A multi-objective approach to search-based stress testing

Nauber Gois
Serviço Federal
de Processamento de Dados
Av. Pontes Vieira, 832
Fortaleza - CE, Brazil, 60130-240
Email: francisco.gois@serpro.gov.br

Pedro Porfírio
Universidade de Fortaleza
Av. Washington Soares, 1321
Fortaleza - CE, Brazil, 60811-905
Email: porfirio@unifor.br

André Coelho
Universidade de Fortaleza
Av. Washington Soares, 1321
Fortaleza - CE, Brazil, 60811-905
Email: acoelho@unifor.br

Abstract—**Nowadays applications need to deal with a large number of concurrent requests. These systems must be stress tested to ensure that they can function correctly under load. In this context, a research field called Search based Software Testing has become increasingly important. Most of the search-based test methods are based of single objective optimization. In case of multi objective optimization of tests, usually researchers assign different weight values to different objectives and combine them as single objective. This paper research to verify the use of multi-objective algorithm in search-based stress testing. The NSGA-II algorithm was implemented in the IAdapter tool using the jMetal framework. IAdapter is a JMeter plugin used for performing search-based stress tests. jMetal is an object-oriented Java-based framework for multi-objective optimization with metaheuristics. One experiment was conducted to validate the proposed approach.**

I. Introduction

Performance problems such as high response times in software applications have a significant effect on the customer's satisfaction. The explosive growth of the Internet has contributed to the increased need for applications that perform at an appropriate speed. Performance problems are often detected late in the application life cycle, and the later they are discovered, the greater the cost to fix them. The use of stress testing is an increasingly common practice owing to the increasing number of users. In this scenario, the inadequate treatment of a workload generated by concurrent or simultaneous access due to several users can result in highly critical failures and negatively affect the customers perception of the company [1] [2] [3] [4].

Software testing is a expensive and difficult activity. The exponential growth in the complexity of software makes the cost of testing has only continued to rise. Test case generation can be seen as a search problem. The test adequacy criterion is transformed into a fitness function and a set of solutions in the search space are evaluated with respect to the fitness function using a metaheuristic search technique. Search-based software testing is the application of metaheuristic search techniques to generate software tests cases or perform test execution [5] [6].

Search-based testing is seen as a promising approach to verifying timing constraints [5]. A common objective of a load search-based test is to find scenarios that produce execution times that violate the specified timing constraints [7]. Experiments involving search based tests are inherently complex and typically time-consuming to set up and execute. Such experiments are also extremely difficult to repeat. People who might want to duplicate published results, for example, must devote substantial resources to setting up and the environmental conditions are likely to be substantially different.

Ussually, search-based test methods are based of single objective optimization. Among them, multiobjective evolutionary algorithms (MOEAs) are widely used for solving MOPs because they produce a complete set of solutions in a single run. The NSGA-II is a GA based on obtaining a new offspring population from the original one by applying the typical genetic operators (selection, crossover, and mutation); then, the individuals in the two populations are sorted according to their rank, and the best solutions are chosen to create a new population.

This paper addresses to validade the use of an multi-objective algorithm in stress testing problem. A tool named IAdapter (www.iadapter.org, github.com/naubergois/newiadapter), a JMeter plugin for performing search-based load tests, was extended [8]. One experiment was conducted to validate the proposed approach. The experiment use NSGA-II algorithm with one objective: Discover application scenarios where there is a high response time for a small number of users. The relevance of finding scenarios with high response times is to enable corrective actions before the application under test is released to a production environment.

The remainder of the paper is organized as follows. Section 2 presents a brief introduction about load, performance, and stress tests. Section 3 presents concepts about the workload model. Section 4 presents details features about search-based stress testing. Section 5 presents concepts about NSGA-II multi-onjective algorithm. Section 6 presents concepts about the proposed solution. Section 7 shows the results of the experiment performed. Conclusions and further work are presented in Section 8.

II. Load, Performance and Stress Testing

Load, performance, and stress testing are typically done to locate bottlenecks in a system, to support a performance-tuning effort, and to collect other performance-related indicators to help stakeholders get informed about the quality of the application being tested [9] [10].

The performance testing aims at verifying a specified system performance. This kind of test is executed by simulating hundreds of simultaneous users or more over a defined time interval [11]. The purpose of this assessment is to demonstrate that the system reaches its performance objectives [9].

In a load testing, the system is evaluated at predefined load levels [11]. The aim of this test is to determine whether the system can reach its performance targets for availability, concurrency, throughput, and response time. Load testing is the closest to real application use [3]. A typical load test can last from several hours to a few days, during which system behavior data like execution logs and various metrics are collected [5].

The stress testing verifies the system behavior against heavy workloads [9], which are executed to evaluate a system beyond its limits, validate system response in activity peaks, and verify whether the system is able to recover from these conditions. It differs from other kinds of testing in that the system is executed on or beyond its breakpoints, forcing the application or the supporting infrastructure to fail [11] [3].

While load testing is the process of assessing non-functional quality related problems under load; performance testing is used to measure and/or evaluate performance related aspects (e.g., response time, throughput and resource utilizations) of algorithms, designs/architectures, modules, configurations, or the overall systems and stress tests puts a system under extreme conditions to verify the robustness of the system and/or detect various functional bugs (e.g., memory leaks and deadlocks) [5].

III. Workload Modeling

The design of a stress test depends intrinsically on the load model applied to the software under test. Based on the objectives, there are two general schools of thought for designing a proper load to achieve such objectives [5]:

- Designing Realistic Loads (Workload Descriptive).
- Designing Fault-Inducing Loads (Workload Generative).

In Designing Realistic Loads, the main goal of testing is to ensure that the system can function correctly once. Designing Fault-Inducing Loads aims to design loads, which are likely to cause functional or non-functional problems [5].

