Experiments in applying evolutionary algorithms to software verification

Woei Shyang, Charles Lakos, Zbigniew Michalewicz, Sven Schellenberg

Abstract—Complex concurrent systems present a significant challenge for software verification. If those systems are safety-critical, the need for software verification becomes particularly pressing, given the serious consequences of unforeseen defects. Complex concurrent systems are characterised by extremely large state spaces. The use of testing techniques for verification means that very little of the state space is explored. On the one hand, model-checking techniques exhaustively examine the state space, but will be stymied by the actual size.

In this paper, we discuss some preliminary experiments on the application of evolutionary algorithms to software verification. This approach does not explore the whole state space, but does use heuristics to guide the search through the most promising parts of the state space for locating errors.

I. INTRODUCTION

A. The challenge of software verification

Complex concurrent systems such as distributed e-commerce applications, factory automation systems, network protocols, etc. typically have an inordinately large number of possible states. This state space needs to be explored in order to achieve a high level of confidence in the correctness of such systems, especially safety-critical systems such as those used in aircraft control, railway signalling, medical monitoring and control systems.

Testing is a traditional approach to verifying software. There, a number of scenarios are fed to the system, and the results are compared with those expected. Such test cases explore only a small fraction of the possible system behaviour — it is never exhaustive and can never guarantee the absence of errors. At best, the careful choice of the test cases can give some qualitative increase in the confidence that the system is correct. This can be an effective technique, depending on the expertise of the tester [15].

Model checking is a radically different approach to system verification — here, properties are evaluated over all reachable states (and even over all sequences of states). Model checking is an exhaustive analysis which can guarantee correctness, but only if the state space is tractable. There are techniques for reducing the size of the state space without compromising the properties of interest, including partial order reduction, equivalence reduction, and modular techniques. The state space can also be reduced by approximation techniques, such as bit-state hashing [11] and state space caching [14]. A more recent approach is Bounded Model Checking [2], which progressively looks for longer counter examples. However, these techniques are rarely sufficient on their own — what is normally explored is a model of the system rather than the system itself. Extracting a model with a manageable state space is not a simple task — the chosen level of abstraction needs to include the characteristics of the system relevant for the properties to be verified. Counter-Example Guided Abstraction Refinement (CEGAR) [3] allows the user to over-abstract, and then progressively refine the model to home in on potential errors.

B. The promise of evolutionary algorithms

In contrast to the above approaches for verifying systems with large state spaces, many different heuristic search techniques have emerged over recent years to deal with large search spaces. They complement exact and complete algorithms, and provide approximate solutions to a problem.

This paper presents preliminary results from experiments in applying population-based heuristic techniques to software verification. Unlike model checking, this approach is not exhaustive but this means that it can be applied even to systems with very large state spaces. Unlike testing, it will not be ad hoc, but will target problematic areas of the state space.

II. BACKGROUND

In this section, we consider some of the previous approaches in applying heuristic techniques to software verification. These include the directed model checking of Edelkamp et al, the genetic algorithms approach of Godefroid and Khurshid, and the ant colony optimisation of Alba and Chicano.

A. Directed model checking — Edelkamp et al

The most prominent work to date in the area of heuristic techniques for model checking has been that of Edelkamp et al in the use of best-first search of the state space (as opposed to breadth-first and depth-first). The choice is made based on the known distance from the initial state and a heuristic estimate of the distance to an error state. The estimate is based on a Hamming distance between the state vectors [6], [16]. This work assumes that errors are present and aims to find the shortest traces that lead to the error state(s). Specifically [6] considers how liveness properties can be explored within a best-first search strategy, while [16] considers how partial order reduction can be incorporated with the best-first strategy. This approach has been coded in a tool called HSP-SPIN [6], an extension of the widely-available SPIN model checker [12].
The above approach utilising classical artificial intelligence search algorithms, has been successful in finding shorter error trails and reducing memory consumption. However, we are interested in the application of modern heuristic techniques, and specifically population-based techniques.

B. Genetic algorithms — Godefroid and Khurshid

Godefroid and Khurshid applied genetic algorithms to the detection of deadlocks in the Dining Philosophers’ Problem [7]. They considered an instance of the problem with 17 philosophers, so that the state space would be too large for exhaustive exploration. Furthermore, philosophers who become hungry make a non-deterministic choice as to which fork they will pick up first. Chromosomes encoded a path through the state space, starting from the (unique) initial state. (More on this below.) The genetic algorithm was combined with a model checker, so that a chromosome could be interpreted as a path through the state space, and executed in the model. In the process, a measure of fitness for this particular path could be determined.

