

Analysing Robot Swarm Behaviour via Probabilistic Model Checking

Savas Konur, Clare Dixon, and Michael Fisher
Department of Computer Science, University of Liverpool, UK
{Konur, CLDixon, MFisher}@liverpool.ac.uk

Published as *Robotics and Autonomous Systems* 60(2):199-213, Feb. 2012

Abstract

An alternative to deploying a single robot of high complexity can be to utilize robot swarms comprising large numbers of identical, and much simpler, robots. Such swarms have been shown to be adaptable, fault-tolerant and widely applicable. However, designing individual robot algorithms to ensure effective and correct overall swarm behaviour is actually very difficult. While mechanisms for assessing the effectiveness of any swarm algorithm before deployment are essential, such mechanisms have traditionally involved either computational simulations of swarm behaviour, or experiments with robot swarms themselves. However, such simulations or experiments cannot, by their nature, analyse *all* possible swarm behaviours. In this paper, we will develop and apply the use of automated *probabilistic formal verification* techniques to robot swarms, involving an exhaustive mathematical analysis, in order to assess whether swarms will indeed behave as required. In particular we consider a foraging robot scenario to which we apply probabilistic model checking.

1 Introduction

Robot swarms, comprising a number of simple and identical robots, have been developed by many researchers and deployed in significant application areas [4, 5, 1]. Although the behaviour of each individual robot is simple, these robots work together to achieve potentially quite complex swarm behaviour. Consequently, understanding individual robot behaviour is easy, but predicting the overall swarm behaviour is much more difficult. In particular, it is very difficult to design an individual robot control procedure that, when replicated across all the robots, will guarantee the required swarm behaviour.

In [29], we introduced the idea of using formal verification techniques, in particularly probabilistic model checking, to assess swarm behaviour in a more comprehensive manner than simply by simulation or testing on selected scenarios. While we introduced the idea in [29], very little verification and analysis was carried out in that short paper. Here we will extend and develop that work in order to show how probabilistic model checking can be effectively used, in a number of ways, to automatically and exhaustively assess swarm activity. Thus, we aim to show that probabilistic model checking is a useful alternative tool to simulation and practical experimentation that not only allows the simulation of particular runs of the system but also enables verification of quite sophisticated temporal and probabilistic properties.

We exhibit our approach by analysing, in depth, an existing swarm algorithm and its related probabilistic model. Specifically, we examine the state-based control algorithm at the heart of the *foraging robots* developed in [34, 35]. These foraging robots aim to search for, grab, and retrieve, food items, bring them back to their *‘nest’* and then rest. Thus, at any time, robots may be *resting* (in the nest), out *searching* for food, *grabbing* the food, bringing food back to their nest (i.e. *depositing*), or returning to the nest having failed to retrieve food (i.e. *homing*). By selecting swarm algorithms that have already been designed, implemented and tested, we can compare our analysis with the simulation-based assessments from [34, 35]. Specifically, we should be able to formally verify behaviours that have been found to be important within these swarm scenarios. Further, [34] proposes a macroscopic probabilistic model for such a system which we also adapt and apply probabilistic model checking to.

The basic foraging algorithm [34], involves several important parameters that must be supplied. These include the *time each robot spends resting*, the *probability of a robot finding food*, the *energy expended by the robot in searching*, the *energy gained by food*, etc. In analysing swarm algorithms of this sort, it is particularly important to see how the settings of such parameters affect global swarm behaviours in terms of, for example, *overall swarm energy* or the ratio of *searchers* to *resters* within the swarm. By such an analysis we can explore under what conditions the swarm exhibits optimal behaviour. In the verification we carry out here, we show that, given specific parameters to be used in practice, we can assess overall swarm behaviour as well as specific analysis relevant to each system.

Traditionally, the analysis of swarm behaviour is carried out either by testing real robot implementations, or by computational simulations; see, for example, [30, 35]. Real implementations are clearly forced to follow a particular architecture while simulations only examine a subset of all the possible swarm behaviours. However, if the swarms are to be used in safety-critical areas, or even where swarm failure might involve financial penalties, the above approaches guarantee very little

about actual swarm behaviours.

Within Computer Science, a general alternative to the examination of systems through simulation or testing is to use *formal verification*. In particular, the variety of formal verification called *model-checking* [11] has become extremely popular. Here, a mathematical model of *all* the possible behaviours of the system in question is built [21], incorporating all the different robots, and then all executions through this model are assessed against a logical formula describing a required property of the system. If there is a possible swarm execution that violates the required property, then this is highlighted and the swarm designer can examine this potentially anomalous behaviour in detail. Additionally model checkers usually allow the exploration of particular runs of the system if required, i.e. a simulation of the system modelled.

The logical properties assessed in standard model checking are *temporal* properties [20]. However, since robot control algorithms usually require not only the formalisation of temporal behaviour, but also of uncertain behaviours, then we choose to use a more sophisticated model checking approach. Specifically, we use the *probabilistic* model checker PRISM [23], through which we can analyse not just the temporal, but also the probabilistic, properties of the swarm.

In summary, we here target an existing robot swarm algorithm for foraging robots, describe the control algorithm within each robot in terms of a *probabilistic model* [34], and then automatically analyse all possible runs through a system of multiple robots via the PRISM model checker. Note that, while we can verify properties of individual robots, we are particularly concerned with *global* swarm behaviour. Thus, we will show that we can not only re-create the simulation results from relevant papers [34], but can also show that certain properties hold for *all* possible runs/configurations, i.e. we can *formally verify* swarm behaviour. This work takes the basic idea from [29], extends it, applies it to different properties, considers and assesses a number of different ways of robot swarm verification via probabilistic model checking. This work also moves beyond just the testing and simulation of robot swarms to formal verification via probabilistic model checking and provides the potential for more detailed and systematic analysis of swarm behaviours.

In Section 2 we introduce the foraging robot scenario while, in Section 3, we outline the model checking approach to verification and describe the probabilistic model checker PRISM which allows us to carry out both simulation and verification activities. In Sections 4 to 6, we consider a range of different mechanisms for modelling the swarm behaviour and the corresponding problems of carrying out verification via model-checking on these. Thus, in Section 4, we examine the basic *microscopic* or *agent-based* model, in which each robot is modelled in detail and the swarm model is constructed from the product of *all* the individual

robot models. In Section 5 we turn to the *macroscopic* or *population-based* model whereby we abstract away from the details of individual robot behaviours and consider modelling the population as a whole, thus allowing us to represent a much larger number of robots. In Section 6, we consider variations, combinations, and extensions of these approaches.

In these sections, particularly Sections 4 and 5, we show how probabilistic temporal formulae can be verified of *all* executions through these models. This provides a route to the formal modelling of swarm behaviour followed by detailed, yet automatic, formal verification assessing logical properties of all executions. Our particular contribution here is to explore the formal verification of probabilistic and population-based models of swarm behaviour via model checking, and to show how formal verification can indeed provide the swarm designer with a powerful tool for comprehensive analysis to increase his/her confidence while designing a particular swarm.

In Section 7 we discuss related work with respect to both foraging robots and verification for robot swarms. Finally, in Section 8, we provide concluding remarks and explore future work.

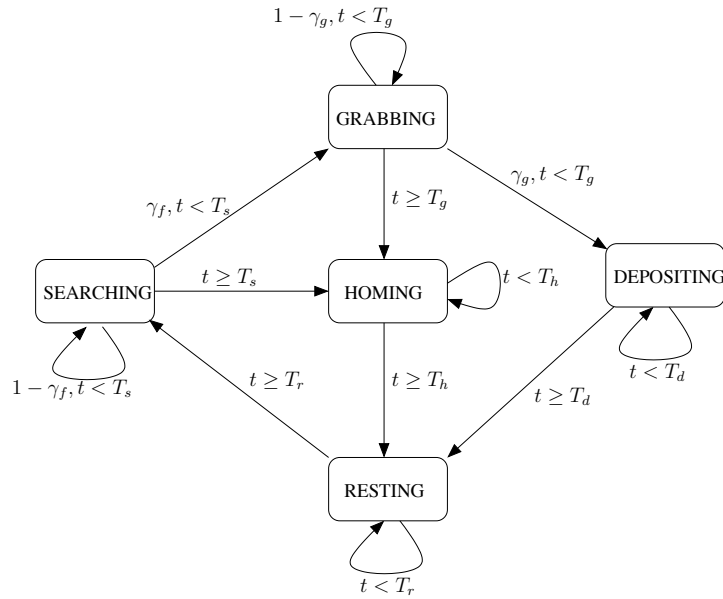


Figure 1: Probabilistic Finite State Machine for a Single Foraging Robot [34].

2 The Foraging Robot Scenario

The foraging robot scenario we base our probabilistic model checking analysis on is that presented in [34]. Here, within a fixed size arena, there are a number of foraging robots, i.e. each robot must search a finite area and bring food items back to the common nest. Food is placed randomly over the arena and more may appear over time. The food items collected will increase the energy of swarm, but searching for food items will use energy up and there is no guarantee that robots will actually *find* any food. The behaviour of each robot in the system is represented by the probabilistic state machine in Figure 1, comprising the states:

SEARCHING, wherein the robot is searching for food items;
GRABBING, wherein the robot attempts to grab a food item it has found;
DEPOSITING, wherein the robot moves home with the food item;
HOMING, wherein the robot moves home without having found food; and
RESTING, wherein the robot rests for a particular time interval.

Associated with transitions between these states are both time-out conditions and probability values:

T_s : amount of time a robot can continue searching;
 T_g : amount of time a robot can attempt grabbing;
 T_d : amount of time spent depositing;
 T_h : amount of time spent homing;
 T_r : amount of time spent resting;
 γ_f : the probability of finding a food item; and
 γ_g : the probability of grabbing a food item.

In an actual swarm model, timeouts may vary for DEPOSITING and HOMING, e.g. due to the distance from the nest, so we consider the average timeouts.

The probabilistic finite state machine in Figure 1 is adapted from [34], and it does not include the obstacle avoidance activities. We consider the case of avoidance in Section 5.2.4. Importantly, in this study, we are not aiming to directly model any *specific* swarm; our thesis is that formal modelling and automated verification of the form carried out here can be useful in the analysis required in swarm robotics design. Thus, while we do not claim to verify *exactly* the detailed model from [34], we show that our approach can at least carry out all the forms of test/simulation reported in [34] and can go much further in terms of full exploration.

Note that, in order to apply model-checking methods we must ensure that the model under study has only finitely-many states. Hence we model timeouts (i.e. T_s, T_g, T_d, T_h, T_r) as a sequence of states representing each state in Figure 1. For

example, if $T_r = 4$ the RESTING state is replaced by four states R_i for $i = 0, \dots, 3$ representing robots who have been resting $i + 1$ moments. To clarify, we note that in Figure 2 the states and transitions on the left are transformed to the combination on the right, signifying a delay of three additional time steps. So, instead of the resting phase taking one time step, it now takes four, moving on to searching.

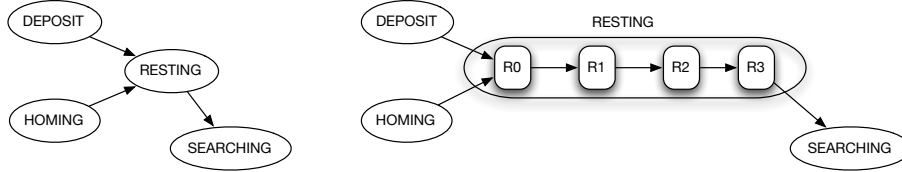


Figure 2: Practical Modelling of Delay — in model checking the subgraph on the left is replaced by that on the right.

