Property-Driven Design for Robot Swarms: A Design Method Based on Prescriptive Modeling and Model Checking

MANUELE BRAMBILLA, ARNE BRUTSCHY, MARCO DORIGO,
and MAURO BIRATTARI, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium

In this article, we present property-driven design, a novel top-down design method for robot swarms based on prescriptive modeling and model checking. Traditionally, robot swarms have been developed using a code-and-fix approach: in a bottom-up iterative process, the developer tests and improves the individual behaviors of the robots until the desired collective behavior is obtained. The code-and-fix approach is unstructured, and the quality of the obtained swarm depends completely on the expertise and ingenuity of the developer who has little scientific or technical support in his activity. Property-driven design aims at providing such scientific and technical support, with many advantages compared to the traditional unstructured approach. Property-driven design is composed of four phases: first, the developer formally specifies the requirements of the robot swarm by stating its desired properties; second, the developer creates a prescriptive model of the swarm and uses model checking to verify that this prescriptive model satisfies the desired properties; third, using the prescriptive model as a blueprint, the developer implements a simulated version of the desired robot swarm and validates the prescriptive model developed in the previous step; fourth, the developer implements the desired robot swarm and validates the previous steps. We demonstrate property-driven design using two case studies: aggregation and foraging.

Categories and Subject Descriptors: I.2.9 [**Artificial Intelligence**]: Robotics

General Terms: Design, Reliability, Verification

Additional Key Words and Phrases: Swarm robotics, top-down design, prescriptive modeling, model checking, aggregation, foraging

ACM Reference Format:
Manuele Brambilla, Arne Brutschy, Marco Dorigo, and Mauro Birattari. 2014. Property-driven design for robot swarms: A design method based on prescriptive modeling and model checking. ACM Trans. Autonom. Adapt. Syst. 9, 4, Article 17 (December 2014), 28 pages.
DOI: http://dx.doi.org/10.1145/2700318

1. INTRODUCTION

Swarm robotics is an approach to the coordination of large groups of robots that takes inspiration from social insects, such as ants, bees, and termites [Şahin 2005].

The research leading to the results presented in this article received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement n.246939 and from the EU project ASCENS, 257414.
Arne Brutschy, Marco Dorigo, and Mauro Birattari acknowledge support from the F.R.S.-FNRS of Belgium's Wallonia-Brussels Federation, of which they are a research fellow, a research director, and a research associate, respectively.
Authors' addresses: M. Brambilla, A. Brutschy, M. Dorigo, and M. Birattari, IRIDIA, CoDE, Université Libre de Bruxelles, 50 Av. Franklin Roosevelt CP 194/6, 1050 Brussels, Belgium; emails: mbrambil@ulb.ac.be; abrutsch@ulb.ac.be; mdorigo@ulb.ac.be; mbiro@ulb.ac.be.

DOI: http://dx.doi.org/10.1145/2700318

Swarm robotics aims at developing systems that are fault tolerant, scalable, and flexible [Dorigo et al. 2014].

Robot swarms are self-organized systems that can be observed at two levels: the individual, also called *microscopic*, level; and the collective, also called *macroscopic*, level. The individual level is the behavior displayed by a single robot. The collective level is the behavior displayed by the swarm and is the result of the interaction of the individual behaviors.

On the one hand, this dual nature of swarm robotics systems is key in achieving fault tolerance, scalability, and flexibility. On the other hand, it is the source of difficult design challenges. In fact, the swarm robotics engineer must think at the collective level but develop at the individual level: developers of robot swarms are caught between collective-level missions, such as "monitor the perimeter of a building for intruders" or "carry these heavy objects from here to there," and individual-level software, as the only controllable components of a robot swarm are the individual behaviors of the robots. Conversely, at the individual level, collective-level goals could be meaningless: for example, for a single robot, it is impossible to monitor an entire building at the same time or transport an object if it is too heavy to be moved. Thus, the developer needs to design the behavior of the individual robots so that their interaction will result in the collective-level behavior that is needed to accomplish the mission.

Unfortunately, the design and development of individual-level behaviors to obtain a desired swarm-level goal is, in general, quite difficult, as it is difficult to predict and thus design the nonlinear interactions of tens or hundreds of individual robots that result in a desired collective behavior. The difficulty to predict and design such interactions and the lack of a centralized controller make traditional system engineering approaches ineffective [Wooldridge and Jennings 1998; Banzhaf and Pillay 2007].

Some approaches to the design of robot swarms have been proposed in the last years. However, as discussed in Section 2, these approaches present limitations, and an effective approach to the top-down design of robot swarms is still missing.

In this article, we present property-driven design, a novel top-down design method for robot swarms based on prescriptive modeling and model checking. In our approach, the developer creates a prescriptive model of the desired robot swarm and uses it as a blueprint for the implementation and improvement of the swarm. The use of model checking allows the developer to formally verify properties directly on the model, reducing the need for testing in simulation or with robots. In property-driven design, different "views" of the system to realize are produced, from the most abstract (the properties of the system) to the most concrete (the final robot swarm).

Property-driven design addresses the shortcomings of the existing approaches:

—It aims at providing a method to formally specify the requirements of the desired robot swarm.
—It reduces the risk of developing the "wrong" robot swarm—that is, a robot swarm that does not satisfy the requirements.
—It promotes the reuse of available models and tested solutions.
—It can be used to develop platform-independent models that help in identifying the best robotic platform to use.
—It helps to shift the focus of the development process from implementation to design.

Property-driven design can be used to design and develop any kind of robot swarms. In particular, its applicability is closely linked to the applicability of mathematical modeling and model checking: property-driven design can be applied to the class of robot swarms that can be described using mathematical modeling and whose properties can be validated using model checking. Whether this class encompasses all possible robot swarms is an open research question.

Property-driven design is a step forward in the development of *swarm engineering*, which is the systematic application of scientific and technical knowledge to specify requirements, design, realize, verify, validate, operate, and maintain an artificial swarm intelligence system [Brambilla et al. 2013].

To illustrate and validate property-driven design, we apply it to two case studies: aggregation and foraging.

In Section 2, we present the related literature on design methods and model checking for swarm robotics. In Section 3, we present model checking and its two components: models and properties. In Section 4, we present property-driven design. In Section 5, we present the two case studies.

2. RELATED WORK

In this section, we first discuss the literature on design methods and then the literature on model checking in swarm robotics.

Design methods. The design of multirobot systems has been addressed in many research papers [Zambonelli et al. 2001; Bordini 2009; Goldberg and Matarić 2001]. However, the design of robot swarms poses challenges that are not present in other multirobot systems. Indeed, the characteristics of robot swarms, such as high number of individuals, strong decentralization, simple behaviors, local communication, and action, are usually regarded as characteristics that make a multirobot system "too complex to manage effectively" [Wooldridge and Jennings 1998].

Traditional multirobot approaches are thus of limited use when developing robot swarms. For this reason, other ad hoc design approaches have been proposed.

Kazadi et al. [2009] developed a design approach based on Hamiltonian vector fields called the *Hamiltonian method*: starting from a mathematical description of a collective behavior, the method derives microscopic rules that minimize or maximize a selected numerical value (e.g., the virtual potential energy of a particular state of the swarm). The Hamiltonian method has the major drawback that it deals only with spatially organizing behaviors such as pattern formation.

Berman et al. [2009] proposed a top-down approach to the design of a task allocation behavior. In this approach, the system is described as a Markov chain in which states represent tasks and edges represent the possibility for a robot to move from one task to another. The probabilities that govern how robots change task are obtained using a stochastic optimization method that minimizes the time needed to converge to the desired allocation. This approach is specific to task allocation and has not been extended to other collective behaviors.

Hamann and Wörn [2008] proposed a method inspired by statistical physics. The authors use Langevin equations to describe the individual behaviors of the robots, and through analytical means, they derive a Fokker-Planck equation describing the collective behavior of the system. A similar approach was adopted also by Berman et al. [2011], who used a set of advection-diffusion-reaction partial differential equations to derive the individual behaviors of a swarm performing task allocation. Both methods are based on advanced mathematical techniques and on the ability of the developer to model the robot interactions. Moreover, such methods rely on ordinary or partial differential equations, which provide reliable results only if it is assumed that the swarm size tends to infinity. In swarm robotics, this is often not the case, since typically robot swarms are composed of no more than a hundred robots and often of just a few tens of robots [Brambilla et al. 2013].

Model checking in swarm robotics. Property-driven design is based on model checking [Baier and Katoen 2008], a technique to prove properties of a system in a formal way.

Winfield et al. [2005] were the first to model a robot swarm in the form of the *and*-composition of individual-level models, as well as linear temporal logic to define properties of individual robots and of the swarm. Dixon et al. [2012] further extended focusing on the use of model checking to analyze a similar model. The approach presented in these works is not scalable, as the number of states of the model increases exponentially with the number of robots. Furthermore, linear temporal logic, which deals only with binary values (true/false), is a suboptimal choice to analyze robot swarms, which are systems characterized by stochastic properties.

Recently, Konur et al. [2012] adopted a different approach. The authors used model checking on a macroscopic model of a robot swarm performing foraging. They specified the desired properties of the system using probabilistic computation tree logic (PCTL), a temporal logic that includes probabilistic aspects (see Appendix II for a description of PCTL). This approach is able to overcome the limits of linear temporal logics. Moreover, the use of a macroscopic model, instead of a microscopic one, allows this approach to deal with systems composed of tens of robots. We used a similar approach in a previous work [Brambilla et al. 2012], in which model checking and PCTL were used to verify properties of a robot swarm performing aggregation.

In a work on the use of Bio-PEPA in swarm robotics [Massink et al. 2013], we were the first to use statistical model checking (presented in Appendix III) to analyze a collective decision-making behavior. Statistical model checking overcomes the scalability issues of complete model checking, allowing us to analyze models of large swarms.

3. MODEL CHECKING

Property-driven design is based on prescriptive modeling and model checking. Model checking is a formal method that allows one to formally prove that a model satisfies a given property. The idea is that a system can be modeled using a formal mathematical model and then is checked against a property defined using a formal logic language. In this work, we use Markov chains to define models and probabilistic temporal logics to define properties. Markov chains can be used to describe the behavior of a robot swarm. Markov chains and probabilistic temporal logics have been proven to be well suited for model checking in swarm robotics [Konur et al. 2012; Brambilla et al. 2012] due to their simplicity and expressive power.

The most common way to describe the behavior of a swarm using a Markov chain is by using a macroscopic Markov chain model: each state of the Markov chain is augmented with a variable that tracks the number of robots in that particular state. Note that, differently from rate equations, this variable tracks the actual number of robots, not the ratio. This avoids rounding problems that might occur with rate equations.

Properties can be defined using probabilistic temporal logics. In this article, we use PCTL.[1] PCTL is well suited for swarm robotics systems, as it can capture both time-related and stochastic aspects. Examples of properties that can be expressed using PCTL are *with a probability greater than 0.75, the system completes the task before 1,000 timesteps* or *there is a probability greater than 0.95 that every request is answered within 10 timesteps*.

