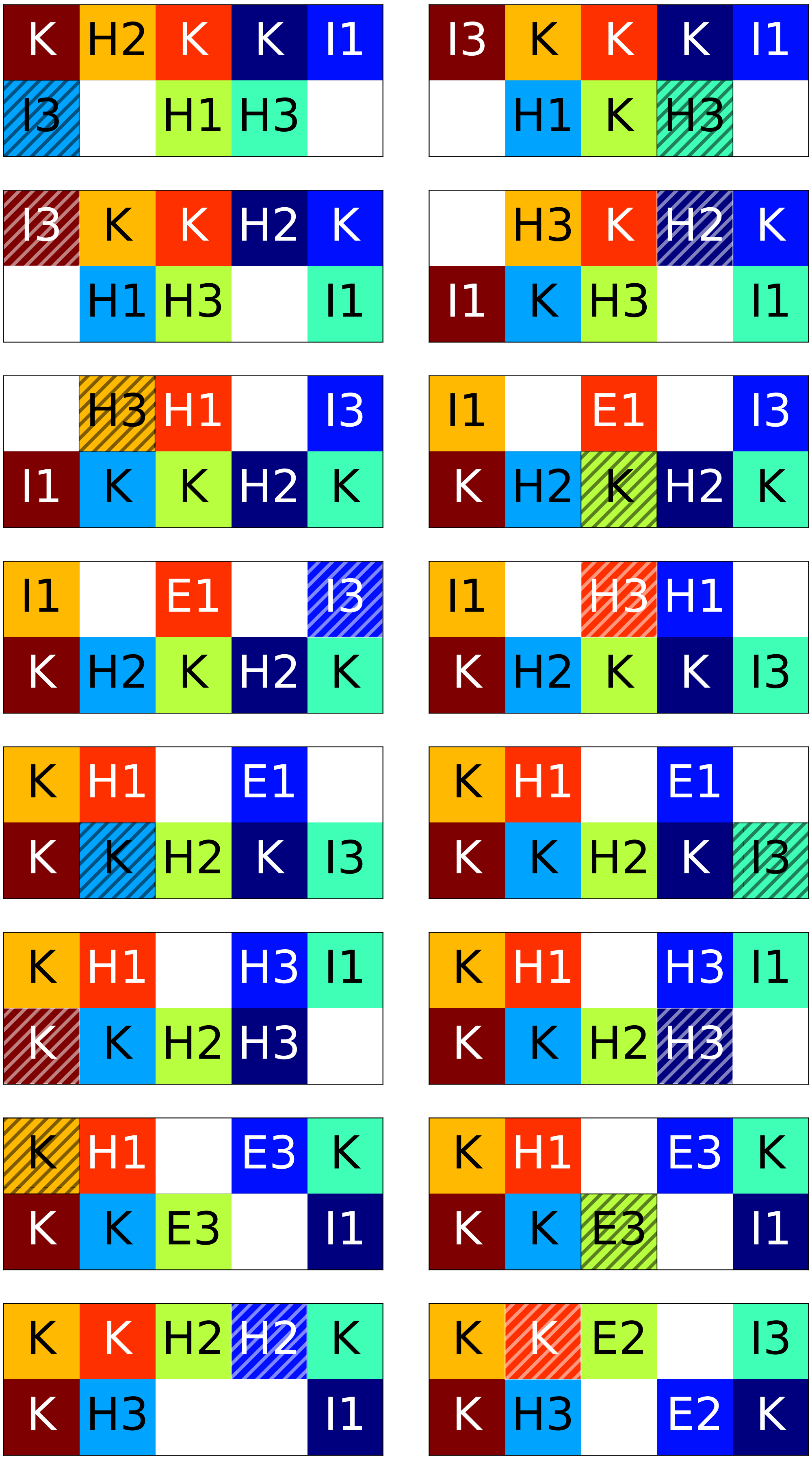


Fig. A.66 (X) Counter-example 36. Dynamic deadlock of period 360. Images show 144-159 of 360.

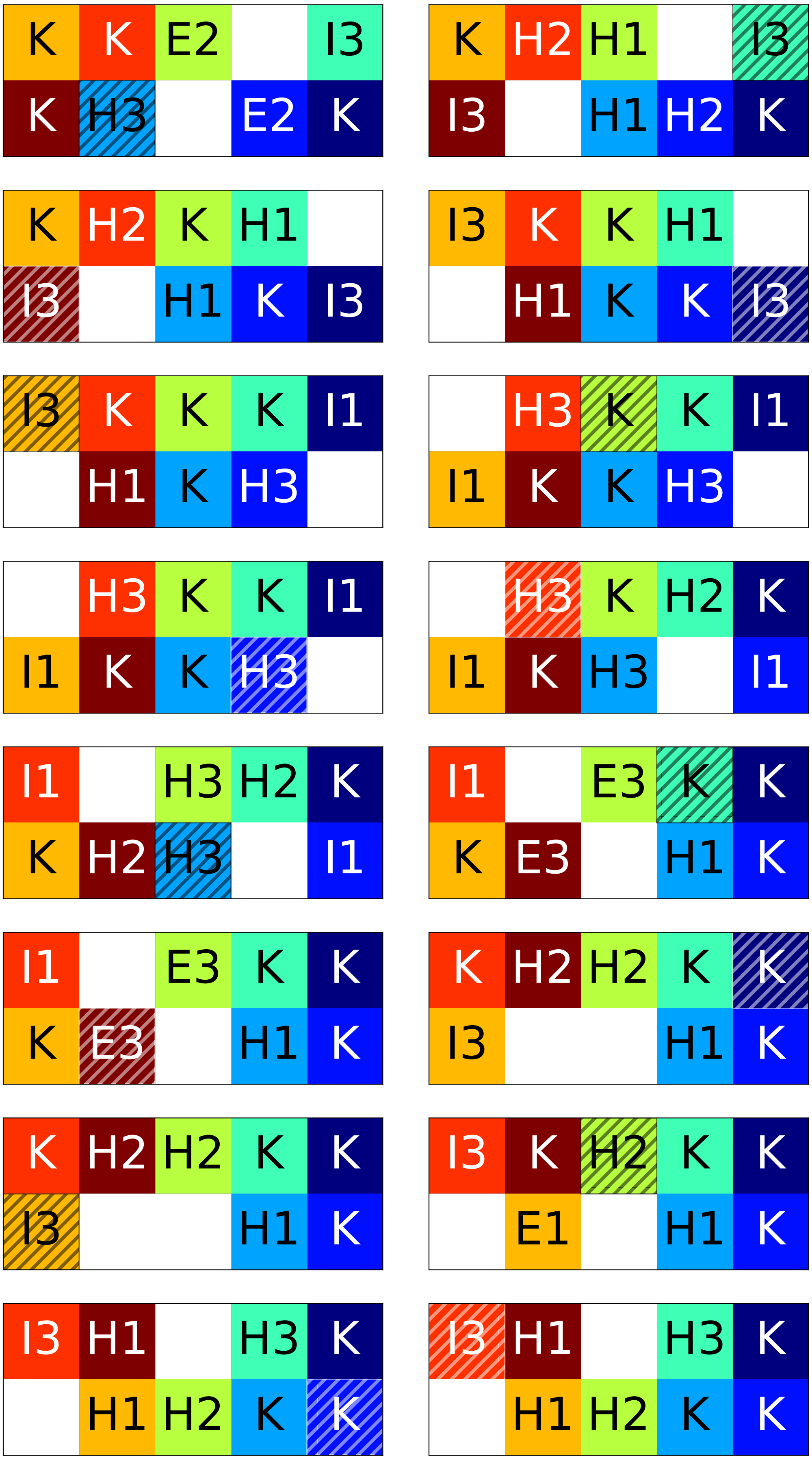


Fig. A.67 (XI) Counter-example 36. Dynamic deadlock of period 360. Images show 160-175 of 360.