Initially all robots are in the state SEARCHING. At each time step, robots move to the GRABBING state with probability γ_f (the chance of a robot finding food). Robots stay in the SEARCHING state with probability $1 - \gamma_f$. If a robot cannot find food within T_s time steps (i.e. $t \geq T_s$), it will move to the HOMING state. In the GRABBING state the robots move to the DEPOSITING state with probability γ_g (the chance of grabbing the food), and stay in the GRABBING state with probability $1 - \gamma_g$. If a robot cannot grab a food item in T_g time steps (i.e. $t \geq T_g$), it will move to the HOMING state. The robots in the HOMING (respectively DEPOSITING, RESTING) state take T_h (respectively T_d, T_r) time steps to return back to the nest (respectively deposit food, rest) before they move to the next state.

3 Model Checking Using PRISM

Formal verification aims to assess a logical requirement (given within *formal logic*) of all behaviours within a system. Within this general field, *model checking* [11] is an algorithmic technique for exhaustively analysing the logical correctness of a finitely-represented system. It checks a logical requirement (typically stated in a form of *temporal logic*) against all possible behaviours of the system in question. Thus a finite structure (such as a finite-state automaton), is used to describe *all* possible behaviours of the system and then model checking assesses this logical requirement against all possible paths through the finite-state automaton (corresponding to all possible runs of the system). Model-checking has come to prominence in recent years as it provides fast, automated, and relatively easy to use verification techniques. These, in turn, are embodied within tools that are widely available, such as SPIN [24], NuSMV [10], Java PathFinder [40], and UPPAAL [3].

Given that the state-machine in Figure 1 describes a *probabilistic* model, we can analyse both *probabilistic* and *temporal* properties of our robots using a *probabilistic model checker*, such as PRISM [23]. This supports three types of probabilistic model: *Discrete-Time Markov Chains (DTMCs)*; *Continuous Time Markov Chains (CTMCs)* and *Markov Decision Processes (MDPs)*. When we assess such models containing detailed probabilistic information, it is natural to use logical requirements which can describe these probabilistic aspects. So, when we verify properties of an individual probabilistic state-machine, our input to PRISM is a probabilistic model (technically a DTMC) and a property which can be represented in a number of *probabilistic temporal logics*, such as PCTL [22].

PCTL is an extension of the well-known branching-time temporal logic CTL [16] which can be used to represent quantities such as “the probability a robot eventually reaches the nest”, “the probability that the energy in the system is greater than E ”, etc.

PCTL formulae are interpreted over a Markov chain (or an Markov decision process). The execution of a Markov chain constructs a set of *paths*, which are *infinite* sequences of states. The i -th element of a path σ is denoted by $\sigma[i]$. The set of paths from a state s is denoted by $Paths(s)$. A probability measure π_m for a set of paths with a common prefix of the length n , $s_0 \rightarrow s_1 \rightarrow \dots \rightarrow s_n$, is defined to be the product of transition probabilities along the prefix, i.e. $\mu(\langle s_0, a_1, s_1 \rangle) \times \dots \times \mu(\langle s_{n-1}, a_n, s_n \rangle)$ [22].

Apart from the usual operators from classical logic such as \wedge (and), \vee (or) and \Rightarrow (implies), PCTL has the *probabilistic operator* $P_{\sim r}$ ¹, where $0 \leq r \leq 1$ is a *probability bound* and $\sim \in \{<, >, \leq, \geq, =\}$. Intuitively, a state, s , of a model satisfies $P_{\sim r}[\varphi]$ if, and only if, the probability of taking a path from s satisfying the *path formula* φ is specified by ‘ $\sim r$ ’. The following path formulae φ are allowed: $\bigcirc\phi$; $\diamond\phi$; $\square\phi$; $\phi U\psi$; and $\phi U^{\leq k}\psi$ (Note that the operators $\diamond\phi$ and $\square\phi$ can actually be derived from $\phi U\psi$).

As an example, the property that “the probability of φ eventually occurring is greater than or equal to b ” can be expressed in PCTL as follows:

$$P_{\geq b}[\text{true } U^{<\infty}\varphi].$$

The informal meanings of such formulae are:

$\bigcirc\phi$ is true at a state on a path if, and only if, ϕ is satisfied in the next state on the path;

$\diamond\phi$ is true at a state on a path if, and only if, ϕ holds at some present/future state on that path;

¹The $P_{\sim r}$ operator is the probabilistic counter-part of path-quantifiers \forall and \exists of CTL.

$\Box\phi$ is true at a state on a path if, and only if, ϕ holds at all present/future states on that path;

$\phi U\psi$ is true at a state on a path if, and only if, ϕ holds on the path up until ψ holds; and

$\phi U^{\leq k}\psi$ is true at a state on a path if, and only if, ψ satisfied within k steps on the path and ϕ is true up until that moment.

The formal semantics of PCTL formulas are given as follows: Assume \mathcal{M} is either a DTMC or an MDP, and V is the corresponding valuation function mapping each state s to a set of propositions. For a given PCTL-formula φ and a state s , the satisfaction relation \models is inductively defined on the structure of φ as follows [22]:

$$\mathcal{M}, s \models p \text{ iff } p \in V(s)$$

$$\mathcal{M}, s \models \neg\phi \text{ iff } \mathcal{M}, s \not\models \phi$$

$$\mathcal{M}, s \models \phi_1 \wedge \phi_2 \text{ iff } \mathcal{M}, s \models \phi_1 \text{ and } \mathcal{M}, s \models \phi_2$$

$$\mathcal{M}, s \models P_{\sim r}[\psi] \text{ iff } \pi_m(\sigma \in Paths(s) \text{ s.t. } \mathcal{M}, \sigma \models \psi) \sim r$$

Path formulas are defined on the following semantics:

$$\mathcal{M}, \sigma \models \bigcirc\phi \text{ iff } \mathcal{M}, \sigma[1] \models \phi$$

$$\mathcal{M}, \sigma \models \phi_1 U \phi_2 \text{ iff } \exists i \geq 0 \text{ s.t. } \mathcal{M}, \sigma[i] \models \phi_2 \text{ and } (\forall j < i) \mathcal{M}, \sigma[j] \models \phi_1$$

$$\mathcal{M}, \sigma \models \phi_1 U^{\leq k} \phi_2 \text{ iff } \exists i \leq k \text{ s.t. } \mathcal{M}, \sigma[i] \models \phi_2 \text{ and } (\forall j < i) \mathcal{M}, \sigma[j] \models \phi_1$$

PRISM can then be used to verify whether or not a PCTL formula holds on a given probabilistic structure representing all the executions of the system being modelled. In addition, PRISM can also be used to compute either the minimum or maximum probability over a range of possible configurations or parameters of a model, producing a form of best/worst-case analysis.

As well as probabilistic queries, PRISM also supports quantitative structures defining *costs* and *rewards*. These structures can be used to reason about quantitative measures such as “expected number of hits”, “expected success rate”, etc. This is achieved using ‘R’ operator, which works in a similar fashion to the ‘P’ operator above. For example, $R^{=?}[C \leq 10]$ returns the expected cumulative reward within 10 units of operation.

One of the key features of PRISM is its underlying symbolic implementation technology using data structures based on binary decision diagrams (BDDs) [6, 7].

These allow both compact representation and efficient manipulation of extremely large probabilistic models, by exploiting structure and regularity derived from their high-level description. Such techniques have been successfully applied to the probabilistic verification of models with as many as 10^{10} states [15].

4 Microscopic Approach

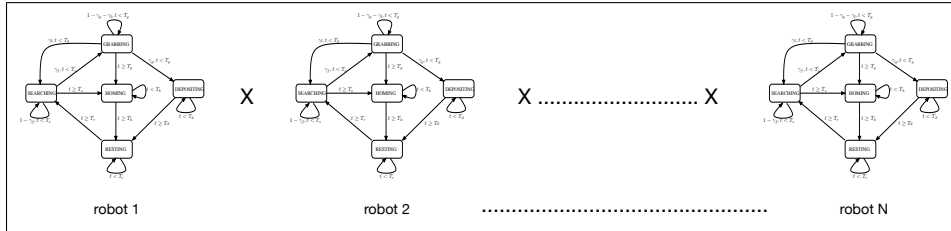


Figure 3: Microscopic View of Swarm Behaviour — product of individual robot behaviours provides state space of the whole system.

To model swarms based on the transition system in Figure 1 we have a number of options. The first is the *microscopic approach*, where we instantiate such a transition system for *each* robot in the swarm and then take the *product of* all these to provide the behaviour of the overall swarm². Global variables can then be used to calculate, for example, the number of robots searching or resting at any moment, or the swarm energy.

Although this approach is convenient for observing the behaviour of single (or small numbers of) robots, it is very inefficient to apply it to the overall swarm behaviour. Most model checkers require the full product of the state space induced by the transition system of individual robots to be generated. A schematic view of this approach is given in Figure 3. The product of the state space quickly leads to an explosion in the number of global states, a well known problem for model checkers [12].

In order to practically evaluate the efficiency of this approach we created a PRISM model for each robot. As described above, PRISM then takes the product of these to create the state space for the overall swarm. In Table 1 we present

²When taking the product of two automata, there are many variations. We take a simple, synchronous view, as follows. For every state in automaton A , say A_1, A_2, A_3, \dots and automaton B , say B_1, B_2, B_3, \dots , then given transitions $A_1 \rightarrow A_2$ labelled by α and $B_1 \rightarrow B_2$ labelled by β , then we can have a transition $A_1B_1 \rightarrow A_2B_2$ labelled by $\alpha\beta$ where A_1B_1, A_2B_2, \dots , are states in the product automaton and $\alpha\beta$ is a consistent label. We do this for every possible pair of transitions.

the model construction time for PRISM and the corresponding state space for each different swarm size³.

Table 1: State space and time for model construction.

| number of robots | model construction time | state space |
|------------------|-------------------------|----------------------|
| 1 | 0.10 sec. | 2.8×10^2 |
| 2 | 0.69 sec. | 8.2×10^6 |
| 3 | 4.35 sec. | 2.3×10^{10} |
| 4 | 12.6 sec. | 6.6×10^{13} |
| 5 | 28.5 sec. | 1.9×10^{17} |
| 6 | 54.5 sec. | 5.4×10^{21} |

As can be seen from Table 1, the number of states is very small for single robots; but the state space explodes rapidly if we increase the number of robots. Even with just six robots the state space reaches the magnitude of 10^{21} .

The situation is even more problematic when undertaking verification, which is even more demanding in terms of computational resources. Table 2 shows that the memory requirement for the verification of a simple property (such as the property stating that the total number of robots is always equal to the initial number of robots) exceeds 3 GB even in the case of only three robots. We have insufficient memory to verify this property for more than three robots.

Table 2: Verification time and memory usage vs. swarm size.

| number of robots | verification time | memory usage |
|------------------|-------------------|--------------|
| 1 | 0.38 sec. | 81 KB |
| 2 | 78 sec. | 156 MB |
| 3 | 835 sec. | > 3 GB |
| 4 | — | — |

These results show that, due to the state explosion problem, the microscopic approach is very inefficient if we wish to observe the behaviour of the overall swarm, and this analysis becomes effectively impossible with even a medium number of robots. However, for a detailed analysis of the interaction between small numbers of robots, this microscopic approach remains useful.

³Experiments were run on a 2.4 Ghz Core 2 Duo computer with 6 GB RAM running Mac OS 10.5.7.