More details about Markov chains, probabilistic logics, and model checking can be found in the appendix.

[1]Note that PCTL can be used only with discrete-time Markov chains. For continuous-time Markov chains, it is necessary to use continuous stochastic logic (CSL). For our goals and purposes, however, the two logics are equivalent. Thus, for the sake of simplicity, we refer to PCTL also when dealing with continuous-time Markov chains, even though this is formally incorrect.

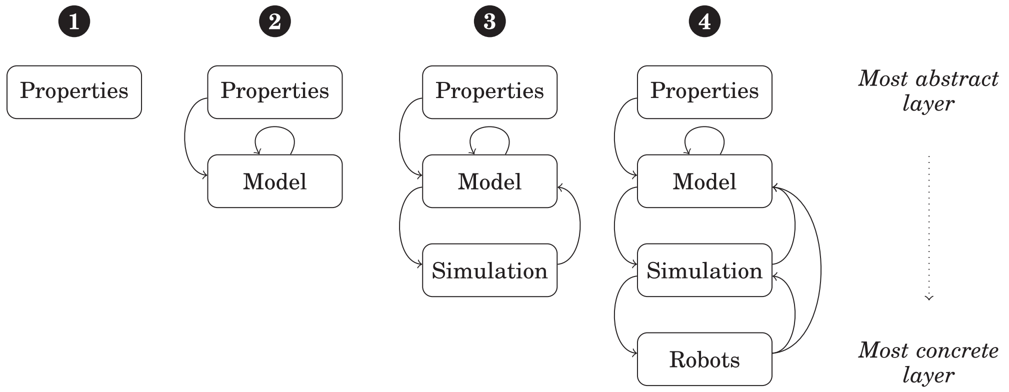


Fig. 1. The four phases of property-driven design.

4. PROPERTY-DRIVEN DESIGN

Property-driven design is composed of four-phases: (1) the requirements of the robot swarm are first formally described in the form of desired properties; (2) subsequently, a prescriptive model of the robot swarm is created; (3) this prescriptive model is used as a blueprint to implement and improve a simulated version of the desired robot swarm; and (4) the final robot swarm is implemented.

A schema showing the different phases of property-driven design is presented in Figure 1.

In each phase of property-driven design, a new layer is added to the system. Layers differ in their level of abstraction: the *properties* layer is the most abstract, in which only the goal characteristics of the robot swarm are stated, and the *robots* layer is the most concrete, in which the actual software for the real robots is developed and deployed. The addition of a new layer brings the system closer to its final state.

Each phase of property-driven design is characterized by a *development/validation* cycle: the focus of the developer is on the newly introduced layer, but all previously developed layers are still active—that is, they are still improved and expanded, should this be needed to guarantee the consistency of the layers. The newly introduced layer provides the developer with further information on the system. This information is used to improve the system being developed, to validate its prescriptive model, and to verify its properties. For example, the development of the system in simulation provides the developer with new data that can be used to improve and validate the prescriptive model and further verify that the desired properties hold.

Phase One: Properties. In this phase, the developer formally specifies the requirements of the robot swarm in the form of desired properties. These properties are the distinguishing features of the robot swarm that the developer wants to realize. They can be task specific, such as *the system eventually completes task X*, or they can express more generic properties, such as *the system keeps working as long as there are at least N robots* or *the system will never be in state Y for more than t timesteps*. The more precise and complete these properties are in this phase, the more the developed robot swarm will meet expectations. Clearly stated requirements help to reduce the risk of developing "the wrong robot swarm." For simplicity, in this article, we assume that requirements do not change during the development of the robot swarm.

Phase Two: Model. In this phase, the developer creates a prescriptive model of the robot swarm. Usually, the prescriptive model describes how robots change state over time, where a state is an abstract simplified description of the actions of a robot (see

Appendix I). To create this prescriptive model, a creative step is necessary in which the developer devises the behavior that the robots will execute. However, in property-driven design, it is not necessary that this first behavior is exhaustively defined, particularly regarding the implementation details. The prescriptive model should be sufficiently detailed to capture the behavior of the robots and their interaction but not too detailed, to avoid unnecessary complication.

Once a first draft of the prescriptive model is produced, the desired properties stated in phase one are verified using model checking. Given that robot swarms are characterized by uncertainty, the most natural choice is to use probabilistic model checking [Baier and Katoen 2008]. As in test-driven development [Beck 2003], at first it is possible that the prescriptive model does not satisfy all of the desired properties. In an iterative process, the developer expands and improves the prescriptive model until the properties are satisfied. The outcome of this process is a prescriptive model of the collective behavior of the robot swarm that satisfies the stated properties.

Note that it might not always be possible to identify all numerical parameters of the model in this phase. If some parameter cannot be identified, the model will be completed after an initial implementation in phase three. Nonetheless, even a partially completed model can be useful as a blueprint to implement the system.

Phase Three: Simulation. In this phase, the developer uses the prescriptive model as a blueprint to implement and improve the robot swarm using a physics-based computer simulation (henceforth, simply simulation). By *blueprint*, we mean that the prescriptive model is used to identify the most relevant aspects of the robot swarm to realize. This allows the developer to focus on these aspects and neglect other minor details. For example, if a prescriptive model shows that by entering state i an individual robot affects the performance of the whole swarm more than by entering state j, the developer can focus on the first and temporarily ignore the second. Moreover, concentrating on the prescriptive model at design time allows the developer to direct his efforts toward high-level decisions rather than toward the implementation.

It is possible that the implementation choices or other unforeseen aspects of the system result in a simulated system that does not behave as predicted by the prescriptive model. In this case, the developer must go back to the previous phases, modify the prescriptive model to consider the results obtained from the simulation, and verify whether the required properties still hold true.

Phase Four: Robots. In this phase, the developer realizes the final robot swarm. Similarly to the transition between the prescriptive model and the simulation, if the implementation on robots reveals that some assumptions made during the previous phases do not hold, it might be necessary to modify the simulated version or the prescriptive model to keep all levels consistent.

5. CASE STUDIES

In this section, we illustrate property-driven design using two very common case studies from the swarm robotics literature [Brambilla et al. 2013]: aggregation and foraging. These case studies have been chosen because they allow us to develop behaviors with different distinguishing characteristics.

In both case studies, we perform model checking using PRISM, a state-of-the-art suite for model checking [Kwiatkowska et al. 2004]. PRISM is free and released as open source software under the GNU General Public License (GPL).[2]

[2]http://www.prismmodelchecker.org.

5.1. Aggregation

In the first case study, we tackle *aggregation*: robots have to cluster in an area of the environment. The robots have neither knowledge of the position of the other robots nor a map of the environment. We choose aggregation as a case study for various reasons: (1) aggregation is a simple case study, and this allows us to focus on the development process; (2) aggregation is a common case study in swarm robotics (for a review, see Brambilla et al. [2013]); and (3) the behaviors used to tackle aggregation possess many of the salient traits of swarm robotics—they are completely distributed, based on simple robot-to-robot interactions, and are characterized by stochasticity and spatial aspects.

The aggregation case study that we discuss in this article is similar to the one presented by Jeanson et al. [2005]. We consider a dodecagonal environment with two black spots of equal size referred to as area a and area b. We call area c the remaining white area. Each of the black spots is large enough to host all the robots. (See Figure 5 for a picture of the environment.) Let N be the number of robots in the swarm. We consider three swarm sizes: $N \in 10, 20, \text{and } 50$. We use three different arenas for the three different swarm sizes, respectively, of 4.91 m^2, 19.63 m^2, and 50.26 m^2. The surfaces of the black areas, respectively, are 0.38 m^2, 0.78 m^2, and 3.14 m^2.

In the following discussion, we will apply the four-phase process explained in Section 4.

Phase One: Properties. The main property that the robot swarm must satisfy is that eventually all robots form an aggregate either on area a or area b. We set a time limit that depends on the number of robots composing the swarm, as we expect that larger swarms will require a longer time to aggregate.

We define the following event of interest,

$$E_1 \equiv [F \leq 100N \quad (S_a = N)|(S_b = N)],$$

and property,

$$P(E_1) \geq 0.75. \tag{1}$$

In less formal terms, E_1 means that in less than or equal to $100N$ seconds ($F \leq 100N$), where N is the total number of robots, the number of robots in area a (S_a) or in area b (S_b) is equal to the total number of robots in the swarm ($S_a = N|S_b = N$). $F \leq t$ is the bounded *eventually* operator, which is true if and only if the subsequent formula becomes true in a future state that is closer to now than t seconds. The property we require from the final swarm is that E_1 is true with a probability greater than or equal to 0.75.

Another property the swarm must satisfy is that the aggregate, once formed, is stable for at least 10 seconds—that is, robots do not change state once the aggregate is formed. We want this to happen more than two thirds of the times in which an aggregate is formed.

We define the following event of interest,

$$E_2 \equiv [G < 10\,(S_a = N)|(S_b = N)],$$

which indicates the event of staying in the aggregate state for less than 10 seconds, and property,

$$(S_a = N)|(S_b = N) \Rightarrow 1 - P(E_2) \geq 0.67. \tag{2}$$

In natural language, Property 2 can be expressed in this way: starting from a state in which the robot swarm is aggregated ($S_a = N|S_b = N$), is it true with probability of at least 0.67 that event E_2 does not occur ($1 - P(E_2) \geq 0.67$)—that is, that the swarm does not leave the aggregate state for at least 10 seconds ($G < 10$)? The operator $G < t$

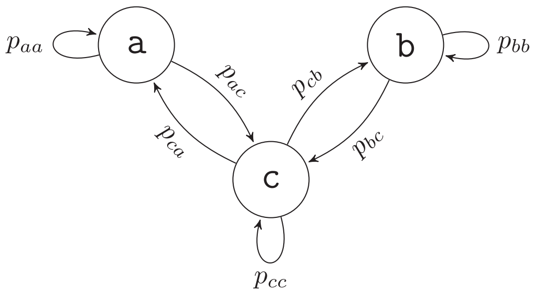


Fig. 2. The prescriptive model for aggregation. Each state is used to count the number of robots in the corresponding area.

is the bounded *always* operator, which is true if and only if the subsequent formula is true for at most t seconds.

Phase Two: Model. We develop a discrete-time macroscopic prescriptive model. In a macroscopic model, each state is used to track the number of robots that are in a specific area of the environment or are performing a specific action (see Appendix I). To develop the prescriptive model for the aggregation case study, we consider the three areas in which the environment is divided. We define three states: a, b, and c, for areas a, b, and c, respectively. To each state, we associate a variable, S_a, S_b, and S_c, respectively, to track the number of robots currently in that area. Note that $S_a + S_b + S_c = N$. Figure 2 provides a graphical representation of the prescriptive model.

In this initial stage of the definition of the prescriptive model, we assume that the system can be effectively described by a *nonspatial* model—that is, a model in which the trajectories of the robots are ignored and a robot can move instantaneously from area c to area a or b, and vice versa. Moreover, for the moment, we also ignore the effects of interferences between robots [Lerman et al. 2005]. In case these assumptions prove to be not realistic and the results obtained with the prescriptive model do not match those obtained in simulation or with the final robot swarm, we will modify them in the following phases, as explained in Section 4.

