Model Checking the Ant Colony Optimisation

Lucio Mauro Duarte, Luciana Foss, Flávio Rech Wagner, Tales Heimfarth

Model Checking the Ant Colony Optimisation

Lucio Mauro Duarte[1] , Luciana Foss[2], Flávio Rech Wagner[1], Tales Heimfarth[3]

[1] Institute of Informatics, Federal University of Rio Grande do Sul – Brazil
[2] Institute of Physics and Mathematics, DINFO, Federal University of Pelotas – Brazil
[3] Dep. of Computer Science, Federal University of Lavras - Brazil
{lmduarte,flavio}@inf.ufrgs.br, luciana.foss@ufpel.edu.br, tales@dcc.ufla.br

Abstract. We present a model for the travelling salesman problem (TSP) solved using the ant colony optimisation (ACO), a bio-inspired mechanism that helps speed up the search for a solution and that can be applied to many other problems. The natural complexity of the TSP combined with the self-organisation and emergent behaviours that result from the application of the ACO make model-checking this system a hard task. We discuss our approach for modelling the ACO in a well-known probabilistic model checker and describe results of verifications carried out using our model and a couple of probabilistic temporal properties. These results demonstrate not only the effectiveness of the ACO applied to the TSP, but also that our modelling approach for the ACO produces the expected behaviour. It also indicates that the same modelling could be used in other scenarios.

Keywords: ant colony optimisation, self-organisation, emergent behaviour, probabilistic model checking.

1 Introduction

Biologically inspired algorithms (or simply *bio-inspired algorithms*) [1] are methods that use mechanisms that resemble behaviours observed in nature, such as food search, evolution and insect swarming. These mechanisms define solving strategies that make applications more robust, flexible and scalable. They have been successfully applied to several problems, such as sensor networks [2][3].

All bio-inspired algorithms present a decentralised control and are composed by several, simple components that act autonomously and interact with each other. The overall behaviour of the system is a result of the interactions between its components and their autonomous decisions, based on environment conditions. Therefore, bio-inspired algorithms combine self-organisation and emergence. *Self-organisation* [4] describes the ability of a system to dynamically modify its internal structure in an autonomous manner. Hence, components automatically adapt to changes without any external intervention. *Emergence* [5] means that the system's behaviour, rather than being simply the sum of the behaviours of its components, *emerges* from local interactions between these components. Systems that contain self-organisation and emergence are called *self-organising emergent systems* (SOES) and are considered one of the most promising answers to the development of massive distributed systems

with decentralised control [6]. However, no appropriate support is provided by current software engineering techniques for the development of such systems.

The main problem in developing an SOES is the dynamic changes in its structure, which makes it difficult to analyse the system using traditional approaches, which require predicting all possible behaviours. Furthermore, even though components of the system and their interactions are quite simple, the great number of possibilities of interactions increases the system's overall complexity.

In this paper we investigate how to provide some guarantee that a system involving a bio-inspired algorithm – thus characterising an SOES - exhibits the expected behaviour. Though simulation is the usual technique for the analysis of these systems, we present an approach based on *probabilistic model checking* [7], which is a technique that extends the traditional model checking [8] by allowing property specification and model analysis to consider probabilities and timing information. The reason for this choice is the need of checking not only qualitative properties (i.e., properties that evaluate as either "true" or "false") but mainly *quantitative properties* regarding issues such as reliability, performance and resource usage. Moreover, modelling bio-inspired mechanisms requires different abstractions to describe the dynamics of the system and possible changes in its structure over time.

We describe a modelling strategy for a bio-inspired algorithm based on the mechanism of *pheromones*. We advocate that, since bio-inspired algorithms are naturally self-organising mechanisms, if there is a well-defined way of model-checking them, then SOESs in general could be more easily modelled and verified.

To evaluate our idea, we have developed an example involving the *travelling salesman problem* (TSP) [9] using the *ant colony optimisation* (ACO) [10] to drive the solution for the problem in a small scenario. Hence, our main goals with this study were: to propose a way of modelling the bio-inspired algorithm, to apply it to a model of the TSP and to show, through the verification of some properties, the correctness of the modelling, which means that the mechanism indeed eventually leads the system to exhibit the expected behaviour. We demonstrate how we have modelled the TSP and the ACO as a *discrete-time Markov chain* (DTMC) in the *Probabilistic Symbolic Model Checker* (PRISM) [11] and present a couple of properties specified using *probabilistic computational tree logic* (PCTL) [12] that we have checked. The results indicate that the pheromone mechanism successfully eventually leads to a behaviour that complies with requirements for solving the problem, thus demonstrating the correct modelling of the bio-inspired algorithm.

