Empirical Analysis of a Genetic Algorithm-based Stress Test Technique

Vahid Garousi
Software Quality Engineering Research Group (SoftQual)
http://www.softqual.ucalgary.ca
Department of Electrical and Computer Engineering, University of Calgary
2500 University Drive NW, Calgary, AB Canada T2N 1N4
vgarousi@ucalgary.ca

ABSTRACT

Evolutionary testing denotes the use of evolutionary algorithms, e.g., Genetic Algorithms (GAs), to support various test automation tasks. Since evolutionary algorithms are heuristics, their performance and output efficiency can vary across multiple runs. Therefore, there is a strong need to empirically investigate the capacity of evolutionary test techniques to achieve the desired objectives (e.g., generate stress test cases) and their scalability in terms of the complexity of the System Under Test (SUT), the inputs, and the control parameters of the search algorithms. In a previous work, we presented a GA-based UML-driven, stress test technique aimed at increasing chances of discovering faults related to network traffic in distributed real-time software. This paper reports a carefully-designed empirical study which was conducted to analyze and improve the applicability, efficiency and effectiveness of the above GA-based stress test technique. Detailed stages and objectives of the empirical analysis are reported. The findings of the study are furthermore used to better calibrate the parameters of the GA-based stress test technique.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**] Testing and Debugging - *Testing tools*

General Terms

Algorithms, Experimentation, Reliability

Keywords

Empirical Analysis, Genetic Algorithms, Stress Testing

1. INTRODUCTION

Distributed Real-Time Systems (DRTS) are becoming more important to our everyday life. Examples include command and control systems, aircraft aviation systems, robotics, and nuclear power plant systems [1]. However, the development and testing of such systems is difficult and takes more time than for systems without real-time constraints or distribution [1]. Based on an analysis of sources of failures in the United States Public Switched Telephone Network (PSTN) [2], it is reported that in the 1992-1994 time period, although only 6% of the outages were overloads, they led to 44% of the PSTN's service downtime. In the system under study, overload was defined as the situation in which service demand exceeds the designed system capacity. So it is evident that although overloads do not happen frequently, the failure resulting from them can be quite expensive.

Distributed nodes of a DRTS regularly need to communicate with each other to perform system functionality. Network communications are not always successful and on time as problems such as congestion, transmission errors, or delays might occur. On the other hand, many Real-Time (RT) and safety-critical systems have hard deadlines for many of their operations, where if the deadlines are not met, serious or even catastrophic consequences will happen (e.g., explosion in a nuclear power plant).

Proposing that the UML [3] design model of a DRTS is in the form of Sequence Diagrams (SD) annotated with precise timing information, and the system's network topology is given in a specific modeling format, we presented in [4] a stress test technique, referred to as *Genetic Algorithm-based Stress Test Technique (GASTT)*, to derive test requirements to stress the DRTS with maximizing the network traffic in a way that will likely reveal RT faults. The GASTT technique itself was an extension to another earlier technique referred to as *Time-Shifting Stress Test Methodology (TSSTM)* [5]. The difference between the two techniques was that GASTT considered the complex timing constraints of RT tasks in DRTSs (such as arrival patterns) when deriving stress test requirements, while TSSTM was only applicable to DRTSs without timing constraints for RT tasks. That mentioned, GASTT was shown [4] to be successful for deriving strenuous but *valid (legal)* test requirements that respect the timing constraints of RT tasks in a System Under Test (SUT). s

The GASTT methodology [4] was designed by making use of a specifically-tailored Genetic Algorithm (GA) to automatically generate test requirements which comply with task timing constraints and lead to high traffic-aware stress. Since GAs are heuristics, their performance and output efficiency can vary across multiple runs, and the GASTT methodology [4] is no exception. Calibration of different GA parameters (e.g., crossover ratio) is also very important in achieving a reliable performance for a GA-based software testing technique. Therefore, the efficiency and effectiveness of a GA and also its parameter calibration should be validated through empirical means.

From a broader perspective, evolutionary testing denotes the use of evolutionary algorithms (e.g., GAs) to support various test automation tasks [6]. In the context of evolutionary testing, as Briand recommended recently [7]: *"it is important to empirically investigate their [evolutionary test techniques'] capacity to achieve the desired objectives (e.g., generate stress test cases) and their scalability in terms of the complexity of the SUT, the inputs, and the control parameters of the search algorithms*".

As an effort in the above direction, this paper presents a carefully-designed empirical study to validate the design choices of the GA underlying GASTT [4]. The empirical study's methodology, objectives and findings are reported. The findings are used to better calibrate the parameters of GASTT. It is hoped that the current work emphasizes yet again the importance of empirical studies in Search-Based Software Engineering (SBSE). Furthermore, the author hopes that the findings of this empirical study can help other SBSE researchers with the empirical analysis of their own techniques.

The remainder of this article is structured as follows. Related works are discussed in Section 2. An overview of GASTT [4] crucial for the presentation of this paper is described in Section 3. The empirical analysis and its results are presented in Section 4. Section 5 concludes the article and discusses some of the future research directions.

2. RELATED WORKS

There are many works in the operations research community (for instance, [8-10]) which empirically study the performance of GAs and other Evolutionary Optimization Techniques (EOT), e.g., the work in [8] empirically evaluates *spatial* (a type of parallel) GAs whose population is distributed on different types of networks.

Focusing on the state-of-the-art in the SBSE community, although a few works in the area have performed empirical studies to evaluate the performance of the EOTs used in their SBSE techniques, however it seems that the importance of EOT empirical studies in SBSE is still somewhat under-estimated. To the best of the author's knowledge, the works in [11-14] are the major works in this direction which present empirical evaluations of EOTs when applied in software testing [12-14] and project planning [11].