The first design attempt is as follows: a robot performs random walk, and when it finds a black area, it stops. A robot stopped on a black area has a fixed probability to leave.

Since the prescriptive model is nonspatial and ignores interference, we consider only the geometric properties of the areas to compute p_{ca}—that is, the instantaneous probability that a robot transitions from S_c to S_a. A robot in area c can either go to area a, go to area b, or stay in area c. This means that a robot in area c has a probability of going from area c to area a equal to $p_{ca} = \frac{A_a}{A_{arena}}$, where A_a is the surface of area a and A_{arena} is the surface of the entire environment; a probability of going from area c to area b equal to $p_{cb} = \frac{A_b}{A_{arena}}$, where A_b is the surface of area b; and a probability of staying in area c equal to $p_{cc} = \frac{A_c}{A_{arena}} = 1 - (p_{ca} + p_{cb})$, where $A_c = A_{arena} - (A_a + A_b)$. Note that $p_{ca} = p_{cb}$, since $A_a = A_b$.

The remaining probabilities depend on the behavior of the robots. The aggregate can be obtained in area a or area b, and thus we set the probabilities of leaving these two areas to be equal: $p_{ac} = p_{bc}$. A robot in area a can only go to area c or stay in area a, and thus $p_{aa} = 1 - p_{ac}$. The same holds for area b. From the preceding scenario, it follows that $p_{aa} = p_{bb}$. The only probability remaining is p_{ac}—in other words, p_{ac} is the only free parameter of the system. Using model checking, we can explore the parameter space of p_{ac} to find the value that results in the highest probability to satisfy E_1. Once

Table I. Model Checking Results for the First Solution with a Fixed p_{ac}

N	A_a	A_{arena}	p_{ca}	p_{ac}	Pr 1 ($P(E_1)$)	Pr 2 ($1 - P(E_2)$)
10	0.38 m^2	4.91 m^2	0.08	0.05	✓ (0.75)	✗ (0.46)
20	0.78 m^2	19.63 m^2	0.06	0.04	✗ (0.39)	✗ (0.07)
50	3.14 m^2	50.26 m^2	0.06	0.04	✗ (0.12)	✗ (3.7×10^{-5})

Note: Column p_{ac} shows the best value of p_{ac}. Columns Pr 1 and Pr 2 show whether Property 1 and 2 are satisfied (✓) or not satisfied (✗) and the exact values of the probabilities of the related event of interest.

Table II. Model Checking Results for the Second Solution Where $p_{ac} = 1 - p_{min-ac} \cdot N_s$

N	A_a	A_{arena}	p_{ca}	p_{min-ac}	Pr 1 $\left(P(E_1)\right)$	Pr 2 $\left(1 - P(E_2)\right)$
10	0.38 m^2	4.91 m^2	0.08	[0.19, 0.24]	✓ (0.95)	✓ (0.92)
20	0.78 m^2	19.63 m^2	0.06	0.12	✓ (0.86)	✓ (0.87)
50	3.14 m^2	50.26 m^2	0.06	0.10	✓ (0.77)	✓ (0.71)

Note: Column p_{min-ac} shows the best value of p_{min-ac}. Column Pr 1 and Pr 2 are defined as in Table I.

p_{ac} is set, we can use model checking to verify whether the desired properties are satisfied.

Table I shows that this first attempt at tackling the aggregation case study is unsuccessful: Property is satisfied only for $N = 10$ and Property 2 is never satisfied. In general, the developed behavior obtains poor results, and the system does not cope well with increasing swarm sizes.

An analysis of the prescriptive model can help us improve the developed behavior. From the obtained results, we observe that a fixed p_{ac} does not promote the formation of a single aggregate. A better solution might be to let a robot decide whether to leave area a or area b according to the number of sensed robots in its proximity: with only few robots nearby, the probability p_{ac} to leave the aggregate is high and vice versa. We thus redefine p_{ac} as

$$p_{ac} = \begin{cases} 1 - p_{min-ac} \cdot N_s & \text{if } (1 - p_{min-ac} \cdot N_s) \in [0.01, 0.99] \\ 0.01 & \text{if } (1 - p_{min-ac} \cdot N_s) < 0.01 \\ 0.99 & \text{if } (1 - p_{min-ac} \cdot N_s) > 0.99, \end{cases}$$

where p_{min-ac} is the minimum staying probability that we want for a robot and N_s is the number of robots in the sensory range of the robot, including itself. Subsequently, using model checking, we find the best value of p_{min-ac} for the different swarm sizes. This is done by evaluating several different values of p_{min-ac} using model checking and identifying the one that gives the highest probability to satisfy the related event of interest.

An analysis of the improved behavior using model checking shows that results are significantly better both for Property 1 and Property 2, as reported in Table II.

With the current prescriptive model, we are also able to define specifications for the hardware capabilities of the robots: a ground sensor, to differentiate between the two black areas a and b and the white area c; a sensor to detect nearby robots; and wheels to move. An example of such a robot is the e-puck [Mondada et al. 2009], which can be extended with a range and bearing board that allows it to perceive the presence of neighboring robots [Gutiérrez et al. 2009].

Table III shows the symbols used in the model of the aggregation case study.

Phase Three: Simulation. In this aggregation case study, the prescriptive model captures well the microscopic behavior of the single robots; thus, it is quite straightforward to implement the robot swarm in simulation. However, several implementation details are not explicitly present in the prescriptive model, such as how the robots perform random walk. These details now have to be defined.

Table III. The Symbols Used in the Aggregation Model

Symbol	Description	Value
S_i	Number of robots in state i	$i \in \{\texttt{a,b,c}\}$
A_i	Size of area i	$i \in \{a, b, c\}$
A_{arena}	Total size of the arena	$A_{arena} \in \{4.91\text{m}^2, 19.63\text{m}^2, 50.26\text{m}^2\}$
N	Total number of robots	$N \in \{10, 20, 50\}$
N_s	Number of robots sensed by an individual robot, including itself	$N_s \in [1, N]$
p_{ij}	Probability that a robots moves from A_i to A_j	$i, j \in \{a, b, c\}$; $p_{ij} \in [0.01, 0.99]$
p_{min-ac}	Parameter of the model	$p_{min-ac} \in [0.01, 0.99]$

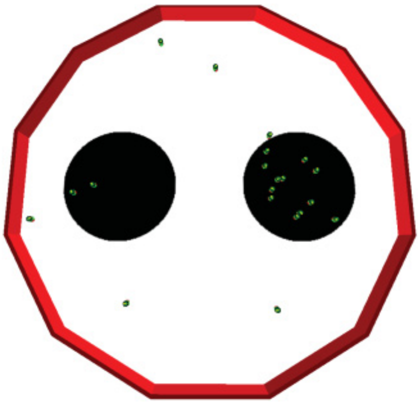


Fig. 3. A screenshot of the simulated version of the robot swarm with 20 robots.

Table IV. A Comparison between Model Checking and Simulation

N	Model Checking	Simulation
10	0.95	100/100
20	0.86	96/100
50	0.77	89/100

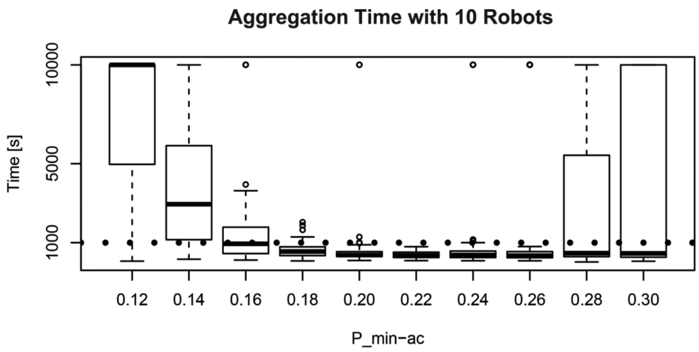
Note: The table presents the probability of E_1 (model checking) compared to the experimental results over 100 runs (simulation). Note that Property 2 is always satisfied in simulation; therefore, it is not listed in the table.

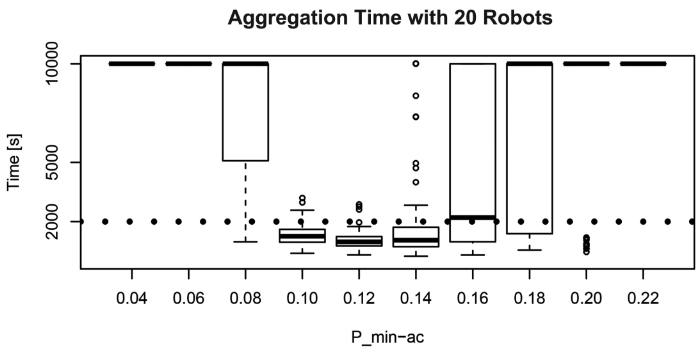
We implement the robot swarm using the ARGoS simulator [Pinciroli et al. 2012]. Figure 3 presents a screenshot of the simulated robot swarm.

We perform three different sets of experiments, one for each swarm size. To validate the prescriptive model, we measure the time necessary to form a complete aggregate on 100 runs with different values of p_{min-ac}. The robots are deployed in a random position at the beginning of each experiment. Each experiment is halted when a complete aggregate is formed or after 10,000 seconds.

As reported in Figure 4, for all the three swarm sizes, the best results are obtained with the value p_{min-ac} predicted using the prescriptive model. The results obtained in simulation are slightly better than those predicted with the model. Table IV provides a comparison.

Three sets of experiments were also performed to test Property 2. We perform 100 runs of the simulated experiments for 10,000 seconds with the three swarm sizes. In the experiments, we measure whether the robot swarm satisfies Property 2—that is,





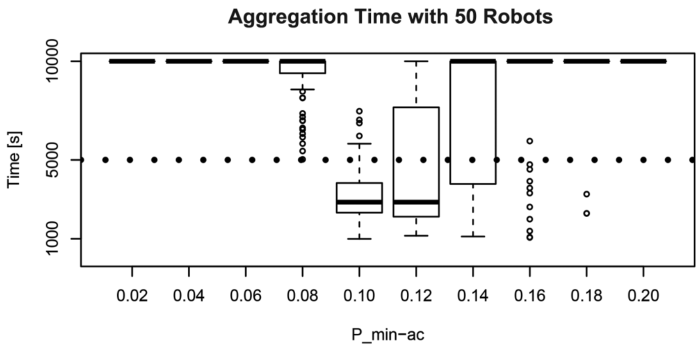


Fig. 4. The results obtained with the ARGoS simulator. The graphs show the time at which the experiment is stopped—that is, either when an aggregate is obtained (all values lower than 10,000 seconds) or when the time limit of 10,000 seconds has been exceeded. Results are presented for different p_{min-ac} over 100 runs for 10, 20, and 50 robots.

whether a complete aggregate, once formed, lasts more than 10 seconds. In all cases in which a complete aggregate was formed before 10,000 seconds, Property 2 was satisfied.

