On Applying Random and Genetic Search in Model Checking

Dana Petcu*,** Daniela Zaharie*

* Department of Computer Science, West University of Timişoara,
** Institute e-Austria Timişoara
{petcu, dzaharie}@info.uvt.ro

Abstract

The aim of this work is to analyze the applicability of stochastic search methods in exploring the state space of a concurrent reactive system to find errors like deadlocks. A recently proposed genetic algorithm [3] is thoroughly analyzed and a fair comparison with pure random search is conducted.

Motivation Model checking typically involves an exhaustive search in the state space of a system to determine whether some property of the system is satisfied or not. The practical limitation of applying model checking to real systems is due to the fact that the number of states contained in the state space of complex systems can be huge. Several approaches have been proposed to address this state explosion problem, like symbolic verification, partial order methods, distributed approaches etc.

There at least two strategies of using model checking in the system design process: complete validation to certify the correctness of the design model and partial validation when the model checker is employed as a debugging aid to find faults. This work addresses the second approach, namely exploring the state space with the aim of error(s) detecting. The global state space is interpreted as an implicitly given graph spanned by a successor generator function, in which paths corresponding to error behaviors are searched. The detection of a safety property violation is equivalent to finding a state in which the property is violated. State space exploration in model checking safety properties can be understood as a search for a path to a failure state in the underlying problem graph. The search terminates once a target state has been found. The algorithms used for finding the property violating states are typically depth-first (DFS) and breadth-first (BFS) searches. These are deterministic search strategies and they use no information on the concrete state space they explore: the selection of a successor node in these algorithms is based on a fixed, deterministic selection scheme (from a stack, respectively, a queue). For large state space such a blind deterministic search is inefficient. To make the search efficient, guided deterministic and stochastic search methods have been recently proposed [2],[3], [5].

A heuristic search method, a guided traversal of the state space, that exploits information of the error description to focus the search, is expected to find errors faster and to require less memory (hence to be able to explore larger search spaces). Heuristic search algorithms take into account additional search information in form of an evaluation function. This function is used to rank the desirability of expanding a state. When the nodes are ordered so that the one with the best evaluation is expanded first and if the evaluation function estimates the cost of the cheapest path from the current state to a desired one, the resulting greedy best-first search (BF) often finds solutions fast.

The idea of exploiting heuristics for model checking has received very little attention so far (see [4] for a review of the papers in this field). HSF-SPIN [1] has been recently presented as a model checker based on heuristic search strategies. Based on a A* algorithm, it utilizes heuristic estimates in order to direct the search for finding software bugs in concurrent systems, and it is able to find shorter trails than DFS. Other heuristics were studied in [2]: interleaving, branch-path, branch-global and most-blocked. For example the most-blocked heuristic is specifically tailored to finding deadlocks: it always expands the state which maximizes the number of blocked processes.

In a randomized search if more than one transition is possible for a state, it is chosen one at random (nondeterministic behaviour). Unlike the complete model checking technique, only a portion of possible behavior is explored. If complete algorithms terminate, they find all the features they are searching for. On the other hand, by their very nature, randomized algorithms can miss important features. If the choice is random search versus nothing at all (because the model is too big), the random search methods can still be a useful analysis tool. Randomized algorithms like that from [5] outperform their deterministic counterparts for search problems representing a wide range of applications.

In [6] is proposed a strategy of distributed random walk - a process of multiple processors randomly, and in an uncoordinated fashion, moving through the state-space looking for safety violations, without recording visited states.

The attractive feature of this approach is that the process operates on a large cluster of computers using extremely low memory sizes, so larger model can be checked. A serious problem of this strategy is that it does not generate an error trail. Different heuristics are also considered to combine BFS with random walk.

Genetic algorithms are stochastic population-based search techniques successfully applied in global optimization. They work with a population of individuals which are candidate solutions, e.g. paths in the graph corresponding to the state space. Unlike the random search, genetic search is based on the evolutionary principle of survival of the most fitted individuals and on generating new individuals by combining information from the existing ones. Thus genetic search can be viewed like a guided random search. Designing a genetic algorithm to solve a given problem involves a set of choices: the choice of an adequate encoding of the potential solutions, the choice of a fitness function which affects to each individual a measure of its potential ability to approach a solution, the choice of adequate selection, crossover and mutation mechanisms.