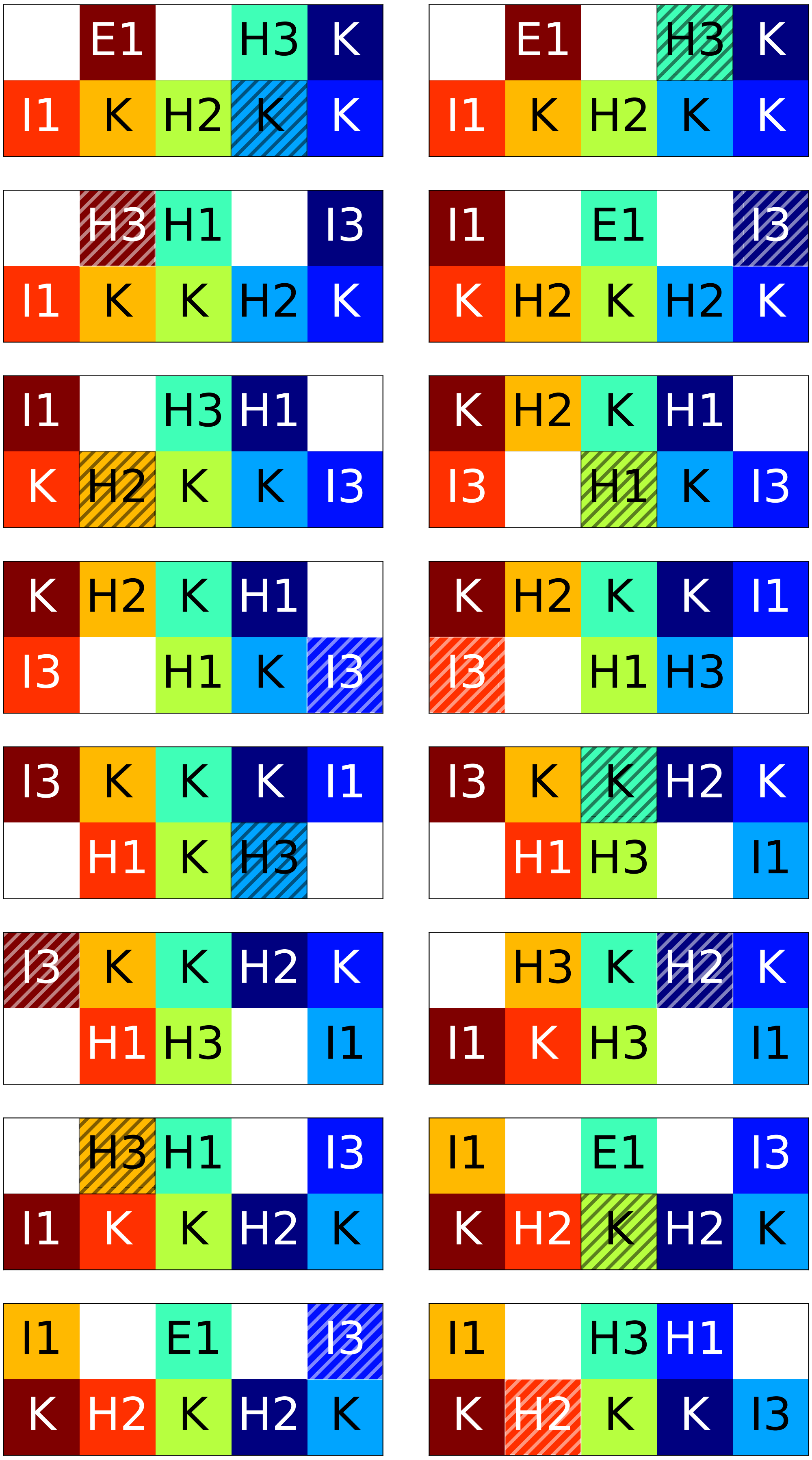


Fig. A.68 (XII) Counter-example 36. Dynamic deadlock of period 360. Images show 176-191 of 360.