Stress testing projects should start with the development of a model for user workload that an application receives. This should take into consideration various performance aspects of the application and the infrastructure that a given workload will impact. A workload is a key component of such a model [3].

The term workload represents the size of the demand that will be imposed on the application under test in an execution. The metric used for measure a workload is dependent on the application domain, such as the length of the video in a transcoding application for multimedia files or the size of the input files in a file compression application [12] [3] [13].

Workload is also defined by the load distribution between the identified transactions at a given time. Workload helps researchers study the system behavior identified in several load models. A workload model can be designed to verify the predictability, repeatability, and scalability of a system [12] [3]. Workload modeling is the attempt to create a simple and generic model that can then be used to generate synthetic workloads. The goal is typically to be able to create workloads that can be used in performance evaluation studies. Sometimes, the synthetic workload is supposed to be similar to those that occur in practice in real systems [12] [3].

There are two kinds of workload models: descriptive and generative. The main difference between the two is that descriptive models just try to mimic the phenomena observed in the workload, whereas generative models try to emulate the process that generated the workload in the first place [11].

In descriptive models, one finds different levels of abstraction on the one hand and different levels of fidelity to the original data on the other hand. The most strictly faithful models try to mimic the data directly using the statistical distribution of the data. The most common strategy used in descriptive modeling is to create a statistical model of an observed workload. This model is applied to all the workload attributes, e.g., computation, memory usage, I/O behavior, communication, etc. [11].

Generative models are indirect in the sense that they do not model the statistical distributions. Instead, they describe how users will behave when they generate the workload. An important benefit of the generative approach is that it facilitates manipulations of the workload. It is often desirable to be able to change the workload conditions as part of the evaluation. Descriptive models do not offer any option regarding how to do so. With the generative models, however, we can modify the workload-generation process to fit the desired conditions [11]. The difference between the workflows of the descriptive and the generative models is that user behavior is not collected from logs, but simulated from a model that can receive feedback from the test execution.

Both load model have their advantages and disadvantages. In general, loads resulting from realistic-load based design techniques (Descriptive models) can be used to detect both functional and non-functional problems. However, the test durations are usually longer and the test analysis is more difficult. Loads resulting from fault-inducing load design techniques (Generative models) take less time to uncover potential functional and non-functional problems, the resulting loads usually only cover a small portion of the testing objectives [2]. The presented research work uses a generative model.

IV. Search-Based Stress Testing

The search for the longest execution time is regarded as a discontinuous, nonlinear, optimization problem, with the input domain of the system under test as a search space [7]. The application of SBST algorithms to stress tests involves finding the best- and worst-case execution times (B/WCET) to determine whether timing constraints are fulfilled [5].

There are two measurement units normally associated with the fitness function in stress test: processor cycles and execution time. The processor cycle approach describes a fitness function in terms of processor cycles. The execution time approach involves executing the application under test and measuring the execution time [5] [14].

Processor cycles measurement is deterministic in the sense that it is independent of system load and results in the same execution times for the same set of input parameters. However, such a measurement is dependent on the compiler and optimizer used, therefore, the processor cycles differ for each platform. Execution time measurement is a non deterministic approach, there is no guarantee to get the same results for the same test inputs [5]. However, stress testing where testers have no access to the production environment should be measured by the execution time measurement [3] [5].

Table I shows a comparison between the research studies on load, performance, and stress tests presented by Afzal et al. [5]. Afzal's work was added with some of the latest research in this area ([15] [16] [17] [18] [19] [8]). The columns represent the type of tool used (prototype or functional tool), and the rows represent the metaheuristic approach used by each research study (genetic algorithm, Tabu search, simulated annealing, or a customized algorithm). The table also sorts the research studies by the type of fitness function used (execution time or processor cycles).

Table I: Distribution of the research studies over the range of applied metaheuristics

	Prototypes		**Functional Tool**
	Execution Time	Processor Cycles	Execution Time
NSGA-II multi-objective algorithm			Our aproach
GA	Alander et al.,1998 [20] Wegener et al., 1996 and 1997 [21][22] Sullivan et al., 1998 [7] Briand et al., 2005 [23] Canfora et al., 2005 [24]	Wegener and Grochtman 1998 [25] Mueller et al., 1998 [26] Puschner et al. [27] Wegener et al., 2000 [28] Gro et al., 2000 [29]	Di Penta, 2007 [30] Garoussi, 2006 [15] Garousi, 2008 [31] Garousi, 2010 [16]
Simulated Annealing (SA)			Tracey, 1998 [32]
Constraint Programming			Alesio, 2014 [18] Alesio, 2013 [17]
GA + Constraint Programming			Alesio, 2015 [19]
GA + SA + Tabu Search			Gois et al. 2016 [8]
Customized Algorithm		Pohlheim, 1999 [33]	

The studies can be grouped into two main groups:

- Search-Based Stress Tesing on Safety-critical systems.
- Search-Based Stress Testing on Industrial systems.

A. Search-Based Stress Tesing on Safety-critical systems

Domains such as avionics, automotive and aerospace feature safety-critical systems, whose failure could result in catastrophic consequences. The importance of software in such systems is permanently increasing due to the need of a higher system flexibility. For this reason, software components of these systems are usually subject to safety certification. In this context, software safety certification has to take into account performance requirements specifying constraints on how the system should react to its environment, and how it should execute on its hardware platform [17].

Usually, embedded computer systems have to fulfil real-time requirements. A faultless function of the systems does not depend only on their logical correctness but also on their temporal correctness. Dynamic aspects like the duration of computations, the memory actually needed during program execution, orthe synchronisation of parallel processes are of major importance for the correct function of real-time systems [22] .