5 Macroscopic Approach

PRISM is generally quite efficient, allowing us to analyse models with as many as 10^{10} states (see, for example, [15]). However, as we saw in Section 4, the naive application of PRISM to robot swarm verification may generate even larger state spaces. In the microscopic approach we built a product state-machine from multiple copies of the state-machine in Figure 1. The size of the resulting model was *huge*. We now consider an alternative approach to solving the problem of verifying the properties of swarms comprising large numbers of robots. Rather than representing each robot as a different probabilistic state machine and then taking the *product* of all these machines to generate the whole system, we use a *counting abstraction* approach (see, for example, [13]). Sometimes called the *population model*, this is particularly useful if there are many identical, independent processes, as is the case in a robot swarm, and allows us to abstract away from low-level probabilistic details and so just consider global population behaviour.

5.1 Modelling the Scenario

Since we know that all the robots are modelled by identical probabilistic state machines, then we will model the whole system by *one* state machine with exactly the SEARCHING, HOMING, etc, states we saw in Figure 1. However, to each of these states we add a counter which is used to record how many robots are *actually* in that state at that moment. Thus, if 20 robots are searching then the counter in the SEARCHING state will be 20. By examining Figure 1 we can calculate how many of these should move to GRABBING, how many should move to HOMING, and how many should remain in SEARCHING at each step. Thus, in addition to the 5 states, each state is labelled with a set of difference equations explaining how the number of robots associated with each state evolves. It is important to note that we are abstracting away from local probabilities and now considering a more global view.

Now we are in a position to define the (difference) equations describing how the numbers of robots in each of the 5 states changes over time.

At time t , let

- $NS_i(t)$: number of robots at the SEARCHING state for i time steps ($i \in \{0, \dots, T_s - 1\}$),
- $NG_i(t)$: number of robots at the GRABBING state for i time steps ($i \in \{0, \dots, T_g - 1\}$),
- $ND_i(t)$: number of robots at the DEPOSITING state for i time steps ($i \in \{0, \dots, T_d - 1\}$),

- $NH_i(t)$: number of robots at the HOMING state for i time steps ($i \in \{0, \dots, T_h - 1\}$), and
- $NR_i(t)$: number of robots at the RESTING state for i time steps ($i \in \{0, \dots, T_r - 1\}$).

Now, assume that $N_s(t)$, $N_g(t)$, $N_d(t)$, $N_h(t)$ and $N_r(t)$ denote the *total* number of robots in the states SEARCHING, GRABBING, DEPOSITING, HOMING and RESTING, respectively at time step t . Then, the number of robots in each state at time step t can be calculated using the following equations:

$$\begin{aligned}
N_s(t) &= \sum_{i=0}^{T_s-1} NS_i(t); & N_g(t) &= \sum_{i=0}^{T_g-1} NG_i(t); \\
N_d(t) &= \sum_{i=0}^{T_d-1} ND_i(t); & N_h(t) &= \sum_{i=0}^{T_h-1} NH_i(t); \\
N_r(t) &= \sum_{i=0}^{T_r-1} NR_i(t)
\end{aligned}$$

and the total number of robots at time t , $N_{total}(t)$ is as follows.

$$N_{total}(t) = N_s(t) + N_g(t) + N_d(t) + N_h(t) + N_r(t)$$

Following Figure 1, $NS_i(t)$, $NG_i(t)$, etc., are calculated by:

$$\begin{aligned}
\mathbf{NS}_i: & \quad NS_0(t+1) = NR_{T_r-1}(t) \\
& \quad NS_1(t+1) = (1 - \gamma_f)NS_0(t) \\
& \quad \dots\dots \\
& \quad NS_{T_s-1}(t+1) = (1 - \gamma_f)NS_{T_s-2}(t) \\
\mathbf{NG}_i: & \quad NG_0(t+1) = \gamma_f \sum_{i=0}^{T_s-2} NS_i(t) \\
& \quad NG_1(t+1) = (1 - \gamma_g)NG_0(t) \\
& \quad \dots\dots \\
& \quad NG_{T_g-1}(t+1) = (1 - \gamma_g)NG_{T_g-2}(t) \\
\mathbf{ND}_i: & \quad ND_0(t+1) = \gamma_g \sum_{i=0}^{T_g-2} NG_i(t) \\
& \quad ND_1(t+1) = ND_0(t) \\
& \quad \dots\dots \\
& \quad ND_{T_d-1}(t+1) = ND_{T_d-2}(t)
\end{aligned}$$

$$\begin{aligned}
\mathbf{NH}_i: \quad & NH_0(t+1) = NG_{T_g-1}(t) + NS_{T_s-1}(t) \\
& NH_1(t+1) = NH_0(t) \\
& \dots\dots \\
& NH_{T_h-1}(t+1) = NH_{T_h-2}(t) \\
\mathbf{NR}_i: \quad & NR_0(t+1) = NH_{T_h-1}(t) + ND_{T_d-1}(t) \\
& NR_1(t+1) = NR_0(t) \\
& \dots\dots \\
& NR_{T_r-1}(t+1) = NR_{T_r-2}(t)
\end{aligned}$$

Note that fractional values are rounded to the nearest whole number.

5.2 Verification

In this section we will analyse different scenarios based on different parameter settings. All these scenarios have been the subject of relevant research which has been carried out on real robots or simulations. Here we will consider them in a verification context.

The probabilistic model developed in [34] was assessed by carrying out simulations, effectively testing only selected scenarios. We now wish to capture these in PCTL and then carry out PRISM verification. There are many properties we can verify, but we will concentrate on the ones derived from corresponding questions in the related papers, e.g. [34].

So, in this section, we present the results from running PRISM on our model with different properties and parameters. We run them both in *simulation mode*, whereby we generate a single random run, and in *verification mode*, whereby we assess all possible runs against a PCTL formula.

However, before discussing the model checking experiments, we will explain how swarm energy is calculated. In a swarm, each robot consumes a certain amount of energy at each time step. We assume that a robot consumes E_s , E_g , E_r and E_h units of energy at each step in the SEARCHING, GRABBING, RESTING and HOMING states, respectively, and each food-item delivers the swarm E_d units of energy (we assume that E_d is net energy, i.e. it is the energy obtained from the food carried minus the energy consumed in the depositing state; we also assume that a robot can carry only one food-item.) The total swarm energy in the next time instance, denoted by $En(t+1)$, is calculated as follows:

$$En(t+1) = En(t) + E_d ND_{T_d-1}(t) - E_s N_s(t) - E_g N_g(t) - E_r N_r(t) - E_h N_h(t)$$

where $ND_{T_d-1}(t)$ denotes the number of robots that have been in DEPOSITING for $T_d - 1$ time steps; and $N_s(t)$, $N_g(t)$, $N_r(t)$ and $N_h(t)$ denote the number

of robots that are in the SEARCHING, GRABBING, RESTING and HOMING states, respectively, at time t .

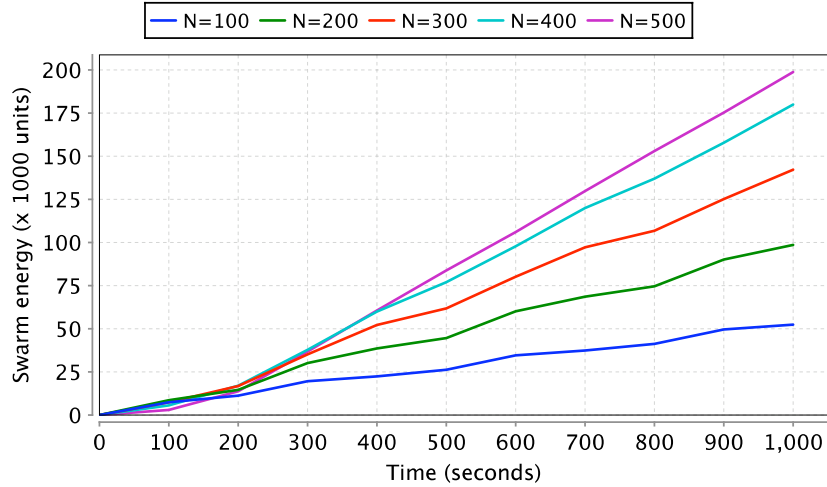


Figure 4: Total swarm energy vs. the total number of robots, where $\gamma_f = 0.5$ and $E_d = 40$.

In the sequel, we take N as the total number of robots. The energy parameters used in the verifications are as follows: $E_d = 50$, $E_r = 2$, $E_h = 6$, $E_s = 12$ and $E_g = 12$. We also take $T_s = T_g = T_h = T_d = T_r = 5$ seconds. We note that these parameters are easy to change for comparison with specific robot simulations or experiments.

5.2.1 Basic Swarm Model (with fixed γ_f and γ_g)

We begin our analysis with a model based on the state structure in Figure 1, which we will subsequently enhance to become more sophisticated aiming to show that we can easily capture a number of different scenarios. We start with a scenario, where γ_f (probability of finding food) and γ_g (probability of grabbing food) are constant. Throughout this scenario, we assume $\gamma_g = 0.7$.

Figure 4 illustrates the simulation of the total swarm energy w.r.t. different number of robots, when $E_d = 40$ (the energy delivered by each food item). The figure shows that the swarm gains energy throughout the foraging process. From this observation, we would conclude that this swarm colony can survive since it will have sufficient energy. However, this does not reflect *all* situations, because the simulation in the figure considers only *some* behaviour of the system. In order

to explore *all* behaviours, our analysis should be based on verification.

Using the verification module of PRISM we can *verify* whether a property holds for all possible runs; or we can determine the actual probability of a property being true. Assume we want to check the scenario above with verification, and assume we want to consider an arbitrary food finding probability. Table 3 illustrates the verification results of the expected energy levels reached within T seconds, expressed by $R_{=?}[C \leq T]$, when we take $\gamma_f \in \{0, 0.1, \dots, 1\}$, $E_d = 40$ and $T = 1000$. In Table 3, $?$ is used to query PRISM to return the value (Here, C is a PRISM operator denoting the accumulation.). As the table shows, the expected energy levels of the swarm are negative, if we consider all possible cases.

| | $R_{=?}[C \leq T]$ | $P_{=?}\square(R_{>0}[S])$ |
|-----|-----------------------|----------------------------|
| N | expected energy | probability |
| 100 | -12.96×10^3 | 0.0 |
| 200 | -34.46×10^3 | 0.0 |
| 300 | -76.97×10^3 | 0.0 |
| 400 | -134.52×10^3 | 0.0 |
| 500 | -197.37×10^3 | 0.0 |

Table 3: Verification results of expected swarm energy for different number of robots ($E_d = 40$, $\gamma_f \in \{0, 0.1, \dots, 1\}$).

| | $R_{=?}[C \leq T]$ | $P_{=?}\square(R_{>0}[S])$ |
|-----|----------------------|----------------------------|
| N | expected energy | probability |
| 100 | 135.00×10^3 | 1.0 |
| 200 | 260.79×10^3 | 1.0 |
| 300 | 361.94×10^3 | 1.0 |
| 400 | 447.97×10^3 | 1.0 |
| 500 | 522.66×10^3 | 1.0 |

Table 4: Verification results of expected swarm energy for different number of robots ($E_d = 50$, $\gamma_f \in \{0.1, 0.2, \dots, 1\}$).

Table 3 also shows the probability that the expected energy's always being positive in the long run, expressed by $P_{=?}\square(R_{>0}[S])$, is 0.0 (Here, ' S ' is an operator which denotes the *steady state (long-run or equilibrium)* behaviour of a model.) This shows that the swarm will not have enough energy in the long run to survive.

Like other model checkers, PRISM provides a visual presentation of the individual executions, and counter-examples. This tool helps us analyse the counter-examples when a property does not hold, and change the model accordingly. As we

show in Table 3, the verification result of $P_{=?}\Box(R_{\geq 0}[S])$ 0.0. When we analysed the traces provided by the PRISM we figured out that more energy-rich foods need to be provided, and the food finding probability (γ_f) should always be positive. We therefore changed the parameters accordingly, and re-ran the verification experiments. The Table 4 illustrates the new results, where we changed the parameters as $E_d = 50$ and $\gamma_f \in \{0.1, 0.2, \dots, 1\}$ (we kept the rest of the parameters same), and modify the model accordingly. The results show that the expected swarm energy (which is the accumulation of all possible paths) never goes below 0.