Antoniol et al. [11] empirically evaluated the use of three different search–based techniques, namely GAs, hill climbing and simulated annealing, for planning resource allocation in large-scale maintenance projects. Results of the study showed that a GA with an ordering-based genome encoding and a tailored cross-over operator appears to provide the most robust solution.

The study in [12] by Harman et al. theoretically and empirically evaluated the impact of input domain (search space) reduction on the performance of search–based test techniques, presenting results from the application of local and global search algorithms to real-world examples. The theoretical analysis predicted that search space reduction would not have a significant effect on random testing, but could enhance the performance of more intelligent search techniques, such as hill climbing and GAs. An empirical study, performed on 360 branches from an open source code and another embedded controller production code, supplied by DaimlerChrysler, was found to support these claims.

Harman and McMinn presented in [13] a theoretical and empirical analysis of evolutionary testing and hill climbing for structural (white-box) test data generation. A new theoretical framework was constructed as a generalization of the theories of *schemata* and *Royal Roads* from the literature of evolutionary computation. The theory is used to predict the situations in which EOT will perform well, i.e.., how good (close to the optimum) the outputs are, and to explain why. These predictions are validated by an empirical observation. The empirical study then goes on to explore the impact of the choice of search technique providing some important and perhaps counter-intuitive findings. The findings of the study are surprising because they indicate that sophisticated search techniques, such as EOT can often be outperformed by far simpler search techniques. However, as the theory indicates, the findings also show that there do exist test data generation scenarios for which the evolutionary approach is ideally suited.

Xiao et al. [14] presented an empirical evaluation of five types of EOTs when used to generate test case data: (1) Genetic Algorithms, (2) Simulated Annealing, (3) Genetic Simulated Annealing, (4) Simulated Annealing with Advanced Adaptive Neighborhood, and (5) a random-search optimization technique. The authors of [14] believed that a number of papers dedicated to approaches and methods providing suggestions regarding selection of the best values of control parameters for a given EOT have been published (e.g., [15, 16]). However, there is no universal *recipe* that can be used for calculating the values of control parameters in all contexts and applications. In many cases, the values have to be adjusted for a given problem at hand and algorithm. A set of optimization experiments was conducted in [14] for different values of parameters for each algorithm, and the parameters that led to the best performance of that algorithm for a given problem (i.e., a SUT) were identified. Among the GA parameters analyzed and calibrated through experiments in [14] were: population size, crossover probability, mutation probability, and termination criterion (number of generations).

Although the works in [11-14] are interesting contributions, they did not evaluate EOTs' operating performance and efficiency w.r.t. the following three criteria: (1) repeatability of results across multiple runs of an algorithm (this is important for heuristic-based EOTs since they have notions of randomness), (2) convergence efficiency across generations towards a stable maximum plateau, and (3) scalability of the EOT at hand, i.e., impacts of variations in the SUT size and complexity. The current empirical study takes into account the above criteria.

From a higher-level perspective, in a keynote speech at the International Symposium on Empirical Software Engineering and Measurement (ESEM) 2007, Briand presented a critical analysis [7] of empirical research in software testing. As he mentioned, empirical studies of software testing should go beyond assessing the cost-effectiveness of test techniques. He stated [7] that, as evolutionary test techniques are heuristics, it is important to empirically investigate their capacity and scalability when used in test automation tasks.

Furthermore, in a road-map paper, Harman [6] mentioned that scalability of results generated by SBSE techniques, robustness of results, and insight into how those techniques work are among the important cross-cutting issues which need to be carefully investigated for the evolutionary testing to be adapted in large-scale industrial settings.

The work reported in this paper is taking into account the above operating performance criteria of GAs when applied to stress test requirement generation using GASTT [4], and is an investigation along the lines as suggested by Briand's critical analysis [7] and Harman's road-map on SBSE [6].

3. AN OVERVIEW OF THE GASTT

The *Genetic Algorithm-based Stress Test Technique (GASTT)* [4] is a stress test technique to derive test requirements to stress test a DRTS with maximizing the network traffic in a way that will most likely reveal RT faults. Due to space constraints, we only present an overview of GASTT [4] crucial for the presentation of the current paper. For comprehensive details on GASTT, readers are referred to [4].

Based on a comparative analysis in [4], we showed that the solution space of stress test requirements for DRTSs is uneven, characterized by multiple peaks and valleys. A Non-Linear Programming (NLP) technique was thus needed that alleviates the above problem by exploring multiple parts of the non-linear problem space. Among existing meta-heuristic optimization methods, we adopted GAs for our test requirement generation problem. This decision was based on a few rationale including the high scalability and flexibility of GAs.

Two of the design details of GASTT [4] which are crucial for the understanding of the empirical analysis in this paper are: *Arrival Patterns (APs)* of RT tasks, and the *maximum search time* for the GA designed for GASTT.

