Available online at www.sciencedirect.com
ScienceDirect
Information Processing Letters 106 (2008) 221–231
Information Processing Letters
www.elsevier.com/locate/ipl

Ant colony optimization with partial order reduction for discovering safety property violations in concurrent models ☆

Francisco Chicano *, Enrique Alba

Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, Spain

Received 16 June 2007; received in revised form 24 October 2007; accepted 21 November 2007

Available online 20 February 2008

Communicated by L. Boasson

Abstract

In this article we analyze the combination of ACOhg, a new metaheuristic algorithm, plus partial order reduction applied to the problem of finding safety property violations in concurrent models using a model checking approach. ACOhg is a new kind of ant colony optimization algorithm inspired by the foraging behavior of real ants equipped with internal resorts to search in very large search landscapes. We here apply ACOhg to concurrent models in scenarios located near the edge of the existing knowledge in detecting property violations. The results state that the combination is computationally beneficial for the search and represents a considerable step forward in this field with respect to exact and other heuristic techniques.

Keywords: Program correctness; Ant colony optimization; Metaheuristics; Model checking; HSF-SPIN

1. Introduction

From the very beginning of computer research, computer engineers have been interested in techniques allowing them to know if a software module fulfills a set of requirements (its specification). These techniques are especially important in critical software, such as airplane, nuclear plants, and spacecraft software controllers, in which people's lives depend on the software system. In addition, modern non-critical software (like communication protocols) is very complex and these techniques have become a necessity in most software companies. *Model checking* [8] is a well-known and fully automatic formal method in which all the possible states of a given model are analyzed (in an explicit or implicit way) in order to prove (or refute) that the model satisfies a given property. This property is specified using a temporal logic like Linear Temporal Logic (LTL) or Computation Tree Logic (CTL).

In a recent work [3], a new kind of ant colony optimization algorithm called ACOhg was applied to the problem of finding safety property violations in concurrent models using a model-checking based approach. The new algorithm is able to get short error paths (good quality solutions) with a low amount of memory in all the studied models. The contribution of the present work is to combine ACOhg with a technique for reducing

☆ This work has been partially funded by the Ministry of Education and Science and FEDER under contract TIN2005-08818-C04-01 (the OPLINK project). Francisco Chicano is supported by a grant (BOJA 68/2003) from the Junta de Andalucía.

* Corresponding author.
E-mail addresses: chicano@lcc.uma.es (F. Chicano), eat@lcc.uma.es (E. Alba).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2007.11.015

the memory consumption of the algorithm: partial order reduction [14]. With this combination we expect to reduce even more the amount of computational resources required by ACOhg to find execution error paths in concurrent models. This is a very important step forward in software engineering, since it allows the construction of efficient tools for checking real software.

This article is organized as follows. In the next section, we present the background and related work. Section 3 formalizes the problem at hands and describes both the ACOhg algorithm and the partial order reduction. In Section 4 we apply ACOhg with and without partial order reduction and analyze the results. Finally, Section 5 summarizes our conclusions and future work.

2. Background

Our proposal is based on explicit state model checking. One of the best known explicit state model checkers is SPIN [19], which takes a software model codified in Promela and a property specified in LTL as inputs. SPIN transforms the model and the negation of the LTL formula into Büchi automata in order to perform their intersection. The resulting intersection automaton is explored to search for a path starting in the initial state and including a cycle of states containing an *accepting state*. If such a path is found, then there exists at least one execution of the model not fulfilling the LTL property (see [19] for more details). If such kind of path does not exist, then the model fulfills the property and the verification ends with success. In SPIN this exploration is performed with Nested-DFS [20], an exhaustive algorithm. When the property to check is a safety property [23], the verification is reduced to a search for one path from the initial state to one accepting state in the Büchi automaton. This path represents an execution of the concurrent model in which the given safety property is violated. Finding such a path is the case in which we are interested.

The amount of states of the intersection automaton is very high even in the case of small models, and it increases exponentially with their size. This fact is known as *the state explosion problem* and limits the size of the model that a model checker can verify. This limit is reached when it is not able to explore more states due to the absence of free computer memory. Several techniques exist to alleviate this problem. They aim at reducing the amount of memory required for the search by following different approaches. On the one hand, there are techniques which reduce the number of states to explore, such as partial order reduction [22] or symmetry reduction [21]. On the other hand, techniques exist that reduce the memory required for storing one state, such as state compression, minimal automaton representation of reachable states, and bitstate hashing [19]. In spite of the fact that they are largely used, exhaustive search techniques such as Nested-DFS are always handicapped since real concurrent programs are too complex even for the most advanced techniques.

In the first stages of the software development, when the probability of finding errors in the software is high, a good practice consists in guiding this exhaustive search by means of heuristics to gain in efficiency when searching for errors. The utilization of heuristics in model checking is well known and called *heuristic* or *directed model checking*. The heuristics are designed to explore first the region of the state space in which an error is likely to be found. This way, the time and memory required to find an error in faulty concurrent models is reduced on average. However, no benefit from heuristics is obtained when the goal is to verify that a given model fulfills a given property. In this case, the state space must be explored exhaustively. Exhaustive algorithms using heuristics, such as A* or Best First (BF), can find errors in models by using less resources than non-heuristic exhaustive approaches like Depth First Search (DFS) and Breadth First Search (BFS) and, in addition, they can verify the model if no error exists (since they are exhaustive algorithms) [11]. However, when the search for errors with a very low amount of computational resources (memory and time) is a priority, non-exhaustive algorithms using heuristic information can be used. One example of this class of algorithms is *Beam-search*, included in the Java PathFinder model checker [16,17]. Non-exhaustive algorithms have been shown to find errors in programs using less computational resources than exhaustive algorithms [2,3].

A well-known class of non-exhaustive algorithms for solving general complex problems is the class of metaheuristic algorithms [6]. They are search algorithms used in global optimization problems which can find good quality solutions in a reasonable time. The search for accepting states in a Büchi automaton can be translated into an optimization problem and, thus, metaheuristic algorithms can be applied. In fact, Genetic Algorithms (GA), a kind of metaheuristic algorithm, have been applied in the past to the search for errors in concurrent programs. In an early proposal, Alba and Troya [5] used GAs for detecting errors (deadlocks, useless states, and useless transitions) in communication protocols. To the best of our knowledge, this is the first application of a metaheuristic algorithm to the problem of finding errors in concurrent models using a model-checking based approach. Later, Godefroid and