We can actually verify a single property by simply considering a random choice of robots. In this case, we assume N is chosen arbitrarily from the set $\{20, 40, \dots, 500\}$. When we perform the verification, the PRISM returns 305×10^3 for the property $R_{=?}[C \leq T]$ (where $T = 1000$ seconds), and returns 1.0 for the property $P_{=?}\Box(R_{\geq 0}[S])$.

Another design condition is that the total number of robots, N , never changes (i.e. equal to the initial number of robots N_{init}). This can be expressed in PCTL as follows:

$$P_{\geq 1}\Box(N = N_{init})$$

where N is the initial number of robots. We also verified this formula using PRISM, thus guaranteeing that the above design condition is indeed satisfied.

5.2.2 Swarm Model with variable γ_f and γ_g

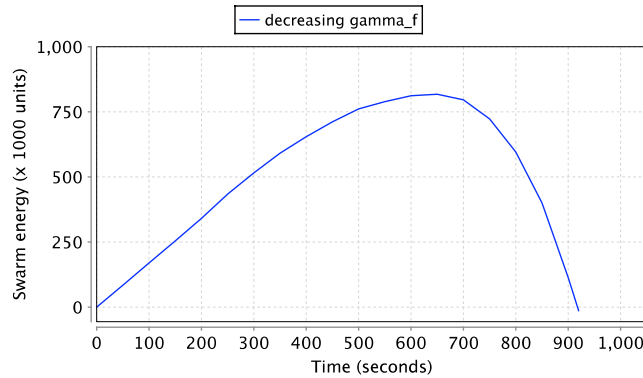


Figure 5: Total swarm energy vs. the (decreasing) probability of finding food (γ_f) for $N = 2000$ (We assume $\gamma_g = 0.7$ and $E_d = 50$).

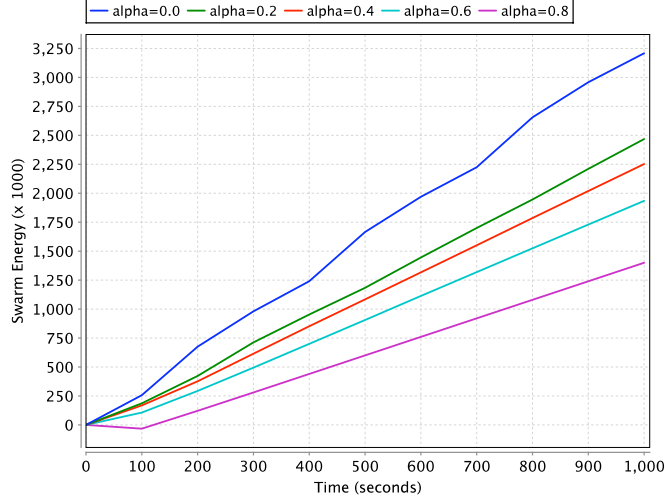


Figure 6: Total swarm energy v.s. α , the density of arena, for $N = 2000$ (We assume γ_f and γ_g are variable, and $E_d = 50$.)

In Figure 4, the γ_f values remained constant throughout the runs. Now, in Figure 5, we use a *variable* γ_f value that decreases over time, i.e. modelling the situation when food gradually becomes more scarce. So, γ_f is 1.0 at $t = 0$, and reduces by 0.001 in every second. The simulation in Figure 5 shows that the total swarm energy initially increases, since the probability of finding food is high, i.e. there is much food available for the swarm. When the probability of finding food decreases, i.e. food availability reduces, the energy gain becomes equal to the energy spent by the swarm. After a while the energy gained becomes less than the energy spent, since the food becomes scarce, and therefore the total swarm energy decreases.

Although γ_f is variable in Figure 5, its decrease depends only on time. This dependency does not allow an infinite execution of the system. In Figure 6 and Figure 7 we instead consider that γ_f and γ_g are variable. Namely, γ_f depends on the *number of robots* foraging (i.e. robots searching, grabbing and depositing). Specifically, we assume that

$$\gamma_f(t) = \left(1 - \frac{\alpha N_{foraging}(t)}{N}\right)$$

where $N_{foraging}(t) = N_s(t) + N_g(t) + N_d(t)$, and α is the *density factor*, such that $\alpha \in \{0, 0.1, \dots, 1\}$. The density factor denotes how densely the arena is populated (informally speaking, it represents the grid size and population ratio).

So, if the arena is densely populated, i.e. α is high, then γ_f will be more sensitive to the changes in the number of foraging robots.

We also assume that γ_g depends on the number of robots grabbing, which is formulated as follows:

$$\gamma_g(t) = \left(1 - \frac{\alpha N_g(t)}{N}\right).$$

Intuitively, if more robots are in the GRABBING state, the probability of a robot being able to grab a food item decreases; if fewer robots are ‘GRABBING’, the robot has more chance to grab a food item (i.e. there is less competition).

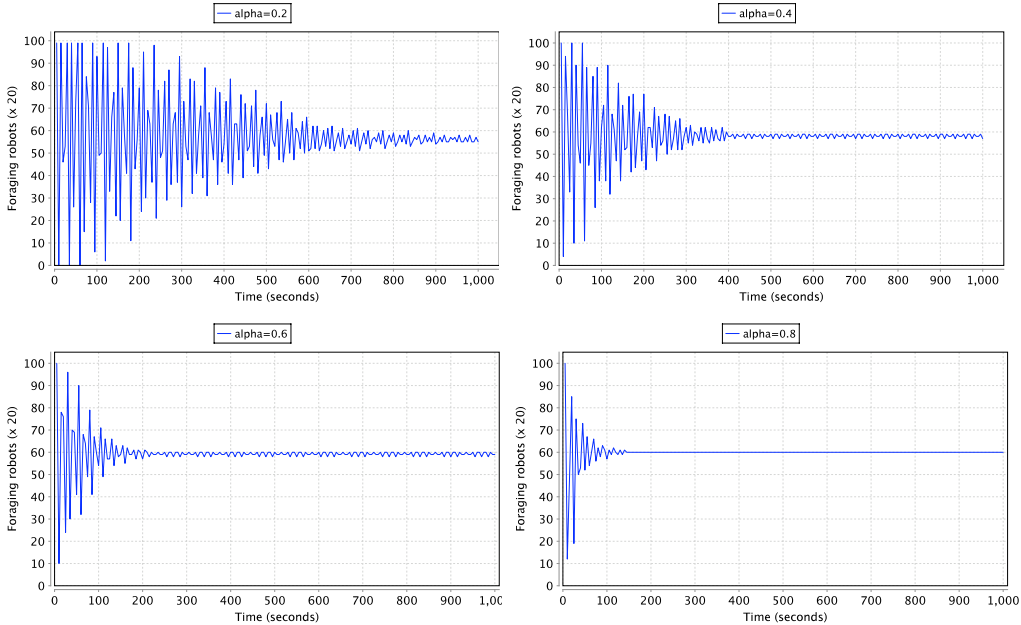


Figure 7: Number of foraging robots v.s. α , the density of arena, for $N = 2000$ (We assume γ_f and γ_g are variable, and $E_d = 50$.)

Figure 6 shows that the energy gain of the swarm decreases if the density of the arena increases whereas Figure 7 shows that the number of foraging robots converges to oscillate above and below some value in all cases; but it will converge more quickly if the density of the arena is high.

Rather than just running individual experiments, we can *verify* some properties. Assume we wish to verify the probability that

“if the number of robots is more than n , then the average swarm energy

exceeds E within t_A time steps”.

This property can be used to make sure that the swarm reaches a desired energy level within a certain time, if there are sufficient number of robots. This property can be expressed in PCTL as follows:

$$P_{=?}\Box(N \geq n \Rightarrow \text{true } U^{\leq t_A}(En_{average} \geq E))$$

We checked the above formula under the assumptions that γ_f, γ_g are variable, $\alpha \in \{0, \dots, 1\}$, $N \in \{20, 40, \dots, 400\}$, $n = 100$, $E = 100 \times 10^3$, and $t_A = 100$. PRISM returned a probability value of 0.97. This result shows that, in this scenario, it is most likely (97%) that the total energy required is reached within the required time, if the swarm has at least 100 robots.

Using the same settings we also queried the following formula

$$P_{=?}\Box(t \geq t_A \wedge N \geq k \Rightarrow N_{foraging} \geq n)$$

which analyses the probability that after t_A time steps the number of foraging robots is always greater than n , if the total size of the swarm is larger than k robots. The verification result 1.0 for $n = 100$, $k = 300$ and $t_A = 100$. This property is also useful for checking whether a certain number of robots are always foraging.

5.2.3 Swarm Model with variable γ_f and γ_g and without Resting Timeout

In the scenarios above, we assumed that if a robot moves to the RESTING state from HOMING or DEPOSITING states, it waits T_r time steps in the RESTING state. We now change this scenario and assume that the robots do not wait in the RESTING state for a fixed amount of time before moving to the SEARCHING state, but the waiting time depends on a probability γ_s . This probability, in turn, depends on the number of robots in the DEPOSITING and HOMING states. Namely, robots move from RESTING to SEARCHING states with probability

$$\gamma_s(t) = \frac{\lambda N D_{T_d-1}(t) - N H_{T_h-1}(t)}{N},$$

and stay in the RESTING state with probability $1 - \gamma_s$. In the above, λ is the *energy gaining parameter*, and $N D_{T_d-1}(t)$ (respectively $N H_{T_h-1}(t)$) denotes the number of robots that have been in DEPOSITING (respectively HOMING) for $T_d - 1$ (respectively $T_h - 1$) time steps. Intuitively, we can think of this new scenario as follows: since each robot brings a food item when in the DEPOSITING state, if more robots move to the DEPOSITING state and fewer robots move to the HOMING state, then more robots will move to the SEARCHING state. As can be seen above,

γ_s is proportional to the *energy gaining parameter* λ . That is, if we increase the value of λ , then more robots will move to the SEARCHING state.

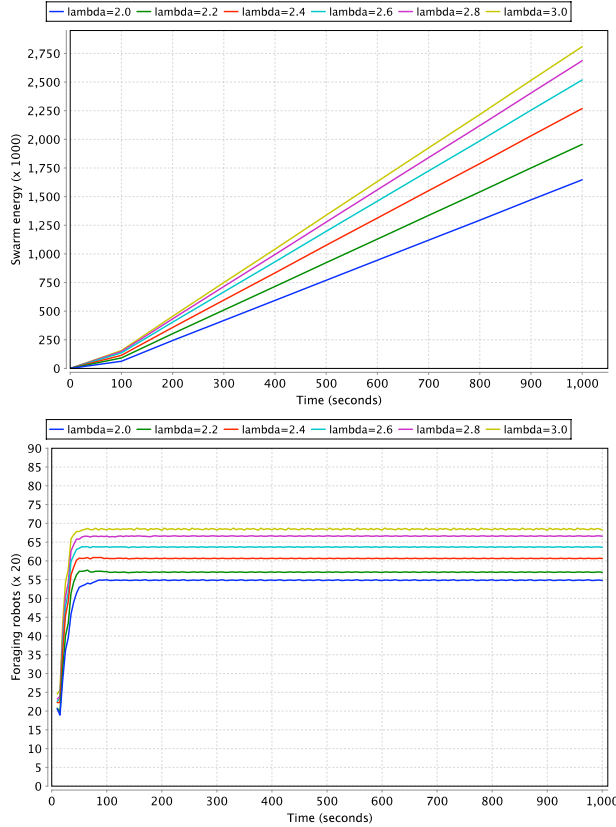


Figure 8: Number of foraging robots and total swarm energy vs food energy gain (λ) for $N = 2000$ (We assume γ_f, γ_g are variable, $\alpha \in \{0, 0.1, \dots, 1\}$ and $E_d = 50$).