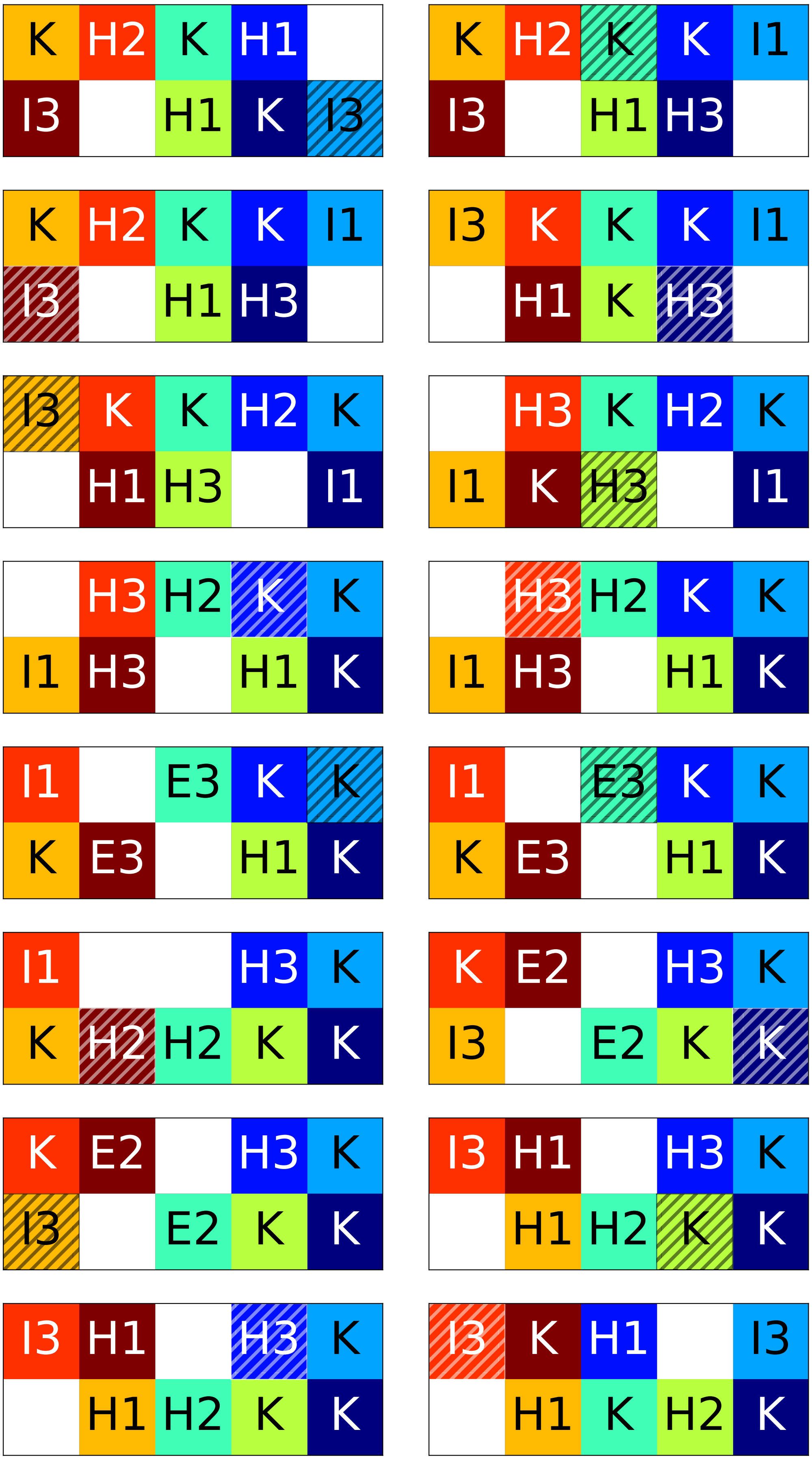


Fig. A.69 (XIII) Counter-example 36. Dynamic deadlock of period 360. Images show 192-207 of 360.

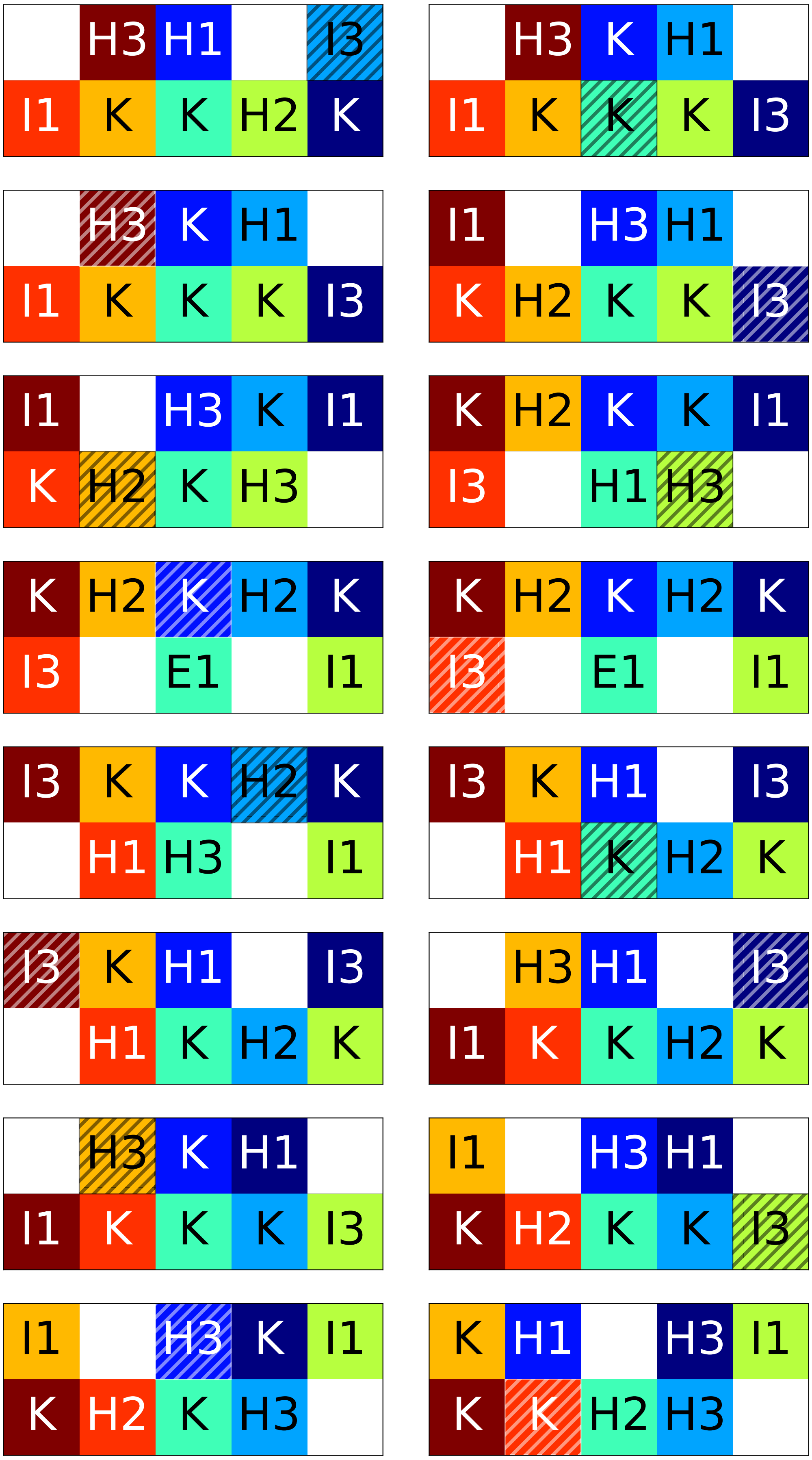


Fig. A.70 (XIV) Counter-example 36. Dynamic deadlock of period 360. Images show 208-223 of 360.