Videos of the simulated experiments are available in the supplementary material [Brambilla et al. 2014].

Phase Four: Robots. We perform 10 experiments with a swarm of 10 e-pucks in an arena identical to the simulated one. An overhead shot of an experiment can be seen in Figure 5. Figure 6 shows a comparison between the time necessary for achieving

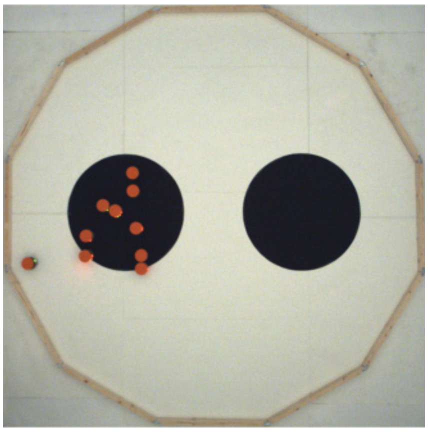


Fig. 5. An overhead shot of an experiment performed with 10 e-puck robots.

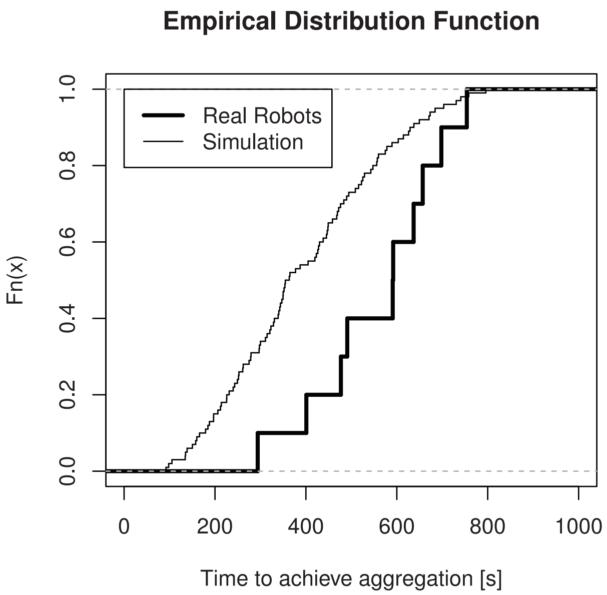


Fig. 6. A graph showing the empirical cumulative distribution $Fn(x)$ of the time necessary to achieve aggregation obtained with robots (10 runs) and in simulation (100 runs). In both cases, $N = 10$ and $p_{min-ac} = 0.22$.

aggregation with the robots and in simulation. A video of a run is available in the supplementary material [Brambilla et al. 2014].

In 10 runs out of 10, both Property 1 and Property 2 were satisfied. The results obtained with the robots are in line with those obtained in simulation.

The robot swarm that we designed is able to aggregate satisfying the required properties. For this reason, there is no need to further update the prescriptive model, and we can declare the process completed.

5.2. Foraging

In the second case study, we tackle foraging. In the simplest form of foraging, robots harvest *objects* and store them in the *nest*. The objects can be scattered in random positions or located in specific areas called *sources*. Foraging can be seen as an abstraction of more complex and realistic applications, such as search and rescue, land mine removal, waste cleaning, and automated warehouse operation.

The number of objects retrieved typically depends on the number of robots: a single robot can perform foraging alone, but additional robots could be added to increase the performance of the swarm, as robots working in parallel are able to retrieve more objects per time unit than a single robot. However, when the density of the robots in the environment increases, the performance of each single robot may be reduced due to interference [Lerman and Galstyan 2002; Pini et al. 2009].

In this case study, we assume that the robotic platform is given: the task must be tackled using e-pucks [Mondada et al. 2009]. The e-puck does not have the manipulation capabilities to interact with physical objects, so we consider an abstract version of foraging: instead of interacting with objects, e-pucks interact with TAM devices [Brutschy 2014]. The TAM is a device similar to a booth, in which a robot can enter. It has a system of light barriers to sense the presence of a robot, as well as an LED that can be used to communicate information about its internal state. In this case study, TAMs are used to simulate the manipulation of objects: an e-puck can enter in a TAM, wait a fixed amount of time, and leave to simulate harvesting or storing an object.

The arena comprises 20 TAMs: 5 TAMs on the north wall act as the nest, and each of these TAMs is a storing location; 15 TAMs on the other walls act as sources, or locations where objects can appear. At any given time, there are O objects available in the arena—that is, a new object appears as soon as one is harvested by a robot. We perform experiments in which $O \in \{2, 4, 6, 8, 10\}$. The number of available storing locations depends on the number of robots currently storing an object: it can vary from 5, when no robot is using a storing location, to 0, if all are in use.

The state of a TAM is encoded using colors: green when the TAM is available for storage; blue when the TAM has an object available for harvesting; red when the TAM is busy—that is, a TAM in which a robot is currently harvesting or storing an object; and off/black when the TAM is unavailable.

The environment is a 2m × 2m square (see Figures 7 and 12). Note that there is no globally perceivable clue in the environment that informs the robots of the position of the nest, which is different from many other foraging studies (see Brambilla et al. [2013] for a review including work on foraging).

To allow robots to see the TAMs, we use e-pucks equipped with an omnidirectional camera.[3] Using the omnidirectional camera, robots can see the LEDs of the TAMs within a range of 0.5m.

In the following discussion, we will apply the four-phase process presented in Section 4. In the foraging case study, we use the continuous-time version of the Markov chain model so that we can model the duration of some actions, such as harvesting and storing an object.

Phase One: Properties. In foraging, the main requirement is that the swarm retrieves at least a certain number of objects within a fixed time. We define the following event of interest,

$$E_3 \equiv \mathit{obj\_ret}[C \leq 600],$$

and property,

$$\mathbb{E}(E_3) \geq k. \tag{3}$$

where *obj_ret* is the number of objects retrieved and $C \leq 600$ indicates that we are interested in the cumulative value over 600 seconds. The desired property is that the

[3]See http://www.gctronic.com for more details.

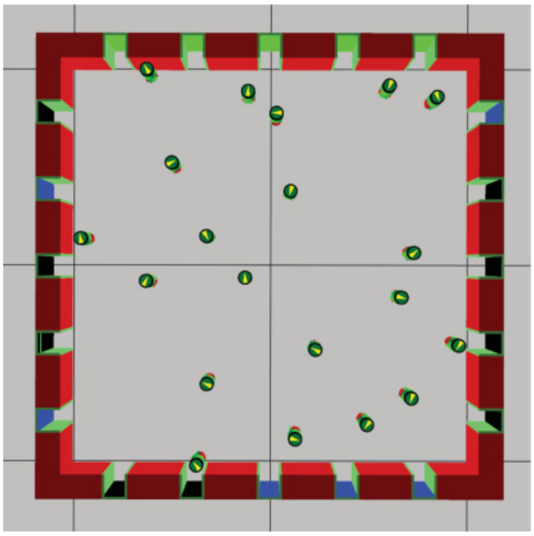


Fig. 7. A screenshot of the simulated version of foraging using 20 robots. Green-colored TAMs signal storage locations, red-colored TAMs signal that the object is busy, blue-colored TAMs signal objects to be taken, and dark TAMs are not available.

Table V. The Value of *k* Necessary to Satisfy Property 1 for Different Values of *N*; the Number of Robots Composing the Swarm; and Values of *O*, the Number of Objects Available in the Environment at Any Given Time

N	O	k	N	O	k
20	2	45	10	6	40
20	4	55	20	6	65
20	6	65	50	6	90
20	8	75	100	6	75
20	10	85			

expected number of objects retrieved in less than 600 seconds ($\mathbb{E}(E_3)$) is greater than or equal to k. $\mathbb{E}$ is the expected value.

The number k of objects that we wish to retrieve depends on N, the number of robots composing the swarm, and on O, the number of objects available in the environment at any given time (Table V). We specify k for two scenarios: one in which the number of robots is fixed while O changes, and one in which O is fixed and the number of robots changes. From Table V, it can be noticed that the number of objects that we want to retrieve does not increase linearly with the number of robots. This choice has been made because we want to test the scalability of the system, but at the same time, we expect that the effects of interferences between robots will become more and more significant as the swarm size increases, reducing the number of object retrieved [Hamann 2013].

Another requirement is on the *worst-case performance*—that is, we want to ensure that the robot swarm is able to retrieve at least a minimum number of objects in 600 seconds. Model checking allows us to formally verify this condition, as we can compute not only the expected value but also its cumulative distribution (or, conversely, the density function).

We define the following event of interest,

$$E_4 \equiv [F \leq 600 obj\_ret > 40],$$

and property,

$$P(E_4) \geq 0.90. \tag{4}$$

In natural language, Property 4 can be expressed as such: is it true with probability greater than or equal to 0.90 ($P \geq 0.90$) that at least 40 objects are retrieved ($obj\_ret > 40$) in less than or equal to 600 seconds ($F \leq 600$)? To simplify the discussion, we verify Property 4 only in the case where $N = 20$ and $O = 6$.

Phase Two: Model. To build the prescriptive model, we consider the different actions that a robot must perform. We then associate a state of the Markov chain model to each of these actions.

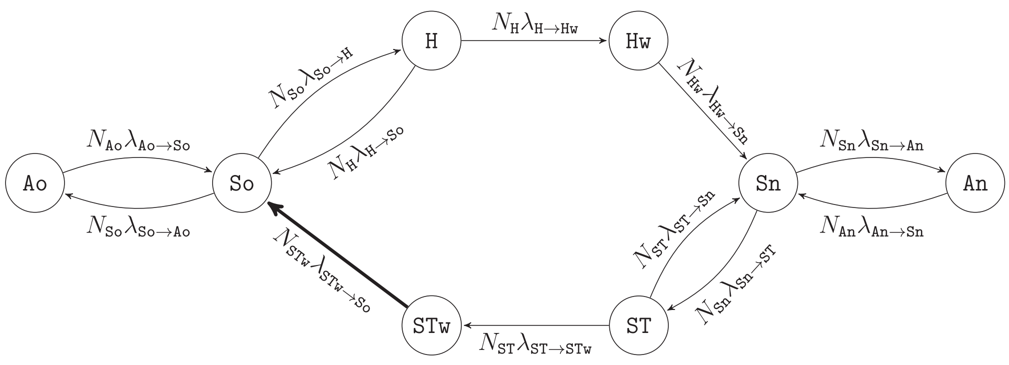
A robot searches for objects by performing a random walk in the environment (So state). Once an object is found, the robot tries to harvest it (H state); in case of multiple objects in range, the robot goes toward the closest one. If the harvest action is unsuccessful, because, for instance, another robot harvests the object, the robot goes back to searching. When the object is reached, the robot waits inside the TAM for a fixed amount of time until the object is harvested (Hw state). Once the robot has harvested an object, it proceeds to search for the nest by performing a random walk (Sn state). As soon as an available storage location is found, the robot tries to store the carried object (ST state); also in this case, the closest storage location is approached if multiple storing locations are seen. If the store action is unsuccessful, the robot searches for another storage location until the object is stored. Similar to the harvest operation, in this case the robot waits inside a TAM for a fixed amount of time until the object is stored (STw state). A successful store operation (transition from STw to So) increases the object counter (obj_ret). The robot then searches for a new object to harvest.