We can simulate the swarm behaviour of the swarm for a particular number of robots and λ value. Assume γ_f and γ_g are defined as in Section 5.2.2, where we take $\alpha \in \{0, 0.1, \dots, 1\}$. Figure 8 shows the number of foraging robots and corresponding energy gains with respect to different λ values. If we increase λ , both the number of foraging robots and the energy of the swarm increases.

For our scenario, we define a PRISM model where λ takes discrete steps with increments of 0.1 in the range of values that λ can take. In this experiment, we take $\lambda \in \{2.0, 2.1, \dots, 3.0\}$ and $N \in \dots \{20, 40, \dots, 200\}$. We also assume that γ_f and γ_g are defined as in Section 5.2.2, where we take $\alpha \in \{0, \dots, 1\}$.

Then we checked the following PCTL property:

$$P_{=?}\Box(R_{=?}[C \leq T]\{N \geq m\} \geq 2 \times R_{=?}[C \leq T]\{N \geq n\})$$

which assesses the probability that the expected energy of the swarm whose size greater than m is at least the double the expected energy of the swarm whose size is greater than n . PRISM actually verifies that this is always the case for $m = 200$ and $n = 100$. This property is an example of comparing the behaviour of two different swarm.

Note that in the subformula $R_{=?}[C \leq T]\{N \geq m\}$, the $\{\}$ represents a *filter* expression. That is, the property is only checked at the executions where $N \geq m$ is satisfied.

We also checked following property:

$$P_{=?}\Diamond(P_{\geq 1}\Box N_{foraging} \geq l\{n \leq N \leq m\})$$

which states that if the number of robots is between n and m , then there is a future time point from which the number of foraging robots is always greater than l . The verification result of this property 0.60 for $n = 100$, $m = 200$ and 50, which implies this is guaranteed only 60% of time.

5.2.4 Collision Avoidance

In the original probabilistic model in [34] there are additional states to deal with avoidance. In particular, the states SEARCHING, GRABBING, DEPOSITING and HOMING each have an associated state AVOIDANCE _{x} where $x \in \{s, g, d, h\}$ where AVOIDANCE _{s} denotes avoidance when searching, AVOIDANCE _{g} denotes avoidance when grabbing etc. Associated with each avoidance state is a parameter T_a which denotes the time taken for avoidance and a probability γ_r of moving to an avoidance state. In the case of the states AVOIDANCE _{s} , AVOIDANCE _{d} and AVOIDANCE _{h} after the time for avoidance is over there is a transition back to the state that the robot was previously in i.e. SEARCHING, DEPOSITING or HOMING (assuming that the searching time T_s , depositing time T_d or homing time T_h is not over). From the state AVOIDANCE _{g} , after the time for avoidance is over there is a transition back to the state SEARCHING (assuming the grabbing time T_g is not over). If, in each case the time for the searching T_s , grabbing T_g , depositing T_d or homing T_h is over a transition is made to the state they would have been in if avoidance had not taken place. That is HOMING if they were in the AVOIDANCE _{s} or AVOIDANCE _{g} states and RESTING if they were in the AVOIDANCE _{s} or AVOIDANCE _{d} states. The probabilistic state machine which models this scenario is presented in Figure 9.

To model the new scenario we must first extend the calculations of how many robots are in each state to take account of these additional states and transitions.

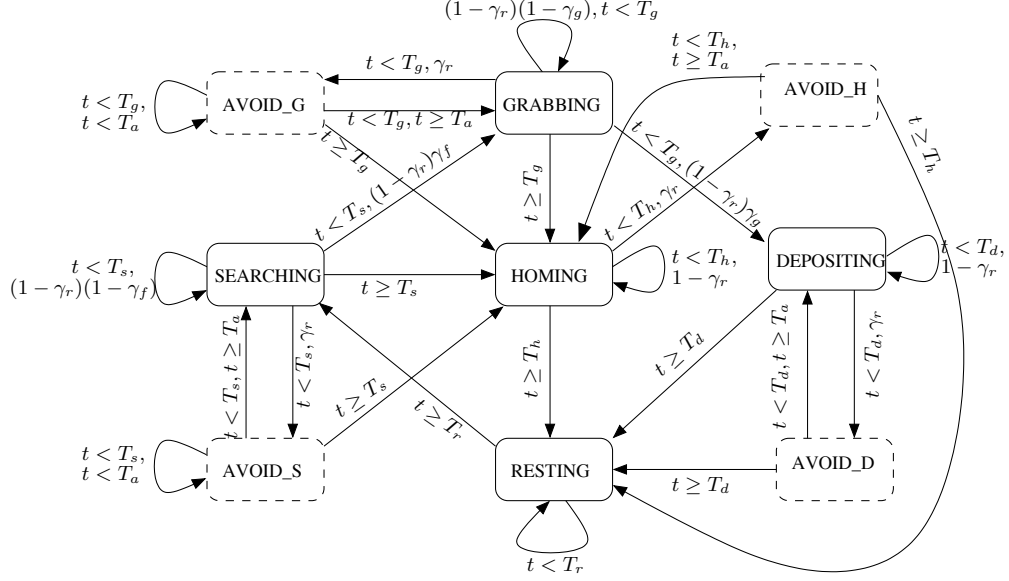


Figure 9: Probabilistic state machine extended with Avoidance states.

Let $NAS_{j,i}(t)$ be the number of robots in the AVOIDANCE_s for i time steps ($i \in \{0, \dots, T_a - 1\}$) having already performed j time steps ($j \in \{0, \dots, T_s - 1\}$) in searching (We can define $NAG_{j,i}(t)$, $NAD_{j,i}(t)$ and $NAH_{j,i}(t)$, similarly.) In order to cover the *collision avoidance*, we modify the equations in Section 5.1 as follows:

$$\begin{aligned} \mathbf{NAS}_{j,i}: \quad & NAS_{j,0}(t+1) = \gamma_r NS_j(t) \\ & NAS_{j+1,1}(t+1) = NAS_{j,0}(t) \\ & \dots\dots\dots \\ & NAS_{j+1,T_a-1}(t+1) = NAS_{j,T_a-2}(t) \end{aligned}$$

$$\begin{aligned} \mathbf{NAG}_{j,i}: \quad & NAG_{j,0}(t+1) = \gamma_r NG_j(t) \\ & NAG_{j+1,1}(t+1) = NAG_{j,0}(t) \\ & \dots\dots\dots \\ & NAG_{j+1,T_a-1}(t+1) = NAG_{j,T_a-2}(t) \end{aligned}$$

$$\begin{aligned} \mathbf{NAD}_{j,i}: \quad & NAD_{j,0}(t+1) = \gamma_r ND_j(t) \\ & NAD_{j+1,1}(t+1) = NAD_{j,0}(t) \\ & \dots\dots\dots \\ & NAD_{j+1,T_a-1}(t+1) = NAD_{j,T_a-2}(t) \end{aligned}$$

$$\begin{aligned}
\mathbf{NAH}_{j,i}: \quad & NAH_{j,0}(t+1) = \gamma_r NH_j(t) \\
& NAH_{j+1,1}(t+1) = NAH_{j,0}(t) \\
& \dots\dots \\
& NAH_{j+1,T_a-1}(t+1) = NAH_{j,T_a-2}(t) \\
\mathbf{NS}_i: \quad & NS_0(t+1) = NR_{T_r-1}(t) \\
& NS_1(t+1) = (1-\gamma_r)(1-\gamma_f)NS_0(t) + NAS_{0,T_a-1}(t) \\
& \dots\dots \\
& NS_{T_s-1}(t+1) = (1-\gamma_r)(1-\gamma_f)NS_{T_s-2}(t) + NAS_{T_s-2,T_a-1}(t) \\
\mathbf{NG}_i: \quad & NG_0(t+1) = (1-\gamma_r)\gamma_f \sum_{i=0}^{T_s-2} NS_i(t) \\
& NG_1(t+1) = (1-\gamma_r)(1-\gamma_g)NG_0(t) + NAG_{0,T_a-1}(t) \\
& \dots\dots \\
& NG_{T_g-1}(t+1) = (1-\gamma_r)(1-\gamma_g)NG_{T_g-2}(t) + NAG_{T_g-2,T_a-1}(t) \\
\mathbf{ND}_i: \quad & ND_0(t+1) = (1-\gamma_r)\gamma_g \sum_{i=0}^{T_g-2} NG_i(t) \\
& ND_1(t+1) = (1-\gamma_r)ND_0(t) + NAD_{0,T_a-1}(t) \\
& \dots\dots \\
& ND_{T_d-1}(t+1) = (1-\gamma_r)ND_{T_d-2}(t) + NAD_{T_d-2,T_a-1}(t) \\
\mathbf{NH}_i: \quad & NH_0(t+1) = NG_{T_g-1}(t) + NS_{T_s-1}(t) + \sum_{j=0}^{T_a-1} NAG_{T_g-1,j}(t) + NAS_{T_s-1,j}(t) \\
& NH_1(t+1) = (1-\gamma_r)NH_0(t) + NAH_{0,T_a-1}(t) \\
& \dots\dots \\
& NH_{T_h-1}(t+1) = (1-\gamma_r)NH_{T_h-2}(t) + NAH_{T_h-2,T_a-1}(t) \\
\mathbf{NR}_i: \quad & NR_0(t+1) = NH_{T_h-1}(t) + ND_{T_d-1}(t) + \sum_{j=0}^{T_a-1} NAH_{T_h-1,j}(t) + NAD_{T_d-1,j}(t) \\
& NR_1(t+1) = NR_0(t) \\
& \dots\dots \\
& NR_{T_r-1}(t+1) = NR_{T_r-2}(t)
\end{aligned}$$

This is modelled in PRISM by extending the original model in Figure 1 with additional states and updating the fomulae calculating the numbers of robots as described above. In this scenario, we assume

$$\gamma_f(t) = \left(1 - \frac{\alpha N_{foraging}(t)}{N}\right); \quad \gamma_g(t) = \left(1 - \frac{\alpha N_g(t)}{N}\right); \quad \gamma_r(t) = \frac{\alpha N_{active}(t)}{N}$$

where N_{active} is defined as the active robots in the arena, i.e. $N_{active}(t) = N - N_r(t)$.

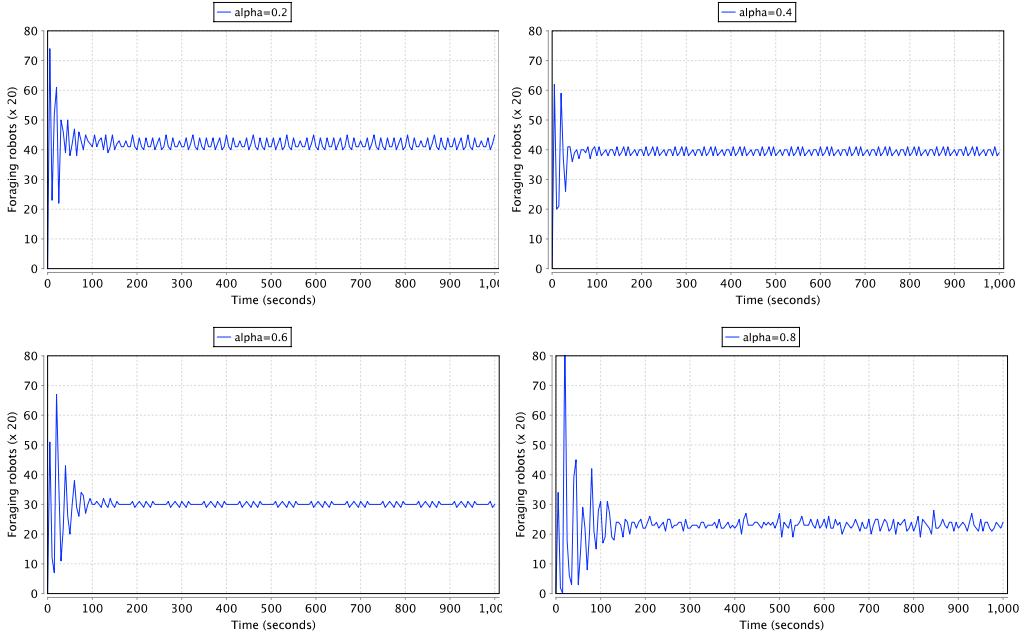


Figure 10: Number of foraging robots v.s. α , the density of arena for $N=2000$.