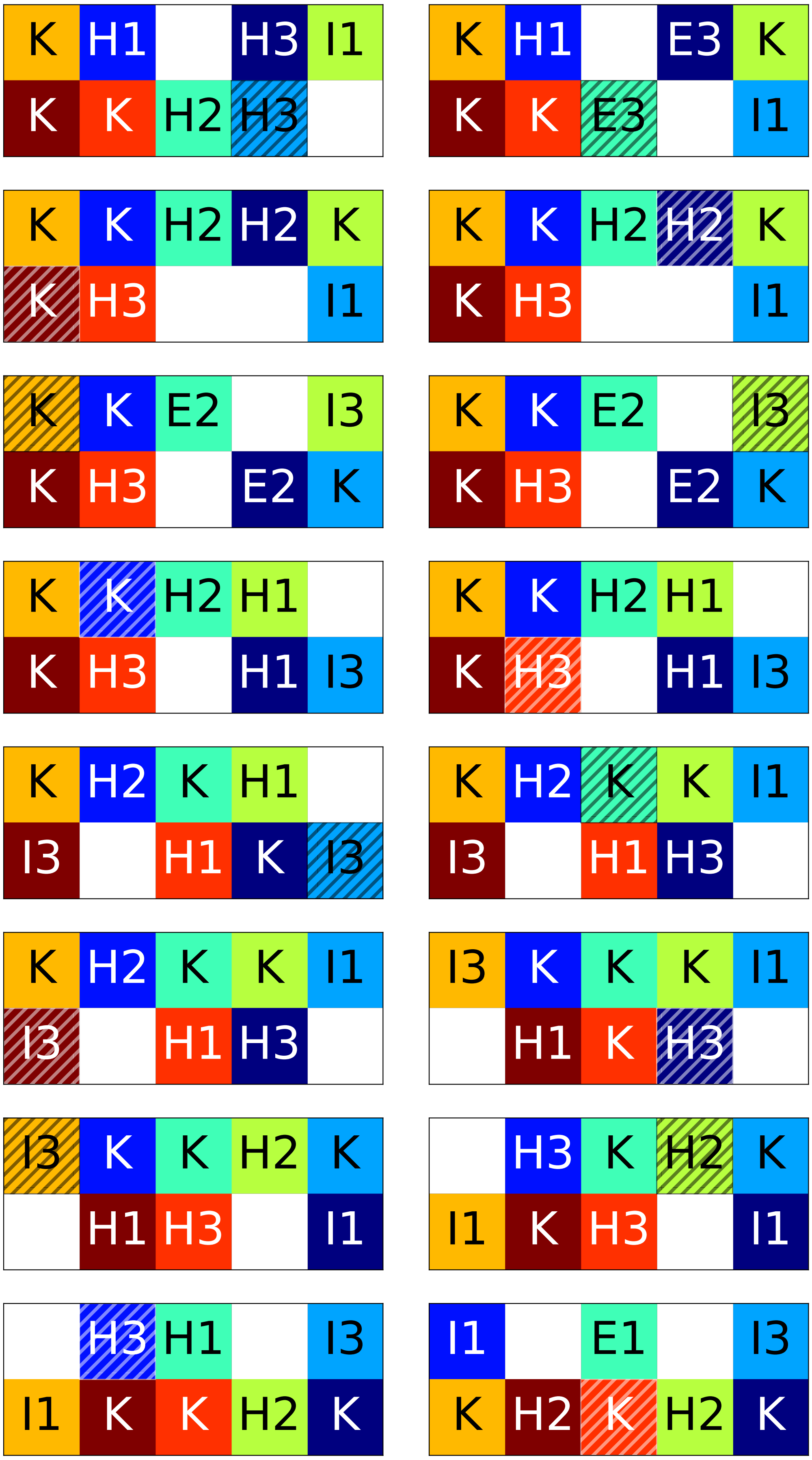


Fig. A.71 (XV) Counter-example 36. Dynamic deadlock of period 360. Images show 224-239 of 360.

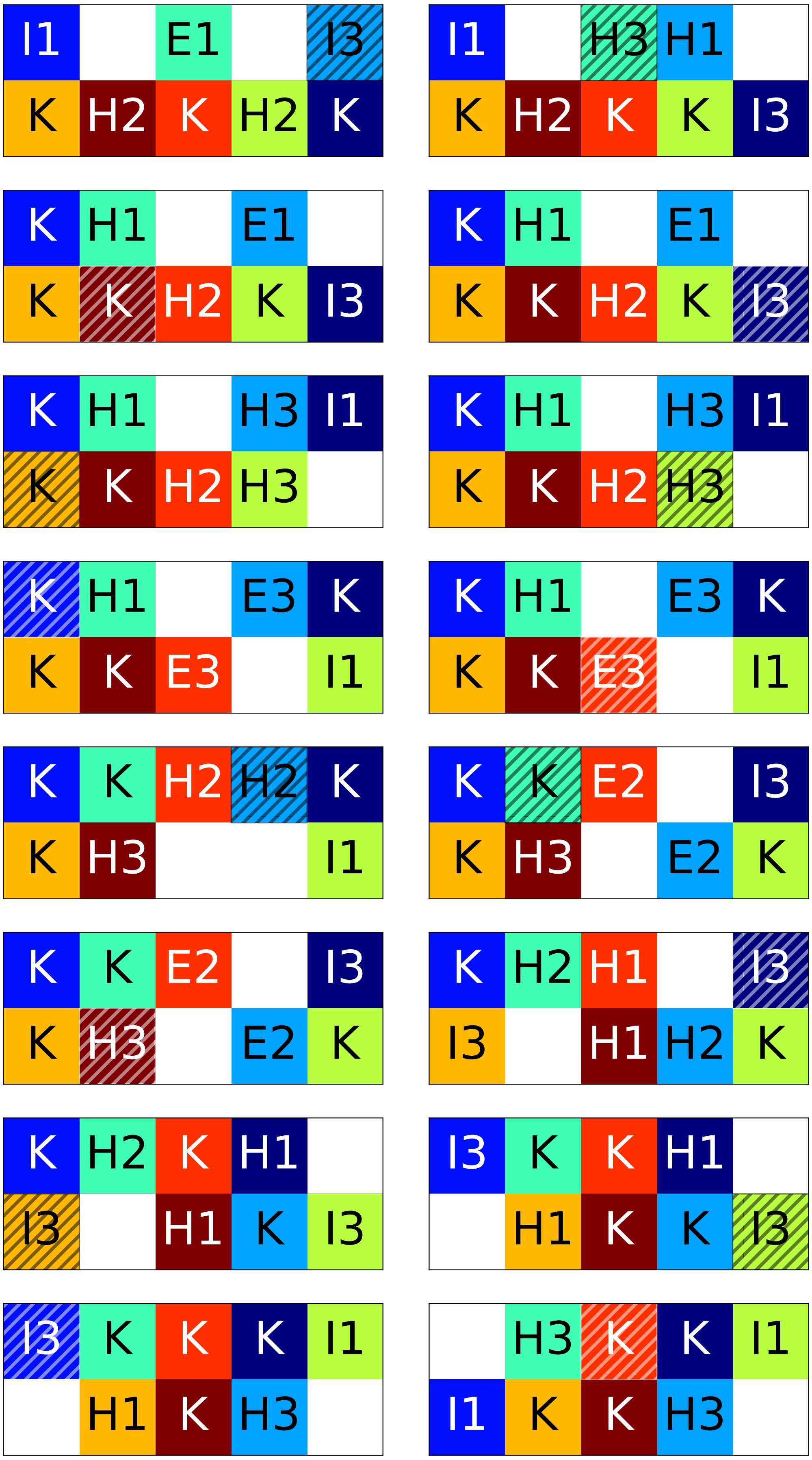


Fig. A.72 (XVI) Counter-example 36. Dynamic deadlock of period 360. Images show 240-255 of 360.