The remaining of this paper is organised as follows: Section 2 presents background information on the subject of this work; Section 3 describes how we have modelled and verified the TSP-ACO experiment and the results obtained; Section 4 presents a discussion about some related work; and Section 5 contains the conclusions.

2 Background

This section presents the basic ideas related to the problem and to the bio-inspired algorithm. It also briefly describes concepts related to probabilistic model checking.

2.1 The ACO Applied to the TSP

The *travelling salesman problem* (TSP) [9] is a classic and well-studied problem in Theoretical Computer Science that can be used to formalise a number of real-world problems. It consists in finding the shortest Hamiltonian circuit in a fully-connected graph $G=(N,E)$, where N represents a set of n cities and E describes the set of routes between pairs of cities. Each route (i,j) in E is assigned a cost $cost(i,j)$, describing the distance between cities i and j. Thus, the total cost of a certain tour $t=(c_0,c_1,...,c_n,c_0)$ is given by the sum in (1), which represents the sum of the costs of every route included in the tour. Therefore, the shortest path[1] would be the one with the lowest total cost.

$$\sum_{i=0}^{n-1} \text{cost}(\text{i},(\text{i}+1) \bmod \text{n}) . \qquad (\mathbf{1})$$

One way proposed to help solve the TSP is by the *ant colony optimisation* (ACO) [10], which is an algorithm that simulates the behaviour of ants looking for food sources. They mark the paths they take by leaving a trail of *pheromone*, which is a natural hormone. When choosing paths, ants take into account the pheromone concentration, which indicates paths more often taken by other ants. Thus, paths leading to food sources close to the nest tend to concentrate higher levels of pheromone and become more attractive to other ants. After some time, the pheromone slowly evaporates, ensuring that paths that do not lead to good food sources (i.e., are not frequently used) will become less attractive over time. Therefore, the pheromone mechanism provides all the information ants need to choose paths and achieve the necessary organisation without external intervention. This means that the behaviour of the whole colony emerges from the interactions through pheromone.

2.2 Probabilistic Model Checking

As in any other SOES, the complexity of the ACO lies not on the behaviour of each component (ant) alone, but on the difficulty of predicting the behaviours that can emerge from local interactions. In order to provide guarantees of some sort about the emergence of a specific behaviour, we can identify two possible approaches: simulation or formal verification. *Simulations* of an abstract system model can be used to drive design choices until the required quality properties are obtained. However, system analysis based on simulation does not provide sound guarantees for the engineering of complex systems such as ACO because it is based only on approximations. In contrast to simulation, *formal verification* techniques, such as *model checking* [8], can provide precise results about the system real behaviour, at the cost of requiring more accurate abstractions.

Probabilistic model checking [7] is a model checking technique more tailored for the analysis of SOESs, as it provides means of dealing with systems that exhibit probabilistic or stochastic behaviour. It mainly differs from traditional model checking in that it involves additional information on probabilities or timing of

[1] We use the term *tour* to represent the traversal of the graph and the term *path* to describe a sequence of cities visited during a tour.

transitions between states. Properties can involve calculations of the probability of certain events occurring during the execution of the system.

There are several commonly used model representations for probabilistic and stochastic systems, most of them based on Markov chains. In Markov chains, transitions between states depend on some probability distribution, where only the current state of the system influences the probability of the next transitions. A particular type of Markov chain is a *discrete-time Markov chain* (DTMC), which is represented by a transition system that defines the probability of moving from one state to another by applying discrete-time steps. The temporal logics *probabilistic computation tree logic* (PCTL) [12] is used to specify properties of a DTMC. PCTL extends the temporal logic CTL [13] with discrete time and probabilities.

One of the most used tools regarding probabilistic model checking is the *Probabilistic Symbolic Model Checker* (PRISM) [11]. Amongst some other features, it supports the specification of PCTL properties and the creation of DTMCs described using a simple, state-based language. It also provides an environment for checking the properties against the models using either simulation or verification.