Robots always try to avoid collisions with obstacles and other robots. Practically, this produces two behaviors: when a robot is trying to enter a TAM (state H or state ST), it follows a vector that is the sum of a vector pointing to the desired destination and a vector pointing away from the closest obstacle. When a robot is performing a random walk without a specific destination instead (state So or state Sn), if it encounters an obstacle or another robot, it starts turning on the spot for a random number of steps and then begins again to move straight. This random number of steps follows a geometric distribution. We model these different reactions in two different ways. In the first case, the action of the robot is not significantly disturbed, as the robot performs only a slight change of trajectory towards its goal. For this reason, this first kind of obstacle avoidance affects only the time to complete the action but does not change the behavior of the robot. In the second case, instead, the robot completely changes its direction to avoid a collision, resulting in a significant change in its behavior and its chance to find objects or the nest. For this reason, the second kind of obstacle avoidance is modeled by adding two states: state Ao in case the robot is avoiding a collision when searching for an object, and state An in case the robot is avoiding a collision when searching for the nest.

Figure 8 provides a graphical representation of the prescriptive model.

We now have the structure of the behavior that the robots should follow. We need to assign values to the transition rates. We compute the transition rates considering the behavior of a single robot. For the macroscopic model, the rates are then multiplied by the current number of robots in the corresponding state, as illustrated in Figure 8. Note that here we use continuous-time Markov chains, so the transitions are defined as rates that take values in $[0, +\infty)$.

All rates involved in the definition of the model depend on the geometrical characteristics of the environment and/or on the behavior of the robots. Unfortunately, differently from the previous case study, we cannot completely define them a priori since it is



H
Hw
Ao
So
Sn
An
STw
ST
$N_{\mathtt{H}}\lambda_{\mathtt{H}\rightarrow\mathtt{Hw}}$
$N_{\mathtt{So}}\lambda_{\mathtt{So}\rightarrow\mathtt{H}}$
$N_{\mathtt{H}}\lambda_{\mathtt{H}\rightarrow\mathtt{So}}$
$N_{\mathtt{Hw}}\lambda_{\mathtt{Hw}\rightarrow\mathtt{Sn}}$
$N_{\mathtt{Ao}}\lambda_{\mathtt{Ao}\rightarrow\mathtt{So}}$
$N_{\mathtt{So}}\lambda_{\mathtt{So}\rightarrow\mathtt{Ao}}$
$N_{\mathtt{Sn}}\lambda_{\mathtt{Sn}\rightarrow\mathtt{An}}$
$N_{\mathtt{An}}\lambda_{\mathtt{An}\rightarrow\mathtt{Sn}}$
$N_{\mathtt{STw}}\lambda_{\mathtt{STw}\rightarrow\mathtt{So}}$
$N_{\mathtt{ST}}\lambda_{\mathtt{ST}\rightarrow\mathtt{Sn}}$
$N_{\mathtt{Sn}}\lambda_{\mathtt{Sn}\rightarrow\mathtt{ST}}$
$N_{\mathtt{ST}}\lambda_{\mathtt{ST}\rightarrow\mathtt{STw}}$

Fig. 8. The continuous-time Markov chain used to model foraging. `H` is the *harvest* state; `Hw` is the *wait to harvest* state; *So* is the *search object* state; `Ao` is the *avoid (while searching for an object)* state; `ST` is the *store* state; `STw` is the *wait to store* state; `Sn` is the *Search nest* state; and `An` is the *Avoid (while searching for the nest)* state. Transitions are labeled with their respective rates multiplied by the number of robots currently in that state: $\lambda_{\mathtt{i}\rightarrow\mathtt{j}}$ is the rate at which an individual robot moves from state `i` to state `j`; and $\mathtt{N}_{\mathtt{i}}$ is the number of robots currently in state `i`. To compute the expected number of objects retrieved, we keep track of the number of times that the transition from `STw` to `So` happens.

impossible to identify the correct value of the parameters involved in their definition without experimental data. In the following, we make some working hypotheses about the system that can be subject to refinements or changes in the subsequent phases, should they prove not to be sufficiently accurate or correct. Since we do not have experimental data, the model that we are creating is largely arbitrary. Other valid choices could have been made. In particular, parameters will be fitted once experimental data are available—that is, in phase three. Note that the goal of this phase is not to create a model that is as precise as possible, but one that can be used to develop and later improve the desired robot swarm.

In the following discussion, we present how each rate is defined. We define $\lambda_{\mathtt{So}\rightarrow\mathtt{H}}$, the rate at which a robot finds an object, as proportional to the density of available objects in the environment:

$$\lambda_{\mathtt{So}\rightarrow\mathtt{H}} = \alpha\frac{O}{A_{arena}},$$

where O is the number of objects available at any given time, A_{arena} is the area of the environment, and α is a parameter.

We define $\lambda_{\mathtt{So}\rightarrow\mathtt{Ao}}$, the rate at which a robot searching for an object finds another robot and then performs obstacle avoidance, as proportional to the density of robots in the environment:

$$\lambda_{\mathtt{So}\rightarrow\mathtt{Ao}} = \beta\frac{N}{A_{arena}},$$

where N is the total number of robots in the environment and β is a parameter.

Rates $\lambda_{\mathtt{Sn}\rightarrow\mathtt{ST}}$ and $\lambda_{\mathtt{Sn}\rightarrow\mathtt{An}}$ are defined similarly, considering the number of storage locations instead of the number of objects:

$$\lambda_{\mathtt{Sn}\rightarrow\mathtt{ST}} = \gamma\frac{D}{A_{arena}},$$

$$\lambda_{\mathtt{Sn}\rightarrow\mathtt{An}} = \delta\frac{N}{A_{arena}},$$

where D is the number of storage locations and γ and δ are two parameters.

Rates $\lambda_{\mathtt{Ao}\rightarrow\mathtt{So}}$ and $\lambda_{\mathtt{An}\rightarrow\mathtt{Sn}}$ depend on the time necessary for a robot to perform obstacle avoidance. As said previously, if a robot encounters an obstacle or another robot while searching for objects or for the nest, it performs obstacle avoidance by turning on the spot for a random number of steps distributed geometrically and then starts again searching for objects or the nest. The rate at which a robot moves from obstacle avoidance back to search is thus

$$\lambda_{\mathtt{Ao}\rightarrow\mathtt{So}} = \lambda_{\mathtt{An}\rightarrow\mathtt{Sn}} = p_{oa},$$

where p_{oa} is the parameter of the geometric distribution.

We define $\lambda_{\mathtt{H}\rightarrow\mathtt{Hw}}$ and $\lambda_{\mathtt{ST}\rightarrow\mathtt{STw}}$, the rates at which a robot going toward an object-TAM or a storage-TAM manages to enter it, as the reciprocal of the time necessary to get in the TAM, counted from the instant in which the robot sees it:

$$\lambda_{\mathtt{H}\rightarrow\mathtt{Hw}} = \lambda_{\mathtt{ST}\rightarrow\mathtt{STw}} = \left(\frac{r}{s}\right)^{-1},$$

where r is the range at which a robot sees a TAM and s is the forward speed of a robot.

A robot trying to enter a TAM is not always successful: other robots may "steal" its object by occupying the TAM before it can do it. This means that not all robots going toward a TAM enter it. Some are interrupted by other robots and thus are forced to search for another available TAM. This is modeled by the transitions $\mathtt{H} \rightarrow \mathtt{So}$ and $\mathtt{ST} \rightarrow \mathtt{Sn}$. We define $\lambda_{\mathtt{H}\rightarrow\mathtt{So}}$ and $\lambda_{\mathtt{ST}\rightarrow\mathtt{Sn}}$, the related rates, as proportional to the density of robots in the environment:

$$\lambda_{\mathtt{H}\rightarrow\mathtt{So}} = \epsilon\frac{N}{A_{arena}},$$

$$\lambda_{\mathtt{ST}\rightarrow\mathtt{Sn}} = \eta\frac{N}{A_{arena}},$$

where ϵ and η are parameters. We expect ϵ and η to have different values, as storage locations are all next to each other, generating more interference, whereas objects to harvest generally are evenly distributed along the walls of the environment.

The last rates that we need to define are $\lambda_{\mathtt{Hw}\rightarrow\mathtt{Sn}}$ and $\lambda_{\mathtt{STw}\rightarrow\mathtt{So}}$, the rates at which robots in a TAM complete their operations and exit. These rates are the reciprocal of the time spent by a robot in a TAM:

$$\lambda_{\mathtt{Hw}\rightarrow\mathtt{Sn}} = \lambda_{\mathtt{STw}\rightarrow\mathtt{So}} = (t_{TAM})^{-1},$$

where t_{TAM} is the time spent by a robot in a TAM.

Since we do not have empirical data to estimate the parameters, at this point we cannot use model checking to compute the expected number of objects retrieved, or whether the desired properties are satisfied. However, even without empirical data, we can use the model to improve the behavior of the robots. For example, by analyzing the model, we can observe that increasing the rate at which the robots find objects or storage locations—that is, increasing $\lambda_{\mathtt{So}\rightarrow\mathtt{H}}$ and $\lambda_{\mathtt{Sn}\rightarrow\mathtt{ST}}$—results in an increase of objects retrieved.

To increase these rates, we cannot modify the number of objects available at any given time or the number of storage locations, as they are given. We could act on the parameters, but it is not clear how to change the behavior of the robots to increase these parameters. Even though we cannot change the dimensions of the environment, we can change the size of the area effectively covered by the robots. In other terms, we can change the behavior of the robots so that they do not cover the whole environment when searching for objects or storage locations. In particular, we could let robots avoid places where they know they will not find anything useful.

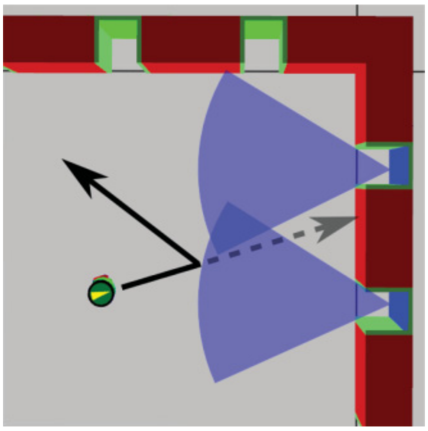


Fig. 9. The modification of the behavior used to reduce interference and reduce the area searched by a robot. A robot (depicted as a green circle with a yellow arrow on top) carrying an object performs obstacle avoidance (full arrow) as soon as it sees another object, instead of reaching to the wall (dotted arrow). The light blue circular sections represent the areas in which a robot sees an object. The same applies to robot searching for the nest, even though this is not displayed in the figure.