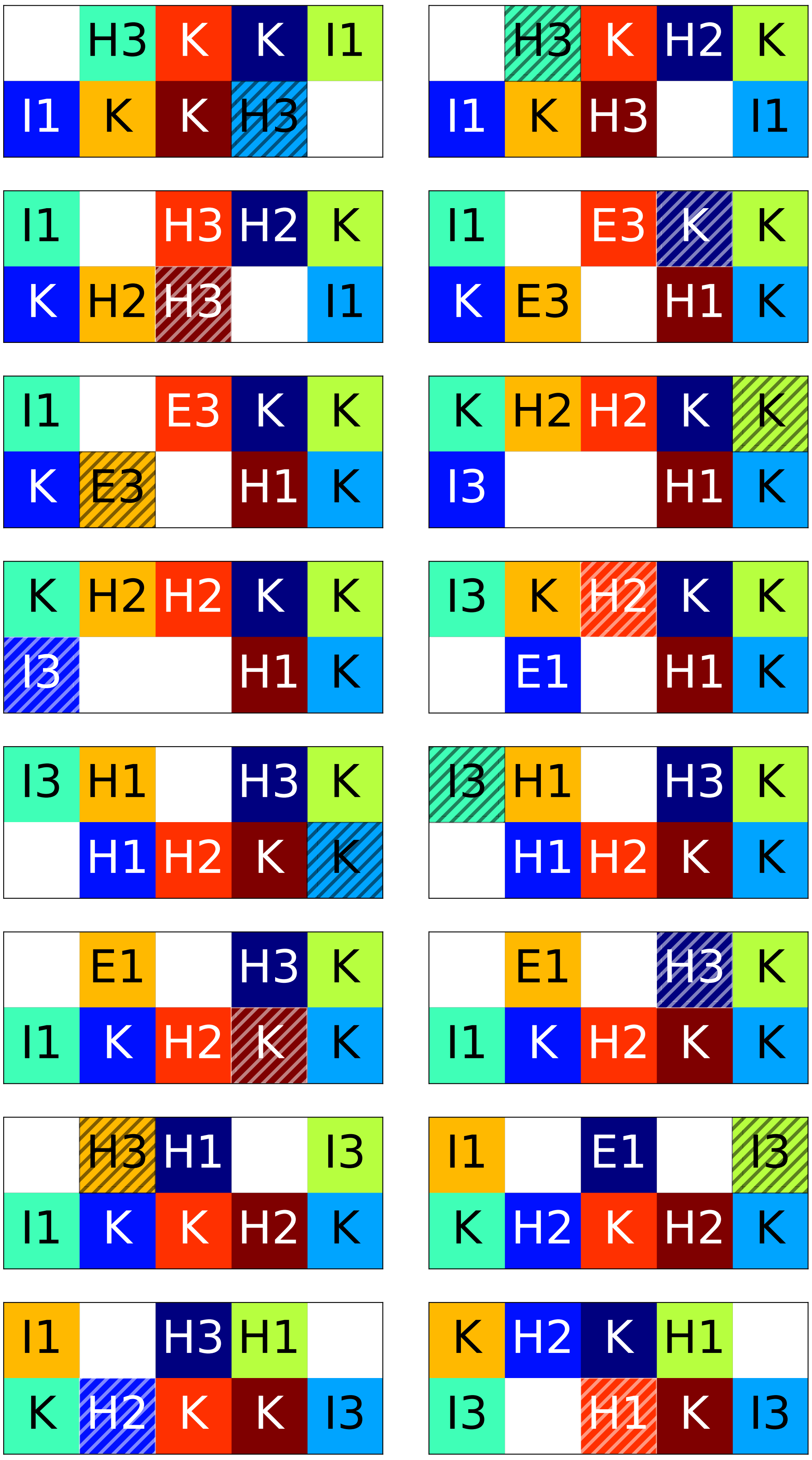


Fig. A.73 (XVII) Counter-example 36. Dynamic deadlock of period 360. Images show 256-271 of 360.