3 Verification of the TSP with ACO

This section presents the results of our modelling of the ACO and how this mechanism was introduced in the TSP model to help find a solution using self-organisation and emergence. We also discuss how we specified properties based on these requirements and verified that they hold in the model.

3.1 Modelling

We modelled the problem using the PRISM language [11] to describe the symmetric version of the TSP (where routes have the same cost on both ways). The graph was composed by 4 cities, which is the minimum number of vertices necessary to introduce some complexity to the problem. To simplify the modelling, we fixed city number 1 as the initial city and modelled the solution with only one ant. This way, rather than having multiple ants travelling in parallel, there was only one ant repeatedly traversing the graph. Therefore, each tour of this sole ant represents the behaviour of a different ant, thus simulating the behaviour of several ants. This abstraction reduces the complexity of the problem but does not affect the analysis results as the effect of the pheromone mechanism works exactly in the same way.

Although we followed the original ACO algorithm [10], calculating the probabilities of paths based on both costs and desirability (amount of pheromone), we used different formulas to simplify the modelling. Our desirability component, called *preference*, refers to the amount of pheromone associated to each route between two cities, which ranges from 1 (*MIN_PREF*) to 10 (*MAX_PREF*). All routes are initialised with *MIN_PREF* and, after a tour, pheromone is deposited only on edges that are part of the path taken, causing preferences to be updated. The local updated preference value after a tour is calculated as presented in (2).

$$p_{ij}' = \min(p_{ij} + inc\_factor(tot\_dist), MAX\_PREF)). \quad (2)$$

In the formula, p_{ij} denotes the preference of route *(i,j)* and *tot_dist* is the sum of the costs of all routes comprising the most recent tour. Function *inc_factor* determines by how much the preference of a route will be increased depending on the total cost of the complete path. It assigns an increase value to paths according to their length group (short, mid-length or long). The group which a path belongs to is determined by the assignment of a cost to each route, which defines a scenario for the TSP.

The local update of routes causes a global update of probabilities. The probability of taking a certain route *(i,j)* is given by (3), where *N_CITIES* defines the number of cities involved, and *visited_cities* is the set of cities already visited in the current tour.

$$prob_{ij} = \frac{p_{ij}}{\sum_{k=1}^{N\_CITIES} p_{ik}} \; s.t. \; k \in \{1,...,N\_CITIES\} - visited\_cities. \quad (3)$$

Considering a scenario with 4 cities, the probability of taking, for instance, route (1,2) when starting the tour in city 1 is given by (4).