The chromosome decoding was as follows: given a point reached in the chromosome and a corresponding state, the model checker determines the number of outgoing transitions from that state, say \( n \); the next \( \log n \) bits of the chromosome are used to select one of the outgoing transitions. If the number of outgoing transitions of a state is not a power of 2, then the next \( \log n \) bits can include spurious encodings. Rather than mapping the decoded integer onto the desired interval, Godefroid and Khurshid deal with decoded integers outside the interval by randomly generating a new number inside the interval and repairing the chromosome to reflect this number.

A significant concern is the chromosome length since this determines the depth of the state space. The approach adopted was to choose (arbitrarily) a chromosome length sufficient to encode paths up to a given maximum depth. The effective length of chromosomes is tracked, namely the maximum number of bits that have been read from any single chromosome up to that point in the search. Mutations and crossovers are only performed on the initial chromosome segments (up to the effective length).

It may be possible to address this concern about a fixed chromosome length by adopting an approach similar to Bounded Model Checking [2]. In other words, we could start off with a shorter chromosome and, only if no solutions are found of that length, would we explore progressively longer chromosomes. Alternatively, the length of chromosome could be the subject of self adaptation.

The fitness function plays the key role of guiding the genetic algorithm towards an optimum, or at least satisfactory, solution. For deadlock detection, Godefroid and Khurshid reasoned that the goal is to find a state with no outgoing transitions, and hence they chose a simple fitness function which was the sum, at each state along a path, of the number of outgoing transitions. This makes sense as long as the number of choices at each state steadily reduces up to the deadlock state.

The parameters used were as follows: the population size was 200, with the best 100 chromosomes reproducing to form the next generation. The default mutation rate was 200, i.e. each bit of a chromosome produced by a crossover is flipped with probability 1/200. (The mutation rate was temporarily increased if the algorithm appeared to be in a local minimum, i.e. the best and worst chromosomes in a population had the same fitness value.) The effective length of chromosomes varied between 75 and 320.

The genetic algorithm was run 50 times, with each run having 50 generations. The number of runs in which a deadlock was detected was then recorded. Godefroid and Khurshid found that the genetic algorithm was more effective in finding deadlocks (in 26 out of 50 runs) as compared to either an exhaustive search or a random search (which found no deadlocks). They experimented with using mutation alone, and found that there was no change in the effectiveness of the algorithm. They also experimented with the inclusion of partial order reduction, and found that this decreased the effectiveness of the algorithm.

C. Ant colony optimisation — Alba and Chicano

Quite recently, another meta-heuristical approach was proposed by Alba and Chicano in [1]. Here, a modified version of an Ant Colony Optimization (ACO) algorithm is used to explore the state space of a Finite State Automaton (FSA). They argue that standard ACO models are not appropriate. Firstly, the number of states of the FSA is unknown and they are generated dynamically as the previous state is examined. By contrast, many standard ACO models initialise the pheromone trail with a value that depends on the number of nodes. Secondly, the vast size of the search space limits the applicability of most ACO models since this would require long potential trails through the state space, and thus consume a lot of memory. The ACO would eventually encounter the same limitations as traditional Model Checking techniques, namely the state space explosion problem.

In order to solve these difficulties, Alba and Chicano introduce a so-called Missionary Technique to explore the FSM. The idea is that the length of the paths explored by the ants are limited at each optimisation step to some parameter value \( \lambda_{\text{ant}} \). Once the exploration of this partial solution is finished, the best trails found so far are used as starting points for the next run. Hence, each run successively builds up the path leading to the error in the model. Since only a partial solution is optimised at a time, the memory consumption is limited to the current ant colony’s memory needs and the error trail.

Once the ants (10 ants are used for the trials) approach their search depth limit \( \lambda_{\text{ant}} \), the paths’ pheromone trails are updated according to their fitness values and the ants optimise the paths, which are now populated with pheromones, again. The best solutions (10) of this last optimisation step are kept and used as a starting point for the ants at the next step. The 10 ants are assigned to the potential 10 starting points by a roulette-wheel mechanism, which, again, depends on the fitness of the trail. That is, trails with better fitness
values are more likely to be expanded than those with lower fitness. Now the ants wander again until they reach a depth of $\lambda_{\text{ant}}$, and thus the search space is expanded without considering the error trail found so far any more.