The concurrent nature of embedded software makes the order of external events triggering the systems tasks is often unpredictable. Such increasing software complexity renders performance analysis and testing increasingly challenging. This aspect is reflected by the fact that most existing testing approaches target system functionality rather than performance [17]. Reactive real-time systems must react to external events within time constraints. Triggered tasks must execute within deadlines. Shousha develops a methodology for the derivation of test cases that aims at maximizing the chance of critical deadline misses [34].

The main goal of Search-Based Stress testing of Safety-critical systems it is finding a combination of inputs that causes the system to delay task completion to the greast extent possible [34]. The followed approaches uses metaheuristics to discover the worst-case execution times. Wegener et al. [21] used genetic algorithms(GA) to search for input situations that produce very long or very short execution times. The fitness function used was the execution time of an individual measured in micro seconds [21]. Alander et al. [20] performed experiments in a simulator environment to measure response time extremes of protection relay software using genetic algorithms. The fitness function used was the response time of the tested software. The results showed that GA generated more input cases with longer response times [20].

Wegener and Grochtmann performed a experimentation to compare GA with random testing. The fitness function used was duration of execution measured in processor cycles. The results showed that, with a large number of input parameters, GA obtained more extreme execution times with less or equal testing effort than random testing [22] [25]. Gro et. al. [29] presented a prediction model which can be used to predict evolutionary testability. The research confirmed that there is a relationship between the complexity of a test object and the ability of a search algorithm to produce input parameters according to B/WCET [29]. Briand et al. [23] used GA to find the sequence of arrival times of events for aperiodic tasks, which will cause the greatest delays in the execution of the target task. A prototype tool named real-time test tool (RTTT) was developed to facilitate the execution of runs of genetic algorithm. Two case studies were conducted and results illustrated that RTTT was a useful tool to stress a system under test [23].

Pohlheim and Wegener used an extension of genetic algorithms with multiple sub-populations, each using a different search strategy. The duration of execution measured in processor cycles was taken as the fitness function. The GA found longer execution times for all the given modules in comparison with systematic testing[33]. Garousi presented a stress test methodology aimed at increasing chances of discovering faults related to distributed traffic in distributed systems. The technique uses as input a specified UML 2.0 model of a system, augmented with timing information.The results indicate that the technique is significantly more effective at detecting distributed traffic-related faults when compared to standard test cases based on an operational profile [15].

Alesio, Nejati and Briand describe a approach based on Constraint Programming (CP) to automate the generation of test cases that reveal, or are likely to, task deadline misses. They evaluate it through a comparison with a state-of-the-art approach based on Genetic Algorithms (GA). In particular, wthe study compares CP and GA in five case studies for efficiency, effectiveness, and scalability. The experimental results show that, on the largest and more complex case studies, CP performs significantly better than GA. The research proposes a tool-supported, efficient and effective approach based on CP to generate stress test cases that maximize the likelihood of task deadline misses [17].

Alesio describe stress test case generation as a search problem over the space of task arrival times. The research search for worst case scenarios maximizing deadline misses where each scenario characterizes a test case. The paper combine two strategies, GA and Constraint Programming (CP). The results show that, in comparison with GA and CP in isolation, GA+CP achieves nearly the same effectiveness as CP and the same efficiency and solution diversity as GA, thus combining the advantages of the two strategies. Alesio concludes that a combined GA+CP approach to stress testing is more likely to scale to large and complex systems [19].

B. *Search-Based Stress Testing on Industrial systems*

Usually, the application of Search-Based Stress Testing on non safety-critical systems deals with the generation of test cases that causes Service Level Agreements violations.

Tracey et al. [32] used simulated annealing (SA) to test four simple programs. The results of the research presented that the use of SA was more effective with larger parameter space. The authors highlighted the need of a detailed comparison of various optimization techniques to explore WCET and BCET of the of the system under test [32].

Di Penta et al. [30] used GA to create test data that violated QoS constraints causing SLA violations. The generated test data included combinations of inputs. The approach was applied to two case studies. The first case study was an audio processing workflow. The second case study, a service producing charts [30].

Gois et al. proposes an hybrid metaheuristic approach using genetic algorithms, simulated annealing, and tabu search algorithms to perform stress testing. A tool named IAdapter, a JMeter plugin used for performing search-based stress tests, was developed. Two experiments were performed to validate the solution. In the first experiment, the signed-rank Wilcoxon non- parametrical procedure was used for comparing the results. The significant level adopted was 0.05. The procedure showed that there was a significant improvement in the results with the Hybrid Metaheuristic approach. In the second experiment, the whole process of stress and performance tests, which took 3 days and about 1800 executions, was carried out without the need for monitoring by a test designer. The tool automatically selected the next scenarios to be run up to the limit of six generations previously established [8].

V. NSGA-II Multiobjective heuristics

Many real optimization problems require optimizing multiple conflicting objectives with each other. There is no single optimal solution, but a set of alternative solutions. The objectives that have to be optimised are often in competition with one another and may be contradictory; we may find ourselves trying to balance the different optimisation objectives of several different goals [35] [36].

The image of all the efficient solutions for is called Pareto front or Pareto curve or surface. The shape of the Pareto surface indicates the nature of the trade-off between the different objective functions. An example of a Pareto curve is reported in Fig. 1. Multiobjective optimization methods have as main purposes to minimize the distance between the non-dominated front and the Pareto optimal front and find a set of solutions that are as diverse as possible.