Task APs are common to DRTSs as they impose constraints on when RT tasks are released (available to start execution), e.g., periodic, and bounded [1]. Without considering those APs in the generation of stress test requirements, stress testing would derive strenuous but they would be *invalid (illegal)* execution scenarios w.r.t. task arrival times. To support automation of APs analysis and incorporating them in stress test case generation, the notion of *Accepted Time Set (ATS)* was defined in [4] for each RT task. The ATS of each task is the set of time instances or time intervals when the task is allowed to be triggered (started), according to its AP. For example the ATS of a task with a *bounded* AP is illustrated in Figure 1. In a *bounded* AP, the inter-arrival time of two consecutive arrivals of a task is bounded by a minimum and a maximum time value. The type and the parameters of the AP illustrated in Figure 1 are (*'bounded', (4, ms), (5, ms))*. In this AP, the minimum and maximum inter-arrival times of two consecutive arrivals are specified as 4 and 5 ms, respectively. The gray eclipses in Figure 1 depict the ATS of the above task, i.e., the time intervals where the task AP is satisfied

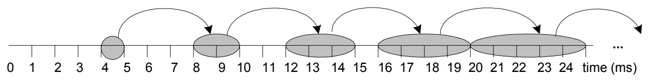
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 time (ms)
...

Figure 1- Accepted Time Set (ATS) of a RT task.

On the other hand, one important issue in the GA design in [4] was the range of the random time values chosen from the ATSs of tasks with APs. As discussed in [4], the ATSs of some types of APs (e.g. periodic, bounded) can be infinite. Therefore, choosing a random value from such an ATS can yield very large values, thus creating implementation problems.

As another direct impact of such unboundedness on our GA [4] is that if the maximum range when generating a set of random numbers is infinite, the probability that all (or a subset) of the generated time values are relatively close to each other is very small. Thus, to eliminate such problems, we introduced in [4] a parameter referred to as the *Maximum Search Time* for our GA. This maximum search time is essentially an integer value (in time units) which enforces an upper bound on the selection of random values for start times of tasks, chosen from their ATSs. The GA maximum search time was used in the GA operators to limit the maximum ranges of generated random time values.

One of the criteria used in our empirical analysis (Section 4) will be to assess the impacts of variations in task APs and the GA maximum search time on the GA performance and repeatability.

4. EMPIRICAL ANALYSIS

To support the application of the GASTT methodology, we implemented a prototype tool called *GARUS (Genetic Algorithm-based test Requirement tool for real-time distribUted Systems)* [17]. This section presents a carefully designed empirical study, using this tool, to validate the design choices of the GA underlying GASTT. A short functional overview for the tool is provided below (Figure 2) but technical details about the tool can be found in [18]. The tool was implemented in C++ and the source code is available from the World Wide Web [17]. The library used to implement the GA-based tool was GAlib [19].

The (stress) test model of a SUT is given in an input file. Such a test model is built from the UML design models of a SUT. The test model, for example, captures different Control Flow Paths (CFPs), and different APs of RT tasks. The tool reads the test model from the input file and creates an object named *tm* of type *TestModel*, initialized with the values from the input test model. Then, an object named *ga* of type *GAlib::SteadyStateGA* is created, such that *tm* is used in the creation of *ga*'s initial population (details in [18]). Note that object *ga* has a collection of chromosomes of type *GARUSGenome,* and each object of type *GARUSGenome* has an ordered set of genes of type *GARUSGene* (these are classes in the tool's class diagram [18]). Furthermore, appropriate values for the *ga*'s parameters (e.g., crossover rate) are set according to the empirical analysis reported in this work. GARUS then evolves *ga* using the defined mutation and crossover operators. When the evolution of *ga* finishes, i.e., after a predefined number of generations, the best individual is saved in an output file.

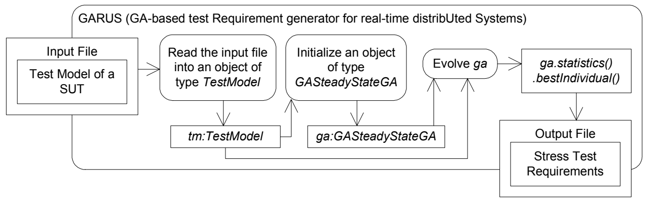
GARUS (GA-based test Requirement generator for real-time distribUted Systems)
Input File
Test Model of a SUT
Read the input file into an object of type TestModel
Initialize an object of type GASteadyStateGA
Evolve ga
ga.statistics() .bestIndividual()
tm:TestModel
ga:GASteadyStateGA
Output File
Stress Test Requirements

Figure 2-A high-level activity diagram of GARUS tool [17].

Along with the output stress test requirement (i.e., the most stressing control flow paths of the RT tasks), GARUS also generates a maximum traffic value (for that test requirement) and a maximum traffic time (when the maximum traffic occurs). The maximum traffic value is in fact the objective function value of the GA's best individual at the completion of the evolution process. The objective function is described in [4] and is referred to as *Instant Stress Test Objective Function (ISTOF)*.

4.1 Evaluation Criteria

In our empirical analysis, we evaluate the GASTT's operating performance and efficiency according to four criteria:

1. *Repeatability of GA results across multiple runs* (Section 4.3)*:* It is important to assess how stable and reliable the results of the GA are across multiple runs. To do so, the GA was executed a large number of times and we assessed the variability of the best chromosome's fitness value.

2. *Convergence efficiency across generations towards a stable maximum plateau* (Section 4.4): In order to assess the design of the selected mutation and cross-over operators in GASTT, as well as the chosen chromosome representation, it is useful to look at the speed of convergence towards a stable maximum fitness plateau [20]. This can be easily computed as, for each generation, GAlib [19] statistics provide min, max, mean, and standard deviation values.

3. *Scalability of the GA (i.e., impacts of variations in size and complexity of a test model)* - (Section 4.5): Assessing how the GA performance and repeatability are affected with SUTs of different size and complexity.

4. *Impacts of variations in parameters other than SUT size and complexity* (Section 4.6-4.7): Assessing how the GA is affected when it is applied to different SUTs with different properties. In our analysis, we investigated the impacts of variations in AP types, arrival pattern parameters (e.g., periodic arrival pattern period and deviation values), and the GA maximum search time (Section 3). Due to space constraints, the impacts of variations in task arrival pattern parameters are not presented in this paper, but can be found in [18].