$$prob_{12} = \frac{p_{12}}{p_{12} + p_{13} + p_{14}}. \quad (4)$$

Note that the formula guarantees that the probabilities of all possible routes from a city add up to 1. The code below presents the PRISM model of the TSP-ACO:

```
1  dtmc
2  const int N_CITIES = 4; const int MIN_PREF = 1;
3  const int MAX_PREF = 10;
4  const int D12 = 1; const int D23 = 8; const int D34 = 2;
5  const int D14 = 4; const int D13 = 4; const int D24 = 3;
6  const int MAX_DIST=(D12+D23+D34+D14+D13+D24); const double EVAP_RATE;
7  const int N_CYCLES;
8  global cont : [0..N_CYCLES] init 0;
9  global p12 : [MIN_PREF..MAX_PREF] init MIN_PREF;
10 global p13 : [MIN_PREF..MAX_PREF] init MIN_PREF;
11 global p14 : [MIN_PREF..MAX_PREF] init MIN_PREF;
12 global p23 : [MIN_PREF..MAX_PREF] init MIN_PREF;
13 global p24 : [MIN_PREF..MAX_PREF] init MIN_PREF;
14 global p34 : [MIN_PREF..MAX_PREF] init MIN_PREF;
15 formula prob12 = (p12/(p14+p13+p12));
16 formula prob13 = (p13/(p14+p13+p12));
17 formula prob14 = (p14/(p14+p13+p12));
18 formula prob123 = (p23/(p23+p24)); formula prob124 = (p24/(p23+p24));
19 formula prob132 = (p23/(p23+p34)); formula prob134 = (p34/(p23+p34));
20 formula prob142 = (p24/(p24+p34)); formula prob143 = (p34/(p24+p34));
21 formula inc_factor = (tot_dist<15) ? 7 : ((tot_dist=15) ? 4 : 1);

22 module traveller
23   loc : [1..N_CITIES+1] init 1;
24   path : [0..6] init 0;
25   tot_dist : [0..MAX_DIST] init 0;
26   [] loc=1 & path=0 -> prob12 : (loc'=2) +
27                        prob13 : (loc'=3) +
28                        prob14 : (loc'=4);
29   [] loc=2 & path=0 ->
30       prob123 : (loc'=3) &
31                 (path'=1) &
```

```
32                (tot_dist'=D12+D23+D34+D14) +
33      prob124 : (loc'=4) &
34                (path'=2) &
35                (tot_dist'=D12+D24+D34+D13);
36    [] loc=3 & path=0 ->
37      prob132 : (loc'=2) &
38                (path'=3) &
39                (tot_dist'=D13+D23+D34+D14) +
40      prob134 : (loc'=4) &
41                (path'=4) &
42                (tot_dist'=D13+D34+D24+D12);
43    [] loc=4 & path=0 ->
44      prob142 : (loc'=2) &
45                (path'=5) &
46                (tot_dist'=D14+D24+D23+D13) +
47      prob143 : (loc'=3) &
48                (path'=6) &
49                (tot_dist'=D14+D34+D23+D12);
50    [] (loc=2|loc=3|loc=4) & path!=0 -> 1.0 : (loc'=5);
51    [] loc=5 & path=1 -> 1.0 :
52       (path'=0) &
53       (p12'=min(p12+inc_factor,MAX_PREF)) &
54       (p23'=min(p23+inc_factor,MAX_PREF)) &
55       (p34'=min(p34+inc_factor,MAX_PREF)) &
56       (p14'=min(p14+inc_factor,MAX_PREF)) &
57       (tot_dist'=0);
58    [] loc=5 & path=2 -> 1.0 :
59       (path'=0) &
60       (p12'=min(p12+inc_factor,MAX_PREF)) &
61       (p24'=min(p24+inc_factor,MAX_PREF)) &
62       (p34'=min(p34+inc_factor,MAX_PREF)) &
63       (p13'=min(p13+inc_factor,MAX_PREF)) &
64       (tot_dist'=0);
65    [] loc=5 & path=3 -> 1.0 :
66       (path'=0) &
67       (p13'=min(p13+inc_factor,MAX_PREF)) &
68       (p23'=min(p23+inc_factor,MAX_PREF)) &
69       (p24'=min(p24+inc_factor,MAX_PREF)) &
70       (p14'=min(p14+inc_factor,MAX_PREF)) &
71       (tot_dist'=0);
72    [] loc=5 & path=4 -> 1.0 :
73       (path'=0) &
74       (p13'=min(p13+inc_factor,MAX_PREF)) &
75       (p34'=min(p34+inc_factor,MAX_PREF)) &
76       (p24'=min(p24+inc_factor,MAX_PREF)) &
77       (p12'=min(p12+inc_factor,MAX_PREF)) &
78       (tot_dist'=0);
79    [] loc=5 & path=5 -> 1.0 :
80       (path'=0) &
81       (p14'=min(p14+inc_factor,MAX_PREF)) &
82       (p24'=min(p24+inc_factor,MAX_PREF)) &
83       (p23'=min(p23+inc_factor,MAX_PREF)) &
84       (p13'=min(p13+inc_factor,MAX_PREF)) &
85       (tot_dist'=0);
86    [] loc=5 & path=6 -> 1.0 :
87       (path'=0) &
88       (p14'=min(p14+inc_factor,MAX_PREF)) &
89       (p34'=min(p34+inc_factor,MAX_PREF)) &
90       (p23'=min(p23+inc_factor,MAX_PREF)) &
91       (p12'=min(p12+inc_factor,MAX_PREF)) &
92       (tot_dist'=0);
93    [] loc=5 & path=0 & cont<N_CYCLES -> 1.0 :
94       (p12'=(max(p12-ceil(EVAP_RATE*p12),MIN_PREF))) &
95       (p23'=(max(p23-ceil(EVAP_RATE*p23),MIN_PREF))) &
96       (p34'=(max(p34-ceil(EVAP_RATE*p34),MIN_PREF))) &
97       (p14'=(max(p14-ceil(EVAP_RATE*p14),MIN_PREF))) &
98       (p13'=(max(p13-ceil(EVAP_RATE*p13),MIN_PREF))) &
99       (p24'=(max(p24-ceil(EVAP_RATE*p24),MIN_PREF))) &
```

```
100    (loc'=1) & (cont'=cont+1);
101  [] loc=5 & path=0 & cont=N_CYCLES -> 1.0 : true;
102  endmodule
```