In the behavior defined earlier, robots searching for objects and for the nest go straight until they find an obstacle such as a wall or another robot. This means that robots carrying an object while searching for the nest may go close to other available objects, interfering with robots not carrying objects. Similarly, robots searching for objects often go close to the storing locations, interfering with the other robots. A possible solution is that robots searching for objects avoid storing locations as soon as they see them, and similarly, robots searching for the nest avoid objects as soon as they see them. This improved behavior is depicted in Figure 9.

This improvement in the behavior can be modeled by decreasing the value of A_{arena}. The rates are updated in the following way:

$$\lambda_{\mathtt{So}\to\mathtt{H}} = \alpha \frac{O}{A_o},$$

where A_o is the area explored by the robots searching for objects and avoiding storage locations, with $A_o < A_{arena}$;

$$\lambda_{\mathtt{Sn}\to\mathtt{ST}} = \gamma \frac{D}{A_n},$$

where A_n is the area explored by the robots searching for storage locations and avoiding objects, with $A_n < A_{arena}$; and all other rates are left unchanged, since, on average, the density of the robots involved in their definition does not change. For example, consider $\lambda_{\mathtt{So}\to\mathtt{Ao}}$. This rate depends on the number of robots N in the total area A_{arena}. The area involved in the definition of this rate is reduced and becomes A_o, as explained previously. However, the number of robots considered changes as well: the number of robots considered in this new version is the number of robots currently searching for objects in the area A_o, which is less than N. In other words, even though the area considered is reduced, on average the density of robots does not change.

More complex improvements, such as task allocation mechanisms, could be implemented to further increase the performance of the system, should the obtained performance not be sufficient. However, for the sake of brevity and clarity, we limit our design process to the simple improvement presented earlier.

Table VI. The Numerical Parameters Used in the Foraging Prescriptive Model

Symbol	Description	Value
O	Number of available objects at any given time	$O \in \{2, 4, 6, 8, 10\}$
N	Number of robots composing the swarm	$N \in \{10, 20, 50, 100\}$
D	Number of storage locations	5
A_{arena}	Area of the environment	4 m^2
α	Coefficient of $\lambda_{\mathtt{So}\rightarrow\mathtt{H}}$	4×10^{-2}
β	Coefficient of $\lambda_{\mathtt{So}\rightarrow\mathtt{Ao}}$	1×10^{-2}
γ	Coefficient of $\lambda_{\mathtt{Sn}\rightarrow\mathtt{ST}}$	3.9×10^{-2}
δ	Coefficient of $\lambda_{\mathtt{Sn}\rightarrow\mathtt{An}}$	9×10^{-3}
p_{oa}	Coefficient of $\lambda_{\mathtt{Ao}\rightarrow\mathtt{So}}$ and $\lambda_{\mathtt{An}\rightarrow\mathtt{Sn}}$	0.20
r	Coefficient of $\lambda_{\mathtt{H}\rightarrow\mathtt{Hw}}$ and $\lambda_{\mathtt{ST}\rightarrow\mathtt{STw}}$	0.5 m
s	Coefficient of $\lambda_{\mathtt{H}\rightarrow\mathtt{Hw}}$ and $\lambda_{\mathtt{ST}\rightarrow\mathtt{STw}}$	0.1 m s^{-1}
ϵ	Coefficient of $\lambda_{\mathtt{H}\rightarrow\mathtt{So}}$	6.5×10^{-2}
η	Coefficient of $\lambda_{\mathtt{ST}\rightarrow\mathtt{Sn}}$	1.01×10^{-1}
$t_{\mathtt{TAM}}$	Time spent in a TAM to perform an action	3 s
A_o	Coefficient of $\lambda_{\mathtt{So}\rightarrow\mathtt{H}}$	3 m^2
A_n	Coefficient of $\lambda_{\mathtt{Sn}\rightarrow\mathtt{ST}}$	3.7 m^2

Note: The values are obtained from the experimental data obtained in phase three.

Phase Three: Simulation. In this phase, we implement the foraging robot swarm using the ARGoS simulator [Pinciroli et al. 2012].

The prescriptive model developed in phase two provides us with a detailed blueprint to implement the robot swarm: the behavior of the individual robot can be implemented using a finite state machine that resembles the Markov chain defined in phase two. Nonetheless, some implementation details, such as how robots stop inside a TAM, have been ignored in the prescriptive model to focus on the more important details at design time and have to be programmed explicitly at this moment.

We measured the number of objects retrieved over 600 seconds on 100 runs. We first performed experiments using the initial behavior and then using the improved one, as explained in the previous phase.

Using the data obtained from all simulation experiments, we estimate the values of the parameters of the model. Table VI shows the parameters used for model checking derived from the experimental data. We can now compute the expected number of objects retrieved in 600 seconds using model checking on the developed model and compare these values with the results obtained from the simulated experiments. Figure 10 shows the results obtained in simulation together with the expected results predicted by the prescriptive model. These results have been obtained using 20 robots with different values of O, which is the number of objects available at any time. Figure 11 shows the results obtained with $O = 6$ and different numbers of robots. Table VII shows whether Property 3 is satisfied.

From Figure 10, it is possible to observe that indeed the improvement introduced in phase two significantly increases the number of objects retrieved. The improved behavior is always significantly better than its counterpart—Wilcoxon test with $p < 0.01$.

The correspondence between the results obtained from the prescriptive model and those obtained from the simulations is quite good, although not perfect. In particular, it seems that the model is not capable of fully capturing the effects of physical interference between the robots. This leads to overestimating the results with small swarms and underestimating them with large swarms. However, for our goals and purposes, the model qualitatively captures the behavior of the robot swarm, and thus it is not necessary to further refine it.

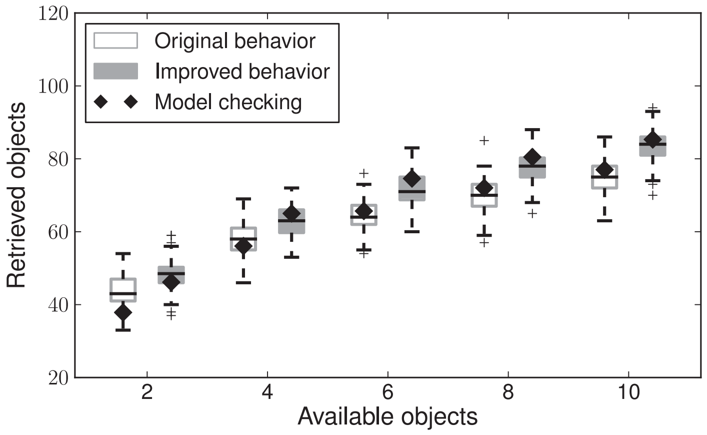


Fig. 10. A comparison between the results obtained using 20 robots with the original behavior and with the improved one for different values of O, which is the number of objects available at any time. Box plots show results obtained over 100 experimental runs using the ARGoS simulator, whereas diamonds show the expected results obtained with PRISM.

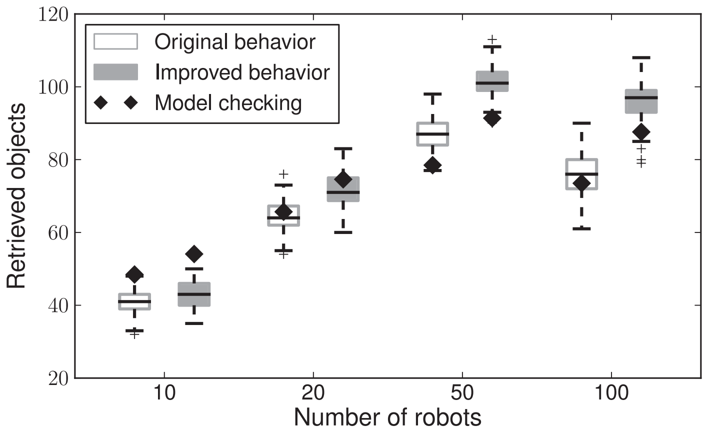


Fig. 11. A comparison between the results obtained using $O = 6$ with the original behavior and with the improved one for different swarm sizes. Box plots show results obtained over 100 experimental runs using the ARGoS simulator, whereas diamonds show the expected results obtained with PRISM.

We also verify Property 4 using model checking and compare it with the results obtained from the simulations. Model checking tells us that Property 4 is not satisfied in the prescriptive model of the original behavior with 20 robots and $O = 2$: the probability to retrieve more than 40 objects is 0.86. This matches the experimental results, where 15 runs out of 100 resulted in fewer than 40 objects retrieved. Instead, with the improved behavior, Property 4 is satisfied: the probability to retrieve more than 40 objects is 0.99. This matches the experimental results, where only 2 runs out of 100 resulted in less than 40 objects retrieved.

All experimental data can be found in the supplementary material [Brambilla et al. 2014].

Phase Four: Robots. We performed 10 experiments with a swarm of 20 e-pucks in an arena identical to the simulated one. An overhead shot of an experiment can be seen in Figure 12. Videos of the performed experiments can be found in the supplementary

Table VII. A Table Presenting Whether the Developed Behaviors Are Able to Satisfy Property 3

N	O	k	Original (MC)	Improved (MC)	Original (Sim)	Improved (Sim)
20	2	45	✗ (37)	✓ (46)	✗ (43)	✓ (48)
20	4	55	✓ (56)	✓ (65)	✓ (58)	✓ (63)
20	6	65	✓ (65)	✓ (74)	✗ (64)	✓ (71)
20	8	75	✗ (72)	✓ (80)	✗ (70)	✓ (78)
20	10	85	✗ (77)	✓ (85)	✗ (75)	✓ (85)
10	6	40	✓ (48)	✓ (54)	✓ (41)	✓ (43)
20	6	65	✓ (65)	✓ (74)	✗ (64)	✓ (71)
50	6	90	✗ (78)	✓ (91)	✗ (87)	✓ (101)
100	6	75	✗ (73)	✓ (87)	✓ (76)	✓ (97)

Note: k is the threshold on the expected number of objects retrieved. Property 3 is satisfied if the values obtained are greater than or equal to k. The results are presented both for the original and the improved behavior, showing whether Property 3 is satisfied (✓) or not satisfied (✗) and the value of the related event of interest. Columns marked with (MC) refer to the results obtained using PRISM, whereas columns marked with (Sim) refer to the results obtained using ARGoS.

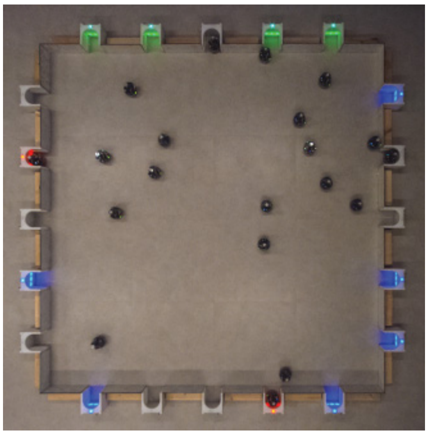


Fig. 12. An overhead shot of an experiment performed with 20 e-puck robots and $O = 6$. Green colored TAMs signal storage locations, red-colored TAMs signal that the object is busy, blue colored TAMs signal objects to be taken, dark TAMs are not available.

material [Brambilla et al. 2014]. Figure 13 shows that the results obtained with real and simulated robots are quite similar. Property 4 is satisfied. There is no need to update the model, and thus we can declare the process completed.