The above algorithm is referred to as ACOhg (for ACO huge graph). The algorithm is run on numerous test cases and the results are compared with exact Model Checking algorithms (BFS, DFS) and other heuristic techniques ($A^*$ and BF) used by Edelkamp et al [6] and Godefroids evolutionary approach [7]. In general, for all the models examined, the ACOhg is able to find an error trail with a probability of 49% to 100% without using the heuristic information of the nodes. This rate dramatically increases when the heuristic information is used to guide the ants (84% to 100% hit rate). With regard to the length of the error trails produced, both ACOhg variations (with and without heuristic information) find near optimal error trails. As for the computational resource consumption, the proposed ACO requires significantly less memory when compared to traditional Model Checking methods, the ACOhg consumes similar amount of memory and rate has been adopted? (We have used a single crossover pool). These results are close to, but not identical to, those of Godefroid and Khurshid. The discrepancy may be due to a number of sources, including a different implementation of non-deterministic choice and random variations. (In three additional runs, we noted success rates of 18, 24 and 25.) Also, there are certain parameters which are not clear from the original paper — are parents mutated before or after being added into the offspring pool? what kind of crossover and rate has been adopted? (We have used a single crossover point with a typical crossover rate of 75%. In other words, crossover is applied to 75% of the parent population.)

B. Selection strategies

We experimented with a variety of different selection strategies. The approach adopted by Godefroid and Khurshid is termed deterministic selection. Here the best half of the population is used to produce the next generation, while the worst half is discarded. We also experimented with roulette selection where the choice of parents is probabilistically dependent on their fitness values (after normalisation), and with tournament selection where a number of chromosomes are chosen arbitrarily (as a tournament) and the fittest chromosome is then used as a parent.

Both roulette selection and tournament selection performed significantly worse than deterministic selection — the former managed 5 successes out of 50, while the latter managed 10. However, there is a significant difference between these techniques and deterministic selection in that deterministic selection discards the worst half of the population. A similar approach, termed truncation, can be applied to roulette selection and tournament selection. Truncation excludes a fraction of the population with poor fitness from selection for the mating pool. (A truncation rate of 50% is implicit in deterministic selection.) We experimented with truncation rates of 10%, 20%, ... 90%. The results are shown in Fig. 1. We note that a truncation rate of 30% produced best results for both roulette and tournament selection, with a success rate of 23 out of 50 in each case.

Given that mutations and crossover can dramatically change the path through the state space encoded by a chromosome, we also experimented with elitism, i.e. retaining the best individuals from one generation into the next (unchanged). It turned out that elitism helped a bit — we achieved a success rate of 22 out of 50 for a truncations rate of 50% and an elitism rate of 50%.
C. Niching and Speciation

A common problem with heuristic search strategies is that they can home in on local optima and then face difficulties in leaving the local optima and finding the global optima. This problem may be addressed by maintaining a certain diversity in the population. In an ecological sense, a niche is a location in an ecosystem that exhibits certain properties. It can be thought of as a region where individuals inhabiting the region compete for limited resources. We would normally expect that similar creatures stay within the same niche.

When applied to population-based heuristics, niching means that the fitness value of an individual is modified by competition with other individuals in the vicinity. Individuals in different niches can thrive independently. There are various approaches for achieving this effect: the fitness value can be scaled (as in Goldberg and Richardson [8]); the best $k$ individuals with a certain minimum separation can be chosen and clusters around these individuals can be formed (as in Yin and Germay [18]); the winner of a subpopulation can take all the resources (as in Pétrowski [17]); or subpopulations can consume available resources (as in De Jong [5]).

We encoded a niching strategy based on the phenotype. Specifically, we formed niches based on the number of new states that a chromosome would add to the mating pool. This choice was made rather than using fitness values because similarity of chromosomes does not guarantee similar fitness values. Thus, given a population, we form a mating pool as follows: chromosomes are classified into buckets according to the number of states that are so far unvisited in the existing mating pool; a bucket is chosen using roulette wheel selection (with greater contribution of unvisited states having a higher probability); a chromosome from the winning bucket is chosen using roulette wheel selection; the winning chromosome is added to the mating pool; and the process is repeated. The adoption of this niching technique produced much better results, with a success rate of 28 and 30 out of 50.

It is postulated that the greater truncation rates produce better results here because the niching techniques lead to a loss of selective pressure. The diverse selector classifies individuals first based on their uniqueness and then does a roulette wheel selection on the winning bucket. Thus, the roulette wheel selection that ultimately selects the individual into the mating pool only performs a fairly localised selection on a subpopulation rather than on the population as a whole.

In addition to niching, which guarantees diversity in the population, we also considered speciation. Here, mating partners are restricted so that distinctive features are maintained. Specifically, we seek to maximise the number of common states visited between the two partners, while avoiding the mating of clones. The addition of this feature results in further improvement of the results, with a success rate of 35 out of 50. The graph of Fig. 2 shows the results of niching and speciation for different truncation rates.

It should be noted that while niching and speciation help to improve the results, they come at a significant penalty in efficiency.

IV. Reflections on the Results

Our experiments indicate that it is possible to make some improvements to the success rates of the genetic algorithm by using truncation, elitism and niching. However, the results are not dramatic, and we here postulate why this might be so.