A model in PRISM is described as a set of constants and global variables, a set of formulas and a set of modules. Each module describes a component of the system and consists of a set of local variables and a list of guarded transition commands. Each of these commands is described in the form `[l]g -> p: <cmds>`, where `l` is an optional label (`[]` for empty), `g` is a guard, which describes the condition on which the transition may occur, `p` is the probability of that transition occurring if enabled (guard is true) and `<cmds>` determines the effect of the transition. This effect is described in terms of changes in values of variables, which might modify values of formulas.

In our model, each constant D_{ij} determines the distance (or cost) between cities *i* and *j*. Constants *EVAP_RATE* and *N_CYCLES* are used during analysis to set the rate at which the pheromone on routes evaporates and the number of cycles to be executed, respectively. Variable *cont* controls the number of cycles executed, whereas each variable *pij* defines the preference of route *(i,j)*. Lines 15-20 describe the application of the formula in (3) to calculate the probability of travelling between cities. In the scenario proposed (lines 4-5), the distances defined for each route in the graph create 3 groups of path lengths: the longest paths have a total distance of 19, mid-length paths have a cost of 15, and the shortest paths have a total distance of 10. Therefore, we simplified the calculation by testing, according to the total distance of a path, which group this path belongs to[2]. The shortest paths receive an increase of 7, mid-length paths of 4, and the longest paths of only 1.

Module *traveller* describes the behaviour of the artificial ant moving from city to city. Variable *loc* determines the city the ant is currently in. Though *N_CITIES* limits the number of cities, we use an extra "city" to apply the necessary updates, as we will explain soon. Variable *tot_dist* determines the total distance travelled during the tour. Variable *path* determines which of the possible paths has been taken during the current tour, according to the following identification, where the sequences of numbers represent sequences of visited cities: *Path 1* = 1-2-3-4-1, *Path 2* = 1-2-4-3-1, *Path 3* = 1-3-2-4-1, *Path 4* = 1-3-4-2-1, *Path 5* = 1-4-2-3-1, and *Path 6* = 1-4-3-2-1. Considering these possible paths and the distances of each route, paths 2 and 4 are the shortest, 1 and 6 are the mid-length, and 3 and 5 are the longest ones.

From city 1 (line 26), there are three possible routes to take. Because all preferences are initialised with the minimum preference, at the beginning, the probabilities are the same for all possible routes. Depending on the next location, which is determined probabilistically, the choices for the next route are different, so that we comply with the requirement that all cities should be visited only once during a tour. For instance, consider that a transition to city 2 has been selected. Then lines 29-35 describe the behaviour at this location. If we had reached city 2 from city 1, then there would still be two cities to visit (3 and 4). Because there are only four cities involved, once the third city is chosen, we can already identify which path was taken. For example, if city 3 is selected, we know for a fact that the next city is necessarily city 4 and that, from there, we will move back to city 1 to complete the tour. Hence,

[2] Command <*cond*> *? val1 : val2* represents a selection operation where value *val1* is returned in case the boolean expression *cond* is evaluated as true and *val2* is returned, otherwise.

the path taken was 1-2-3-4-1, which is Path 1 (line 31). Since we know the path, we can determine the total distance travelled (line 32).

We defined a special location (city number 5), which is used to apply the necessary updates. The idea is that reaching location 5 represents that a tour is completed. Depending on the path taken, the preferences of the routes involved are updated according to (2) (lines 51-92). Variables *path* and *tot_dist* are reset to signal that the preference update has been executed, which allows the evaporation update to happen. Using the defined evaporation rate, preferences are updated once again according to the formulas presented in lines 94-99. The evaporation occurs at the end of every tour until *cont* reaches the predetermined number of cycles, when the model enters a sink state (line 101). This finite behaviour is necessary to avoid state-space explosion and to allow the execution of a bounded model checking process.