Recently, the authors of [3] proposed the use of genetic algorithms to exploit heuristics for guiding a search in the state space of a concurrent reactive system towards errors like deadlocks and assertion violations. The population is a set of paths in the state space graph, which through the basic transformations (selection, crossover and mutation) are tuned towards an error trail. When the aim is deadlock detection an individual corresponding to a given path is considered to be as fitted to the problem as the total number of enabled outgoing transitions (determined during the execution of the path by the model checker) is lower.

The resulting strategy is able to find errors more efficiently than deterministic search. Although no comparison is done with guided search algorithms, the authors explain that genetic algorithms can simulate bestfirst search.

The aim of this work is twofold: (i) to analyze the influence of selection type on the ability of the genetic algorithm proposed in [3] to detect deadlocks; (ii) to conduct a fair comparative analysis between the genetic approach and the random approach.

The experimental setup *The test problem.* We used as a test the classical dinning philosopher's problem having as aim to find a deadlock with a reduced depth. A systematic search can be applied for instances of size up to $n = 10$. For larger sizes a strategy like DFS needs unreasonable time and space resources. On the other hand by using guided or random search good results have been reported for sizes larger than 10 [4], [5].

The genetic algorithm (GA) approach. We followed closely the genetic algorithm proposed in [3]. Thus the candidate solutions are finite strings of bits (chromosomes) which encode finite sequences of transitions in the state space starting from the initial state. At first the entire population is randomly initialized. When a given individual have to be evaluated the genetic algorithm call the model checker which executes the path encoded by the individual starting from the initial state. At each state the model checker find the number of enabled outgoing transitions and send this information to the genetic algorithm. The genetic algorithm decodes a part of the chromosome it is currently processing and informs the model checker which transition to take: if there are n enabled transitions from the current state being processed during the state-space search, the next $\log_2 n$ bits are read from the current chromosome. This substring of bits is decoded and the obtained value indicates which is

Table 1: The number of deadlocks detected by GA.

Selection type	No. of deadlocks (depth<65)			No. of deadlocks (depth<50)		
	Total	1st gen.	r_e	Total	1st gen.	r_e
Truncated ($f_c = 0.5$)	21/30	8/21	0.61	9/30	5/9	0.44
Truncated ($f_c = 0.05$)	23/30	11/23	0.52	7/30	4/7	0.42
Tournament	21/30	7/21	0.67	9/30	2/9	0.78

the next transition to explore (if the obtained value is greater than n, a new random value in the acceptable domain is generated). The model checker then checks whether the state following that transition is an error state. If so, the current path is saved and the user is notified. Otherwise, the model checker repeats this process from the new state. Since a chromosome can only encode a finite number of transitions, the state space is explored up to a fixed depth. Whenever the model checker has explored a path up to this maximum depth, it prompts the genetic algorithm to evaluate the fitness of the current chromosome. During the path execution the number of possible transitions corresponding to all expanded states are summed up. This value represent a heuristic measure of the proximity of the path to a deadlock trail. The fitness of a chromosome is higher if this sum is lower. Once the fitness of the current chromosome has been computed, another chromosome of the current population is evaluated using the same process.

After all chromosomes have been evaluated a truncated selection is used to construct a population of parents. The particularity of the truncated selection is that all chromosomes are decreasingly sorted according to their fitness and a fraction of them (e.g. 50%) are used as parents for the next generation. From the parents population, by one cut-point crossover, is obtained a set of offsprings. Then is applied a gene-pool mutation and the next generation is constructed in a elitist manner: a fraction f_c of the best chromosomes in the old population is copied in the new population, the rest of population being filled with new generated offsprings.

The random search (RS) approach. We implemented a pure random search approach characterized by the fact that at each state the transition to be applied is randomly selected from the set of all possible transitions.