5.3. Discussion

The two case studies presented in this article show that using property-driven design, we were able to develop two robot swarms that tackle successfully aggregation and foraging: all required properties are satisfied.

As shown, with property-driven design, it is possible to analyze and develop behaviors characterized both by numerical and nonnumerical parameters. For example, in the aggregation case study, we analyzed the effects of changing the probability to leave a black area, whereas in the foraging case, we analyzed the effects of changing the exploration behavior.

One of the main advantages of property-driven design is that it allows the developer to focus on the design of the system rather than on its implementation: by focusing the efforts on the abstract model of the system, the developer can concentrate on creation of important aspects of the system. This is important because “whereas a simulation

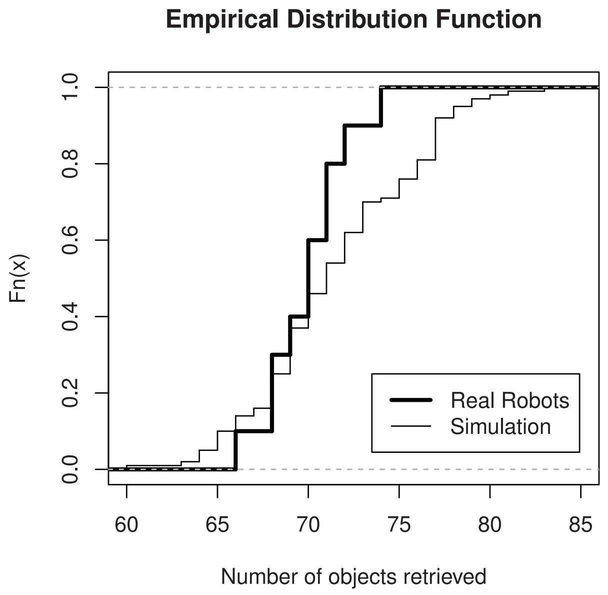


Fig. 13. A graph showing the empirical cumulative distribution $Fn(x)$ of the number of object retrieved using robots (10 runs) and in simulation (100 runs). In both cases, $N = 20$ and $O = 6$.

should include as much detail as possible, a good model should include as little as possible" [Smith 1978]. As an example, in the foraging case study, we leveraged the prescriptive model developed in phase two of property-driven design to identify possible improvements of the partially realized system: we were able to identify the important aspects of the system being designed, which allowed us to avoid wasting time on improving other nonrelevant aspects.

Another advantage of property-driven design is the reduced risk of developing a robot swarm that does not satisfy the requirements. Up to now, there was no clear way to specify the requirements of a robot swarm. In property-driven design, requirements are specified in a formal way at the beginning of the development process in terms of desired properties. Moreover, thanks to model checking, it is possible to evaluate whether the robot swarm fulfills such properties at each step of the design and development process. This advantage of property-driven design has been highlighted in both case studies.

Finally, property-driven design also addresses the problem of low reusability of behaviors in swarm robotics. Usually, behaviors for robot swarms are developed in a disposable way. This is because there is no clear distinction between the design and the implementation. Thus, if a different hardware platform is available, or a slightly different task is tackled, it is necessary to start from scratch. With property-driven design, however, the prescriptive model developed in phase two can be partially or completely reused: (1) the model is hardware independent, so it can be adapted to the available robots, or even guide the process of deciding the best robot to use, and (2) the model can be extended to deal with new properties and verify if they are satisfied even without testing the system in simulation or with robots. The reusability of the prescriptive model reduces the risk that designers "reinvent the wheel" each time they develop a robot swarm. For example, the model of foraging developed in the presented case study could be easily adapted to robots with more sophisticated manipulation capabilities. In the future, it is also possible to imagine a set of publicly available models for swarm robotics applications that can be reused and modified by other developers.

The development process of the case studies presented in this article highlights some issues with property-driven design.

The main issue is that ultimately the step from the prescriptive model to its implementation remains in the hands of the developer: property-driven design does not offer an automatic way to identify the behavior of the individual necessary to obtain a desired collective behavior. Nonetheless, the prescriptive model can be used as a blueprint for the implementation process, providing the developer with a valuable tool to obtain robot swarms with provable properties.

Another issue is the strong reliance on modeling. This strong reliance on modeling might limit the applicability of property-driven design to those systems that can be described using mathematical models and to properties that can be specified using probabilistic temporal logics. Additionally, modeling robot swarms is a difficult task on its own: robot-to-robot interactions, spatial and temporal features, and interference are difficult to completely describe using models. Luckily, modeling robot swarms has been the focus of a large number of studies (see two reviews of the literature by Brambilla et al. [2013] and Lerman et al. [2005]) that provide a solid theoretical foundation on which to base property-driven design.

6. CONCLUSIONS

Property-driven design is a top-down design method based on prescriptive modeling and model checking: the desired robot swarm is first described using a set of properties. Subsequently, a prescriptive model of the robot swarm is created, and the prescriptive model is used as a blueprint for the implementation of the robot swarm first in simulation and then with robots.

Property-driven design is conceived to be part of swarm engineering: the systematic application of scientific and technical knowledge to specify requirements, design, realize, verify, validate, operate, and maintain a swarm intelligence system. Up to now, the design and development of a robot swarm has been performed using a code-and-fix approach based completely on the ingenuity and experience of the developer who has little scientific or technical support in his activity. Property-driven design aims at providing such scientific and technical support, with many advantages compared to the traditional unstructured approach.

In this article, we demonstrated, by tackling two different case studies, that property-driven design is an effective method for the design and development of robot swarms. In the future, we plan to apply property-driven design to different and more complex tasks, possibly using different modeling approaches. In addition, we aim to integrate this design method with automatic design approaches, such as evolutionary robotics or automatic modular design [Francesca et al. 2014a, 2014b], to automatically obtain robot swarms that satisfy some desired properties. This could provide a solution to the open problem of deriving the individual behaviors from a swarm-level model. Additionally, to further promote the use of model checking in swarm robotics, we plan to provide guidelines on how to apply model checking to design and analyze robot swarms. In particular, we plan to provide guidelines on how to choose the best suited modeling and specification languages for different collective behaviors. Finally, we plan to study whether model-driven design [Miller and Mukerji 2003], an approach to software engineering where software is designed through a series of transformations from platform-independent models to executable platform-specific models, can be used together with property-driven design to further ease the design and development of robot swarms.

APPENDIX

Property-driven design is based on prescriptive modeling and model checking. Model checking requires two components: a model of the system to check and a set of properties that the system must satisfy. Among the several possible languages and tools, we chose Markov chains and probabilistic temporal logics. In this appendix, we give a short introduction to Markov chains, probabilistic temporal logics, and model checking.

I. The Model: Markov Chains

A common way to model swarm robotics systems is through the use of Markov chains [Lerman et al. 2005]. Markov chains are used to model the behavior of the robots: *states* might represent an action that a robot performs, such as `random walk` or `grasp object`, or they might represent an area in which a robot is located, such as `in the nest`. Transitions link two states and are activated through *transition conditions* like `obstacle seen` and `object grasped`.

Markov chains can be used to model a swarm robotics system in two ways: as a *microscopic Markov model* (i.e., a model that considers each individual robot) and as a *macroscopic Markov model* (i.e., a model that considers the swarm as a whole).

A microscopic Markov model describes the behavior of the individual robots and of their interactions. In a microscopic Markov model, the collective behavior of the swarm is the *and*-composition of the individual Markov chains.

A macroscopic Markov model describes the swarm as a whole, without considering the individual robots composing it. In general, as explained by Lerman et al. [2005], a macroscopic Markov model is composed of an augmented Markov chain: the Markov chain describing the behavior of a generic individual of the swarm is augmented by associating a counter to each state. Counters are used to keep track of the number of robots that are in the associated state in any given moment. This is different from rate equations, in which only the ratio is tracked. Examples of macroscopic Markov models can be found in Section 5, particularly in Figures 2 and 8.

When compared to macroscopic Markov models, microscopic Markov models give a finer description of the robots and their interactions. However, in a microscopic Markov model, the number of states grows exponentially with the number of robots, making microscopic Markov models computationally intractable in most cases. Since microscopic Markov models are difficult to analyze, the great majority of models in swarm robotics are macroscopic [Brambilla et al. 2013; Lerman et al. 2005].

Time in Markov chains can be modeled in two different ways: discrete and continuous. In discrete-time Markov chains (DTMCs), time assumes only values in $\mathbb{Z}^+$, whereas in continuous-time Markov chains (CTMCs), time can assume any value in $\mathbb{R}^+$. Practically, one of the main differences between DTMC and CTMC is in what transition parameters represent. In DTMC, transition parameters represent the probability p of moving from one state to another; for this reason, transition parameters for DTMC must be in the interval $[0, 1]$. In CTMC, instead, transitions parameters represent the rate λ at which the event of moving from one state to another happens—that is, the time taken to move from one state to another follows an exponential distribution of parameter $\lambda \in (0, \inf)$.

The choice of how to model time depends on the system to describe: in case time is not the main aspect and can be easily discretized, DTMCs are more convenient; on the contrary, when it is important to keep precisely track of time, CTMCs should be preferred. Note that generally it is possible to model the same system using a DTMC or a CTMC without losing expressive power [Serfozo 1979].

II. The Properties: Probabilistic Temporal Logics

The most common way to formally express properties for model checking is through the use of logic predicates. Among the many formal logic systems, we consider probabilistic computation tree logic (PCTL).

PCTL [Hansson and Jonsson 1994] is based on the concept of computation tree, a potentially infinite rooted tree in which the root is the initial state of a corresponding

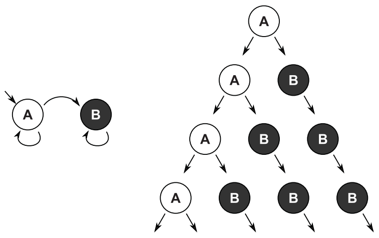


Fig. 14. A simple Markov chain (on the left) and part of its computation tree (on the right).

Markov chain, and each node is a possible state of the system. Edges link a state with its next possible states. Each path on the tree represents a possible execution of the system. Since a sequence of nodes represents the time evolution of a system, the transition from one node to a following one is usually called a *timestep*. In DTMCs, this timestep is fixed, whereas in CTMCs, this timestep is exponentially distributed with a parameter depending on the current state. An example of a simple Markov chain and its computation tree is displayed in Figure 14.

A computation tree can be used to express temporal properties, such as *eventually the system will reach state X,* or *if the system starts from state α, then it will never reach state β*. Such properties can be expressed using computation tree logic (CTL).