Figure 10 shows the number of foraging robots vs different α values for 2000 robots. The figure shows that if the density of the arena increases, the number of foraging robots decreases, because more robots move into avoidance states. We can observe that the average number of foraging robots are less than the case of without avoidance states. This is because some of the robots avoid from collision, and stop foraging.

5.3 Verification vs. Statistical Simulation

In this section, we present some statistics on the verification experiments we carried out in Section 5.2. Namely, we provide the resources used to construct the relevant model and to perform verification.

Some of the verification experiments in Section 5.2 can be approximated using statistical simulations, such as *Monte Carlo*. A statistical simulation works on a statistical sampling scheme, where the problem is solved using a set of randomly generated samples, and measuring what fraction of the random set satisfy a property [19].

Although statistical simulation can be useful in some cases, this method has some limitations. Verification (and model checking) explores *all* situations; whereas statistical simulation can explore *some* situations, and cannot observe *all* behaviours. In some critical scenarios, this might cause some problems. For example, the *safety* properties such as “it is guaranteed that something bad will never happen” need to be evaluated under all possible situations, which cannot be achieved by the simulation method.

The other limitation is that simulations need to be executed for a certain amount of time. On the other hand, we perform verification and model checking *unboundedly*. That is, the execution of a verification experiment should not necessarily be restricted by a time bound. Thanks to the underlying mathematical foundation, model checkers automatically construct a finite structure which is mathematically equivalent to the infinite run of the system.

Table 5: Space and time usage in model construction.

| model construction | | | | | |
|--------------------|---|--------|-------------|---------------|------------|
| Model | scenario | states | transitions | trans. matrix | time |
| 1 | random γ_f | 34,937 | 35,186 | 1,966,611 | 390 sec. |
| 2 | variable γ_f, γ_g | 10,221 | 10,418 | 362,545 | 1,814 sec. |
| 3 | variable γ_f, γ_g no resting timeout | 17,076 | 18,285 | 52,487 | 171 sec. |

Table 6: Comparison of space and time usages of verification and statistical simulation.

| Prop. | Model | verification | | | statistical simulation | | |
|-------|-------|-------------------|-----------|--------|------------------------|----------|--------|
| | | result | time | memory | result | time | memory |
| I | 1 | true | 37 sec. | 750 KB | true | 125 sec. | 162 MB |
| II | 1 | 305×10^3 | 2473 sec. | 101 MB | 301×10^3 | 127 sec. | 164 MB |
| III | 1 | 1.0 | 9276 sec. | 748 KB | N/A | N/A | N/A |
| IV | 2 | 1.0 | 5.8 sec. | 420 KB | 1.0 | 148 sec. | 158 MB |
| V | 2 | 0.97 | 67 sec. | 24 MB | N/A | N/A | N/A |
| VI | 3 | 0.60 | 4 sec. | 23 MB | N/A | N/A | N/A |
| VII | 3 | 1.0 | 131 sec. | 32 MB | N/A | N/A | N/A |

In Table 6, we compare the verification results with statistical simulations ⁴.

⁴We remind that experiments were run on a 2.4 Ghz Core 2 Duo computer with 6 GB RAM running Mac OS 10.5.7.

To perform the simulations, we use the *statistical model checking* environment of PRISM, which works very similar to the Monte Carlo method ⁵. The advantage of using PRISM's own simulation environment is that it allows us to use PCTL formulas to perform simulation, which cannot be done using Monte Carlo, or similar methods. This gives us the opportunity to compare verification and statistical simulation directly in the same model and using the same logical formulae.

In Table 6, the properties I, II, III, IV, V and VI are defined as follows (assume N is the total number of robots at any time, N_{init} is the initial number of robots, $N_{foraging}$ is the number of foraging robots, $En_{average}$ is the average energy, and $R = ?$ is the expected energy):

- I $P_{\geq 1}(N = N_{init})$
- II $R = ?[C \leq T]$
- III $P = ?\Box(R_{>0}[S])$
- IV $P = ?\Box(t \geq t_A \wedge N \geq k \Rightarrow N_{foraging} \geq n)$
- V $P = ?\Box(N \geq n \Rightarrow \text{true } U^{\leq t_A}(En_{average} \geq E))$
- VI $P = ?\Diamond(P_{\geq 1}\Box N_{foraging} \geq l\{n \leq N \leq m\})$
- VII $P = ?\Box(R = ?[C \leq T]\{N \geq m\} \geq 2 \times R = ?[C \leq T]\{N \geq n\})$

where $t_a = 100$ seconds, $T = 1000$ seconds, $E = 100 \times 10^3$, $n = 100$, $m = 200$, $k = 300$, $l = 50$. Note that the properties above were checked against the corresponding models in Table 5, where each model was constructed according to the parameter settings of the related scenario.

Despite the fact that verification is in general slower than simulation, Table 6 shows that the performance of the verification experiments we carried out is very promising (Note that the verification results are based on unbounded execution; whereas the simulation results are based on 1000 seconds of the execution of the system.). Even in some cases, verification performs better than simulation. Actually, the abstraction technique that we employ prevents the *state explosion* problem, and makes the resulting state space significantly less than the product space. We remind that in the microscopic approach the state space raised to the magnitude of

⁵We could actually implement a statistical simulation, such as the Monte Carlo method, using MATLAB. In this case, some of the functionalities of PRISM need to be simulated using some MATLAB features. For example, randomness of sampling can be obtained by choosing a suitable random set, and the probabilistic choice is simulated by comparing a random number output of the `rand` function with a probabilistic threshold.

10^{21} for as low as 6 robots (see Table 1). However, in the macroscopic approach, we could reduce the state space to the level of 10^5 states for much higher number of robots. This proves the effectiveness of our abstraction method.

Note also that the simulation method cannot handle some of the verification experiments, because the complex logical formulae, such as III, V, VI, VII, are not supported by the simulation tool (In Table 6, they are marked as N/A). This is actually another limitation of the statistical simulation.

5.4 Discussion

The size of a probabilistic model (i.e. the number of states/transitions) is critical to the efficiency of probabilistic verification on it, since both the time and memory required to perform verification are proportional to the model size. Actually, the complexity of model checking a PRISM model against a PCTL formula is *linear* in the size of the formula and *polynomial* in the size of the model [22]. Therefore, the complexity of the verification problems discussed in this paper is *polynomial* in the total number of states and transitions in the probabilistic transition systems.

This result suggests to use an efficient way of modeling. Unfortunately, it is very easy to create models that are extremely large. As discussed in Section 4, if we consider the behaviour of each robot individually, modeled as a transition system, and then consider the overall system behaviour, modeled by the product of all individual transition systems, we end up a huge state space. We observed that this makes the verification very difficult. Actually, Table 1 shows that having the product of transition systems rapidly increases the state space, i.e. the model size. Also, Table 2 shows that the verification becomes impossible after a certain size.

Although it has been proved that the complexity of the verification polynomially depends on the model size, the verification becomes intractable for this type of modeling because the model size explodes very quickly. Actually, the *state explosion* is a common problem in model checking. Therefore, our macroscopic approach provides a remedy for this problem. Here, instead of modeling the robots individually, we model the common behaviour in a single state machine. This method is feasible because all the individual robots exhibit same behaviour.

The advantage of this method is that the model size does not depend on the number of robots, because we only have one transition system. Since the model size, i.e. the number of states and transitions in the state machine, does not increase rapidly, the resulting state space is under control and the state explosion problem is prevented. This is indeed illustrated in Table 5, where the model sizes are significantly lower than the of Table 1, and unlike the microscopic case model sizes do not increase exponentially. Therefore, the verification becomes tractable.

We also remark that in Section 5.2 we have shown that verification can be used

in the analysis of various scenarios discussed above. Our aim here is not to show that one particular scenario is better than the others; we rather show that we can model a number of scenarios that may be of interest to swarm designers and in addition to providing simulations can *verify* relevant properties of these scenarios.

This technique can be applied to other scenarios as well. For example, obstacles and interference between robots can be considered. One possible approach for modelling this scenario is to take timeouts as a function of number of foraging robots. For example, when there are obstacles and interference, timeout for searching increases, that is, robots spends more time searching.

Finally, it is worth mentioning that we can also use performance metrics other than energy, such as number of objects collected per second, number of objects collected per unit of energy, foraging efficiency, etc.

6 Other Approaches

As we have seen already, the counting/population abstraction allows us to model large numbers of robots. However, the probabilistic elements within the original model are “smoothed” out to provide difference equations over discrete values. While this provides a very tractable approach, it is useful to then consider what can be done if we re-introduce probabilistic aspects into the counting/population model. In fact this allows us to develop many, quite complex, formulations but we will just outline two such in the sections below.

6.1 Uncertainty in the Counting Abstraction

When we model the probabilistic behaviour of a robot, can we be certain that our probability estimates are correct? Surely there is always the possibility that they miss important aspects? And then, when we move to the counting/population abstraction, are all aspects taken care of? Concerning this second problem, imagine that we have a transition from state S_1 to S_2 which has a probability of 0.000001. When we move to the counting abstraction then this transition effectively rounds to zero, even with as many as 10000 robots. So, we can re-introduce some uncertainty in both the above cases by having probabilistic transition even within our counting abstraction. Consider Figure 11.

Here the state transitions on the left-hand side represent a straightforward counting abstraction with N robots moving from state S_1 to state S_2 at every step. However, if the original modelling is uncertain, or if we have lost some possibilities due to rounding real numbers to integers, then we might wish to re-introduce various possibilities. So, we might adapt the state transitions to be like those on the right-

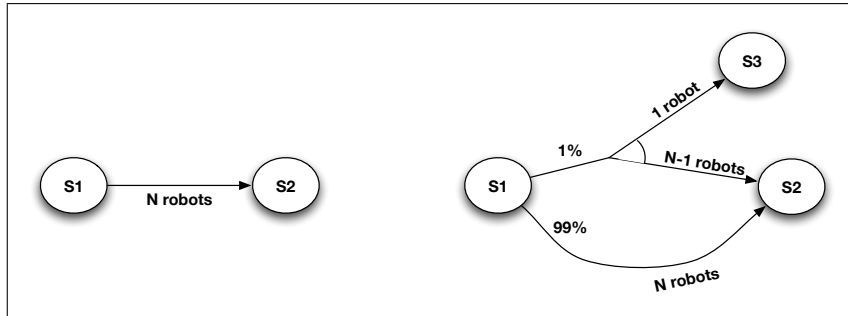


Figure 11: Introducing Probabilistic Transitions into a Counting/Population Model.

hand side where there is now a small (here just 1%) chance that $N - 1$ robots will move as above, but that 1 robot will also move to a different state (S3 in Figure 11). In such a way we might re-introduce some more realistic extreme conditions into the overall modelling.