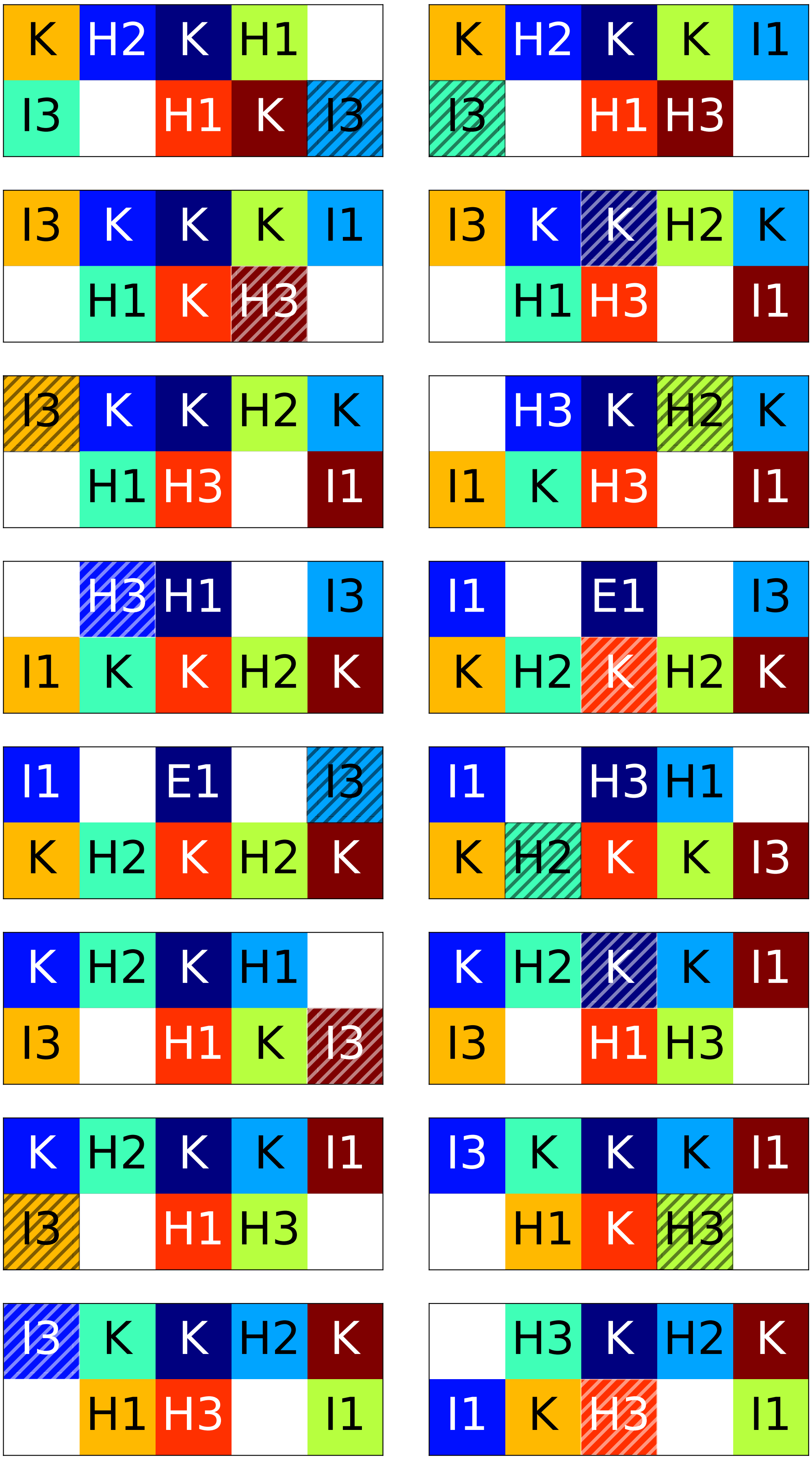


Fig. A.74 (XVIII) Counter-example 36. Dynamic deadlock of period 360. Images show 272-287 of 360.

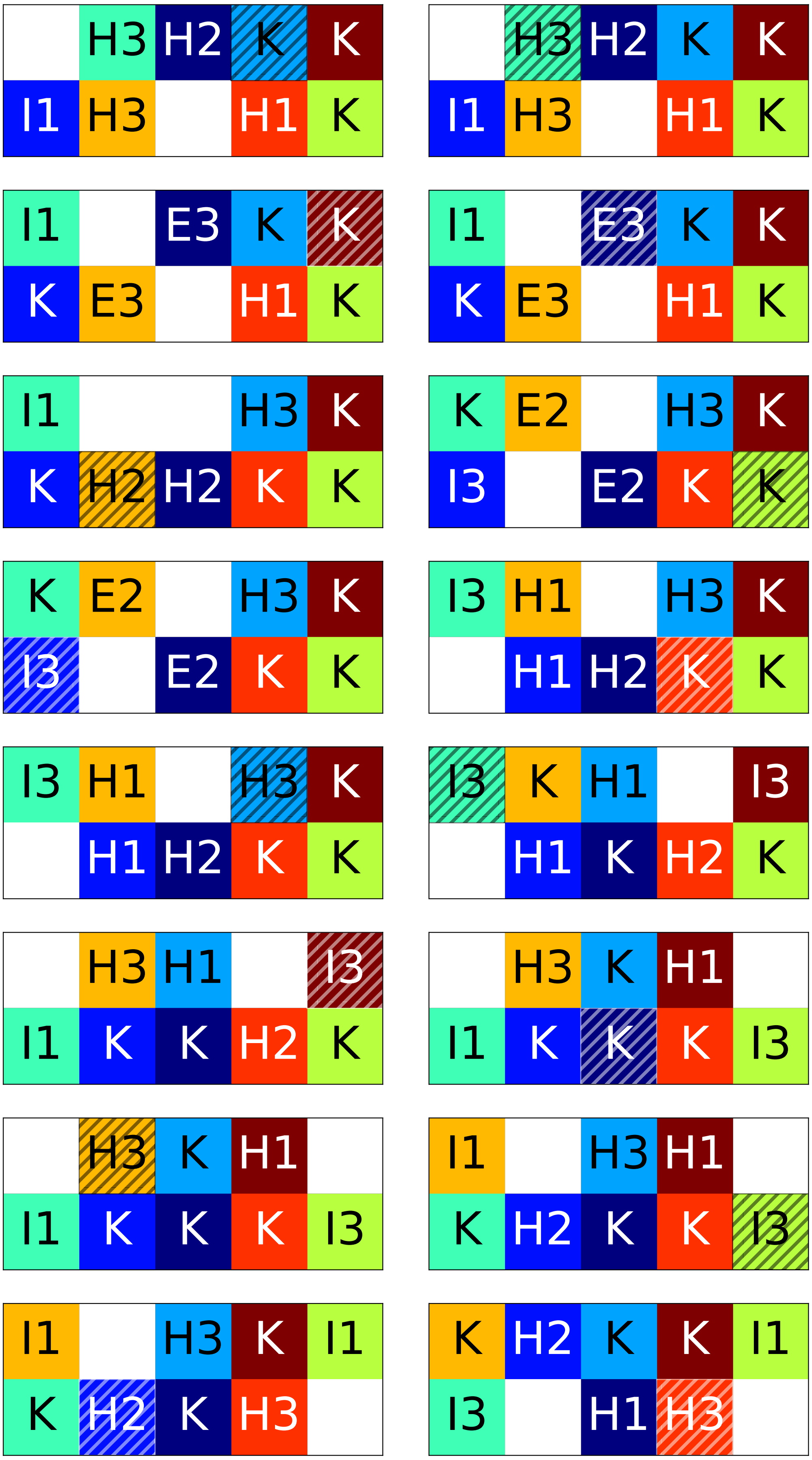


Fig. A.75 (XIX) Counter-example 36. Dynamic deadlock of period 360. Images show 288-303 of 360.