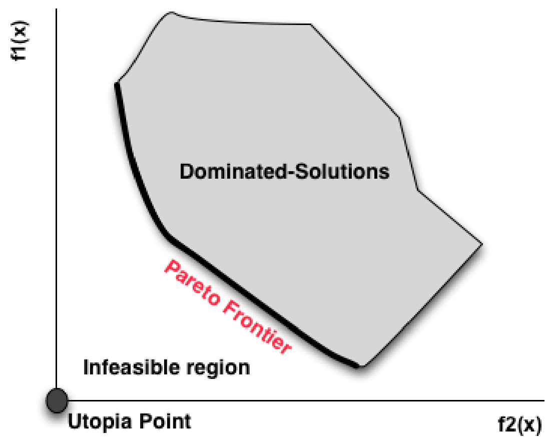


Figure 1: An mimization pareto front example

What distinguishes multi-objective EAs from singleobjective metaheuristics is how they rank and select individuals in the population. If there is only one objective, individuals are naturally ranked according to this objective, and it is clear which individuals are best and should be selected as parents. In case of multiple objectives, it is still necessary to rank the individuals, but it is no longer obvious how to do this. Most people probably agree that a good approximation to the Pareto front is characterized by:

- a small distance of the solutions to the true Pareto frontier,
- a wide range of solutions, i.e., an approximation of the extreme values, and
- a good distribution of solutions, i.e., an even spread along the Pareto frontier.

Multi-objective metaheuristics rank individuals according to how much they contribute to the above goals.

The implementation adapted for this paper is based on the NSGA-II algorithm. Deb et al. proposed the NSGA II taking into account the need to reduce computational complexity in non-dominated classification, introduce elitism and eliminate subjectivity in the allocation of the sharing parameter [37]. NSGA II (Non-dominated Sorting Genetic Algorithm II) is a multiobjective algorithm, based on Genetic Algorithms and that implements the concept of Dominance, that is, to classify the Total Population in fronts according to the degree of dominance. According to NSGA II, the individuals that are located on the first front are considered the best solutions of that generation, while in the last front are the worst. Using this concept, one can find more consistent results, points closer to the Pareto region, and that are better adapted to the type of problem.

The NSGA algorithm II applies a fitness evaluation in an initial population (Figure 2- ❶ and ❷). The population are ranked using multiple tournament selection, which consists of comparing two solutions (Figure 2- ❸). In order to estimate the density of the solutions surrounding a particular solution in the population, the common distance between the previous solution and the posterior solution is calculated over each of the objectives. This distance serves as an estimate of the size of the largest cuboid that includes solution i without including any other solution of the population. A solution i wins another solution if:

- Solution i has a better rank, then $Rank_i < Rank_j$.
- Both solutions have the same rank, but i has a Distance better than j, then $Rank_i = Rank_j$ and $Distance_i > Distance_j$.

At the end of each analysis a certain group of individuals are classified as belonging to a specific category called the front and upon completion of the classification process all individuals will be inserted into one of the n fronts. Front 1 is made up of all non-dominated solutions. Front 2 can be achieved by considering all non-dominated solutions excluding solutions from front 1. For the determination of front 3, solutions previously classified on front 1 and 2 are excluded, and so on until all individuals have been Classified on some front.

After selection, recombination and mutation are performed as in conventional Genetic Algorithms (Figure 2- ❹). The two sets (father and son of the same dimension) are united in a single population (dimension 2) and the classification is applied in dominance fronts. In this way, elitism is guaranteed preserving the best solutions (fronts not dominated) in the later population (Figure 2- ❻).

However, not all fronts can be included in the new population. Thus, Deb et al. proposed a method called crowd distance , which combines the fronts not included in the set, to compose the last spaces of the current population, guaranteeing the diversity of the population [37]. The NSGA-II algorithm creates a set of front lines, each front containing only non–dominating solutions. Within a front, individuals are rewarded for being ‘spread out’. The algorithm also ensures that the lowest ranked individual of a front still has a better fitness value than the highest ranked individual of the next front [38].

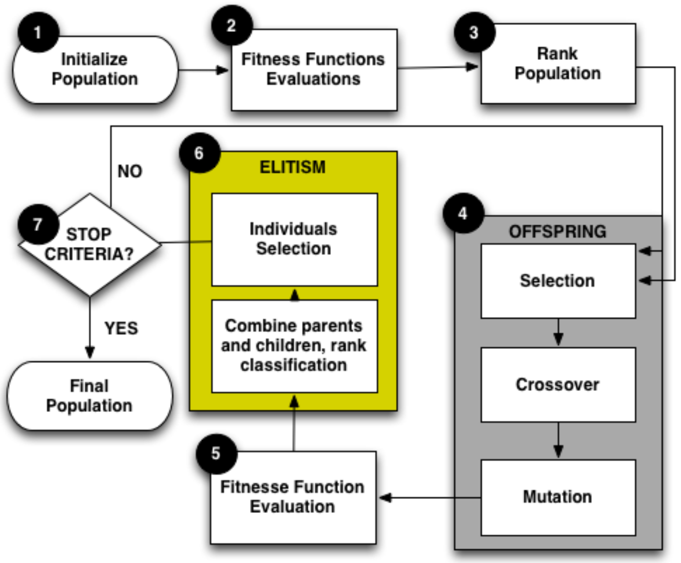


Figure 2: NSGA-II Algorithm

VI. Search-based stress testing applications using NSGA-II

The proposed solution makes it possible to create a model that perform search-based stress tests with two distinct objectives. In the solution, each workload represent an individual in the search space. The solution implements the NSGA-II algorithm in IAdapter plugin. IAdapter is a JMeter plugin designed to perform search-based stress tests. The plugin is available at www.github.com/naubergois/newiadapter. The NSGA-II implementation used an adapted implementation of the jMetal framework (http://jmetal.sourceforge.net/). The next subsections present details about the solution life cycle, the solution representation, the crossover and mutant operators and the IAdapter Components.

A. Solution life cycle

Figure. 3 presents the proposed solution life cycle. Given an initial population (Figure 3 -❶), the NSGA-II algorithm implementation receive a set of workloads (Figure 3 -❷). The

NSGA-II implementation generates a new set of individuals based on crossover or/and mutant operators (Figure 3 -❸). JMeterEngine run each workload (Figure 3 -❺) and the NSGA-II algorithm ranks and classifies each workload based on objective functions (Figure 3 -❻). After all these steps the cycle begins until the maximum number of generations it is reached (Figure 3 -❼).