3.2 Property Specification

For the TSP, considering an origin *Orig* and a destination *Dest*, such that $Orig \neq Dest$, two main requirements can be defined:

1. If an ant a_1 starts a tour at time t_1, with probability p_1 of finding the shortest path, and an ant a_2 takes off at time t_2, with a probability p_2 of finding the shortest path, such that $t_1 < t_2$, then $p_1 \leq p_2$;

2. The majority of ants travelling through the graph will eventually follow the shortest path from *Orig* to *Dest*.

Based on these requirements, we have specified properties in PCTL devised to verify whether the behaviour described in the model fulfils the requirements. These properties are presented below, where *"stopped"* is a label that represents the formula *(cont=T & loc=5)*, which defines the end of cycle *T*, where *T* is an undefined constant used during verification. Label *"R12"* represents the formula *((p12>p14) & (p12>p23))*, *"R13"* the formula *((p13>p14) & (p13>p23))*, *"R24"* the formula *((p24>p14) & (p24>p23))*, and *"R34"* the formula *((p34>p14) & (p34>p23))*. Label *"shortest_paths"* represents formula *((path=2)|(path=4))*.

P1: *P=? [F ("stopped" & "R12" & "R13" & "R24" & "R34")]*
P2: *P=? [F ("stopped" & "shortest_paths")]*

Property *P1* is based on requirement 1 and refers to the probability of the preferences of routes that compose the shortest paths being higher than those of other routes when cycle *T* ends. This property checks whether the pheromone update and pheromone evaporation processes guarantee that edges that compose the shortest paths will receive higher concentrations of pheromone, thus having higher probability of being taken. Property *P2*, based on requirement 2, asks the probability of taking one of the shortest paths when cycle *T* ends. Hence, this property can be used to identify how this probability varies from one cycle to the next.

3.3 Verification

Equipped with the model presented in Sec. 3.1 and having the properties described in Sec. 3.2, we were able to carry out experiments using the PRISM tool. Our first experiment was to compare different evaporation rates and analyse how they affect the results of our properties. The objectives were to evaluate the effect of the evaporation rate to guarantee the preference for the shortest paths (2 and 4) and to determine the minimum value of the evaporation rate that guarantees the requirements are fulfilled. For this analysis, we used a model with 20 cycles, which was enough to detect a pattern of increase or decrease of probabilities[3], and T ranging from 0 to 20. Table 1 presents the results for each property for evaporation rates ranging from 0 to 0.8 (we ignore values 0.9 and 1, as they have the same results as those of value 0.8).

Analysing the results, we can see that value 0 (i.e., preference does not decay with time) results in property *P1* having almost 100% probability. This is because no route has its preference decreased as cycles go by, which means all routes of the scenario will reach saturation irrespective of being in the shortest paths or not. The effect of having an evaporation rate can be seen in the results of value 0.1, where already there was a clear tendency to select the shortest paths. With value 0.2 we achieved the probability that the majority of tours involve the shortest paths, but it was with value 0.3 that a consistent majority was achieved. With value 0.4 we reached the highest probability of taking the shortest paths. With values from 0.5 to 0.8, it is possible to see that the probability of the shortest paths decreased as the evaporation rate increased, resulting from an effect similar to that of not having pheromone decay.

Considering the results, we decided to adopt value 0.3 because our main objective was that routes belonging to the shortest paths had higher preference than other routes, so that ants would tend to take those routes. Hence, we needed to take into account the value of property *P1*, and its highest value, considering values 0.3, 0.4 and 0.5 (those where *P2* is above 51%), was obtained with 0.3.

Adopting value 0.3 as the evaporation rate, we produced a model with 20 cycles and verified the results for property *P2* at each cycle. The goal was to obtain evidence that indeed the probability of taking the shortest paths tended to grow after each cycle. The results are displayed on the graph of Fig. 1.

Table 1. Property results for different evaporation rates.

Ev.	**0.0**	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**
P1	0.993	0.807	0.841	0.793	0.713	0.484	0.346	0	**0**
P2	**0.346**	**0.480**	**0.501**	**0.548**	**0.562**	**0.558**	**0.481**	**0.417**	**0.333**

The graph clearly shows the expected behaviour, as the probability of the shortest paths increase continually. Hence, the behaviour described in the model leads to the global behaviour expected, where the majority of ants choose the shortest paths.