Note that the items 1-3 above are generic cross-cutting concerns which heuristics-based SBSE techniques should, in general, be empirically evaluated to. However, the 4th item is specific to the technique at hand (stress testing).

4.2 Experimental Test Models

Using the above four criteria (Section 4.1), we analyzed GASTT's operating performance and efficiency by running GARUS on a set of 20 hypothetical *experimental* test models (described below), which were used as a test-bed for our experiments.

Note that we chose to use experimental test models in our analysis, since there are not many publicly-available real or prototype DRTS design models or source codes. Furthermore, the use of experimental test models enabled us to conveniently change different empirical parameters (e.g., the SUT complexity) and to accordingly study their impacts on the GA. The set of 20 experimental Test Models (TMs) were designed based on the following three variation criteria:

1. Variations in test model (SUT) size and complexity
2. Variations in AP types and parameters
3. Variations in the GA's maximum search time

We devised a set of *variability parameters*, and used them in our experiments to incorporate variability in different TMs based on the above criteria. Eight of those variability parameters are shown in Table 1 which correspond to the above "SUT size and complexity" criterion (#1). Each parameter in this group corresponds to a size/complexity perspective of a SUT, e.g., number of *Independent SD Sets (ISDSs)*[1], number of SDs (each corresponding to a RT task), and minimum/maximum numbers of *Distributed Concurrent Control Flow Path* (*DCCFPs*)[2] per SD. A *DTUPP (Distributed Traffic Usage Pattern Point)* is the predicted traffic usage value of a RT task (when executed) at each time instant. A large SUT might have many ISDSs (by setting large values for *nISDSs*), while another large SUT can have many DCCFPs per SD (by setting large values for *minnDCCFPs* and *maxnDCCFPs*). Parameters prefixed with *min* and *max* serve as statistical means, which enable us to incorporate statistically-controlled randomness into the sizes of our experimental TMs. For example, we can control the minimum and maximum number of DCCFPs per SD in a SUT by *minnDCCFPs* and *maxnDCCFPs* parameters. Such a statistical range for number of SDs per ISDSs, DCCFPs per SD, and DTUPPs per DCCFP also conforms to real-world models, where for example, there are variant numbers of DCCFPs per SDs.

Six TMs (*tm1…tm6*) of different size and complexity were generated based on the statistical information in Table 1 and using a random test model generator (*RandTMGen*) developed in C++. For example, *tm1* (Table 1) has 2 ISDSs, 5 SDs (i.e., RT tasks), has min/max ISDS size of 3 and 4 SDs, respectively, and so on.

Test Models / Parameters	*tm1*	*tm2*	*tm3*	*tm4*	*tm5*	*tm6*
nISDSs	2	**100**	10	10	10	2
nSDs	5	50	**200**	50	10	5
minISDSsize	3	2	2	**20**	2	2
maxISDSsize	4	5	5	**30**	5	5
minnDCCFPs	1	1	2	1	**10**	1
maxnDCCFPs	5	3	5	3	**50**	5
minnDTUPPs	2	1	1	1	1	**50**
maxnDTUPPs	6	10	10	10	10	**100**

Table 1-Six experimental test models.

12 other TMs (*tm7…tm18*) were generated using *RandTMGen* based on the variation criterion #2, and the following sub-criteria:

- *Different-AP-Types* (resulted in *tm7…tm11*): TMs in which tasks had either no, same or different AP types, e.g., none of the RT tasks in *tm7* had specific APs. In *tm8*, *tm9* and *tm10*, all tasks had periodic, bounded, and irregular APs, respectively. *tm11* had tasks with different APs.

- *Same-AP-Different-Parameters* (resulted in *tm12…tm18*)*:* Each TM group in this set of TMs had a same AP type, but different AP parameters (e.g., period values of a periodic AP).

All *tm7…tm18* had the same size and complexity parameters. To incorporate the variation criteria #3, two other TMs (*tm19* and *tm20*) were generated whose size and complexity were the same as *tm1* but their GA maximum search times were 5 and 150 time units, respectively.

[1] As defined in [4], an ISDS is a largest (maximal) set of SDs, in which any two SDs are independent, thus enabling all the SDs in the set to run concurrently.

[2] A DCCFP is a *Concurrent Control Flow Path (CCFP)* where all messages are distributed [4]. A CCFP is a generalized form of conventional Control Flow Path in which messages can be triggered concurrently.

4.3 Repeatability of the GA Results

We investigated the repeatability of GA results by analyzing the variation in maximum objective function (ISTOF) values and maximum-traffic times of the best GA individual (chromosome) after the GASTT execution was finished, and then assessing the extent to which those values were repeatable.

Figure 3-(a) depicts the distributions of maximum ISTOF and stress time values for 1000 runs of *tm1* (Section 4.2). From the ISTOF distribution, we can see that the maximum fitness values for most of the runs are between 60 and 72 units of traffic (e.g., KB). Descriptive statistics of the distributions in Figure 3 and other figures in this paper can be found in [18].

Such a variation in fitness values across runs is expected when using GAs on complex optimization problems such as GASTT. However, though the variation above is not negligible, one would expect based on Figure 3-(a) that, with a few runs, a chromosome with a fitness value close to the observed maximum would likely be identified. Since each run lasts a few seconds (Section 4.5), relying on multiple runs to generate a stress test requirement should perhaps take a few minutes for very large SUTs and should not lead to practical problems.