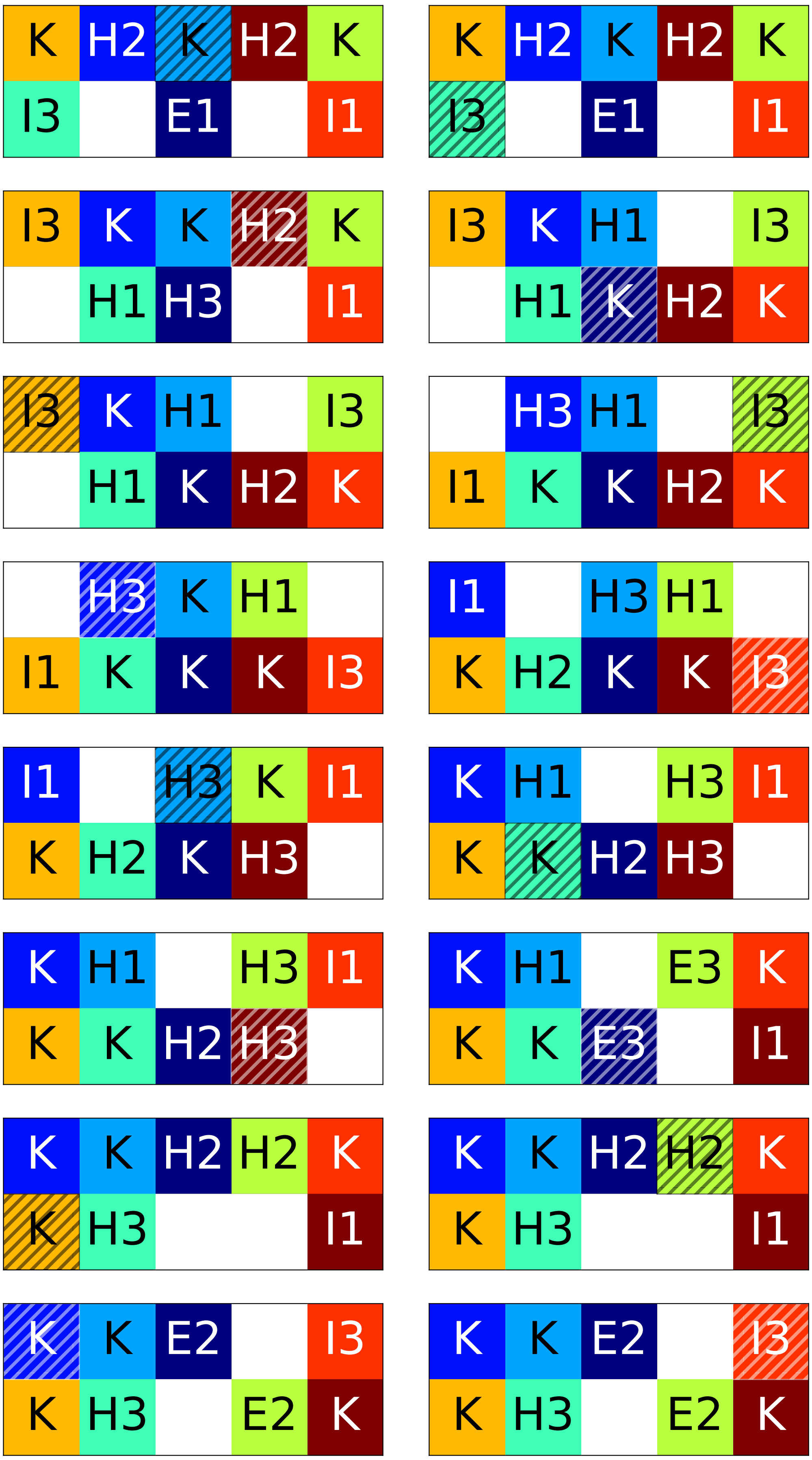


Fig. A.76 (XX) Counter-example 36. Dynamic deadlock of period 360. Images show 304-319 of 360.

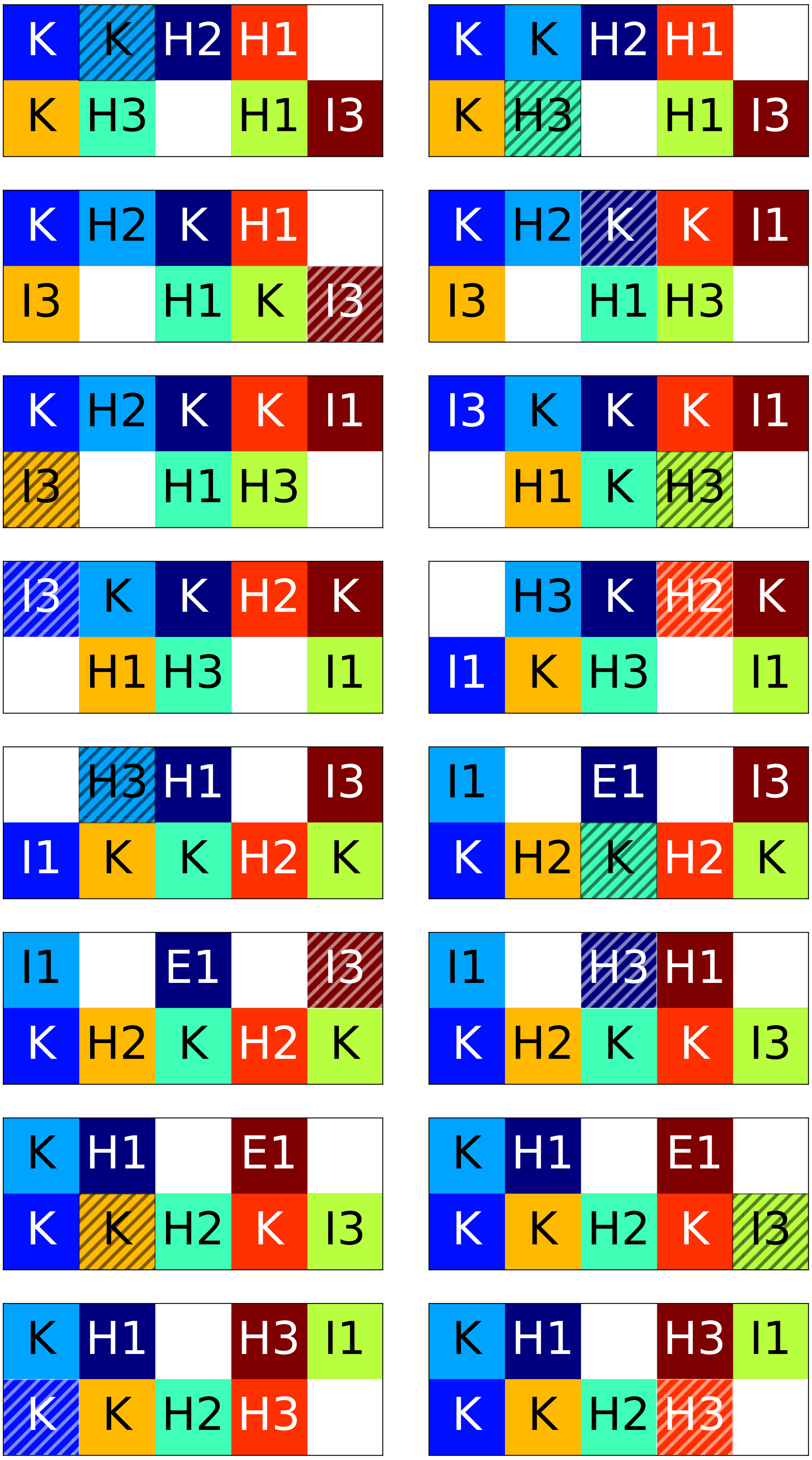


Fig. A.77 (XXI) Counter-example 36. Dynamic deadlock of period 360. Images show 320-335 of 360.