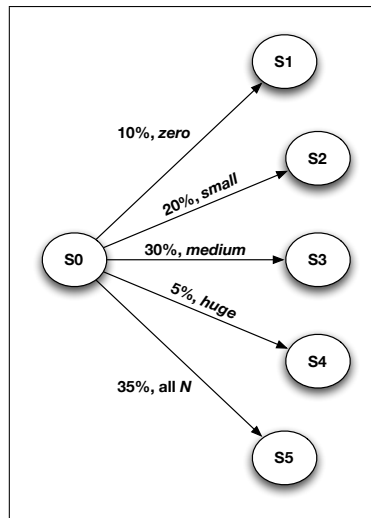


Figure 12: General Population Abstractions.

6.2 Abstracting from the Abstraction

As we move to a counting/population abstraction, we move from considering detailed probabilities to considering the exact numbers of robots that are in certain states. But, is it really important to know *exactly* how many robots are in each

state? Possibly all we need to know is that there are *zero*, a *small* non-zero number, a *medium* sized number, a *large* number, or *all* the robots in a certain state. This would lead us on to some model such as in Figure 12 where there are a number of (probabilistic) possibilities, but where each possibility only has a broad categorisation of the number of robots. This, then, allows us to reason about general population movement in a probabilistic context without being concerned over exact robot numbers. This can be useful when our swarm population is *very* large.

7 Related Work

Related work falls into two main areas: comparison of our results with that of other foraging robot research, and the application of formal methods, in particular model checking, to robot swarms.

7.1 Foraging Robots

Foraging robots have been studied in a number of other papers, for example [32, 30, 33, 34, 35]. The foraging robot scenario we focus on here is from [34]. In that paper the authors' develop a probabilistic model for a swarm of foraging robots. They then consider the net energy of the swarm and the number of robots searching, resting and homing using the probabilistic model and a simulation. However, as stated previously, we here ignore avoidance. Note that we could add states to represent avoidance to the transition system represented in Figure 1 and probabilities that the avoidance state is reached to match the model in [34]. In Section 5 of that paper the energy of the swarm over time is considered for different resting times (Figure 8 and Figure 9) for the probabilistic model and a simulation. This is similar to the experiments we have carried out. In that paper it is shown that there is an optimum resting time to maximise swarm energy given the other swarm parameters.

Figure 9 of [34] plots the number of robots searching, resting and homing for a particular value of the resting parameter. This shows that, in the probabilistic model, the number of robots varies at first before reaching a steady state. The simulations vary over time but deviate above and below the values predicted by the model. In our paper, Figure 8 plots the number of robots over time related to differing probabilities of finding food and food energy factor. These both show the initial oscillation followed by an eventual steady state, a pattern that was also observed for the probabilistic model in [34].

Foraging robots are also considered in [35] where the authors examine simple mechanisms for adjusting the ratio of resting to foraging robots. These mechanisms

include successful food retrieval by the robot itself, successful food retrieval by other robots and collisions with other robots. The authors find that the use of shared information about food retrieval by team mates achieves the highest net energy and the fastest change of ratio of resters to foragers when the food density changes. This is analogous to the experiments we carried out in Section 5.2.3. Since we model the robots as a swarm we cannot isolate the parameter of successful food retrieval by the robot itself. Further, as we do not model avoidance, we cannot consider collisions with other robots. However, we can examine the number of robots who have been successful or unsuccessful in retrieving food by looking at the number of robots in the DEPOSITING and HOMING states. In this case, rather than adjusting time-outs for searching and resting (T_s , T_r respectively) we can change the probability of leaving the RESTING state depending on the number of robots in the DEPOSITING and HOMING states.

In [30] a foraging robot scenario similar to that described above is used. Here, robots have a probability, P , of leaving the nest and P can be changed dynamically depending on whether the robot itself has had previous success or failure in finding food; success increases P while failure decreases it. Two hypotheses are tested. First, whether the efficiency of the swarm is increased by dynamically changing P (self organisation) and, second, whether two classes of robot emerge: one specialised in foraging and one specialised in resting with high and low values of P respectively (task allocation). Experiments with real robots have confirmed both of these hypotheses. In our paper, Section 5.2.3 highlights scenarios where the probability of leaving the nest is dependent on robots having found food or not. As we model the swarm of robots rather than individual robots we cannot directly retrieve the probability of an individual robot leaving the nest or consider the specialisation of robots (hypothesis 2 of [30]).

There are numerous other papers that discuss foraging of robot swarms including [32], which develops a mathematical model for a group of foraging robots and considers the effects of interference on the performance of the swarm. In addition, similar works have been carried out, which focus on different aspects of robots, such as coordination [17], motion planning and control [31], high-level behaviour [26], etc.

7.2 Formal Verification of Robot Swarms

We here highlight previous work in which the formal verification of robot swarms has been considered, typically using model checking or deductive techniques for temporal logics. We begin by examining our previous work in this area. We have formally specified the alpha algorithm [37] for a wireless connected swarm using temporal logics. Each robot has range limited wireless communication and can

only receive and broadcast messages to robots within range.

In [41] we specified this swarm algorithm using propositional linear-time temporal logic. In [9] this temporal specification of swarm algorithms was used to explore ways to generate implementations from a formal specification.

In [2] we considered the state transition system for a swarm of foraging robots, from [35] and represented this using both propositional and first-order temporal logics. A number of properties are then verified using a resolution based theorem prover for these logics. Whilst this models the transition system for *each* robot, similar to that in Section 4 of this paper, it focuses on the state robots are in rather than specific location or movement details. In the propositional setting we could only represent a small number of robots due to the state explosion problem also encountered in this paper. In the first-order case, the robots are represented using variables and, although this allows us to represent an *infinite* number of robots, syntactic restrictions of the monodic first-order temporal logic used in the prover added further representational limitations.

In [27, 28], Kloetzer and Belta adopt a model checking approach to analysing the motion of robot swarms. A hierarchical framework is suggested to abstract away from the many details of the problem. First, a continuous abstraction is used to capture the main features of the swarm’s position and size (the example considered uses the centroid and variance of robot positions to achieve this). Next this continuous abstraction is abstracted further, providing a discrete model to which model checking can then be applied. That approach differs in a number of ways to this paper. Firstly it concentrates on robot motion and behaviour such as obstacle avoidance, swarm cohesion, and collision avoidance. Our work has abstracted away from considerations of robot location in that we do not represent details of the physical space. Thus, while Kloetzer and Belta concentrate on detailed movement, navigation, and collision avoidance of robots, we concentrate on the high-level behavioural choices of the robots, rather than their low-level navigation. We also consider much larger numbers of robots within a swarm and highlight the probabilistic aspects of their behaviours.

A paper related to that of Kloetzer and Belta is that of Fainekos et al. [18]. Again, the emphasis here is on the analysis of robot motion rather than swarm behaviours. This approach again involves the production of a discrete representation (specifically, a finite state transition system) of the continuous space of movement. A model checker is used to produce traces that satisfy particular properties (e.g., visiting regions in a particular order, eventually visiting a region but avoiding other regions on the way). These are then used to produce a continuous movement plan whilst maintaining the required property. The key difference between that work and ours is again that they focus on motion and do not mention robot swarms. A related approach is given in [25] which, after assessing a number modelling and

verification techniques, focusses on model checking. There the underlying transition system has states that relate to the robots' behaviour, but only a small number (three) of robots is considered.

In [8], Casadei and Viroli investigate “collective sorting”, an online distributed algorithm with emergent properties, using PRISM. They model individual elements and then take a product of finite-state structures to produce the overall state-space. Importantly, they use approximation techniques to avoid state-space explosion. Although not directly addressing robot swarms, this work also highlights how probabilistic model checking can be very useful in analysing the emergent properties on self organising systems. In contrast, we tackle a realistic robot architecture, address large numbers in a swarm (through the population-based approach) and verify important swarm coherence properties.

Other formalisms have been considered to specify and verify aspects of realistic robot swarms. In [39], Rouff et al. compare a number of formal methods for representing and verifying part of the Autonomous Nano Technology Swarm (ANTS) mission aimed at sending small swarms of spacecraft to study the asteroid belt. As a result of this, in [38] four formal methods were selected, namely Communicating Sequential Processes (CSP), Weighted Synchronous Calculus of Communicating Systems (WSCCS), X-Machines and Unity Logic. These are proposed for use alongside techniques from *agent-oriented software engineering*. While the authors do not apply these techniques, they conclude that there is a need to develop new formal techniques alongside specialised sets of models and software processes based on a number of formal methods and other areas software engineering. Importantly, they do not tackle specific swarm architectures, carry out full verification experiments, or address the probabilistic aspects inherent in any practical undertaking.

Finally, in [33], Lerman et al. use distributed stochastic processes to model swarm robots. An individual robot controller is used to develop a probabilistic model and is applied it to the area of “stick pulling”, foraging and aggregation. The modelling is macroscopic, i.e. it directly describes the collective behaviour of the swarm and so bears a relationship to our work. Their models are compared with simulations and experiments on real robots with good agreement between the models and simulations or experiments. The paper [36] also develops a stochastic approach.

8 Concluding Remarks

In this paper we have taken a probabilistic state transition system for existing foraging swarm robots from [34] and used it as the basis for verification of global

swarm behaviour using the PRISM model-checker. Rather than instantiating such a transition system for each robot and performing local verification, we primarily adopt a macroscopic (alternatively termed “population-based” or “counting abstraction”) approach, where we represent the whole swarm using one transition system calculating the number of robots in each state (based on a combination of the number of robots in the previous state and the probability that robots change state). This allows us to analyse the global foraging robot scenario for a number of parameters. In particular, we investigate the changes to swarm energy relating to changing the probability of finding food and differing resting timeouts. We also experimented with variable probabilities including the probability of moving from resting to searching states and the probability of grabbing.

It is important to note that the use of this counting abstraction allows us to analyse the behaviour of large numbers of robots — otherwise, we would not have been able to carry out local verification even with tens of robots composed together. Using this approach we can formally verify that certain behaviours will *always* happen. This is not something that can easily be achieved with either simulation or testing. Our overall point is that formal verification of this form provides an essential tool for the swarm algorithm designer, especially if reliable and predictable swarm behaviour is required. Finally, we considered several additional abstractions incorporating uncertainty into the counting abstraction and moving away from a calculation of the actual robot numbers in each state to a qualitative notion of population size in each state.

While foraging robots are taken as a particular case study, the approach is viable for any variety of robot whose essential control can be characterised in a finite-state form. Thus, while our states are of the form ‘Resting’, ‘Searching’, ‘Homing’, etc., we could just have easily have used any other finite set of states that the robot might be in. For example, a swarm of *excavation* robots might have states such as ‘Digging’, ‘Waiting’ and ‘Stuck’, while for more cooperative robots we might have ‘Searching’, ‘Coordinated’, and ‘SwarmConnected’. Between all such states, as long as there are straightforward probabilistic values, based on a uniform distribution, and as long as any timing delays are discrete and finite then we believe the approach describe here could be used for verification in such scenarios. Our collaboration with robotics experts confirms that many simple swarm robots can indeed be modelled in this way [14].

While this paper describes the essential approach, work is under way to develop a more *user friendly* front-end, as this interface will make it easier for designers to provide PRISM models and PCTL properties. This involves developing a toolbox that will allow simple robot algorithms to be described and then expanded for input to PRISM. A beta version of such a front-end, called “SwarmChecker”, was developed [42], and it is under development to support more features.

Our future work in this area includes extending the analysis presented here to incorporate further robot interactions, and develop PRISM models for other domains such as “stick pulling”.

Acknowledgements.