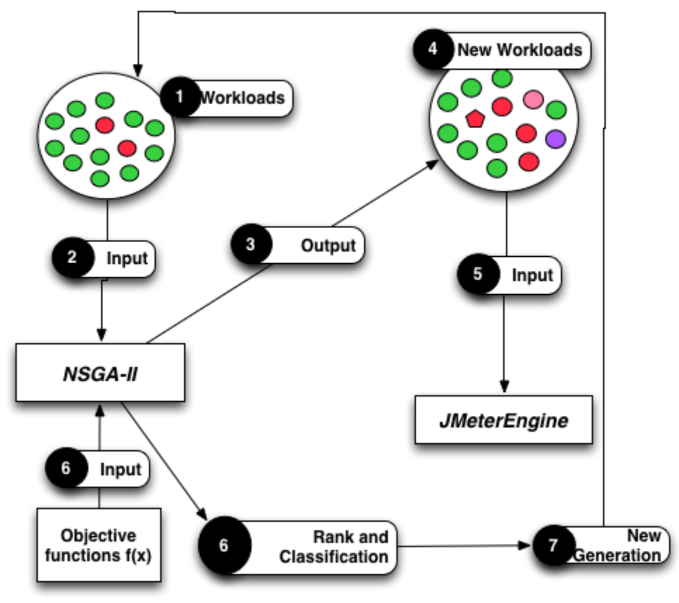


Figure 3: Test module life cycle.

B. Solution representation

Figure. 4 presents the solution representation and an example using the crossover operation. In the example, solution 1 (Figure 4 -❸) has the Login scenario with 2 users, the Search scenario with 4 users, Include scenario with 1 user and the Delete scenario with 2 users. After the crossover operation with solution 2 (Figure 4 -❹), We obtain a solution with the Login scenario with 2 users, the Search scenario with 4 users, the Update scenario with 3 users and the Include scenario with 5 users (Figure 4 -❺). Figure. 4 -❻ shows the strategy used by the proposed solution to obtain the neighbors for the Tabu search and simulated annealing algorithms. The neighbors are obtained by the modification of a single position (scenario or number of users) in the vector.

C. IAdapter Components

WorkLoadThreadGroup is a component that creates an initial population and configures the algorithms used in IAdapter. Fig. 5 presents the main screen of the WorkLoadThreadGroup component. The component has a name ❶, a set of configuration tabs ❷, a list of individuals by generation ❸, a button to generate an initial population ❹, and a button to export the results ❺.

WorkLoadThreadGroup component uses the GeneticAlgorithm, TabuSearch and SimulateAnnealing classes. The WorkLoadSaver component is responsible for saving all data in the

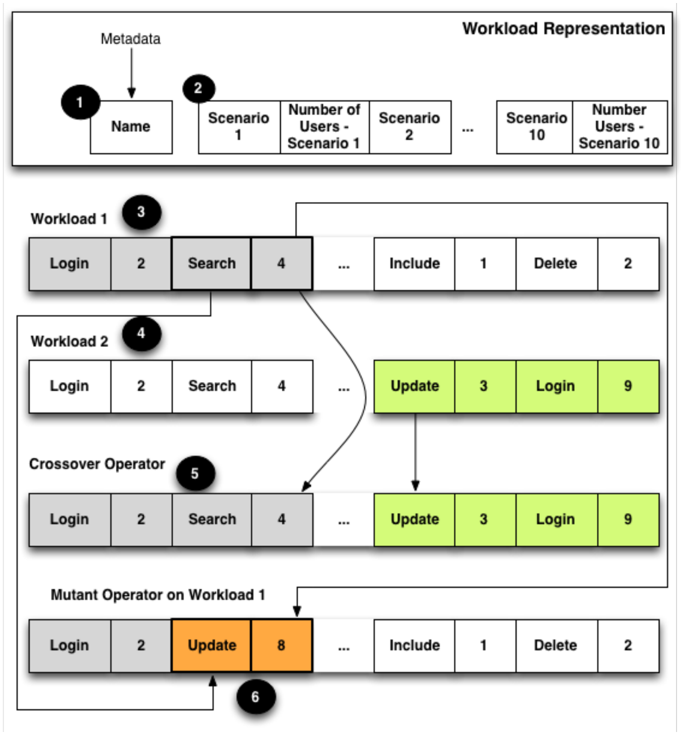
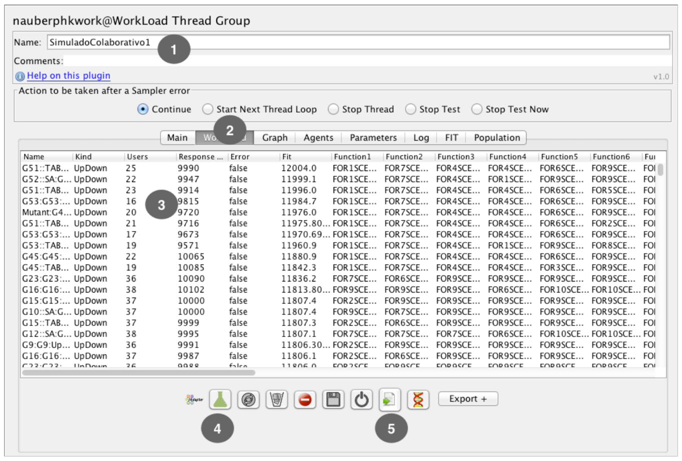


Figure 4: Solution representation, crossover and mutant operators

database. The operation of the component only requires its inclusion in the test script.