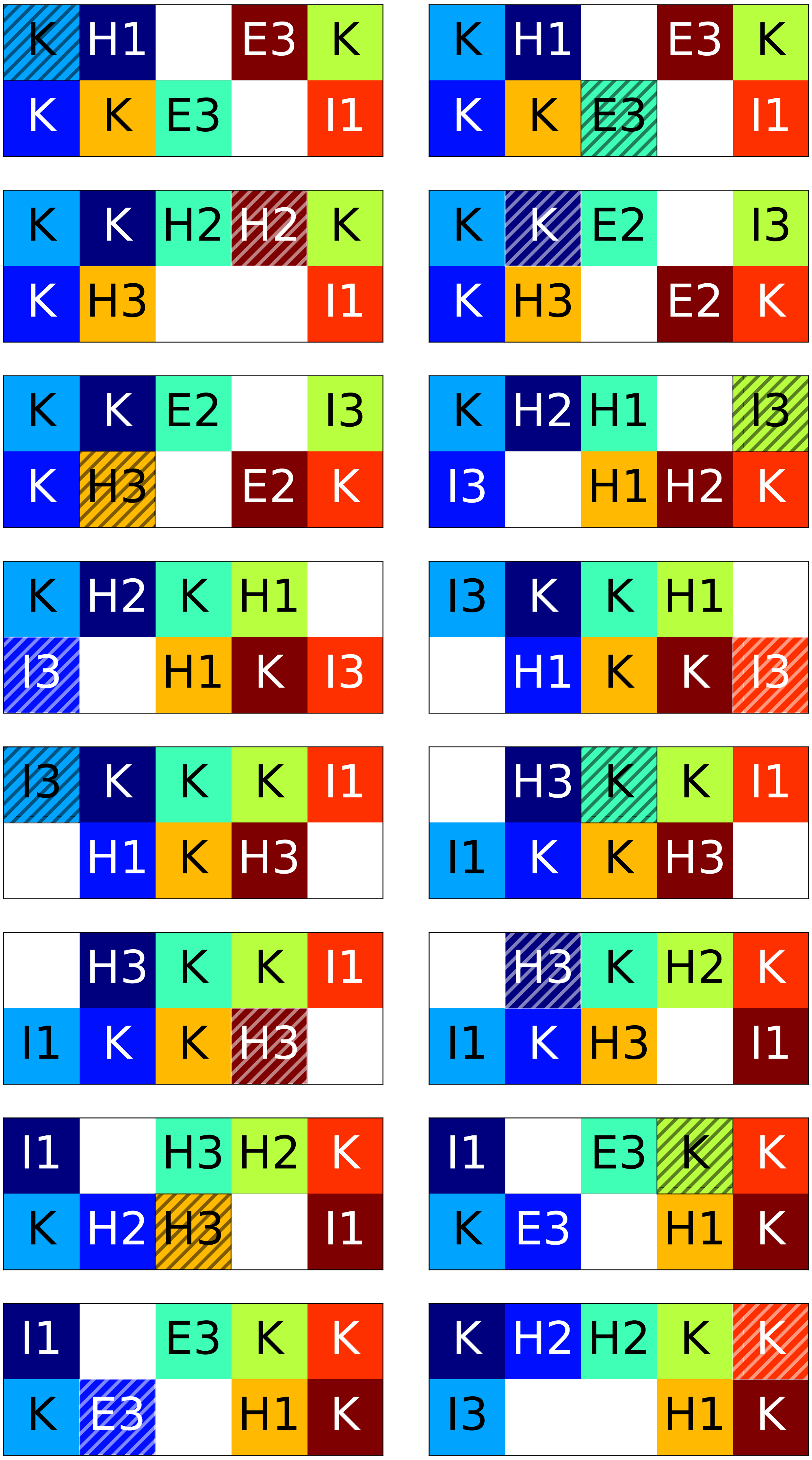


Fig. A.78 (XXII) Counter-example 36. Dynamic deadlock of period 360. Images show 336-351 of 360.

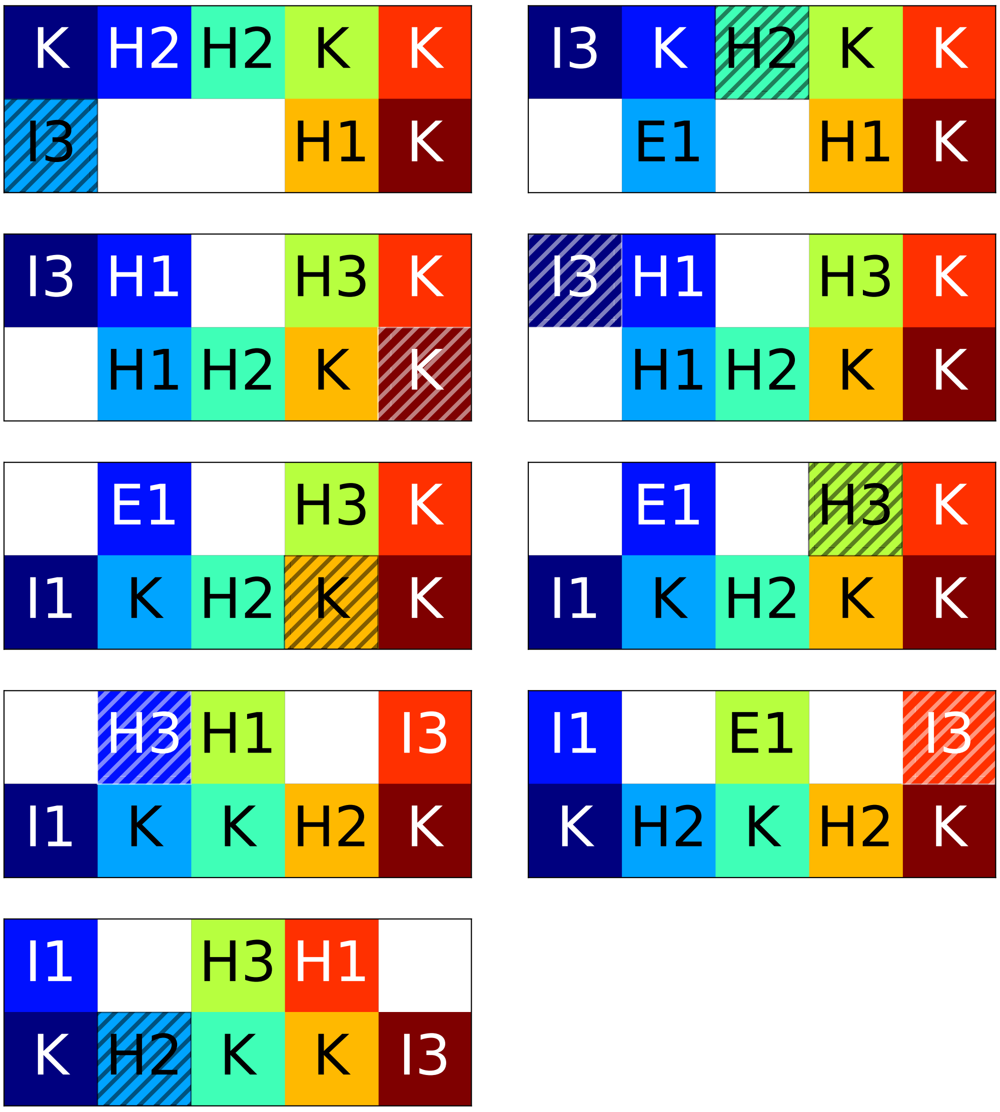


Fig. A.79 (XXIII) Counter-example 36. Dynamic deadlock of period 360. Images show 352-360 of 360.