[3] In a simulation of property *P2* considering 1000 cycles, the model reached a firm majority ($\cong$ 55%) after 20 cycles and remained around this value from that point on.

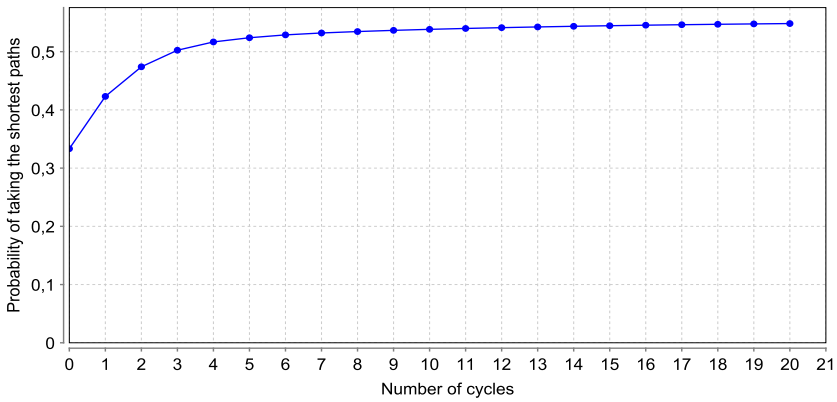

Fig. 1. Analysis of property P2 with evaporation rate 0.3 and 20 cycles.

3.4 Analysis and Discussion

Our experiments showed that the model successfully fulfills the requirements of the system, considering the specification provided. We obtained numerical, accurate evidence that the majority of ants (or travellers) tended to take the shortest paths. Furthermore, we obtained results that indicate the effective action of the pheromones on paths by comparing different evaporation rates.

Though we worked with only four cities, we believe that the modelling could be easily extended to a higher number of cities. The limitation, however, capacity of the machine, as the number of states increases considerably just with the addition of another city. It seems that better abstractions should be defined so as to avoid state-space explosion and allow the verification of larger models. Nevertheless, we are confident that the effectiveness of the ACO modelling is not limited to four cities.

Considering our requirements, requirement 1 can be checked by verifying the result of property *P1*, since it is possible to obtain the probability of higher concentrations being deposited on routes composing the shortest paths along a certain number of cycles (by varying the value of *T*). Requirement 2 can be verified using property *P2*, analysing whether the result reaches a value higher than 50% and tends to not go below this value during the subsequent cycles.

4 Related Work

Considering the modelling of bio-inspired mechanisms, there are a few approaches worth mentioning. In [14], the authors present bio-inspired techniques for self-organisation and self-governance for autonomic networks. Simulation is used to analyse properties like traffic- and node-load and the amount of bandwidth in each route in the network (self-management of resources). In [15], the authors propose self-organising mechanisms based on properties of cellular systems to model

hardware systems that can grow, self-replicate and self-repair. Hardware simulation is performed to show how the artificial organisms evolve.

Formalisms like Brane Calculus [16] and P Systems [17] are inspired by the structure and dynamics of biological membranes and used to model biological processes. There are some approaches proposing the application of model checking to these formalisms, but they are still quite restrictive.

With respect to using ACO to solve the TSP, some approaches have been proposed, such as [10], [18], [19] and [20]. In all these approaches, simulation is used to check the solution to compare it with those of other approaches. Though simulation is in general faster than model checking, it provides only approximate results. Model checking, on the other hand, provides a solid confidence on the results, which is particularly important when dealing with behaviours that cannot be easily predicted beforehand and may violate critical properties. As far as we are aware of, there is no work on applying model checking to verify properties of the TSP-ACO.

5 Conclusions

We presented a probabilistic modelling of the ACO, a bio-inspired mechanism, applied to the TSP. The ACO attributes a self-organising characteristic to the system as it adjusts the probabilities of paths automatically and autonomously, deriving emergent behaviours. Results of experiments show that our model, when checked against a couple of quantitative properties, indeed presents the expected behaviour. This behaviour corresponds to guaranteeing that most of the times the shortest paths are probabilistically taken. We analysed the effect of the pheromone evaporation rate on the variations of path probabilities and determined the most appropriate value for this rate. We also presented a comparison of path probabilities during multiple cycles of execution, representing multiple ants travelling from city to city to complete tours.