WorkLoadController represents a scenario of the test. All actions necessary to test an application should be included in this component. All instances of the component need to login into the application under test and bring the application back to its original state.



nauberphkwork@WorkLoad Thread Group

Name: SimuladoColaborativo1

Comments:

Help on this plugin

v1.0

Action to be taken after a Sampler error

[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Name	Kind	Users	Response ...	Error	Fit	Function1	Function2	Function3	Function4	Function5	Function6	Fu
G51::TAB...	UpDown	25	9990	false	12004.0	FOR1SCE...	FOR7SCE...	FOR4SCE...	FOR4SCE...	FOR6SCE...	FOR9SCE...	FO
G52::SA:G...	UpDown	22	9947	false	11999.1	FOR1SCE...	FOR7SCE...	FOR4SCE...	FOR1SCE...	FOR9SCE...	FOR9SCE...	FO
G51::TAB...	UpDown	23	9914	false	11996.0	FOR1SCE...	FOR7SCE...	FOR4SCE...	FOR4SCE...	FOR6SCE...	FOR5SCE...	FO
G53:G53:...	UpDown	16	9815	false	11984.7	FOR1SCE...	FOR7SCE...	FOR4SCE...	FOR4SCE...	FOR6SCE...	FOR9SCE...	FO
Mutant:G4...	UpDown	20	9720	false	11976.0	FOR1SCE...	FOR7SCE...	FOR4SCE...	FOR4SCE...	FOR6SCE...	FOR9SCE...	FO
G51::TAB...	UpDown	21	9716	false	11975.80...	FOR1SCE...	FOR7SCE...	FOR4SCE...	FOR4SCE...	FOR6SCE...	FOR2SCE...	FO
G53:G53:...	UpDown	17	9673	false	11970.69...	FOR1SCE...	FOR7SCE...	FOR4SCE...	FOR4SCE...	FOR6SCE...	FOR9SCE...	FO
G53::TAB...	UpDown	19	9571	false	11960.9	FOR1SCE...	FOR7SCE...	FOR4SCE...	FOR1SCE...	FOR9SCE...	FOR8SCE...	FO
G45:G45:...	UpDown	22	10065	false	11880.9	FOR1SCE...	FOR7SCE...	FOR4SCE...	FOR4SCE...	FOR6SCE...	FOR9SCE...	FO
G45::TAB...	UpDown	19	10085	false	11842.3	FOR1SCE...	FOR7SCE...	FOR4SCE...	FOR4SCE...	FOR3SCE...	FOR9SCE...	FO
G23:G23:...	UpDown	36	10090	false	11836.2	FOR7SCE...	FOR6SCE...	FOR9SCE...	FOR9SCE...	FOR9SCE...	FOR9SCE...	FO
G16:G16:...	UpDown	38	10102	false	11813.80...	FOR9SCE...	FOR9SCE...	FOR9SCE...	FOR6SCE...	FOR10SCE...	FOR10SCE...	FO
G15:G15:...	UpDown	37	10000	false	11807.4	FOR2SCE...	FOR9SCE...	FOR9SCE...	FOR9SCE...	FOR9SCE...	FOR9SCE...	FO
G10::SA:G...	UpDown	37	10000	false	11807.4	FOR9SCE...	FOR7SCE...	FOR9SCE...	FOR9SCE...	FOR9SCE...	FOR9SCE...	FO
G15::TAB...	UpDown	37	9999	false	11807.3	FOR2SCE...	FOR6SCE...	FOR9SCE...	FOR9SCE...	FOR9SCE...	FOR9SCE...	FO
G12::SA:G...	UpDown	38	9995	false	11807.1	FOR7SCE...	FOR7SCE...	FOR7SCE...	FOR6SCE...	FOR10SCE...	FOR10SCE...	FO
G9:G9:Up...	UpDown	36	9991	false	11806.30...	FOR2SCE...	FOR9SCE...	FOR9SCE...	FOR9SCE...	FOR9SCE...	FOR9SCE...	FO
G16:G16:...	UpDown	37	9987	false	11806.1	FOR2SCE...	FOR6SCE...	FOR9SCE...	FOR9SCE...	FOR9SCE...	FOR9SCE...	FO
G23:G23:...	UpDown	36	9988	false	11806.0	FOR2SCE	FOR9SCE	FOR9SCE	FOR9SCE	FOR9SCE	FOR9SCE	FO

Figure 5: WorkLoadThreadGroup component

VII. Experimental Results

In this section, We present the result of the experiment which we carried out to verify the multiobjective NSGA-II implementation. The experiment was conducted to validate the use of NSGA-II multiobjective algorithm with a real implemented application. The chosen application was the JPetStore, available at https://hub.docker.com/r/pocking/jpetstore/. The maximum tolerated response time in the test was 500 seconds. The whole process of stress tests, which run for 3 days and 492 executions, was carried out without the need for monitoring by a test designer. The tool automatically selected the next scenarios to be run up to the limit of 123 generations previously established by algorithm. The experiments use the follow application features:

- Enter in the Catalog: the application presents the catalog of pets.
- Fish: The application shows the recorded fish items.
- Register: a new user is registered into the system.
- Dogs: The application shows the recorded dogs supplies.
- Shopping Cart: the application displays the shopping cart.
- Add or Remove in Shopping Cart: the application adds and removes items from shopping cart.

The experiments used an initial population of 4 individuals by metaheuristic. The genetic algorithm used the top 10 individuals from each generation in the crossover operation The mutation operation was applied to 10% of the population on each generation. The objective functions applied is intended to minimize the number of users and maximize the response time of the scenarios being tested. A penalty is applied when an application under test takes a longer time to respond than the maximum response time expected.