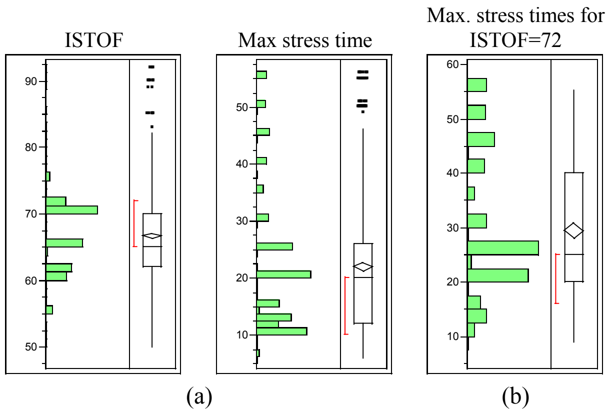

Figure 3-Empirical data for repeatability of the GA results.

Corresponding portions of max stress time values for the most frequent maximum ISTOF value (72 units of traffic) are shown in Figure 3-(b). As we can see, these maximum stress time values are scattered across the time scale (e.g., from 10 to 60 units of time). This highlights that a single ISTOF value (maximum stress traffic) can happen in different time instances, thus suggesting the search landscape for the GA is rather complex for this type of problem. Thus, a testing strategy to further explore would be to cover all (or a subset of) such test requirements with maximum ISTOF values in different time instances. Indeed, although their ISTOF values are the same, a SUT's reaction to different test requirements might vary, since different DCCFPs (and hence set of messages) in different time instances may be triggered. This might in turn lead to uncovering different RT faults in the SUT.

4.4 Convergence Efficiency of the GA

Another interesting property of the GA is the number of generations required to reach a stable maximum fitness plateau. We empirically analyzed the GA's convergence efficiency w.r.t. various GA configuration parameters (e.g., crossover rate, mutation rate, and population size) [18], but due to space constraint, we only report the results w.r.t. the GA's crossover rate.

The distributions of these generation numbers over 1000 runs of *tm1* for two different GA crossover rates of %70 and %50 are shown in Figure 4, where the x-axis is the generation number and the y-axis is the probability of reaching a stable maximum fitness plateau for a given generation number. We discuss next how this part of our empirical analysis was used to better calibrate the parameters of the GASTT (in specific, its crossover rate) to generate efficient and reliable results.

Selecting an appropriate crossover rate for every GA is critical [21]. If the crossover rates are too high, desirable genes will not be able to accumulate within a single chromosome whereas if the rates are too low, the search space will not be fully explored [21]. The work by Grefenstette et al. [22] recommends that crossover rates should range between %45 and %95. Consistent with the findings of Grefenstette, we applied several choices in the above range (%50 and %70 were two of them) and used our empirical analysis to choose the most appropriate one as described below.

Note that, in both the above cases, the number of GA generations (stopping criteria) was fixed to 100 which was itself calibrated through another empirical analysis [18]. According to Figure 4, for the crossover rate=%50, we can see that in 100th generations, the GA does not converge in all 1000 runs, i.e., the probability of achieving a stable maximum fitness plateau after 100 generations is %0.129. Although this is a small probability, but it can undermine the GA's performance in producing reliable and repeatable test requirements.

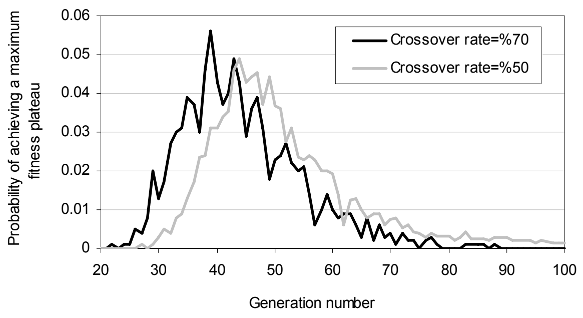

Figure 4-Histogram of the generation numbers when a stable maximum fitness plateau is reached.

On the other hand, for the crossover rate=%70, no more than 100 generations was required to efficiently converge to a maximum fitness plateau. In this case, the minimum, maximum and average values are 20, 91, and 52, respectively. Therefore, we can state that in this case, on the average, 52 generations of the GA are required to converge to the final result (stress test requirement). In this case, the variation around this average is limited and no more than 100 generations will be required (even for our large experimental models). This number is in conformance with the experiments reported in the GA literature (e.g., [9]) but is however likely to be dependent on the number and complexity of SDs as well as their ATSs.

We thus decided to choose crossover rate=%70 for the GA at hand. But note that this rate might need to be revised depending on the different parameters (e.g., number of GA generations).

After setting crossover rate=%70, we further observed in our experiments that, from the initial to the final populations, the maximum fitness values typically increase by about 80%, which can be considered a large improvement. So, though we cannot guarantee that a GA does find the global maximum, it clearly generates test requirements that is close to the global maximum and will significantly stress the SUT.

4.5 Scalability of the GA

The GA's scalability was evaluated by analyzing the impact of variations in TM size and complexity using the following four empirical objectives:

a) GA execution time
b) Repeatability of maximum ISTOF values
c) Repeatability of maximum stress time values
d) Convergence efficiency across generations

It should be noted that the goals of the empirical objectives (b)-(d) above differ from our empirical analyses in Sections 4.3 and 4.4 in that the objectives (b)-(d) aim at analyzing the repeatability and convergence efficiency aspects of the GA w.r.t. its scalability aspect, i.e., how are the repeatability and convergence efficiency of the GA affected for TMs of different size and complexity? While, the analyses in Sections 4.3 and 4.4 investigated the repeatability and convergence efficiency for multiple runs of a single TM (i.e., *tm1*).