A. Disconnect between change in genotype and phenotype

The basic assumption behind genetic algorithms is that there is a close relationship between the chromosome encoding (or genotype) and the associated characteristics of the individual (or phenotype). Thus, the common genetic operation of mutation assumes that a small change in genotype will lead to a small change in phenotype. In the encoding that we have investigated in our experiments, these basic assumptions do not hold. A chromosome encodes a path through the state space. It is broken up into a number of blocks (of bits), each block making a selection of transitions from a reached state of the system. Not only does the present state influence the outcome of the most recently decoded transition, but it also determines how the subsequent bits are interpreted and decoded. Thus, even the partitioning into blocks is dependent on the states reached. A single bit change in the chromosome, can lead to a completely different path through the state space.
Such a scenario can be contrasted with the Hamming Cliff effect [4]. There, transitions to certain neighbouring solutions require an inordinately large change in genotype. Both cases are undesirable because changes in genotype should ideally translate into proportionate changes in phenotype.

B. Loose Parent-Child relationship

Similarly to the preceding subsection, another common genetic operation is that of crossover, where the genetic material of two parents is meant to be combined in the offspring. Again, however, the encoding that has been explored means that the head of the child chromosome (contributed by one parent) will completely change the context in which the tail of the chromosome (contributed by the other parent) will be interpreted. It is therefore not too surprising that Godefroid and Khurshid observed no difference in performance between a genetic algorithm that utilises crossover in contrast with one that derives the solution by random mutation alone. Such an observation suggests that an unhindered crossover operator tends to lead towards destructive recombination; one in which the offspring solutions are less viable than either parent.

C. Parallels with the Travelling Salesman problem

The above observations call into question the appropriateness of the given encoding of paths in the state space, at least with respect to genetic algorithms. They are reminiscent of the issues involved in solving the Travelling Salesman problem using genetic algorithms. There, the simple encoding of the tour as a list of cities means that crossover might result in cities being omitted or duplicated. In response to this, an ordinal representation of the tour was proposed, where the $i$-th element of the list is a number in the range from 1 to $n-i+1$. Given an initial ordered list of cities, say

$$C = (123456789),$$

suppose that some subset of the cities have been traversed, say

$$C' = (12348),$$

leaving

$$C - C' = (5679).$$

A partial tour of the form

$$1 - 2 - 4 - 3 - 8$$

would be represented by the partial list of ordinal values

$$L' = (11214).$$

If the next city to be included is 5, then the ordinal value 1 is added to the list, indicating that the first element of $C - C'$ is the next city to be visited. Thus, a complete tour

$$1 - 2 - 4 - 3 - 8 - 5 - 9 - 6 - 7$$

is represented as a list $L$ of references,

$$L = (112141311).$$

The main advantage of the ordinal representation is that the classical crossover operation works! Any two tours in the ordinal representation, cut after some position and crossed together, would produce two offspring, each of them being a legal tour. For example, the two parents

$$p_1 = (1121|41311)$$

$$p_2 = (5155|53321),$$

which correspond to the tours

$$1 - 2 - 4 - 3 - 8 - 5 - 9 - 6 - 7$$

and

$$5 - 1 - 7 - 8 - 9 - 4 - 6 - 3 - 2,$$

with the crossover point marked by ‘|’, would produce the following offspring:

$$o_1 = (112153321)$$

$$o_2 = (515541311),$$

these offspring correspond to

$$1 - 2 - 4 - 3 - 9 - 7 - 8 - 6 - 5$$

and

$$5 - 1 - 7 - 8 - 6 - 2 - 9 - 3 - 4.$$
where the search is made for progressively longer error trails. Alternatively, it may be possible to use the Missionary technique of Alba and Chicano [1].

We also feel that there is much to learn from the experimental approach described in this paper. For example, we need to know whether the fitness measure is discriminat-}

ing enough. Does it provide enough guidance for selecting between alternative paths? This is especially important if a progressive technique is adopted.

There are three main directions for the future work. The first one consists of various improvements to our current version of evolutionary algorithm. These improvements include experiments with progressive search, improved encoding scheme, problem-specific operators, and adaptive parameter values. It seems, for example, that the mutation rate is of key importance for the performance of the system, so experiments with self-adaptive mutation rate might be of significance. The second direction aims at rigorous comparisons between ACO and EAs, paying particular attention to the evaluation function which discriminates among the potential solutions. Finally, we plan to examine a hybrid approaches by combining features of ACO and EAs. In particular, we would be interested in running these two meta-heuristics in parallel with a migration element between these two approaches, and incorporation of path-building activity from ACO as one of the operators in EA.

REFERENCES