The experiment used the following objective equations:

$$objectivefunction1 = -3 * numberOfUsers \\ -penalty \tag{1}$$

$$objectivefunction2 = responsetime \\ -penalty \tag{2}$$

The first objective function seeks to find workloads with greater number of users and shorter response time. The second objective function seeks to find workloads with fewer users and longer response times. The penalty is calculated by the follow equation:

$$penalty = 100 * \Delta \\ \Delta = (t_{CurrentResponseTime} - t_{MaximumResponseTimeExpected}) \tag{3}$$

A. Experiment Research Questions

The following research question is addressed:

- Does the NSGA-II algorithm find relevant workload scenarios according to the two test objectives?

B. Variables

The independent variable is the NSGA-II algorithm used in the test. The dependent variables are: the optimal workload scenario found by the algorithm.

Table II: Pareto Frontier workloads results

N.	OBJ. 1	OBJ. 2	Dogs Users	Enter Catalog Users	Fish Users	Register Users	Add Rem. Cart	Cart Users
❶	-3	245	1					
❷	-33	400	1	7		3		
❸	-87	416	5	15	4	5		
❹	-102	434	16	17				1
❺	-105	436	15	4	8	3	5	
❻	-111	472	7	13	7	3	7	
❼	-141	493	7	11	11	7	7	4
❽	-112	496	6		12	8	19	9
❾	-255	499		54	12		7	12

C. Hypotheses

- With regard to multi-objectives applied in the experiment:
  - H_0 (A null hypothesis) :The NSGA-II doesn't found workloads that meet the two objective functions used in the experiment.
  - H_1 : The NSGA-II algorithm found workloads that meet the two objective functions used in the experiment.

D. Results

Fig. 6 and Table II present the results obtained in the experiment. The experiment found 9 optimal workloads (Pareto Frontier) that present a lower number of users with high response times. The workload number 1 with a single user accessing the dog scenario presented the response time of 245 seconds. The workload number 2 with a single user accessing the dog scenario, 7 users accessing the Enter Catalog feature and 3 users in register functionality presented the response time of 400 seconds.

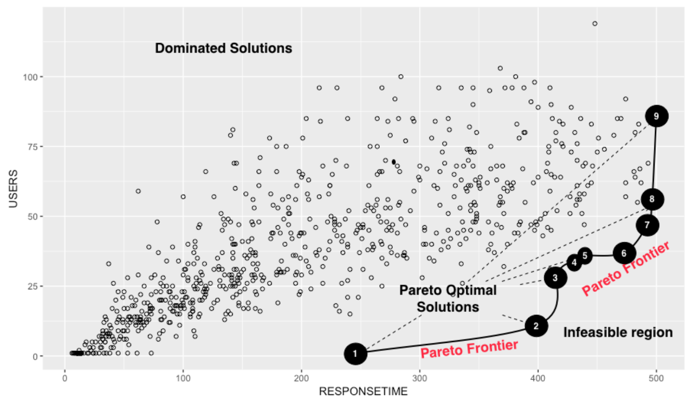


Figure 6: Experiment Pareto Frontier

E. Threats to validity

- Construct Validity: In this work, we just evaluate the use of one multiobjective algorithm. However, several multiobjective algorithms could be applied. There is still a reasonable distance between the Pareto frontier and the data obtained for the second objective, and more experiments are needed to validate the results.

- Conclusion Validity: The workloads found by NSGA-II may be the result of the elitist strategy used in the algorithm that can be applied in the genetic algorithm.

VIII. Conclusion

The experiment verified the use of an multi-objective algorithm in search-based stress testing problem. A tool named IAdapter, a JMeter plugin for performing search-based load tests, was extended. One experiment was conducted to validate the proposed approach. The experiment use NSGA-II algorithm to discover application scenarios where there is a high response time for a small number of users. The whole process of stress tests, which run for 3 days and 492 executions, was carried out without the need for monitoring by a test designer. The experiment found 9 optimal workloads that present a lower number of users with high response times. The results of the experiment can help in the decision making of the service levels to be defined for the application. Future works intend to research the use of other algorithms such as SPEA2, PAES, PESA-II.

References

[1] D. Draheim, J. Grundy, J. Hosking, C. Lutteroth, and G. Weber, "Realistic load testing of Web applications," in *Conference on Software Maintenance and Reengineering (CSMR'06)*, 2006.

[2] Z. Jiang, "Automated analysis of load testing results," Ph.D. dissertation, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1831726

[3] I. Molyneaux, *The Art of Application Performance Testing: Help for Programmers and Quality Assurance*, 1st ed. "O'Reilly Media, Inc.", Jan. 2009.

[4] A. Wert, M. Oehler, C. Heger, and R. Farahbod, "Automatic detection of performance anti-patterns in inter-component communications," *QoSA 2014 - Proceedings of the 10th International ACM SIGSOFT Conference on Quality of Software Architectures (Part of CompArch 2014)*, pp. 3–12, 2014. [Online]. Available: http://dx.doi.org/10.1145/2602576.2602579

[5] W. Afzal, R. Torkar, and R. Feldt, "A systematic review of search-based testing for non-functional system properties," *Information and Software Technology*, vol. 51, no. 6, pp. 957–976, 2009.

[6] G. Gay, "Challenges in Using Search-Based Test Generation to Identify Real Faults in Mockito," pp. 1–6.

[7] M. O. Sullivan, S. Vössner, J. Wegener, and D.-b. Ag, "Testing Temporal Correctness of Real-Time Systems — A New Approach Using Genetic Algorithms and Cluster Analysis —," pp. 1–20.

[8] N. Gois, P. Porfirio, A. Coelho, and T. Barbosa, "Improving Stress Search Based Testing using a Hybrid Metaheuristic Approach," in *Proceedings of the 2016 Latin American Computing Conference (CLEI)*, 2016, pp. 718–728.