Due to space constraints, we report next only the results of the above empirical objective (a). The empirical outcomes of the other three objectives are reported in [18]. To investigate the impact of TM size and complexity on the execution time of the GA, the average execution times over 1000 runs, by running GARUS with *tm1…tm6* on an 863MHz Intel Pentium III processor with 512MB DRAM memory are depicted in Figure 5. Since minimums and maximums of the statistics for each TM were relatively close to the corresponding average value, we use the average values to discuss next the impacts of TM size on the GA execution time.

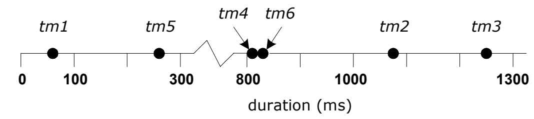

Figure 5-The average GA execution times of experimental test models *tm1…tm6*.

Average duration of the GA run of *tm1* (58 ms) is the smallest among all. This is expected since *tm1* has the smallest size in terms of TM components (ISDSs, SDs, and DCCFPs). *tm3* has the highest average execution time among the six TM runs. Durations of *tm2*, *tm6*, *tm4*, and *tm5* are next in decreasing order. Based on the above order of execution values, we can make the following observations:

- The execution time of the GA is strongly sensitive to an increase in number of SDs in a TM. The more SDs in a TM, the longer a single run of the GA takes (e.g., *tm3*). This can be explained as the number of genes per chromosome in the GA is the same as the number of SDs in a TM. Thus, as the execution results indicate, the execution time of our GA sharply increases when the number of genes per chromosome increases. Such an increase impacts all functional components of the GA, including its operators and the fitness evaluator [4].
- As expected, the execution time of our GA is also highly dependent on the number of ISDSs (e.g. *tm2*). As the number of ISDSs increases, the size of initial population grows, and so does the number of the mutations and crossovers applied in each generation. The number of times the operators are applied is determined by the mutation and crossover rates and the size of initial population.
- The execution time of the GA is also dependent on an increase in number of SDs per ISDS (e.g., *tm4*), as well as an increase in number of DTUPPs per DCCFP (e.g., *tm6*). As the number of SDs per ISDS increases, the number of non-null genes per chromosome will increase. This will, in turn, lead to more mutations and crossovers and an increase in computation for the fitness evaluator. Similarly, an increase in number of DTUPPs per DCCFP will lead to an increase in fitness function's computation time.
- The execution time of the GA is not as dependent on an increase in number of DCCFPs per SD (e.g., *tm5*), when compared to other TM components. This can be explained as there will not be any change in chromosome size, nor in the initial population in that case. Even the frequency of mutations and crossovers will not change. For example, as the mutation operator chooses a random DCCFP among all DCCFPs of a SD, there will be no effect in terms of execution time if the number of DCCFPs per SD increases. The small difference between average durations of *tm5* and *tm1* in Figure 5 is due to the fact that *tm5's* number of SDs is slightly more than that of *tm1*.

In summary, the following are three high-level observations from our experiments on the scalability analysis of the GA:

1. As the size of the test model gets larger, the variation in maximum ISTOF values (objective function) across executions remains constant.
2. The GA can reach a stable maximum plateau even when the size of a specific component (SD, ISDS, DCCFP, etc) of a given model is large (up to 100 ISDSs in a SUT, 200 SDs, 30 SDs (RT tasks) in an ISDS, and 50 DCCFPs in a SD).
3. Test model size does not have an impact on the convergence efficiency across generations, and the GA is able to reach a stable maximum fitness plateau after about 50 generations on average, independent of test model size.

4.6 Impacts of Arrival Pattern Types

Similar to the GA scalability empirical analysis (Section 4.5), we analyzed the impacts of different arrival pattern types on the GA w.r.t. the following four empirical objectives:

a) GA execution time
b) Repeatability of maximum ISTOF values
c) Repeatability of maximum stress time values
d) Convergence efficiency across generations

Due to space constraints, we report next only the results of the above empirical objective (a).

We measured the average, minimum and maximum execution times over all the 1000 runs, by running GARUS with test models *tm7…tm11* on a PC with the same specifications as described in Section 4.5. According to our empirical results, minimums and maximums of the above statistics for each test model run were relatively close to the corresponding average value. Therefore, we use the average values next to discuss the impacts of variations in arrival patterns on GA execution time. To better illustrate the differences, the average values are depicted in Figure 6.

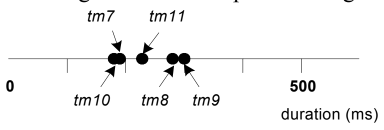

Figure 6-Average values of GA execution times for five experimental test models.

Recall from Section 4.2 that test models *tm7…tm11* were designed in a way that they all had the same size and complexity parameters, but different arrival patterns for their RT tasks. Referring to Figure 6, the average execution times of *tm7…tm11* are relatively close to each other (within 110 ms). This indicates that execution time is not strongly dependent on task arrival pattern "types" in a SUT. Furthermore, as we discuss below, the difference in execution times are mainly due to the implementation details of a method of class *AP* in GARUS.
The execution times of two of these test models (*tm8* and *tm9*), are slightly higher than those of *tm7* and *tm10*. The difference between the two TM groups (*tm8* and *tm9* versus *tm7* and *tm10*) can be explained by an implementation detail of GARUS (the source code is available in [17]). Function `getARandomArrivalTime`, a member function of class *AP*, is overridden in each of *AP*'s subclasses (e.g., *periodicAP*, and *boundedAP*). The time complexity of this function in *noAP* and *irregularAP* classes is *O(1)*, i.e., choosing a random value from a range or an array, respectively. However, the implementation of the function in *periodicAP* and *boundedAP* classes required some extra considerations (related to the ATSs of periodic and bounded APs), and thus the time complexities of the function are not constant anymore, but dependent on the specific arrival pattern parameters.

