

Ant Colony Optimization for Testing Concurrent Systems: Analysis of Scalability

Francisco Chicano and Enrique Alba

University of Málaga
{chicano,eat}@lcc.uma.es

1 Introduction

Model Checking [4] is a well-known and fully automatic technique for checking software properties, usually given as temporal logic formulae on the program variables. Some examples of properties are the absence of deadlocks, the absence of starvation, the fulfilment of an invariant, etc. The use of this technique is a must when developing software that controls critical systems, such as an airplane or a spacecraft. Most of model checkers found in the literature use exhaustive deterministic algorithms to check the properties. One example is the Nested Depth First Search (NDFS). The memory required for the verification with exhaustive algorithms usually grows in an exponential way with the size of the system to verify. This fact is known as the state explosion problem and limits the size of the system that a model checker can verify. When the search for errors with a low amount of computational resources (mainly memory) is a priority (for example, in the first stages of the implementation of a program), non-exhaustive algorithms using heuristic information can be used. Non-exhaustive algorithms can find errors in programs using less computational resources than exhaustive algorithms, but they cannot be used for verifying a property: when no error is found using a non-exhaustive algorithm we still cannot ensure that no error exists.

Metaheuristic algorithms [2] are a well-known set of techniques used for finding near optimal solutions in NP-hard optimization problems in which exhaustive techniques are unviable. In the past, we have proposed the use of a new variant of Ant Colony Optimization Algorithm, called ACOhg, to the problem of searching for property violations in concurrent systems [1]. In this work we analyze the scalability of this approach using different scalable Promela models. The main objective of this study is to empirically find out the trend of the resources required by our proposal when the size of the concurrent models checked increase.

2 Problem Formulation

In this communication we tackle the problem of searching for general property violations in concurrent systems. This problem can be translated into the search of a path in a graph (the so called Büchi automaton) starting in an initial node and ending in an objective node (called accepting node) and an additional cycle involving the objective node; or a path that leads to an end node. Whether a node is accepting or not depends on the concrete property we have to check. Due to room problems we cannot explain all the details of the problem here, we refer the interested reader to [3]. We can formalize the problem as follows.

Let $G = (S, T)$ be a directed graph where S is the set of nodes and $T \subseteq S \times S$ is the set of arcs. Let $q \in S$ be the *initial node* of the graph, $F \subseteq S$ a set of distinguished nodes that we call *accepting nodes*, and $E \subseteq S$ a set of nodes that we call *end nodes*. We denote with $T(s)$ the set of successors of node s . A finite path over the graph is a sequence of nodes $\pi = \pi_1\pi_2 \dots \pi_n$ where $\pi_i \in S$ for $i = 1, 2, \dots, n$ and $\pi_i \in T(\pi_{i-1})$ for $i = 2, \dots, n$. We denote with π_i the i th node of the sequence and we use $|\pi|$ to refer to the length of the path, that is, the number of nodes of π . We say that a path π is a *starting path* if the first node of the path is the initial node of the graph, that is, $\pi_1 = q$. We will use π_* to refer to the last node of the sequence π , that is, $\pi_* = \pi_{|\pi|}$. We say that a path π is a *cycle* if the first and the last nodes of π are the same one, that is, $\pi_1 = \pi_*$.

Given a directed graph G , the problem at hand consists in finding a starting path π ($\pi_1 = q$) for which one of the following propositions holds:

- π ends in an accepting node and there exists a cycle ν containing the accepting node. That is, $\pi_* \in F \wedge \pi_* = \nu_1 = \nu_*$.
- π ends in an end node. That is, $\pi_* \in E$.

3 Algorithmic Proposal

In order to solve the previously defined problem we propose an algorithm that we call ACOhg-mc. This algorithm is based on ACOhg, a new kind of Ant Colony Optimization algorithm proposed in [1] that can deal with construction graphs of unknown size or too large to fit into the computer memory. The search that ACOhg-mc performs is composed of two different phases. In the first one, ACOhg is used for finding accepting nodes in the construction graph. In this phase, the search of ACOhg starts in the initial node of the graph q . If the algorithm finds accepting nodes, in a second phase a new search is performed using ACOhg again for each accepting node discovered. In this second search the objective is to find a cycle involving the accepting node. The search starts in one accepting node and the algorithm searches for the same node in order to find a cycle. That is, the initial node of the search and the only objective node are the same: the accepting node. If a cycle is found ACOhg-mc returns the complete accepting path. If no cycle is found for any of the accepting nodes ACOhg-mc runs again the first phase after including the accepting nodes in a tabu list. This tabu list prevents the algorithm from searching again cycles containing the just explored accepting nodes. If one of the accepting nodes in the tabu list is reached it will not be included in the list of accepting nodes to be explored in the second phase. ACOhg-mc alternates between the two phases until no accepting node is found in the first one. For a detailed description of this algorithm we refer the interested reader to [3]. NDFS usually finds very long counterexamples of the checked property and in some cases the amount of memory required for their search is high. With ACOhg-mc we want to find shorter counterexamples using a low amount of memory.

4 Experiments and Results

For the experiments we have selected four Promela models, all of them scalable: `marriers`, `elev`, `giop`, and `phi`. The first model, `marriers`, can reach a deadlock situation; `elev` violates a response property (specified in linear temporal logic); `giop` and `phi` violates both the absence of deadlocks and a response property. The models have a parameter n that allows us to increase their size. This parameter is related to the number of processes of the concurrent system they model. We have implemented ACOhg-mc inside HSF-SPIN, an experimental model checker based on SPIN. Since reducing the memory required in the search is the main concern for the formal methods community we show in Figure 1 the average memory required by ACOhg-mc to find the errors. For the sake of clarity we use the suffix `-d` and `-r` to denote the search for a deadlock situation and a response property violation, respectively. For obtaining these average values we performed 100 independent runs of the algorithm. We have used the same parameterization of ACOhg-mc for all the models.

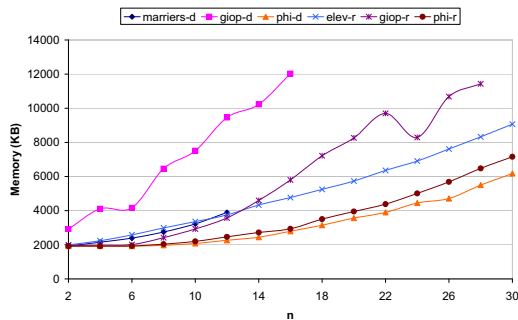


Fig. 1. Memory required by ACOhg-mc

As we can observe, the increase in the memory required by ACOhg-mc is polynomial with respect to the parameter of the model. This is a very important result because the size of the model (the number of nodes) usually grows in an exponential way with the parameter of the model. We performed the same scalability analysis using NDFS and the growth of the memory curve is quasi-exponential in most of the models. In fact, ACOhg-mc is able to find errors in models for which NDFS cannot. With respect to the length of the error trails, the results of ACOhg-mc are by far shorter than the ones of NDFS. Short counterexamples are preferred to understand the error. The maximum CPU time required by ACOhg-mc in the experiments performed was 32 seconds. NDFS is faster but less effective, more memory-consuming, and the counterexamples are longer.

References

1. E. Alba and F. Chicano. Finding safety errors with ACO. In *Proc. of GECCO*, pages 1066–1073, 2007.
2. C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
3. F. Chicano and E. Alba. Finding liveness errors with ACO. In *Proc. of CEC*, pages 3002–3009, 2008.
4. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, January 2000.