[9] C. Sandler, T. Badgett, and T. Thomas, "The Art of Software Testing," p. 200, Sep. 2004.

[10] M. Corporation, "Performance Testing Guidance for Web Applications," United States?, p. 288, Nov. 2007. [Online]. Available: http://www.amazon.com/Performance-Testing-Guidance-Web-Applications/dp/0735625700http://msdn.microsoft.com/en-us/library/bb924375.aspx

[11] G. a. Di Lucca and A. R. Fasolino, "Testing Web-based applications: The state of the art and future trends," *Information and Software Technology*, vol. 48, pp. 1172–1186, 2006.

[12] D. G. Feitelson, *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge University Press, 2013.

[13] M. C. Gonçalves, "Um Processo de Inferência de Desempenho para Apoiar o Planejamento da Capacidade de Aplicações na Nuvem Um Processo de Inferência de Desempenho para Apoiar o Planejamento da Capacidade de Aplicações na Nuvem," 2014.

[14] N. J. Tracey, "A search-based automated test-data generation framework for safety-critical software," Ph.D. dissertation, Citeseer, 2000.

[15] V. Garousi, "Traffic-aware Stress Testing of Distributed Real-Time Systems based on UML Models using Genetic Algorithms," Ph.D. dissertation, 2006.

[16] ——, "A Genetic Algorithm-Based Stress Test Requirements Generator Tool and Its Empirical Evaluation," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 778–797, Nov. 2010.

[17] S. Di Alesio, S. Nejati, L. Briand, and A. Gotlieb, "Stress testing of task deadlines: A constraint programming approach," *IEEE Xplore*, pp. 158–167, 2013.

[18] ——, "Worst-Case Scheduling of Software Tasks – A Constraint Optimization Model to Support Performance Testing," *Principles and Practice of Constraint Programming*, pp. 813–830.

[19] S. D. I. Alesio, L. C. Briand, S. Nejati, and A. Gotlieb, "Combining Genetic Algorithms and Constraint Programming," *ACM Transactions on Software Engineering and Methodology*, vol. 25, no. 1, 2015.

[20] J. T. J. Alander, T. Mantere, and P. Turunen, "Genetic Algorithm Based Software Testing," in *Neural Nets and Genetic Algorithms*, 1998.

[21] J. Wegener, H. Sthamer, B. F. Jones, and D. E. Eyres, "Testing real-time systems using genetic algorithms," *Software Quality Journal*, vol. 6, no. 2, pp. 127–135, 1997. [Online]. Available: http://www.springerlink.com/index/uh26067rt3516765.pdf

[22] B. J. J. Wegener, K. Grimm, M. Grochtmann, H. Sthamer, "Systematic testing of real-time systems," *EuroSTAR'96: Proceedings of the Fourth International Conference on Software Testing Analysis and Review*, 1996.

[23] L. C. Briand, Y. Labiche, and M. Shousha, "Stress testing real-time systems with genetic algorithms," *Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05*, p. 1021, 2005.

[24] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "2005., Canfora, G., An approach for QoS-aware service composition based on genetic algorithms."

[25] J. Wegener and M. Grochtmann, "Verifying timing constraints of real-time systems by means of evolutionary testing," *Real-Time Systems*, vol. 15, no. 3, pp. 275–298, 1998.

[26] F. Mueller and J. Wegener, "A comparison of static analysis and evolutionary testing for the verification of timing constraints," *Proceedings. Fourth IEEE Real-Time Technology and Applications Symposium (Cat. No.98TB100245)*, 1998.

[27] P. Puschner and R. Nossal, "Testing the results of static worst-case execution-time analysis," *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*, 1998.

[28] H. Wegener, Joachim and Pitschinetz, Roman and Sthamer, "Automated Testing of Real-Time Tasks," *Proceedings of the 1st International Workshop on Automated Program Analysis, Testing and Verification (WAPATV'00)*, 2000.

[29] H. Gross, B. F. Jones, and D. E. Eyres, "Structural performance measure of evolutionary testing applied to worst-case timing of real-time systems," *Software, IEE Proceedings-*, vol. 147, no. 2, pp. 25–30, 2000.

[30] M. D. Penta, G. Canfora, and G. Esposito, "Search-based testing of service level agreements," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, 2007, pp. 1090–1097.

[31] V. Garousi, "Empirical analysis of a genetic algorithm-based stress test technique," *Proceedings of the 10th annual conference on Genetic and evolutionary computation - GECCO '08*, p. 1743, 2008.

[32] N. J. Tracey, J. a. Clark, and K. C. Mander, "Automated Programme Flaw Finding using Simulated Annealing," 1998.

[33] H. Pohlheim, M. Conrad, and A. Griep, "Evolutionary Safety Testing of Embedded Control Software by Automatically Generating Compact Test Data Sequences," *Analysis*, no. 724, pp. 804—-814, 2005.

[34] M. Shousha, "Performance stress testing of real-time systems using genetic algorithms," Ph.D. dissertation, Carleton University Ottawa, 2003.

[35] M. Harman and P. McMinn, "A theoretical and empirical study of search-based testing: Local, global, and hybrid search," *IEEE Transactions on Software Engineering*, vol. 36, no. 2, pp. 226–247, 2010.

[36] A. A. El-Sawy, M. A. Hussein, E.-S. M. Zaki, and A. A. A. Mousa, "Local search-inspired rough sets for improving multiobjective evolutionary algorithm," *Applied Mathematics*, vol. 5, no. 13, p. 1993, 2014.

[37] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," *Parallel Problem Solving from Nature PPSN VI*, pp. 849–858, 2000. [Online]. Available: http://repository.ias.ac.in/83498/

[38] M. Harman and P. Mcminn, "A Multi – Objective Approach To Search – Based Test Data Generation," *Gecco'07*, pp. 1098–1105, 2007.