The execution time of *tm11*, in which each task can have an arbitrary arrival pattern, is placed somehow close to the average value of the other four TMs (*tm7*, *tm8, tm9,* and *tm10*). This is as predicted since the APs of RT tasks in *tm11* are a mix of APs in the other four, thus leading to such an impact in its average execution time.

4.7 Impacts of Maximum Search Time

We report in this section the impact of variations in GA maximum search time on execution time, repeatability of outputs (maximum ISTOF values), and also the number of generations to reach a stable maximum plateau. Recall from Section 3 that the GA maximum search time is a GA parameter that limits the range of the random time values chosen from the ATSs of RT tasks with arrival pattern in the SUT.

We compare the GA results w.r.t the above three criteria for *tm1*, *tm19* and *tm20* in Figure 7. As described in Section 4.2, *tm19* and *tm20* have the same size and complexity as *tm1*, but the maximum search time values for *tm19* and *tm20* are 5 and 150 time units, respectively, instead of 50 in *tm1*. Therefore, comparing GA results for these three TMs should reveal the impact of variations in maximum search time. There are 9 graphs (3 rows in 3 columns) in Figure 7, where rows and columns correspond to different maximum search times, and GA performance variables.

In terms of execution time, it seems that variations in maximum search time do not have a considerable impact. Across 1000 runs, all three TMs (*tm1*, *tm19* and *tm20*) show execution times in the range [45 ms, 130 ms]. Since a change in maximum search time only changes the range in which a random time from an ATS (of a task) is selected, it is not surprising that there is no effect on the workload of different GA components (in the GARUS tool [17]).

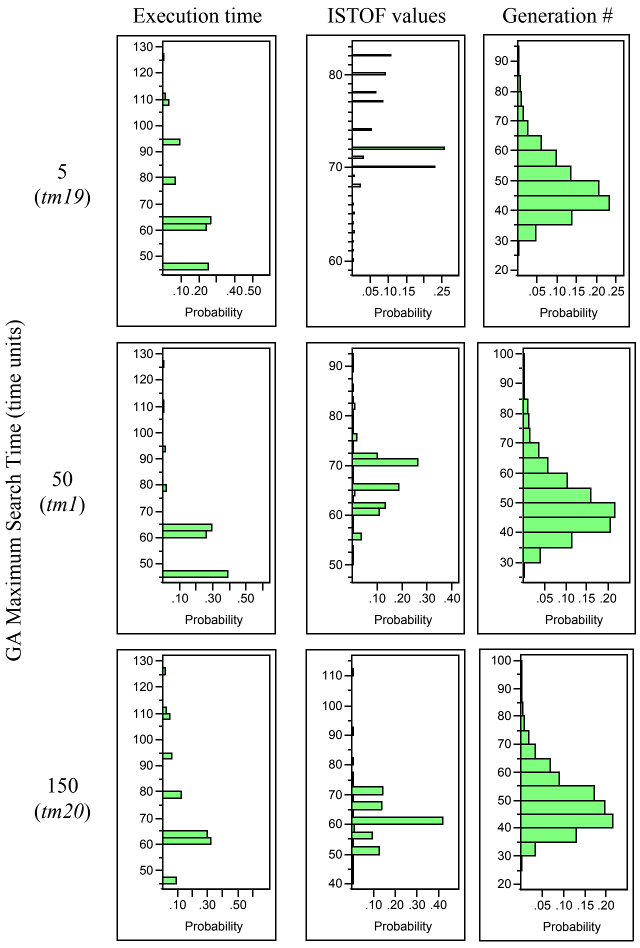

Figure 7- Impact of variations in maximum search time.

As the maximum search time increases across the three test models (5 in *tm19* to 50 in *tm1* and 150 in *tm20*), the maximum of maximum ISTOF values across 1000 runs of a TM increases, i.e. 82 traffic units for *tm19*, 91 traffic units for *tm1* and 110 traffic units for *tm20.* This can be explained by an increase in the size of GA's time search range from *tm19* to *tm1* and *tm20*: a larger time search range allows the GA to search a more broad time range to find the best chromosome (i.e., stress test requirement). From another perspective, the difference between the maximum and minimum of maximum ISTOF values also increases with the maximum search time. The differences between the maximum and minimum of maximum ISTOF values for *tm19*, *tm1* and *tm20* are 20 (82-62), 41 (91-50), and 69 (110-41) respectively. This can also be explained by the increase in the size of time search range.

In terms of number of generations to reach a stable maximum plateau, we can see that the increase in maximum search time slightly *delays* convergence across generations, i.e., the maximum plateau generation number in *tm19* runs is reached at 91, while it is 99 for both *tm1* and *tm20* runs.

5. CONCLUSIONS AND FUTURE WORKS

We presented in [4] a GA-based UML-driven, stress test technique aimed at increasing chances of discovering faults related to network traffic in distributed real-time software. This paper reported a carefully-designed empirical study which was conducted to analyze and improve the applicability, efficiency and effectiveness of the above GA-based stress test technique when applied to distributed real-time software.