Results The first experiment have the aim to analyze the ability of the GA to find a deadlock up to some fixed depths for the dinning philosopher's problem with $n = 17$. In truncated selection we controlled the fraction of conserved individuals (f_c). For a population with m individuals mf_c are conserved from the previous generation while the rest of $m(1 - f_c)$ are new offsprings. If $f_c > 0$ the selection is elitist and as f_c is higher as the selection pressure is higher. The value used in [3] is $f_c = 0.5$. We analyzed the influence of the selection pressure by testing the case $f_c = 0.05$ and another type of selection with lower pressure: the tournament selection (the parents are established by iterating the selection of two individuals and by choosing the best one between them). In each case the results were collected from 30 independent runs. Besides the number of deadlocks found during the entire evolution (the maximum number of generations was set to 10) the number of deadlocks found in the first generation has been registered. To measure the contribution of the evolutionary mechanisms to guide the search toward deadlocks we use the ratio between the number of deadlocks found after the first generation and the total number of found deadlocks (r_e). This ratio offer a more objective measure of the influence of evolution in the search process than the total number of found deadlocks. The results obtained for a population of $m = 100$ individuals (strings of 350 bits) are presented in Table 1. For truncated selection ($f_c = 0.5$), these results are similar with that reported in [3]. With respect to the ratio r_e the tournament selection seems to work better than the truncated selection. This can be explained by the lower selection pressure thus by the higher exploration ability.

The second experiment uses the random search to solve the same problem. To assure a fair comparison with the GA approach, in each of the 30 runs, m independent random walks have been executed. In this way the random search is equivalent with a GA applied to a population of m

individuals but stopped after the evaluation of the initial population. The results in Table 2 suggest that GA manages to find short trails to deadlock (see the case depth<50) when compared to random search. However when the number of independent random walks is sufficiently high (e.g. $m = 1000$) the difference between GA and RS is not so clear as reported in [3]. This can be explained by the fact that the comparison reported in [3] is in the detriment of RS since only one random walk is executed while the population in GA has 200 individuals evolved during 50 generations.

Table 2: The number of deadlocks detected by random search

No. of independent random walks	No. of deadlocks/ No. of runs depth<65	No. of deadlocks/No. of runs depth <50
100	7/30	3/30
500	27/30	5/30
1000	30/30	6/30

Directions for future work Starting from our results and from that reported in [3] it follows that some improvements have to be made to the GA approach such that it could become a competitive technique in large state space exploration.

The purpose of future experiments concerning the improvement of GA algorithm for deadlock detection is to identify heuristics and properties of state spaces that help a genetic search to detect deadlocks. One possible direction is to introduce in the GA algorithm a fitness function based on the heuristics proposed in [4].

Another direction is to replace the random search from the distributed random walk from [6] with a selection based on chromosomes, the first advantage being to retain the error trail.

It is also feasible to further improve the order of explored paths by using parallel versions of the genetic algorithms. Several strategies can be adopted in this direction [7]. The task farm seem to be at a first glance the most appropriate approach. This assumption must be confirmed or invalidated by experiments.

References

[1] Stefan Edelkamp, Alberto Lluch-Lafuente and Stefan Leue. Trail-Directed Model Checking. *Electronic Notes in Theoretical Computer Science* **55** (3), 2001, http://www.elsevier.nl/locate/entcs/volume55.html

[2] Stefan Edelkamp and Tilman Mehler. Byte Code Distance Heuristics and Trail Direction for Model Checking Java Programs. http://tele.informatik.uni-freiburg.de/leue/dirmodcheck.html

[3] Patrice Godefroid and Sarfraz Khurshid. Exploring very large state spaces using genetic algorithms. In 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), *Lecture Notes in Computer Science* **2280**. Springer, 2002.

[4] Alberto Lluch Lafuente. Directed Search for the Verification of Communication Protocols Dissertation PhD thesis. AlbertLudwigsUniversitat Freiburg. June 2003

[5] David Owen, Tim Menzies, Mats Heimdahl, Jimin Gao. On the advantages of approximate vs. complete verification: Bigger models, faster, less memory, usually accurate. In *IEEE NASE SEW* 2003. Available from http://menzies.us/pdf/03lurchc.pdf.

[6] Hemanthkumar Sivaraj, Ganesh Gopalakrishnan. Random Walk Based Heuristic Algorithms for Distributed Memory Model Checking *Electronic Notes in Theoretical Computer Science* **89** (1), 2003, http://www.elsevier.nl/locate/entcs/volume89.html

[7] Michael Tomassini. Parallel and Distributed Evolutionary Algorithms: A Review In *Evolutionary Algorithms in Engineering and Computer Science*, K. Miettinen, M. Makela, P. Neittaanmaki and J. Periaux (eds.), 113–133, J. Wiley and Sons, Chichester, 1999.