PCTL extends CTL by introducing probabilities. It is thus possible to express properties such as *α will eventually become true with probability 0.45,* or *there is a 0.7 probability that α will hold true for 10 seconds*. A formal introduction to the syntax of PCTL can be found in the work of Ciesinski and Größer [2004].

Understanding the details of PCTL is not essential to understand the general principles of the work presented in this article. Nonetheless, we think that a simple formal introduction (based on Hansson and Jonsson [1994] and Ciesinski and Größer [2004]) can be useful to those interested in using PCTL for model checking in swarm robotics. The syntax of PCTL is composed of *state formulas*, generally identified by the lowercase Greek letter ϕ, and *path formulas*, generally identified by the uppercase Greek letter Φ. Path formulas are infinite sequences of states ordered over time (i.e., $\Phi_k = \phi_0 \rightarrow \phi_1 \rightarrow \cdots \rightarrow \phi_i \rightarrow \cdots$). The standard logic constants and operators are available (e.g., $\top$, $\wedge$, $\Rightarrow$, $\neg$) together with operators for probabilistic and temporal properties. Formulas like $\forall\phi$ and $\exists\phi$ are replaced by $P(\phi) \bowtie p$, where P indicates the probability operator $p \in [0, 1] \subset \mathbb{R}$ is a probability limit and $\bowtie$ is a placeholder for $\{>, \geq, \leq, <\}$. Consider, for example, $P(\phi) \geq p$, which asserts that the probability that ϕ holds true is at least p. Two temporal operators are available in PCTL: X, called *Next*, and $\mathcal{U}^{\leq k}$, called *bounded until*. $X\phi$ denotes that ϕ holds in the next state, whereas $\phi_1\mathcal{U}^{\leq k}\phi_2$ denotes that ϕ_1 has to hold from now until, within at most k time units, ϕ_2 becomes true. This implies that ϕ_2 will eventually become true in the future. Other useful temporal operators can be derived from these two. Of particular interest are F, eventually, and G, always.

Probabilistic and temporal operators make PCTL a flexible and powerful logic that can be used to express many interesting properties of particular interest to swarm robotics systems.

III. Model Checking: Complete and Statistical Model Checking

Having defined a model and a set of properties, we now have all of the elements to perform model checking.

Model checking can be used in safe-critical applications in which simulations and experiments might not be enough to guarantee the correctness of a system. In fact, simulations and experiments can only test a subset of all possible execution scenarios of a system. Model checking, instead, formally verifies that a property holds true for all possible executions of a system.

Model checking has several advantages compared to more traditional ways of analyzing models of swarm robotics systems, such as fluid flow analysis [Zarzhitsky et al. 2005].

Not only does model checking allow the user to verify that a model satisfies a specific probabilistic property, such as $P(\phi) \geq 0.75?\,\texttt{TRUE}$), but it also allows the user to obtain quantitative results—that is, to compute with which probability the model satisfies it ($P(\phi)?\,0.86$). This characteristic is useful to find the best parameters of a model that maximize the probability of satisfying a specific property. Moreover, it is possible to augment a Markov chain using *rewards*, real valued quantities that can be assigned to states or transitions. Using model checking, it is possible to compute not only the steady-state value of these rewards but also their probability distribution. This would be impossible with analysis based on rate equations, as they only provide the expected value of the observed variable. Model checking can also be used to produce counter examples: traces of execution of a system in which a property is not satisfied. Additionally, the use of PCTL allows the developer to express properties that are difficult or even impossible to express using algebraic mathematics.

A limit of model checking is that generally it is computationally unfeasible to analyze models composed of a high number of states. State-of-the-art model checkers cannot handle models larger than 10^{10} states [Kwiatkowska et al. 2004]. A way to overcome this problem is *statistical model checking*. Statistical model checking, also known as approximate model checking, is a novel approach to model checking [Nimal 2010]. Compared to traditional model checking, statistical model checking does not completely explore the state space of a model. Instead, it samples a large but limited number of executions of the model and uses statistical estimators to compute the result.

Using statistical model checking, it is thus possible to perform model checking on very large models, such as microscopic models of swarm robotics systems. In Massink et al. [2013], we have applied statistical model checking to a model of a swarm robotics system and have showed that the obtained results were consistent with those obtained using other approaches, such as physics-based simulation, Monte Carlo simulation, and ordinary differential equations.

REFERENCES

C. Baier and J.-P. Katoen. 2008. *Principles of Model Checking*. MIT Press, Cambridge, MA.

W. Banzhaf and N. Pillay. 2007. Why complex systems engineering needs biological development. *Complexity* 13, 2, 12–21.

K. Beck. 2003. *Test-Driven Development: By Example*. Addison-Wesley, Boston, MA.

S. Berman, A. Halasz, M. A. Hsieh, and V. Kumar. 2009. Optimized stochastic policies for task allocation in swarms of robots. *IEEE Transactions on Robotics* 25, 4, 927–937.

S. Berman, V. Kumar, and R. Nagpal. 2011. Design of control policies for spatially inhomogeneous robot swarms with application to commercial pollination. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'11)*. 378–385.

R. H. Bordini. 2009. *Multi-Agent Programming: Languages, Tools and Applications*, Vol. 2. Springer, New York. NY.

M. Brambilla, M. Dorigo, and M. Birattari. 2014. Property-Driven Design for Robot Swarms A Design Method Based on Prescriptive Modeling and Model Checking. Supplementary material available at http://iridia.ulb.ac.be/supp/IridiaSupp2014-003/index.html.

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

M. Brambilla, C. Pinciroli, M. Birattari, and M. Dorigo. 2012. Property-driven design for swarm robotics. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'12)*. 139–146.

A. Brutschy. 2014. *The TAM: A Device for Task Abstraction in Swarm Robotics Research*. Technical Report TR/IRIDIA/2010-015.005. IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

F. Ciesinski and M. Größer. 2004. On probabilistic computation tree logic. In *Validation of Stochastic Systems*. Lecture Notes in Computer Science, Vol. 2925. Springer, 333–355.

C. Dixon, A. Winfield, and M. Fisher. 2012. Towards temporal verification of swarm robotic systems. *Robotics and Autonomous Systems* 60, 11, 1429–1441.

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

G. Francesca, M. Brambilla, A. Brutschy, L. Garattoni, R. Miletitch, G. Podevijn, A. Reina, T. Soleymani, M. Salvaro, C. Pinciroli, V. Trianni, and M. Birattari. 2014a. An experiment in automatic design of robot swarms. In *Swarm Intelligence*. Lecture Notes in Computer Science, Vol. 8667. Springer, 25–37.

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

D. Goldberg and M. J. Matarić. 2001. Design and evaluation of robust behavior-based controllers for distributed multi-robot collection tasks. In *Robot Teams: From Diversity to Polymorphism*. A. K. Peters, Natick, MA, 315–344.

A. Gutiérrez, A. Campo, M. Dorigo, J. Donate, F. Monasterio-Huelin, and L. Magdalena. 2009. Open e-puck range & bearing miniaturized board for local communication in swarm robotics. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'09)*. 3111–3116.

H. Hamann. 2013. Towards swarm calculus: Urn models of collective decisions and universal properties of swarm performance. *Swarm Intelligence* 7, 2–3, 145–172.

H. Hamann and H. Wörn. 2008. A framework of space–time continuous models for algorithm design in swarm robotics. *Swarm Intelligence* 2, 2, 209–239.

H. Hansson and B. Jonsson. 1994. A logic for reasoning about time and reliability. *Formal Aspects of Computing* 6, 5, 512–535.

R. Jeanson, C. Rivault, J.-L. Deneubourg, S. Blanco, R. Fournier, C. Jost, and G. Theraulaz. 2005. Self-organized aggregation in cockroaches. *Animal Behaviour* 69, 1, 169–180.

S. Kazadi, J. R. Lee, and J. Lee. 2009. Model independence in swarm robotics. *International Journal of Intelligent Computing and Cybernetics: Special Issue on Swarm Robotics* 2, 4, 672–694.

S. Konur, C. Dixon, and M. Fisher. 2012. Analysing robot swarm behaviour via probabilistic model checking. *Robotics and Autonomous Systems* 60, 2, 199–213.

M. Kwiatkowska, G. Norman, and D. Parker. 2004. Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer* 6, 2, 128–142.

K. Lerman and A. Galstyan. 2002. Mathematical model of foraging in a group of robots: Effect of interference. *Autonomous Robots* 13, 2, 127–141.

K. Lerman, A. Martinoli, and A. Galstyan. 2005. A review of probabilistic macroscopic models for swarm robotic systems. In *Swarm Robotics*. Lecture Notes in Computer Science, Vol. 3342. Springer, 143–152.

M. Massink, M. Brambilla, D. Latella, M. Dorigo, and M. Birattari. 2013. On the use of Bio-PEPA for modelling and analysing collective behaviours in swarm robotics. *Swarm Intelligence* 7, 2–3, 201–228.

J. Miller and J. Mukerji. 2003. MDA Guide V1.0.1. Retrieved November 2, 2014, from http://www.omg.org/cgi-bin/doc?omg/03-06-01.

F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli. 2009. The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, Vol. 1. 59–65.

V. Nimal. 2010. *Statistical Approaches for Probabilistic Model Checking*. MSc Mini-Project Dissertation. Oxford University Computing Laboratory, Oxford, UK.

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

G. Pini, A. Brutschy, M. Birattari, and M. Dorigo. 2009. Interference reduction through task partitioning in a robotic swarm. In *Proceedings of the 6th International Conference on Informatics in Control, Automation, and Robotics (ICINCO'09)*. 52–59.

E. Şahin. 2005. Swarm robotics: From sources of inspiration to domains of application. In *Swarm Robotics*. Lecture Notes in Computer Science, Vol. 3342. Springer, 10–20.

R. F. Serfozo. 1979. An equivalence between continuous and discrete time Markov decision processes. *Operations Research* 27, 3, 616–620.

J. M. Smith. 1978. *Models in Ecology*. Cambridge University Press, Cambridge, MA.

A. F. T. Winfield, J. Sa, M. C. Fernandez-Gago, C. Dixon, and M. Fisher. 2005. On formal specification of emergent behaviours in swarm robotic systems. *International Journal of Advanced Robotic Systems* 2, 4, 363–370.

M. Wooldridge and N. R. Jennings. 1998. Pitfalls of agent-oriented development. In *Proceedings of the 2nd International Conference on Autonomous Agents*. 385–391.

F. Zambonelli, N. Jennings, and M. Wooldridge. 2001. Organisational abstractions for the analysis and design of multi-agent systems. In *Agent-Oriented Software Engineering*. Lecture Notes in Computer Science, Vol. 1957. Springer, 407–422.

D. Zarzhitsky, D. Spears, D. Thayer, and W. Spears. 2005. Agent-based chemical plume tracing using fluid dynamics. In *Formal Approaches to Agent-Based Systems*. Lecture Notes in Computer Science, Vol. 3228. Springer, 146–160.

Received January 2014; revised September 2014; accepted September 2014