In this empirical analysis, we evaluated the above GA's operating performance and efficiency according to four criteria: (1) Repeatability of GA results across multiple, (2) Convergence efficiency across generations towards a stable maximum plateau, (3) Impacts of variations in size and complexity of a SUT (Scalability of the GA), and (4) Impacts of variations in parameters other than SUT size and complexity.

We presented in Section 4.4 one example scenario of how the findings of this study can be used to better calibrate a parameter (i.e., crossover rate) of the GA-based stress test technique in [4]. Other findings from our analysis were also used to calibrate other parameters of the GA, such as the population size, and mutation rate. More comprehensive details can be found in [18].

As a future work direction, we plan to perform similar empirical analyses for other evolutionary test techniques. We also plan to adopt and use more sophisticated ideas for empirical analysis form the operations research community (e.g., [8-10]).

6. ACKNOWLEDGMENTS

This work was supported in part by the Alberta Ingenuity New Faculty Award no. 200600673 and also by the Discovery Grant no. 341511-07 from the Natural Sciences and Engineering Research Council of Canada (NSERC).

7. REFERENCES

[1] J. J. P. Tsai, Y. Bi, S. J. H. Yang, and R. A. W. Smith, *Distributed Real-Time Systems: Monitoring, Visualization, Debugging, and Analysis*: John Wiley & Sons, 1996.

[2] R. Kuhn, "Sources of Failure in the Public Switched Telephone Network," *IEEE Computer,* vol. 30, no. 4, pp. 31-36, 1997.

[3] Object Management Group (OMG), "UML 2.1.1 Superstructure Specification," 2007.

[4] V. Garousi, L. Briand, and Y. Labiche, "Traffic-aware Stress Testing of Distributed Real-Time Systems Based on UML Models using Genetic Algorithms," *Elsevier Journal of Systems and Software, Special Issue on Model-Based Software Testing,* vol. 81, no. 2, pp. 161-185, 2008.

[5] V. Garousi, L. Briand, and Y. Labiche, "Traffic-aware Stress Testing of Distributed Systems based on UML Models," *Proc. of Int. Conf. on Software Engineering*, pp. 391-400, 2006.

[6] M. Harman, "The current State and Future of Search-Based Software Engineering," *Proc. of Int. Conf. on Software Engineering, Future of Software Engineering*, pp. 342-357, 2007.

[7] L. Briand, "A Critical Analysis of Empirical Research in Software Testing," *Keynote address, Int. Symp. on Empirical Software Engineering and Measurement,* 2007.

[8] Y. Min, X. Jin, X. Su, and B. Peng, "Empirical Analysis of the Spatial Genetic Algorithm on Small-World Networks," *Proc. of Int. Conf. on Computational Science*, pp. 1032-1039, 2006.

[9] G. Rudolph, "Convergence Analysis of Canonical Genetic Algorithms," *IEEE Transactions on Neural Networks,* vol. 5, no. 1, pp. 96-101, 1994.

[10] K. Sastry, M. Pelikan, and D. E. Goldberg, "Empirical Analysis of Ideal Recombination on Random Decomposable Problems," *Genetic and Evolutionary Computation Conference*, pp. 1388-1395, 2007.

[11] G. Antoniol, M. Di Penta, and M. Harman, "Search-Based Techniques Applied to Optimization of Project Planning for a Massive Maintenance Project," *Proc. of Int. Conf. on Software Maintenance*, pp. 240-249, 2005.

[12] M. Harman, Y. Hassoun, K. Lakhotia, P. McMinn, and J. Wegener, "The Impact of Input Domain Reduction on Search-Based Test Data Generation," *Proc. of Int. Symp. on the Foundations of Software Engineering*, pp. 155-164, 2007

[13] M. Harman and P. McMinn, "A Theoretical & Empirical Analysis of Evolutionary Testing and Hill Climbing for Structural Test Data Generation " *Proc. of Int. Symp. on Software Testing and Analysis* pp. 73-83, 2007.

[14] M. Xiao, M. E. M. Reformat, and J. Miller, "Empirical Evaluation of Optimization Algorithms when used in Goal-Oriented Automated Test Data Generation Techniques," *Empirical Software Engineering,* vol. 12, no. 2, pp. 183-239, 2007

[15] K. De Jong, "The Analysis of the Behavior of a Class of Genetic Adaptive Systems," Ph.D. Dissertation, Dept. of Computer Science, University of Michigan, Ann Arbor, 1975.

[16] J. J. Greffenstette, "Optimization of Control Parameters for Genetic Algorithms," *IEEE Trans. on Systems, Man, and Cybernetics,* vol. 16, no. 1, pp. 122-128, 1986.

[17] V. Garousi, "GARUS (Genetic Algorithm-based test Requirement tool for real-time distribUted Systems)," *http://www.enel.ucalgary.ca/~vgarousi/tools/GARUS*, 2006.

[18] V. Garousi, "Traffic-aware Stress Testing of Distributed Real-Time Systems Based on UML Models using Genetic Algorithms," Ph.D. Thesis, Department of Systems and Computer Engineering, Carleton University, 2006.

[19] M. Wall, "GAlib: A C++ Library of Genetic Algorithm Components," Documentation version 2.4, Massachusetts Institute of Technology 1996.

[20] S. J. Louis and G. J. E. Rawlins, "Predicting Convergence Time for Genetic Algorithms," Technical Report 370, Computer Science Department, Indiana University 1993.

[21] R. L. Haupt and S. E. Haupt, *Practical Genetic Algorithms*: Wiley-Interscience, 1998.

[22] J. J. Grefenstette and H. G. Cobb, "Genetic Algorithms for Tracking Changing Environments," *Proceeding of International Conference on Genetic Algorithms*, pp. 523-530, 1993.