This work was partially supported in the UK by EPSRC research project EP/F033567. The authors would also like to thank Alan Winfield and Jin Sa for useful discussions and clarifications.

References

- [1] Special Issue on Swarm Robotics. *Swarm Intelligence*, 2(2-4):69–72, 2008.
- [2] A. Behdenna, C. Dixon, and M. Fisher. Deductive Verification of Simple Foraging Robotic Behaviours. *International Journal of Intelligent Computing and Cybernetics*, 2(4):604–643, 2009.
- [3] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL — a Tool Suite for Automatic Verification of Real-Time Systems. In *Proc. Workshop on Verification and Control of Hybrid Systems III*, number 1066 in LNCS, pages 232–243. Springer Verlag, 1995.
- [4] G. Beni. From Swarm Intelligence to Swarm Robotics. In *Proc. International Workshop on Swarm Robotics (SAB)*, volume 3342 of LNCS, pages 1–9. Springer, 2005.
- [5] E. Bonabeau, M. Dorigo, and G. Theraulaz. Swarm Intelligence: From Natural to Artificial Systems. *Journal of Artificial Societies and Social Simulation*, 4(1), 2001.
- [6] R. E. Bryant. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.
- [7] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation*, 98(2):142–170, 1992.
- [8] M. Casadei and M. Viroli. Using Probabilistic Model Checking and Simulation for Designing Self-Organizing Systems. In *Proc. ACM Symposium on Applied Computing (SAC)*, pages 2103–2104. ACM, 2009.

- [9] D. Chen. *A Simulation Environment for Swarm Robotic System based on Temporal Logic Specifications*. Master’s thesis, University of the West of England, November 2005.
- [10] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A New Symbolic Model Verifier. In *Proc. 11th International Conference on Computer Aided Verification (CAV)*, volume 1633 of *LNCS*, pages 495–499. Springer, 1999.
- [11] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
- [12] E. M. Clarke and O. Grumberg. Avoiding The State Explosion Problem in Temporal Logic Model Checking. In *ACM Symposium on Principles of Distributed Computing*, pages 294–303. (PODC), 1987.
- [13] G. Delzanno. Constraint-based Verification of Parametrized Cache Coherence Protocols. *Formal Methods in System Design*, 23(3):257–301, 2003.
- [14] C. Dixon, A. Winfield, and M. Fisher. Towards temporal verification of emergent behaviours in swarm robotic systems. In *Proc. Towards Autonomous Robotic Systems*, pages 336–347. (TAROS), 2011.
- [15] M. Dufлот, M. Kwiatkowska, G. Norman, and D. Parker. A Formal Analysis of Bluetooth Device Discovery. *International Journal of Software Tools and Technology Transfer*, 8(6):621–632, 2006.
- [16] E. A. Emerson and J. Y. Halpern. “sometimes” and “not never” revisited: on branching versus linear time temporal logic. *J. ACM*, 33:151–178, 1986.
- [17] J. M. Esposito and M. Kim. Using formal modeling with an automated analysis tool to design and parametrically analyze a multirobot coordination protocol: A case study. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 37(3):285–297, 2007.
- [18] G. Fainekos, H. Kress-Gazit, and G. Pappas. Temporal Logic Motion Planning for Mobile Robots. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 2020–2025. IEEE Computer Society Press, 2005.
- [19] A. Fehnker and P. Gao. Formal verification and simulation for performance analysis for probabilistic broadcast protocols. In *Proc. 5th International Conference on Ad-Hoc, Mobile, and Wireless Networks (ADHOC-NOW’06)*, volume 4104 of *LNCS*, pages 128–141. Springer, 2006.

- [20] M. Fisher, D. Gabbay, and L. Vila, editors. *Handbook of Temporal Reasoning in Artificial Intelligence*, volume 1 of *Advances in Artificial Intelligence*. Elsevier Publishers, 2005.
- [21] H. Hamann. *Space-Time Continuous Models of Swarm Robotic Systems*. Springer, 2010.
- [22] H. Hansson and B. Jonsson. A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computing*, 6:102–111, 1994.
- [23] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A Tool for Automatic Verification of Probabilistic Systems. In *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.
- [24] G. J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.
- [25] S. Jeyaraman, A. Tsourdos, R. Zbikowski, and B. White. Kripke Modelling Approaches of a Multiple Robots System with Minimalist Communication: A Formal Approach of Choice. *International Journal of Systems Science*, 37(6):339–349, 2006.
- [26] B. Johnson and H. Kress-Gazit. Probabilistic analysis of correctness of high-level robot behavior with sensor error. In *Proceedings of Robotics: Science and Systems*. USA, 2011.
- [27] M. Kloetzer and C. Belta. Hierarchical Abstractions for Robotic Swarms. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 952–957. IEEE Computer Society Press, 2006.
- [28] M. Kloetzer and C. Belta. Temporal Logic Planning and Control of Robotic Swarms by Hierarchical Abstractions. *IEEE Transactions On Robotics*, 23:320–330, April 2007.
- [29] S. Konur, C. Dixon, and M. Fisher. Formal Verification of Probabilistic Swarm Behaviours. In *Proc. 7th International Conference on Swarm Intelligence (ANTS)*, volume 6234 of *LNCS*, pages 440–447. Springer, 2010.
- [30] T. H. Labella, M. Dorigo, and J.-L. Deneubourg. Efficiency and Task Allocation in Prey Retrieval. In *Biologically Inspired Approaches to Advanced Information Technology*, volume 3141 of *LNCS*, pages 274–289. Springer, 2004.

- [31] M. Lahijanian, J. Wasniewski, S. B. Andersson, and C. Belta. Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees. In *IEEE International Conference on Robotics and Automation*, pages 3227–3232. ICRA, 2010.
- [32] K. Lerman and A. Galstyan. Mathematical Model of Foraging in a Group of Robots: Effect of Interference. *Autonomous Robots*, 13(2):127–141, 2002.
- [33] K. Lerman, A. Martinoli, and A. Galstyan. A Review of Probabilistic Macroscopic Models for Swarm Robotic Systems. In *Proc. International Workshop on Swarm Robotics (SAB)*, volume 3342 of *LNCS*, pages 143–152. Springer, 2005.
- [34] W. Liu, A. Winfield, and J. Sa. Modelling Swarm Robotic Systems: A Study in Collective Foraging. In *Proc. Towards Autonomous Robotic Systems*, pages 25–32. (TAROS), 2007.
- [35] W. Liu, A. Winfield, J. Sa, J. Chen, and L. Dou. Strategies for Energy Optimisation in a Swarm of Foraging Robots. In *Proc. 2nd International Workshop on Swarm Robotics (SAB)*, volume 4433 of *LNCS*, pages 14–26. Springer, 2007.
- [36] A. Martinoli, K. Easton, and W. Agassounon. Modeling Swarm Robotic Systems: A Case Study in Collaborative Distributed Manipulation. *International Journal of Robotics Research*, 23(4):415–436, 2004.
- [37] J. Nembrini, A. F. T. Winfield, and C. Melhuish. Minimalist Coherent Swarming of Wireless Connected Autonomous Mobile Robots. In *Proc. 7th International Conference on Simulation of Adaptive Behavior (ICSAB)*, pages 373–382. MIT Press, August 2002.
- [38] C. Rouff, M. Hinchey, J. Pena, and A. Ruiz-Cortes. Using Formal Methods and Agent-Oriented Software Engineering for Modeling NASA Swarm-Based Systems. In *Proc. International Swarm Intelligence Symposium (SIS)*, pages 348–355. IEEE Computer Society Press, 2007.
- [39] C. Rouff, A. Vanderbilt, W. Truszkowski, J. Rash, and M. Hinchey. Verification of NASA Emergent Systems. In *Proc. 9th IEEE International Conference on Engineering Complex Computer Systems Navigating Complexity in the e-Engineering Age*, pages 231–238. IEEE Computer Society Press, 2004.
- [40] W. Visser, K. Havelund, G. Brat, and S. Park. Model Checking Programs. In *Proc. 15th IEEE International Conference on Automated Software Engineering (ASE)*, pages 3–12. IEEE Computer Society Press, 2000.

- [41] A. Winfield, J. Sa, M.-C. Fernández Gago, C. Dixon, and M. Fisher. On Formal Specification of Emergent Behaviours in Swarm Robotic Systems. *International Journal of Advanced Robotic Systems*, 2(4):363–370, December 2005.
- [42] F. Zhang. Swarmchecker: A robot swarm front-end for prism. Master’s thesis, Department of Computer Science, University of Liverpool, 2010.

APPENDIX

We have proposed using a probabilistic model checker to investigate temporal and probabilistic properties of a particular probabilistic model. This can easily be applied to other probabilistic models. States in the probabilistic model correspond to one or more states in the PRISM model which are represented by variables. Conditions such as exiting a state S within n timesteps can be modelled using PRISM by splitting S into n states S_0, \dots, S_{n-1} . Variables can be declared using the syntax

$$s : [0..9] \textit{ init } 0;$$

which says that s is an integer variable with values between 0 and 9. Transitions can be modelled using the following syntax

$$\square s = 0 \rightarrow 0.3 : (s' = 1) + 0.7 : (s' = 2);$$

stating that if the variable s has value 0 then the value of s will be 1 in the next moment with probability 0.3 and it will be 2 at the next moment with probability 0.7. The uniform probabilities in the scenarios we considered are calculated using this syntax.

To model a system using a cross product of individual probabilistic model we define a **module** in PRISM for each probabilistic model and instantiate it multiple times. For the counting abstraction approach only one module is required but we need to add calculations (using the keyword **formula**) to define how the number of robots in each state are calculated.

The variables in Section 5.1 are declared as states. Due to the space restrictions, we can only provide some part of the PRISM code. Below we show how the variable \mathbf{NG}_i is expressed in PRISM, where we take $i = 5$ (Note that we do not

include the end points of the timeouts).

$$\begin{aligned}
(N_G0' &= \text{ceil}(\text{gamma}_f * N_S0) + \text{ceil}(\text{gamma}_f * N_S1) + \\
&\quad \text{ceil}(\text{gamma}_f * N_S2) + \text{ceil}(\text{gamma}_f * N_S3))\& \\
(N_G1' &= \text{floor}((1 - \text{gamma}_g) * N_G0))\& \\
(N_G2' &= \text{floor}((1 - \text{gamma}_g) * N_G1))\& \\
(N_G3' &= \text{floor}((1 - \text{gamma}_g) * N_G2))\& \\
(N_G4' &= \text{floor}((1 - \text{gamma}_g) * N_G3))\&
\end{aligned}$$

Note that the state values must be integers. We therefore round up the real numbers. We deal with the other variables similarly. We now denote how gamma_f (γ_f) is declared in PRISM. Below we consider the case that gamma_f is variable (see Section 5.2.2)

$$\text{formula } \text{gamma}_f = 1 - (N_foraging * \text{alpha}/N);$$

where $\text{alpha} \in \{0, 0.1, \dots, 1\}$ is the density factor, $N_foraging$ is the total number of foraging robots, and N is the total number of robots. The other cases of the probabilities are dealt with similarly.

We augment the probabilistic models developed in PRISM costs or reward structures to calculate quantitative measures relating to model behaviour. For example, the energy of the swarm is implemented as below:

rewards

$$\text{true} : (E_food * N_D4 - E_r * N_R - E_h * N_H - E_s * N_S - E_g * N_G)/1000;$$

endrewards

where $E_R = E_R0 + E_R1 + E_R2 + E_R3 + E_R4$ (resp. E_H , E_S and E_G). The above reward structure calculates the instantaneous energy of the swarm. The expected energy at a state is the the sum of all the individual rewards. Note that we divided the result by 1000, because we want to have the result per 1000 units.