We explored the use of model checking because it can provide a higher degree of confidence when compared with other techniques, such as simulation, where results are only approximate. However, we had to apply some simplifications during the modelling phase so as to avoid state-space explosion. These restrictions did not affect the results of our experiments, but might be an issue in other applications. As problems grow more complex, creating abstractions precise enough to guarantee a good level of confidence and yet sufficiently coarse to prevent intractability becomes an even harder task. We still need more experience in applying this modelling approach, considering other scenarios, so that we can better understand the real limits of modelling characteristics as self-organisation and emergent behaviours.

Though our scenario was quite simple, it was enough to demonstrate that the application of the pheromone mechanism leads the system to the expected behaviour. From this result, we intend to study how to apply the same mechanism to other known problems where it can be useful (e.g., routing in sensor networks). We aim to define a modelling pattern for this mechanism to facilitate its use.

We also plan to study how other bio-inspired mechanisms could be modelled and used for the verification of quantitative properties, such as modelling ideas used in swarm intelligence. As bio-inspired mechanisms are essentially self-organising

mechanisms, it seems that having a modelling approach for these mechanisms could be a step towards providing support for the verification of self-organising emergent systems in general.

Acknowledgments. This work was supported by grant MCT/CNPq/CT-INFO 551031/2007-7.

References

1. Olariu, S., Zomaya, A.: Handbook of Bioinspired Algorithms and Applications. Oxford University Press (2007)
2. Heimfarth, T., Danne, K., Rammig, F.: An OS for Mobile Ad hoc Networks Using Ant Based Hueristic to Distribute Mobile Services. ICAS-ICNS, 77 (2005)
3. Janacik, P., Heimfarth, T., Rammig, F.: Emergent Topology Control Based on Division of Labour in Ants. AINA'06 1, 733-740 (2006)
4. Camazine, S., Deneubourg, J., Franks, N., et al.: Self-Organization in Biological Systems. Princeton University Press (2001)
5. Lewes, G.: Problems of Life and Mind. Kessinger Publishing (2004)
6. De Wolf, T., Holvoet, T.: Emergence Versus Self-Organisation: Different Concepts But Promising When Combined Engineering. Self Org. Sys.: Method. and Applications 3464, Springer-Verlag, 1-15 (2005)
7. Vardi, M.: Automatic Verification of Probabilistic Concurrent Finite State Programs. 26th Annual Symp. on Found. of Comp. Sci., 327-338 (1985)
8. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press (1999)
9. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton University Press (2006)
10. Dorigo, M., Gambardella, L.: Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Trans. on Evol. Comp. 1, 53-66 (1997)
11. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: Probabilistic Symbolic Model Checker. TOOLS'02, LNCS 2324, 200-204 (2002)
12. Hansson, H., Jonsson, B.: A Logic for Reasoning About Time and Reliability. Formal Aspects of Computing 6, 512-535 (1994)
13. Ben-Ari, M., Manna, Z., Pnueli, A.: The Temporal Logic of Branching Time. Acta Informatica 20, 207-226 (1983)
14. Balasubramaniam, S., Botvich, D., Donnelly, et al.: Biologically Inspired Self-Governance and Self-Organisation for Autonomic Networks. 1st Intl Conf. on Bio-Inspired Mod. of Net., Inform. and Comp. Sys., 1-30 (2006)
15. Stauffer, A., Mange, D., Rossier, J., Vannel, F.: Bio-inspired Self-Organizing Cellular Systems. BioSystems 94, 164-169 (2008)
16. Cardelli, L.: Brane Calculi. Int. Conf. on Computational Methods in Systems Biology, LNCS 3082, 257-278 (2004)
17. Păun, G.: Computing with Membranes. Journal of Computing and System Sciences 61(1), 108-143 (2000)
18. Shang, G., Lei, Z., Fengting, Z., et al.: Solving Traveling Salesman Problem by Ant Colony Optimization Algorithm with Association Rule. 3rd Int. Conf. on Natural Computation, 693-698 (2007)
19. Li, Y., Gong, S.: Dynamic Ant Colony Optimization for TSP. Int. J. Adv. Manuf. Technol. 22, 528-533 (2003)
20. Ugur, A., Aydin, D.: An Interactive Simulation and Analysis Software for Solving TSP using Ant Colony Optimization Algorithms. Adv. in Eng. Soft. 40, 341-349 (2009)